Vizualizacija projektnih građevnih blokova broda primjenom programa otvorenog koda linaetal-fsb/d3v

Pavlović, Tomislav

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Tomislav Pavlović

Zagreb, 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Izv. prof. dr. sc. Pero Prebeg, dipl. ing.

Student:

Tomislav Pavlović

Zagreb, 2020.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru dr.sc. Peri Prebegu na brojnim konzultacijama, prijedlozima i korekcijama prilikom izrade završnog rada.

Također se želim zahvaliti svojoj obitelji i prijateljima na razumijevanju, strpljenju i pomoći tijekom studija.

Tomislav Pavlović

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija brodogradnje

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

ZAVRŠNI ZADATAK

Student: **Tomislav Pavlović** Mat. br.: 0035202907

Naslov rada na hrvatskom jeziku: **Vizualizacija projektnih građevnih blokova broda primjenom programa otvorenog koda linaetal-fsb/d3v**

Naslov rada na engleskom jeziku: **Use of open source program linaetal-fsb/d3v for visualization of ship deign building blocks**

Opis zadatka:

Metoda projektnih građevnih blokova (eng. *Design Building Blocks - DBB*) projektantu broda omogućuje separiranje brodskih funkcija u diskretne elemente koje je moguće pozicionirati na odgovarajuća mjesta na brodu. Program otvorenog koda linaetal-fsb/d3v temeljen je na *Qt for Python* tehnologiji, koja omogućuje trodimenzijsku (3D) vizualizaciju geometrijskih entiteta primjenom *OpenGL* tehnologije te jednostavno definiranje grafičkog sučelja za pojedine specifične namjene. U radu je potrebno, proširenjem funkcionalnosti programa otvorenog koda linaetal-fsb/d3v u programskom jeziku Python, omogućiti trodimenzijsku vizualizaciju projektnih građevnih blokova DBB metode zajedno s triangulariziranim modelom forme fregate te interaktivno modificiranje projektnih građevnih blokova.

Zadatak obuhvaća sljedeće:

- upoznavanje s programom otvorenog koda linaetal-fsb/d3v
- upoznavanje s DBB metodom i karakteristikama mogućih vrsta projektnih građevnih blokova broda
- izradu Python modula koji omogućuje 3D vizualizaciju pojedinih paluba unutar triangularizirane forme fregate u programu otvorenog koda linaetal-fsb/d3v
- izradu Python modula koji omogućuje 3D vizualizaciju pojedinih vrsta projektnih građevnih blokova broda u programu otvorenog koda linaetal-fsb/d3v
- izradu Python modula koji omogućuje izračun volumnih karakteristika projektnih građevnih blokova, koji su omeđeni brodskom oplatom zadanom u triangualiziranom obliku
- izradu Python modula koji omogućuje interaktivnu modifikaciju (barem pomicanje, dodavanje i brisanje) projektnih građevnih blokova broda u programu otvorenog koda linaetal-fsb/d3v.

U radu treba navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan:
28. studenog 2019.

Datum predaje rada:
1. rok: 21. veljače 2020.
2. rok (izvanredni): 1. srpnja 2020.
3. rok: 17. rujna 2020.

Predviđeni datumi obrane:
1. rok: 24.2. – 28.2.2020.
2. rok (izvanredni): 3.7.2020.
3. rok: 21.9. - 25.9.2020.

Zadatak zadao:
Doc. dr. sc. Pero Prebeg

Predsjednica Povjerenstva:
Prof. dr. sc. Nastia Degiuli

SADRŽAJ

POPIS SLIKA

SAŽETAK

U ovom radu dan je pregled osnovnih karakteristika programa otvorenog koda *linaetal-fsb/d3v* i metoda projektnih građevnih blokova blokova (eng. *Design Building Blocks - DBB*). U okviru rada izrađeni su *Python* moduli koji omogućuje 3D vizualizaciju pojedinih paluba i projektnih građevnih blokova broda unutar triangularizirane forme broda. Osim toga, omogućeno je i računanje volumnih karakteristika projektnih građevnih blokova, koji su omeđeni brodskom oplatom zadanom u triangualiziranom obliku. Korisniku programa omogućeno je i interaktivno mijenjanje osnovnih karakteristika građevnih blokova. U radu je dan i opis algoritama razvijenih za rezanje mreže bloka pomoću mreže forme, za izračun volumena zatvorene triangulizirane mreže, te algoritmi za osnovnu interaktivnu manipulaciju trianguliziranih mreža.

Ključne riječi: Python, Qt, Openmesh, poligonska mreža, vizualizacija, d3v, DBB

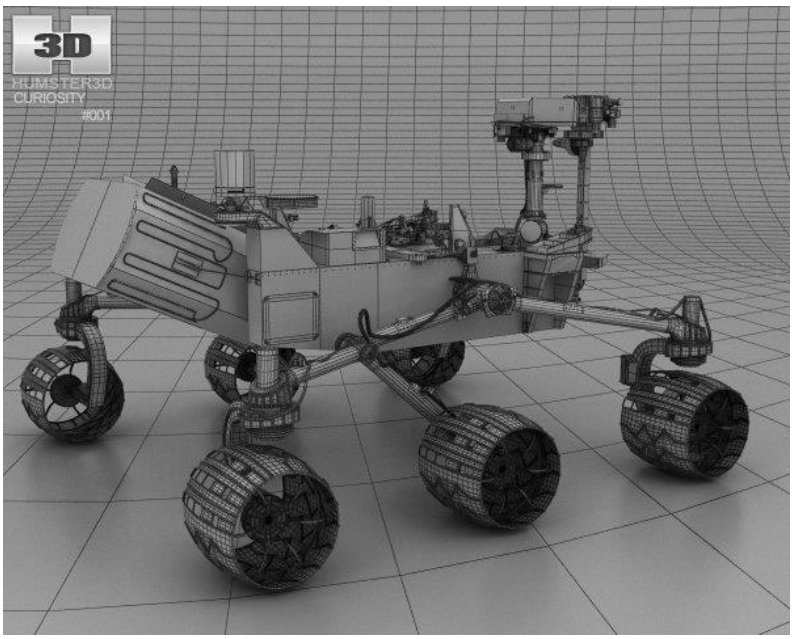
SUMMARY

This paper gives an overview of basic characteristics of the open source linaetal-fsb/d3v program and the Design Building Block (DBB) method. New Python modules were made which allow 3D visualization of decks and design building blocks inside of a triangulated ship hull. Also, volume calculation of design building blocks bounded by the ship's triangulated hull has been implemented. The user can also interactively change the basic characteristics of building blocks. The paper also describes algorithms developed to cut a block mesh using the hull's mesh, closed triangulated mesh volume calculation and algorithms for basic interactive manipulation of triangulated meshes

Key words: Python, Qt, Openmesh, polygon mesh, visualization, d3v, DBB

1. PREGLED RAČUNALNE GRAFIKE, PREDNOSTI PYTHON JEZIKA, QT I POLIGONSKIH MREŽA.

Računalna grafika (RG) je jedna od glavnih grana računalne znanosti koja se bavi generacijom slika i oblika pomoću računala. Termin „Računalna grafika“ su prvi put upotrijebili istraživači Verne Hudson i William Fetter u 1960. Općenito govoreći, RG se bavi gotovo svime na računalu što nije tekst ili zvuk. Njeni začetci počinju u 1950 kao pomoćni sustavi u vojnim radarima, avijaciji i raketama razvijenim tijekom i nakon Drugog svjetskog rata. No u razdoblju od 1980 – 1990 dolazi do eksponencijalne komercijalizacije, koja se može pripisati razvitku i popularnosti osobnih računala i razvitku 3D tehnologija. Daljnjim razvojem raznih tehnologija Računalna grafika je postala nezamjenjiva komponenta modernog društva i moderne infrastrukture. Danas se Računalna grafika koristi gotovo svugdje gdje se nalazi i računalo. Od najobičnijih semafora i natpisa nad uličnim dućanima, IPhonea i osobnih računala do modernih CAD(*Computer Aided Design*) i FEM(*Finite Element Method*) programa i simulacija pomoću kojih smo izradili objekte poslane na druge planete. [1]



Slika 1. Mars curiosity rover 3D model ([2])

Ovaj rad se bavi CAD aspektom Računalne grafike, te će se kroz ovo poglavlje objasniti neke osnovne programe i termine potrebne za izvršavanje danog zadatka.

1.1. Python jezik

Python je interpretiran jezik visoke razine i opće namjere kojeg je razvio 1991 godine Guido Van Rossum i dobio je svoje ime po popularnoj televizijskoj seriji *Monty Python*. *Python* je karakteriziran veoma čitljivim, učinkovitim, jasnim i logičnim kodom za velike i male projekte. *Python* podržava objektno, strukturno i aspektno orijentirano programiranje. Također u sebi ima integriran velik broj modula koji još dodatno povećava njegovu fleksibilnost.[3] *Python* svoju čitkost i jasnoću dobiva iz činjenice da je *Python* interpretiran jezik(jezik koji je pisan u nekom drugom jeziku), a ne kompiliran jezik (jezik koji direktno prevodi svoj kod u strojni jezik) [4]. Algoritmi potrebni za izvršavanje zadatka su pisani uglavnom pomoću *Python* modula *Openmesh* i *Numpy*.

1.2. Qt for Python(Pyside2)

Qt je više-platformna aplikacija za izradu grafičkih korisničkih sučelja i grafičkih elemenata. *Qt* sam po sebi nije programski jezik, nego okruženje pisano u *C++* jeziku. U sebi sadrži alate *Qt Creator* i *Qt Designer* za brzo i jasno stvaranje grafičkih sučelja.[5]

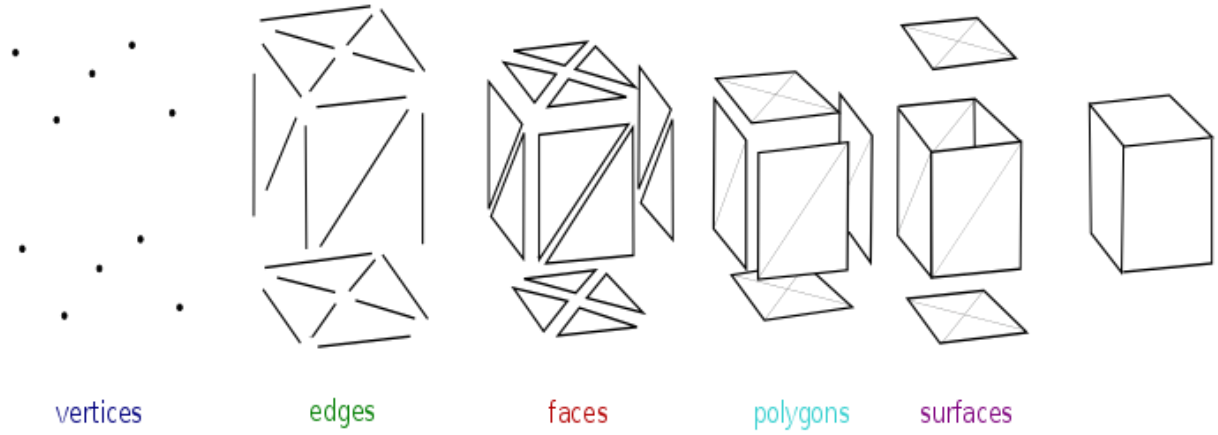
Trenutno podržavane platforme su:

1. Linux
2. OS X
3. Windows
4. QNX
5. Android
6. iOS
7. BlackBerry
8. Sailfish OS
9. i drugi

Također, *Qt* u sebi ima *bindinge*(sposobnost jezika da koristi module nekog drugog jezika[6]) za više programskih jezika. Sposobnosti *Qt*-a da radi na bilo kojoj podržavanoj platformi i na bilo kojem podržavanom jeziku su pretvorile *Qt* u industrijski standard za izradu grafičkih sučelja. Grafičko sučelje i vizualizacija poligonskih mreža u d3v-u su izrađena pomoću *Qt*-a.

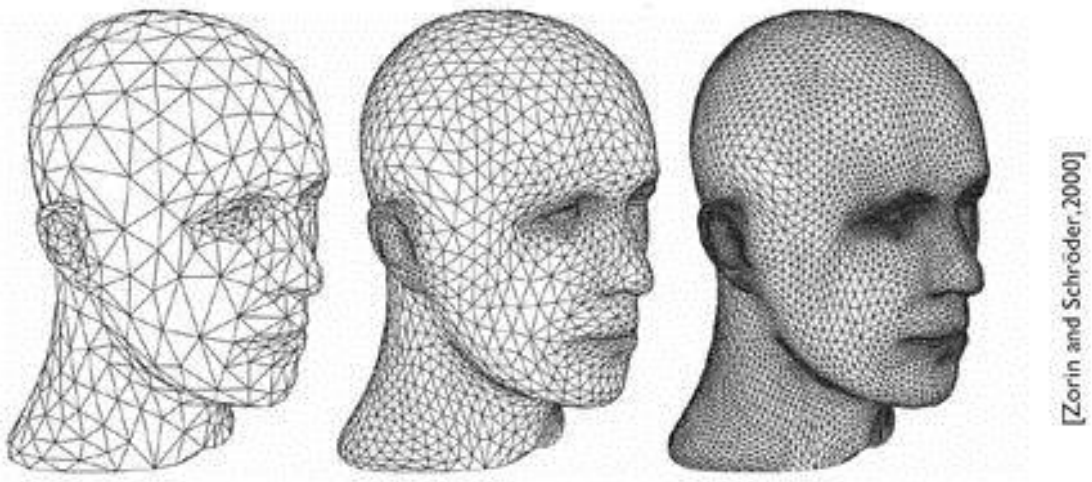
1.3. Poligonska Mreža

U 3D Računalnoj grafici i modeliranju poligonska mreža je skup točaka(*vertices*), rubova(*edges*) i lica(*faces*) koji zajedno definiraju Poliedarski objekt. Lica se mogu raditi od trokuta, četverokuta ili jednostavnih n-terokuta.[7]



Slika 2. Elementi Mreže ([7])

Općenito govoreći mreže sa većom gustoćom elemenata su detaljnije, ali su operacije sa njima sporije jer algoritmi moraju proći kroz više elemenata.



Slika 3. Promjena gustoće mreže ([8])

1.3.1. Openmesh modul

Openmesh je modul sa jednostavnom i učinkovitom strukturom podataka za korištenje poligonskih mreža i korisnik ima mogućnost izrade vlastite vrste mreže potrebne za specifične aplikacije. To se postiže dinamičnim svojstvima mreže koje korisnik može mijenjati za vrijeme izvršavanja programa po volji.

Glavne prednosti koda *Openmesha*:

1. Ne postoje ograničenja za trokutaste mreže, korištenje općih poligonskih mreža.
2. Eksplicitno predstavljanje točaka, rubova, polurubova i lica.
3. Lagano pristupanje susjednim elementima točke.
4. Mogućnost izrade ne raznolikih točaka (dva lica se susreću u samo jednoj točki)

Također sadrži poveznice za *Python* koji će se koristiti u ovom radu [9], *Openmesh* je usko vezan sa *Python* modulom *Numpy*.

1.3.2. Numpy modul za Python

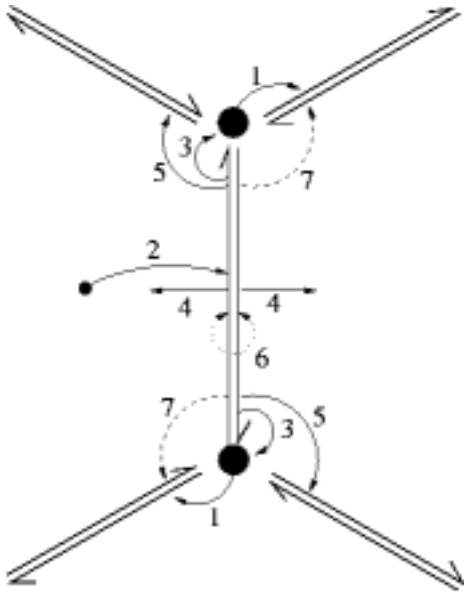
Numpy je temeljni paket otvorenog koda za znanstveno računarstvo u *Python* jeziku. Modul ima mogućnost rada sa višedimenzijskim poljima i objektima koji proizlaze iz njih (npr. matrice) [10].

Moguće operacije sa nizovima:

1. Matematičke
2. Logičke
3. Manipulacije oblika
4. Redanje
5. I/O
6. Fourierove transformacije
7. Linearna algebra
8. Statistika
9. Simulacije Slučajnosti

1.3.3. Polurubna struktura podataka u Openmehsa

Openmesh koristi polurubnu (*halfedge*) strukturu zapisivanja elemenata mreže (točke, lica i rubovi) i njihovu međusobnu povezanost [11]. Za razliku od struktura zasnivane na licima, koje svoju povezanost na točke i susjedna lica spremaju u lica, strukture zasnivane na rubovima te informacije spremaju u rubove. Svaki rub referencira svoje dvije točke, lica kojima pripada i sljedeća dva ruba u tim lica. Ako se taj rub podijeli na dva nova ruba sa suprotnim orijentacijama dobiva se polurubna struktura:



Slika 4. Polurubna Struktura ([11])

- Svaki *verteks* referencira jedan izlazni polurub, tj. polurub koji počine u toj točki (1)
- Svako lice referencira jedan od polurubova koji ga okružuje (2)
- Svaki polurub daje *handle*(referencu) za:
  - Točku prema kojem je usmjeren (3)
  - Lice kojem pripada(4)
  - Slijedeći polurub u licu (suprotno od kazaljke na satu) (5)
  - Suprotni polurub (6)
  - (proizvoljno: prošli polurub u licu (7))

Pomoću poveznica između elemenata moguće je kružiti po licima u svrhu brojanja svih njegovih točaka, polurubova, rubova ili susjednih lica. Ova funkcionalnost Openmesha se izražava preko Cirkulatora i Iteratora mreže.

1.3.4. Pristupi izrade mreža

Postoje dva osnovna pristupa izradi mreže:

- Dodavanje elemenata praznoj mreži
- Definiranje povezanosti točaka prije izrade mreže

U prvom pristupu se generira prazan trokutasti *mesh*(mrežni) objekt *openmesh.TriMesh*(). Postoje i mreže sa četverokutima, ali mreže sa trokutima imaju veću broj razvijenih funkcionalnosti u ovom modulu. Potom se od niza točaka(koji su pohranjeni u *numpy* nizu) generiraju reference na točke te mreže pomoću metode *mesh.add_verteks()*. Nakon toga se od izrađenih referenca točka pomoću metode *mesh.add_face()* generiraju lica. Važno je spomenuti da redoslijed unosa referenci točaka označuje orijentaciju lica, te da je važno osigurati pravilnu polurubnu strukturu, tj. da susjedna lica imaju suprotnu orijentaciju (polurubovi su im suprotno usmjereni). Ukoliko ovaj uvjet nije ispunjen, *Openmesh* javlja grešku. Nakon unosa lica mreža je gotova te se može koristiti.

Sve točke mreže se mogu dobiti pomoću metode *mesh.points()*, koja daje *numpy* niz sa svim točkama u mreži, no također je važno da je pozicija svake točke u tom nizu odgovara vrijednosti indeksa reference točke na koji je vezan. Indeks se može dobiti pomoću metode *verteks_handle.idx()*, a referenca točke se iz indeksa može dobiti pomoću metode *mesh.verteks_handle(idx)*. Analogno se dobivaju indeksi i reference rubova, lica i polurubova. Referenece su važne jer se u njima nalazi puno funkcionalnosti *Openmesha*. Ako se želi iz objekta mreže izvući podatak o povezanosti točaka sa licima poziva se *mesh.face_verteks_indices()* metoda. To je *numpy* niz u kojemu se nalaze indeksi točaka za svako lice koji je u mreži. Također pozicija te vrijednosti u nizu odgovara identitetu reference lica. Ista pravila vrijede za rubove i polurubove, te njihove identitete i reference.

Moguće je dobiti susjedne elemente nekog elementa, npr. za susjedna lica se zove metoda *mesh.face_face_indices()* u kojoj je za svaki identitet lica navedeni identiteti lica s kojima je povezan.

Drugi pristup izradi mreže je da se odmah definiranu indeksi točaka koji su povezani na indekse lica, tj. generirati *numpy* niz, u kojemu je po gornjim pravilima, navedena povezanost između točaka. Potom se generira mrežni objekt *openmesh.TriMesh(points, face_vertex_indices)* sa izrađenim nizovima, dok *Openmesh* iz tih podataka automatski generira sve ostale podatke i povezanosti.

2. UPOZNAVANJE SA LINAETAL-FSB/D3V PROGRAMOM I DBB METODOM

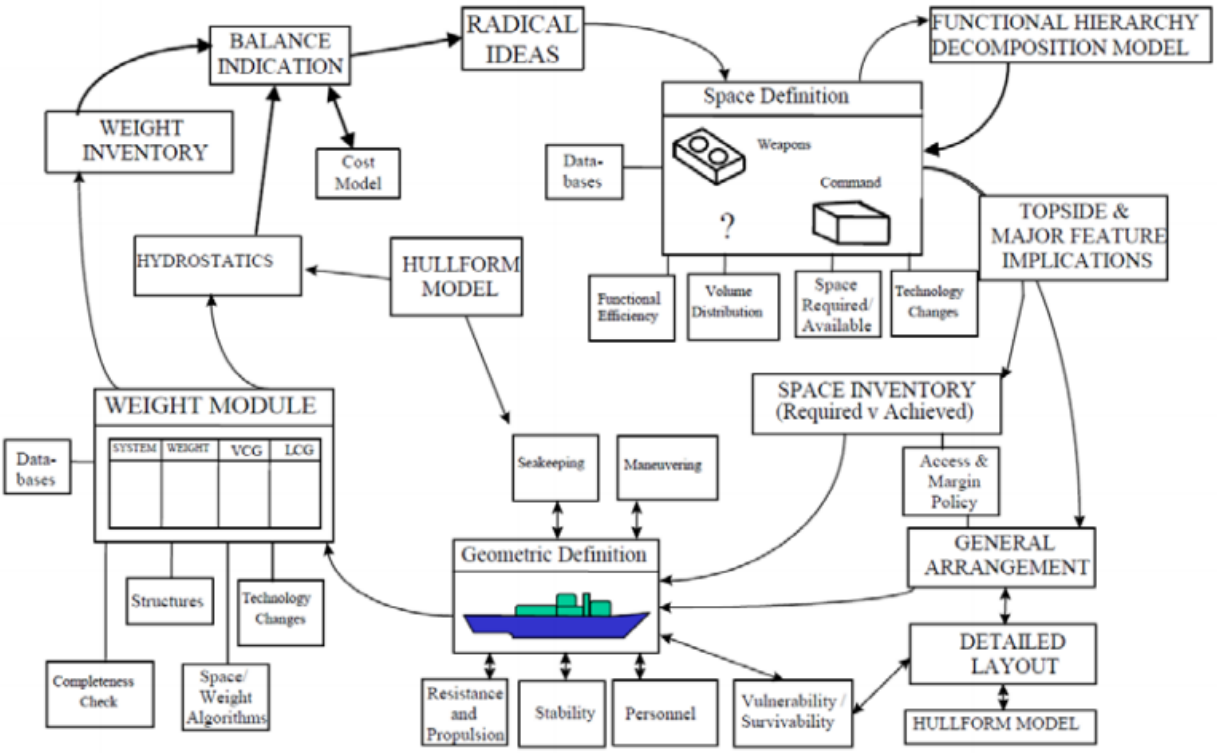
Za vizualizaciju brodske forme i njezinih elemenata korišten je *Linaetal-fsb/d3v* program, te su u njega dodane nove funkcionalnosti potrebne za interaktivnu manipulaciju sa elementima forme i učitavanje DBB (*Design Building Block*) datoteke koja sadrži podatke potrebne za generiranje forme, blokova i paluba u obliku tekstualnih datoteka koje praktično ne zauzimaju nikakvu memoriju na računalu. U daljnjem tekstu su detaljnije objašnjeni *Linaetal-fsb/d3v* program i DBB metodi konstrukcije broda .

2.1. Pregled DBB metode za projektiranje broda

Osnovna ideja iza DBB metode je da se funkcije i podfunkcije broda podijele na diskretne, fizički izvedive elemente. Ti se elementi tada mogu postaviti i pomicati prema željama projektanta. Brodske funkcije su u centru procesa i glavni su pokretatelji metode, za razliku od tradicionalnih sekvencionalnih procesa, gdje se pozicije pojedinih funkcijskih cjelina određuju tek pri kasnijim stadijima projektiranja [12]. Ova metoda nije ovisna o tipu broda, potiče inovaciju i nova rješenja, šireći moguća rješenja smještaja prostora [13].

Pregled koraka DBB metode: [14]

- Potreba za novim početnim dizajnom je stvorena i predlaže se varijanta koja zadovoljava zahtjeve.
- Koristeći nove ideje ili povijesne podatke, generira se niz blokova. Svaki blok sadrži geometrijske i tehničke karakteristike u skladu s predviđenom funkcijom tog bloka.
- Izrađuje se prostor za konstruiranje, te se blokovi unutar njega manipuliraju po želji ili potrebi.
- Cjelokupna ravnoteža strukture i njen performans su istraženi koristeći jednostavne i fleksibilne algoritme i ukoliko je potrebno, vanjske programe za analizu.
- Projektant mijenja konfiguraciju dizajna sve dok nije zadovoljan .
- Provodi se rastavljanje *Design building blokova* u dizajnu na veće detalje u svrhu povećanja točnosti dobivenog rješenja.



Slika 5. Pristup građevnih blokova za površinske brodove ([12])

DBB metoda dopušta projektantu da koristi svoje iskustvo i kreativnost, povećavajući doprinos krajnjem dizajnu, ali odgovornost projektanta. Ovaj pristup je karakteriziran kao „mekan“, jer dopušta velike promjene u odlukama koje su donesene, dizajn se može lagano ažurirati i poboljšati u ovim vrlo ranim koracima, dok je dizajn još uvijek lagano promjenjiv.

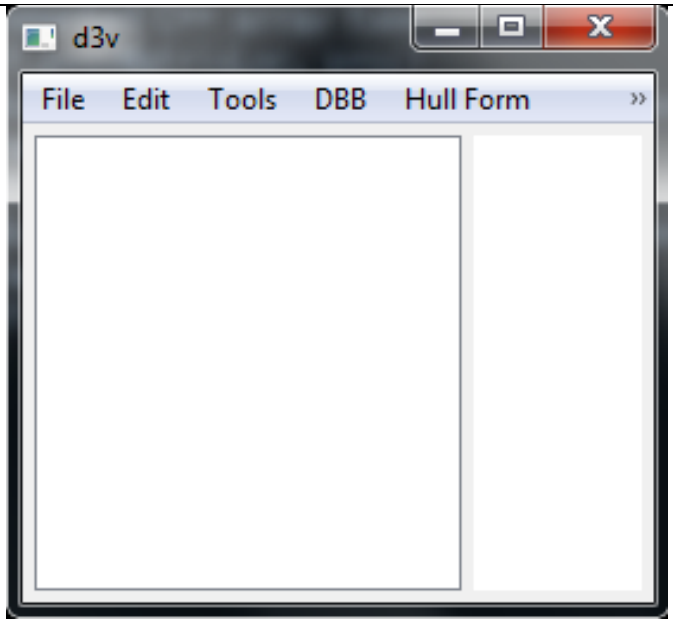
Glavne koraci DBB metode prikazani su na slici 6:

Design Preparation
Selection of Design Style
Topside and Major Feature Design Phase
Design Space Creation
Weapons and Sensor Placement
Engine and Machinery Compartment Placement
Aircraft Systems Sizing and Placement
Superstructure Sizing and Placement
Super Building Block Based Design Phase
Composition of Functional Super Building Blocks
Selection of Design Algorithms
Assessment of Margin Requirements
Placement of Super Building Blocks
Design Balance & Audit
Initial Performance Analysis for Master B.B.
Building Block Based Design Phase
Decomposition of Super Building Blocks by function
Selection of Design Algorithms
Assessment of Margins and Access Policy
Placement of Building Blocks
Design Balance & Audit
Further Performance Analysis for Master B.B.
General Arrangement Phase
Drawing Preparation

Slika 6. Glavni koraci DBB-a ([15])

2.2. Linaetal-fsb/d3v program za vizualizaciju

Linaetal-fsb/d3v (*design visualizer*) je modularna *Python* aplikacija otvorenog programskog koda, prvenstveno namijenjena za 3D vizualizaciju inženjerskih modela u fazi projektiranja. Slobodno je dostupna na poveznici https://github.com/linaetal-fsb/d3v. Inicijalno je nastala kao rezultat dugogodišnje suradnje Fakulteta strojarstva i brodogradnja s USCS.d.o.o te obrtom Linaetal. Struktura programa temeljena je na dvanaestogodišnjem iskustvu razvoja programa *ShipExplorer*. Program je u ranoj fazi razvoja no već je moguća vizualizacija s ograničenim brojem funkcionalnosti. Implementacija je bazirana na bibliotekama *QtForPython (PySide2)* i *OpenMesh*. [16]



Slika 7. Linaetal-fsb/d3v prozor

Funkcionalnosti *Linaetal-fsb/d3v* prije implementacija funkcionalnosti koje su napravljeni u ovom radu su:

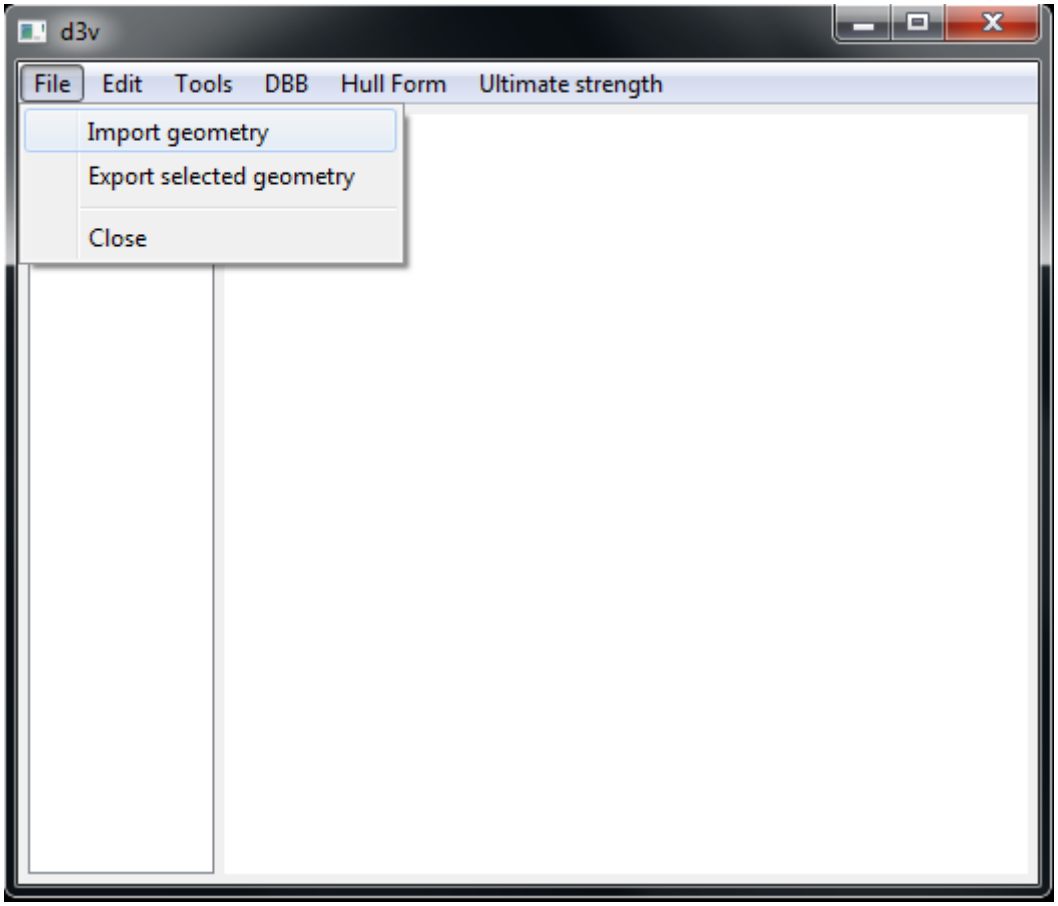
- Vizualizacija mreža u raznim bojama
- Rotacija, translacija i povećavanje/smanjivanje pogleda
- Učitavanje geometrije forme iz datoteka
- Interaktivna selekcija individualnih mreža mišem
- Izračunavanje hidrostatičkih značajki forme
- Dodavanje jednostavnog bloka fiksnih dimenzija

2.3. Moduli Linaetal-fsb/d3v i njihova funkcionalnost

Linaetal-fsb/d3v koristi više modula za ostvarivanje svojih funkcija, no ovdje će se opisati samo oni koji imaju ulogu u trenutnoj funkcionalnosti i imaju važnu ulogu u dodavanju novih. Prvi modul s zove *main.py* i to je modul u kojemu se poziva glavni prozor aplikacije *Linaetal-fsb/d3v*. Modul koji je zadužen za generiranje forme broda iz datoteke je *hullform.py*.

Modul *dbbcommands.py* je zadužen za interakciju između menija u prozoru aplikacije i modulom *dbb.py* i nekih drugih osnovnih funkcionalnosti. U sebi sadrži klasu

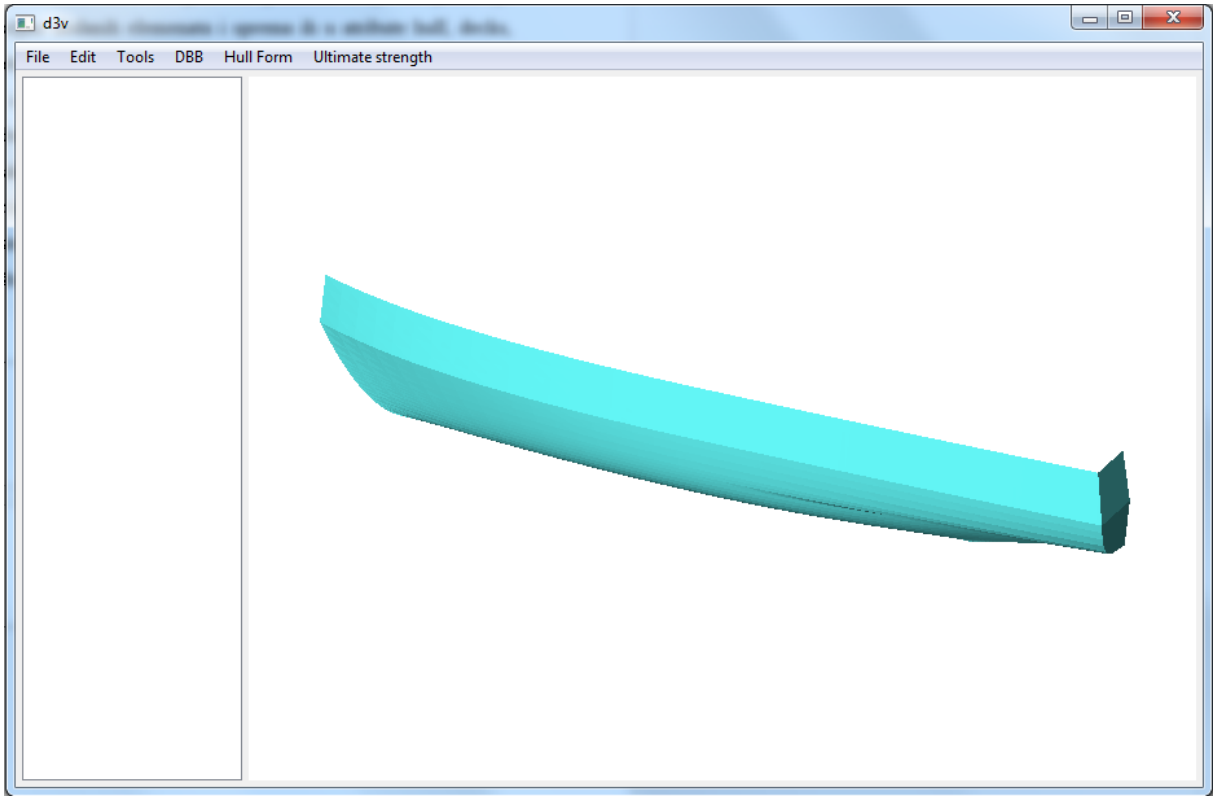
DBBCommand() u kojoj se nalaze padajući meniji u glavnom prozoru, te se tu mogu dodavati novi prozori po potrebi. Također u sebi sadrži akcije koje će se izvršiti prilikom aktivacije naredbi u padajućim menijima. Klasa *DBBImporter()* je zadužena za učitavanje datoteka iz glavnog prozora prilikom aktivacije komande *Import Geometry* i inicijalizaciju klase *dbbproblem().*



Slika 8. Učitavanje geometrije

Klasa *dbbproblem()* nalazi se u modulu *dbb.py* koja sadrži klase za opis blokove, palube i forme. Već spomenuta klasa *dbbproblem()* prilikom svoje inicijalizacije stvara mreže učitanih, ili proizvoljno dodanih elemenata i sprema ih u atribute *hull*, *decks*, *dbbs*. Atribut *hull* sadrži DBB objekt za formu broda, dok atributi *dbbs* i *decks* su liste sa svim DBB objektima blokova i paluba. Konkretno, ti DBB objekti *DBBHullForm()*, *DBBDeck()* i *DBB()* u sebi sadrže bitne informacije potrebne za njihovo definiranje, te metode za generiranje mreže iz tih podataka. Važno je napomenuti da prije implementacija novih funkcionalnosti *DBBDeck klasa* se nije koristila, te nije postojala sposobnost vizualizacije paluba DBB

metodom. Klasa *DBBHullForm()*, pomoću modula *hullform.py*, stvoriti mrežu forme iz učitane datoteke.



Slika 9. Učitana forma broda

3. IMPLEMENTACIJA NOVIH FUNKCIONALNOSTI U LINAETAL-FSB/D3V

Nove funkcije se ostvaruju modificiranjem postojećih modula i stvaranjem *myfunctions.py* i *dbbmenus.py* modula. *Myfunctions.py* je modul koji sadrži algoritme za manipulaciju i generiranje mreža. *Dbbmenus.py* je modul u kojemu su zapisani novi meniji koji se otvaraju prilikom aktivacija nekih komandi u svrhu poboljšavanja korisničkog sučelja. U svrhe proširivanje funkcionalnosti *Linaetal*-fsb/d3v potrebno je implementirati sljedeće nove funkcionalnosti:

- Stvaranje DBB datoteke
- Vizualizacija i generacija blokova DBB metodom
- Vizualizacija i generacija paluba DBB metodom
- Interaktivno dodavanje blokova
- Aproksimacija mreže forme pomoću bloka
- Interaktivno brisanje selektiranih objekata
- Interaktivno pomicanje selektiranih blokova
- Provjera zatvorenosti selektiranih blokova
- Izračun volumena selektiranih blokova

3.1. Sadržaj DBB datoteke

DBB (*Design Building Block*) je datoteka stvorena u svrhu brze i lagane pohrane bitnih podataka za generaciju raznih elemenata broda u obliku trokutastih mreža pomoću *Openmesha* i *Pythona*, te njihove eventualne vizualizacije. Podatci su pohranjeni u csv(*comma seperated values*) formatu koji je jedan od najraširenijih i najjednostavnijih formata, koji postoji još od 1972. Csv datoteke koriste zarez za odvajanje podataka (inače zvan *delimiter*), a svaki red predstavlja novi unos podataka i se mogu ručno mijenjati koristeći bilo kakav editor teksta. [17]

3.1.1. DesignBuildingBlock.dbb

Glavna datoteka je *DesignBuildingBlock.dbb* datoteka u kojoj su pohranjene relativne putanje do ostalih datoteka koje sadrže pojedinačne podatke elemenata.

3.1.2. Shipd.huf

Prva od tih datoteka je *shipd.huf* datoteka koja u sebi sadrži sljedeće varijable:

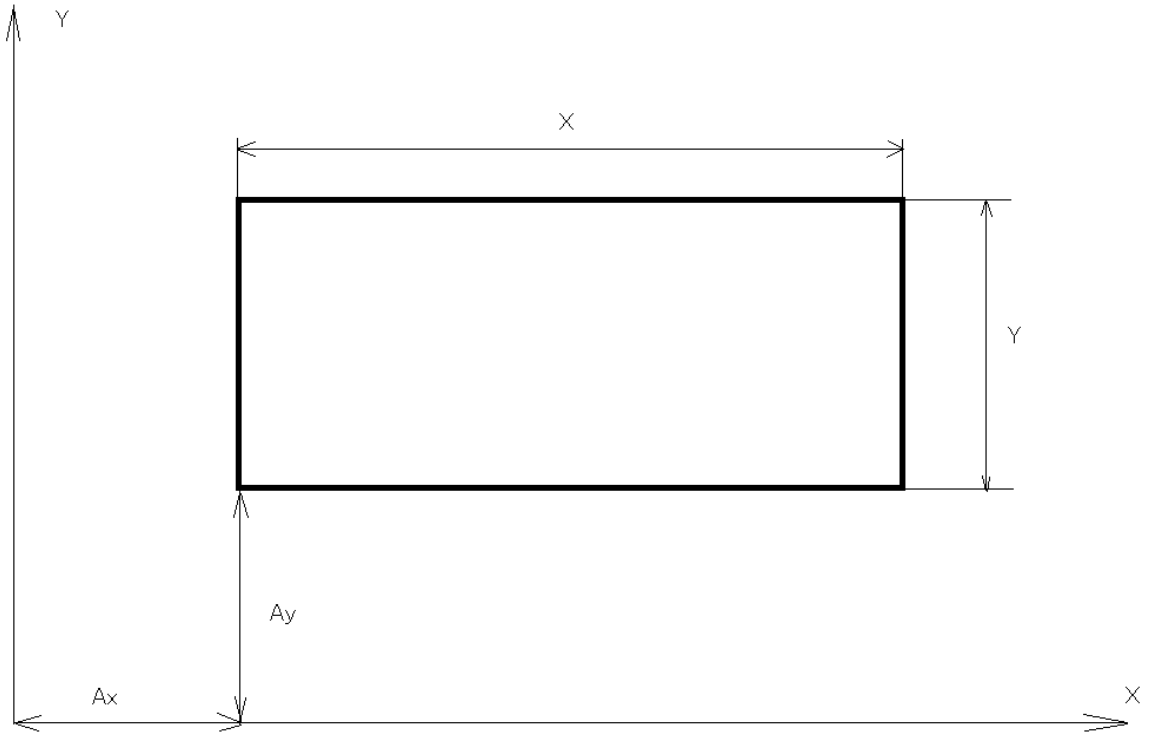
- Id
- Name
- LOA
- BOA
- D
- Bulkheads
- BHPos
- Decks
- DeckPos
- HullData
- MaxSpeed
- MediumDensity
- MediumKinematicViscosity
- BulbArea
- AppMargin
- SeaMargin
- PropCoeff

Iz ovih podataka modul *hullform.py* u *Linaetal-fsb/d3v* programu izrađuje trokutastu mrežu korištenu za vizualizaciju i daljnje računanje.

3.1.3. Block_data.csv

Block_data.csv je datoteka u kojoj su pohranjeni podatci o broju, položaju i dimenzijama pojedinačnih blokova na formi. Prvi red označava imena varijabli, dok svaki red nakon toga označuje novi blok sa pripadnim podacima. Varijable *Ax* i *Ay* označuju položaj vrha bloka koji je odabran za poziciju, dok varijable *x* i *y* označuju dimenzije bloka u pozitivnim smjerovima x i y osi. *Deck* se odnosi na palubu na kojoj se dotični blok nalazi.

Pretpostavljeno je da blokovi leže između dvaju paluba, pa je visina bloka određena razlikom u visini paluba, a pozicija po z koordinati je jednaka visini na kojoj se paluba nalazi. Konkretni brojevi sa pozicijama paluba se nalaze u *shipd.huf* datoteci, pod imenom *DeckPos*.



Slika 10. Block_data format

3.1.4. Unit_block_points.csv i Unit_block_fvi.csv

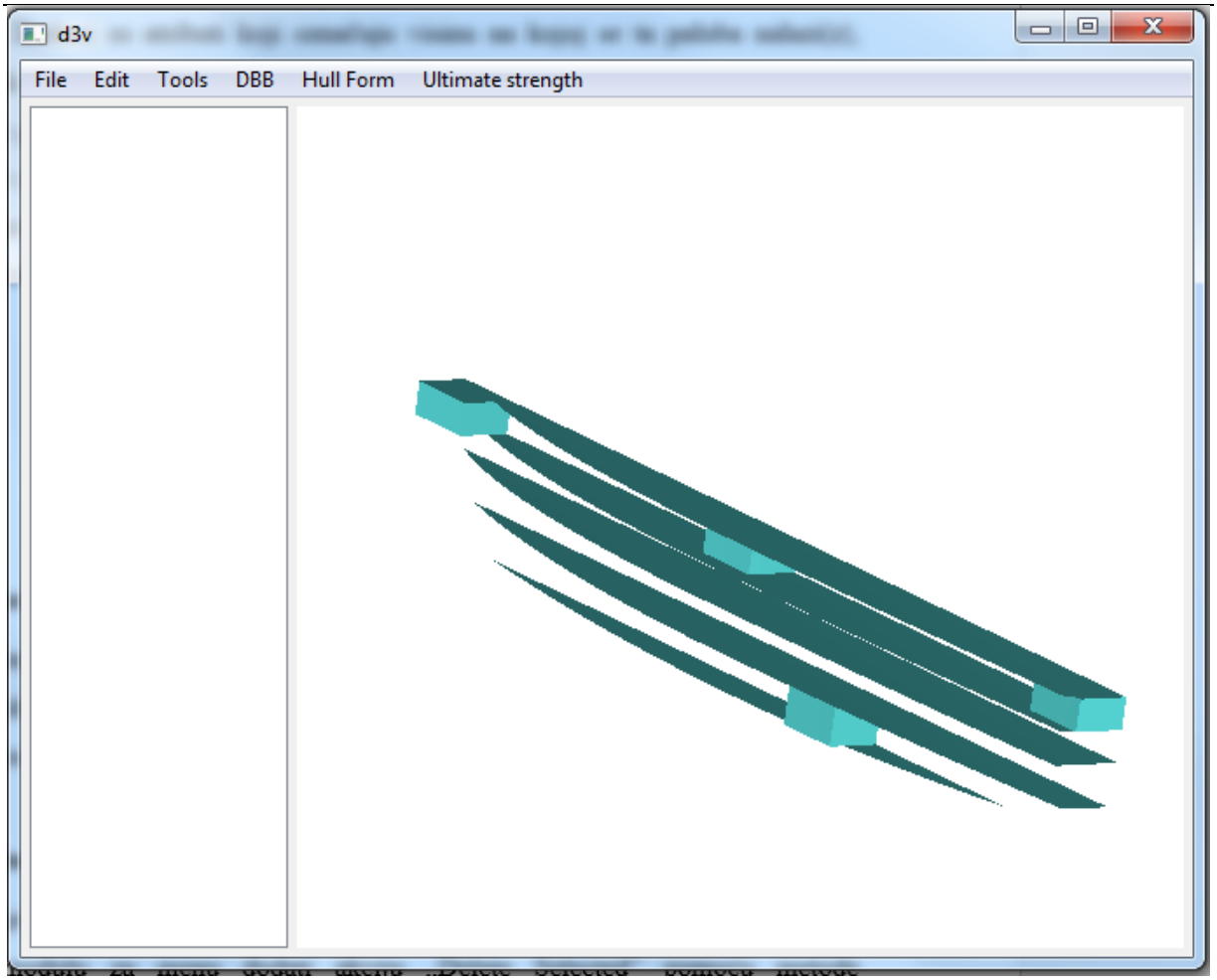
U ovim datotekama se nalaze točke i identiteti točaka za svaki identitet lica za jedinične podijeljene blokove dobivene pomoću funkcije *make_block_csv_file()*. Također u datoteci *Unit Blocks* mogu naći različite verzije ovih dvaju datoteka, jedan par za svaku razinu podjele, koja se kopirati u glavnu datoteku ako se ukaže potreba za većom točnošću aproksimacije forme pomoću algoritama opisanih u poglavlju 4. Ovi blokovi se pretvaraju u mrežu pomoću funkcije *make_block_from_csv()*.

3.2. Vizualizacija i interaktivno generiranje blokova DBB metodom

Kako bi se blok mogao vizualizirati, potrebno je proširiti klasu *DBB()*. Dodani su atributi koji u sebi sadrže dimenzije bloka (*block_dims*), poziciju bloka (*position*) i putanju do dbb datoteke (*folderpath*). Također su promijenjene neke od postojećih metoda i dodane nove. Metoda genMesh(), čiji je zadatak napraviti mrežu, sada pomoću učitavanja jediničnog podijeljenog bloka, te njegovim skaliranjem i pomicanjem radi mrežu pomoću algoritma opisanog u poglavlju 4.1.9, koristeći podatke o poziciji paluba i atribute koji opisuju dimenzije i položaj bloka. Nakon učitavanja *dbb* datoteke klasa *dbbproblem()* u *dbb.py* modulu tokom svoje inicijalizacije generira *DBB()* objekt za svaki blok u *Block_data.csv* datoteci. Ukoliko želimo promijeniti poziciju, dimenzije i palubu na kojoj se nalaze, potrebno je jednostavno promijeniti vrijednosti u *block_data.csv* datoteci ili ukoliko želimo dodati novi blok, popuniti sljedeći red sa potrebnim podacima.

3.3. Vizualizacija i generiranje paluba DBB metodom

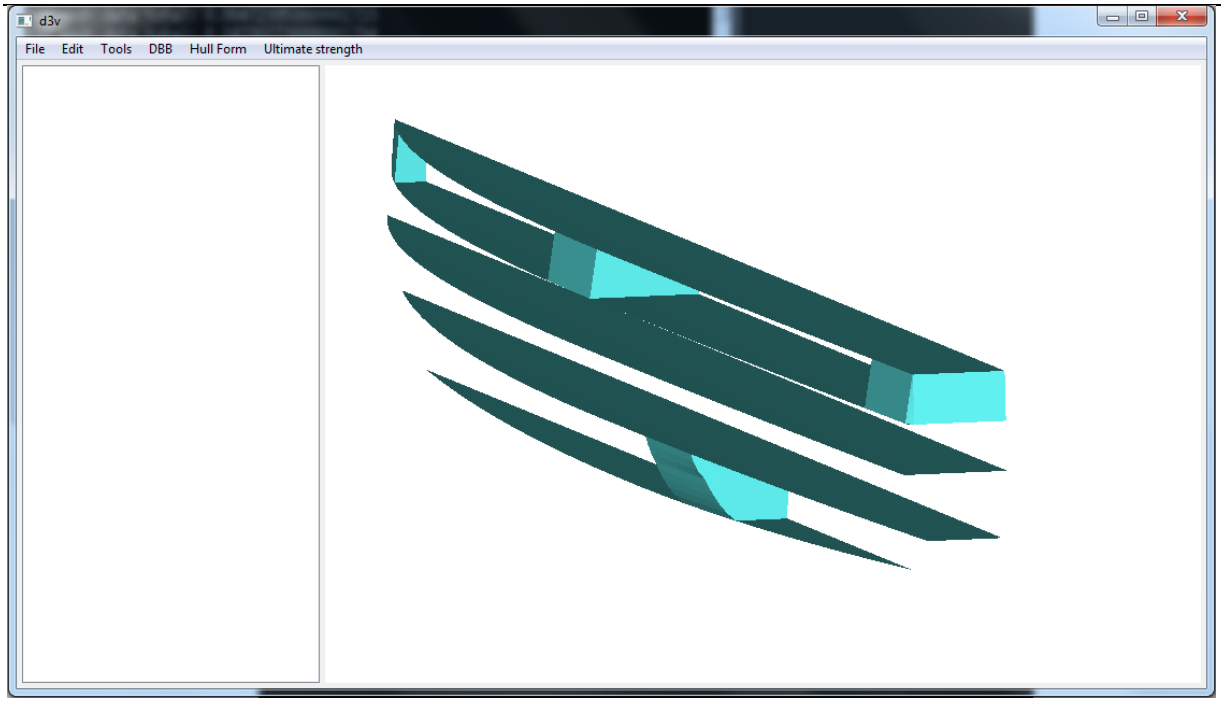
Analogno vizualizaciji i generiranju bloka, potrebno je dodati nove atribute i metode u *DBBDeck()* klasu. Dodani su atributi koji označuju visinu na kojoj se ta paluba nalazi(z), indeks palube(*deck indeks*) koji se broji od palube i formi kojoj pripada(*hullform*). Od metoda je dodana metoda *genMesh()* koja koristi algoritam iz poglavlja 4.1.5 da generira mrežu te palube. Zgodno je za primijetiti da su, prilikom generiranja mreže brodske forme pomoću modula *hullform.py*, napravljene točke svih vodnih linija i na pozicijama paluba te se te točke mogu koristiti za stvaranje mreže.



Slika 11. Vizualizacija paluba i blokova bez forme

3.4. Aproksimacija mreže forme pomoću bloka

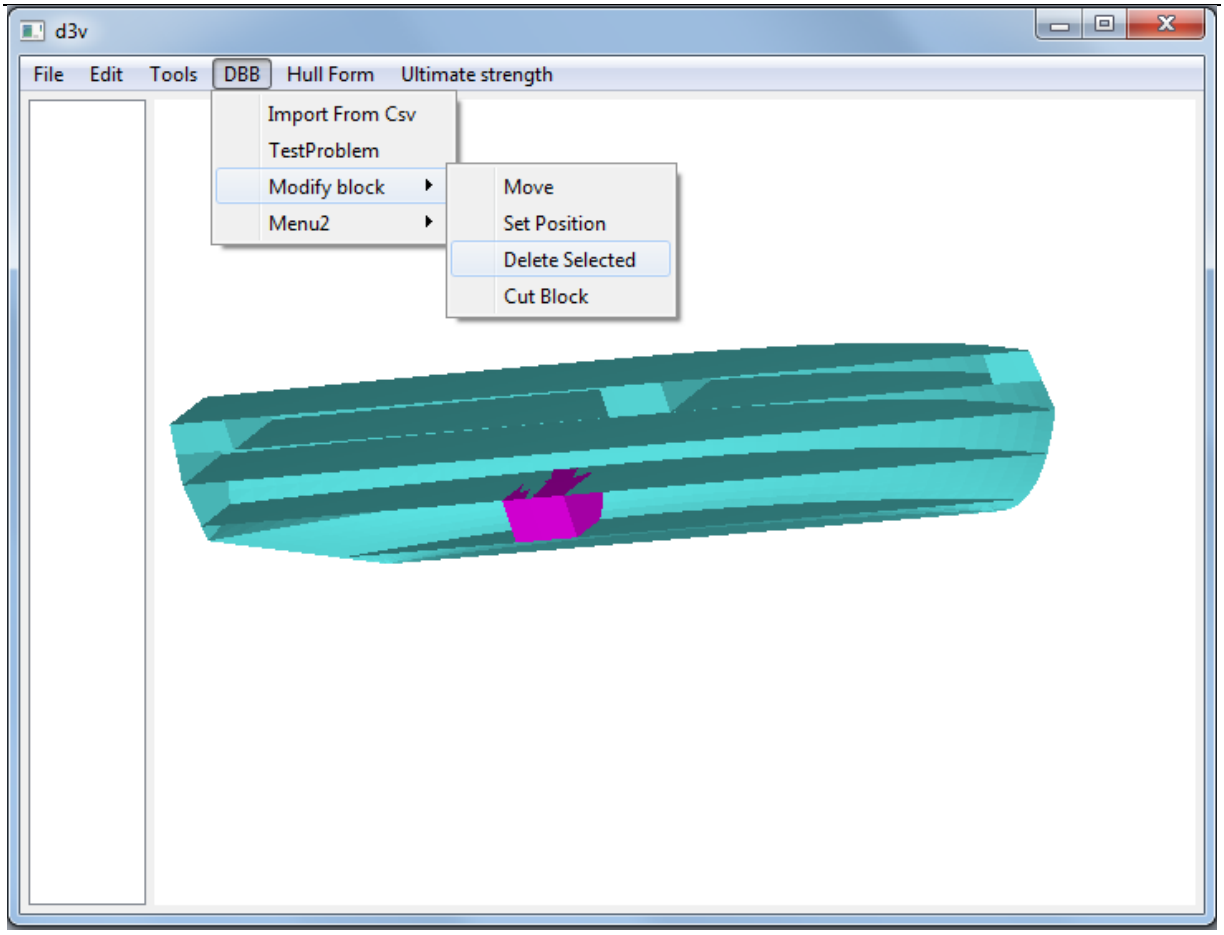
Svrha ove funkcionalnosti je deformirati mrežu bloka da na što bolji način aproksimira mrežu forme. To se postiže korištenjem algoritma opisanog u poglavlju 4.1.15, koji je dodan u *DBB()* klasu pomoću metode *cutMesh()* i automatski se poziva pri inicijalizaciji objekta.



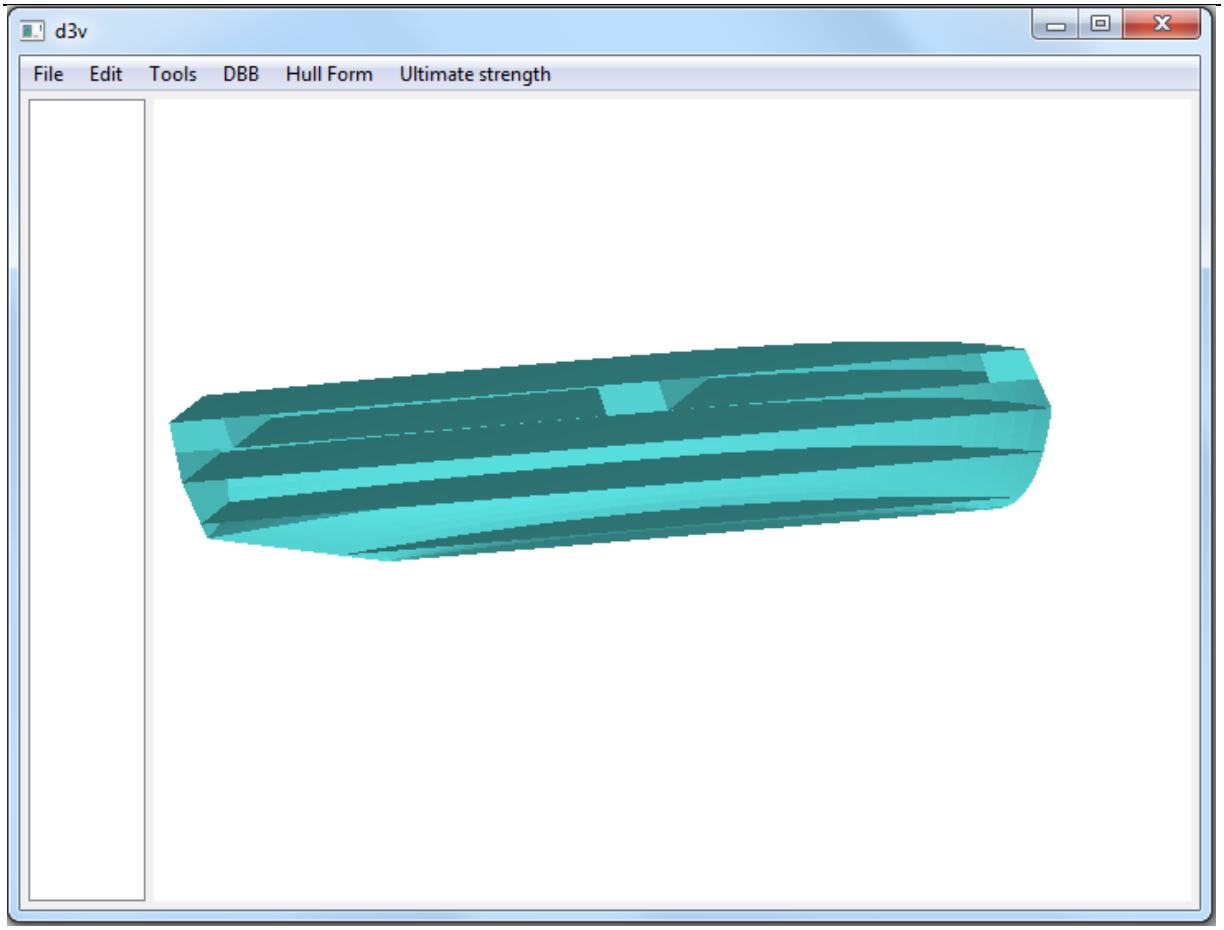
Slika 12. Aproksimacija mreže forme pomoću blokova

3.5. Interaktivno brisanje mišem selektiranih blokova

Kako bi se brisanje selektiranog objekta implementiralo, potrebno je klasi *DBBCommand()* u *dbbcomands.py* modulu za menu dodati akciju „*Delete Selected*“ pomoću metode *menu.addAction(„Delete Selected“).* Nakon što je akcija napravljena u glavnom prozoru, potrebno je odrediti što točno ta akcija radi kada je aktivirana. U tu svrhu se koristi metoda *onDeleteSelected()*, koja prvo provjerava postoji li nešto selektirano i ako postoji taj objekt briše iz liste u *dbbproblem()* klasi u kojoj su spremljeni DBB objekti. Tada se akcija povezuje pomoću metode *menu.triggered.connect(action).* Nakon izvršavanja vizualizacija se osvježava.



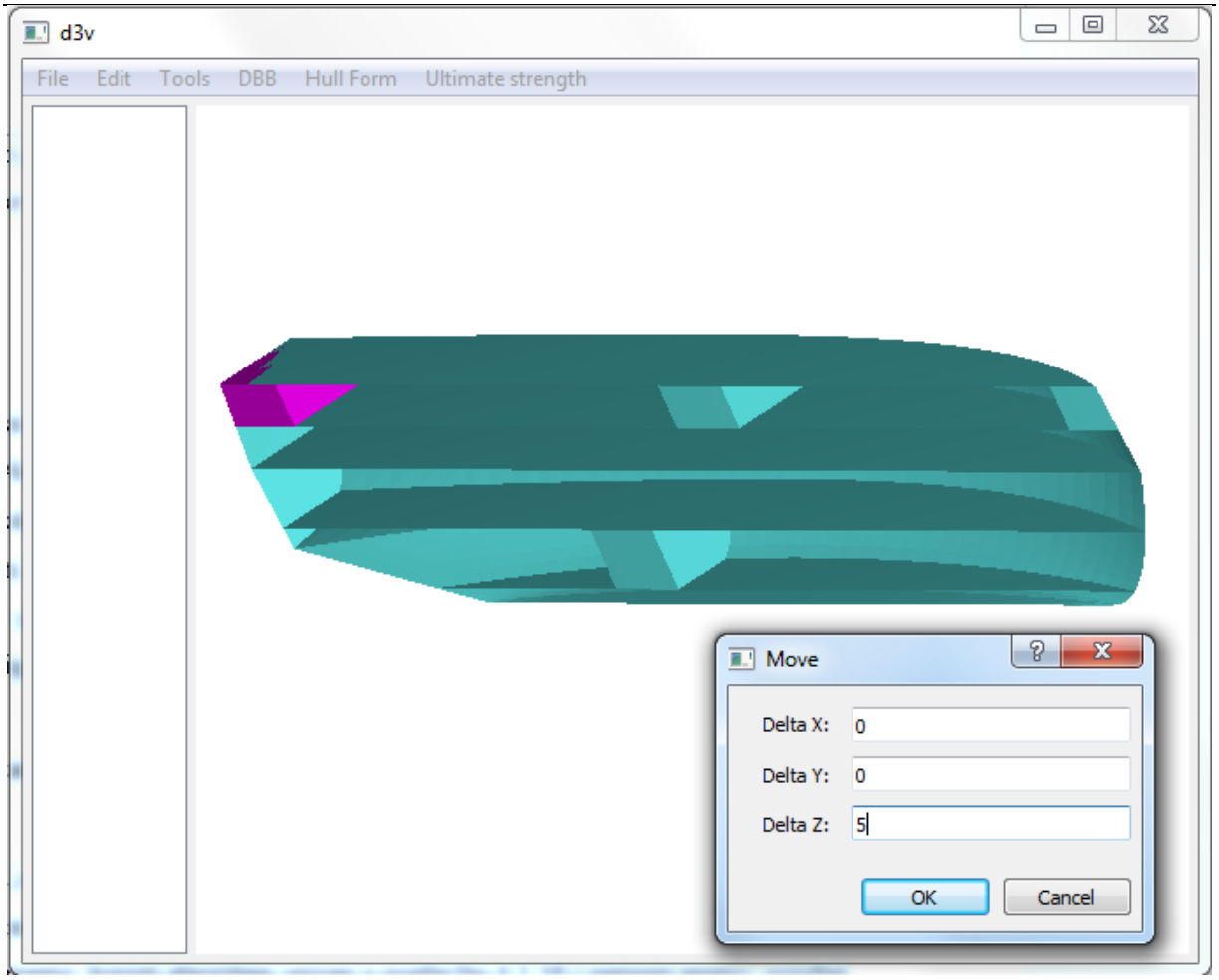
Slika 13. Brisanje bloka



Slika 14. Obrisan blok

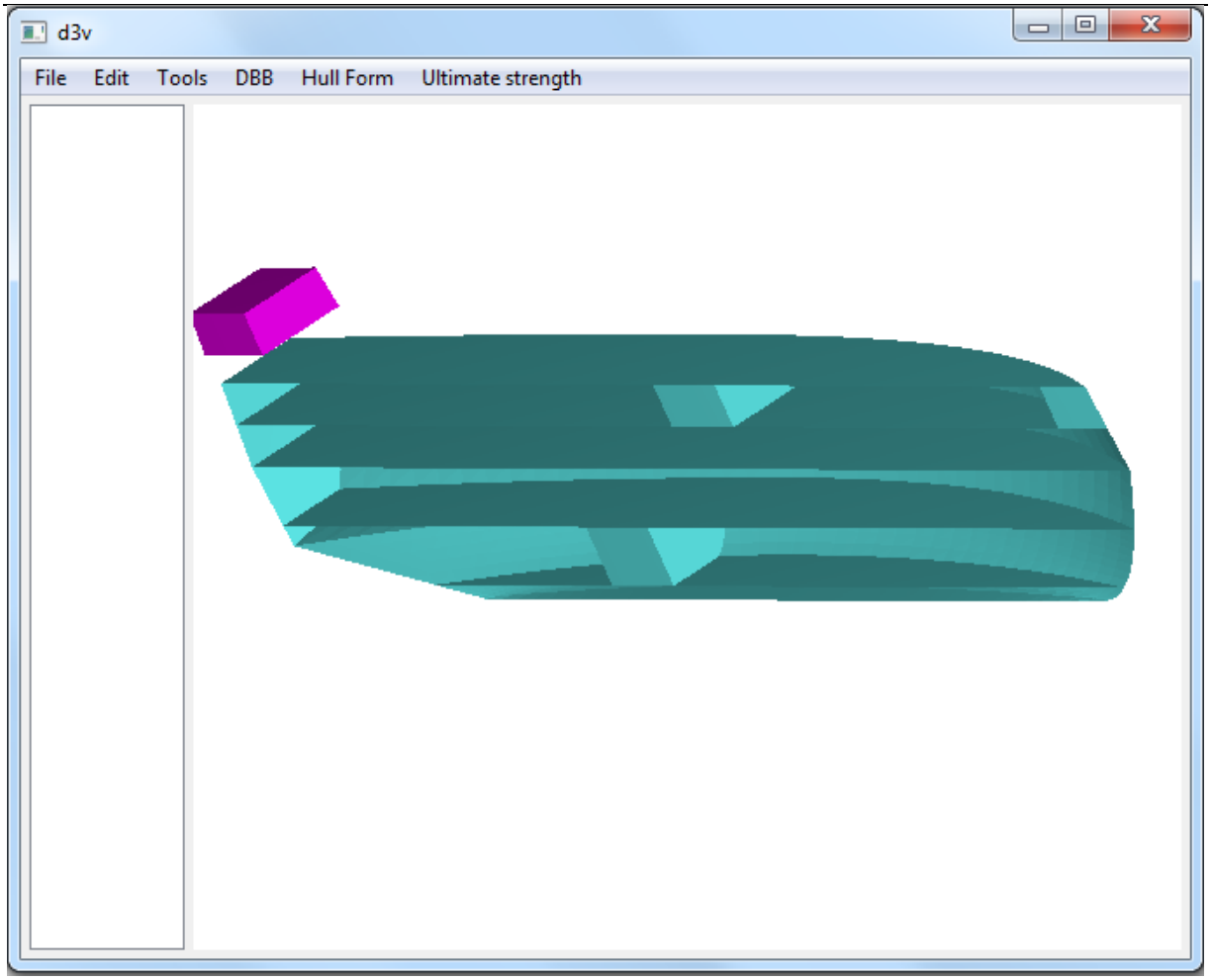
3.6. Interaktivno pomicanje mišem selektiranih blokova

Da bi se selektirani blok mogao pomicati, potrebno je klasi *DBBCommand()* u *dbbcomands.py* modulu za meni dodati akciju „*Move*“ pomoću metode *menu.addAction(„Move“).* Kao i kod brisanja, potrebno je definirati što se događa prilikom aktivacije, te je napravljena metoda *onMove()* koja prvo provjerava imamo li neku mrežu selektiranu i ako imamo iz *dbbmenus.py* modula pokreće novi meni napravljen u *QtCreatoru*:



Slika 15. Meni pomicanja

Ukoliko vrijednosti nisu brojevi ili polja nisu popunjena, meni javlja grešku i traži ponovni unos vrijednosti. Ako su vrijednosti prihvatljive, meni se zatvara i vraća unesene vrijednosti. Nakon toga se pokreće selektiranom objektu pokreće metoda *move()* koja pomiče blok za vraćene vrijednosti. Metoda *move()* koristi algoritam opisan u poglavlju 4.1.11, te atributu koji označuje poziciju pridodaje vrijednosti vraćene iz menija. Akcija sa na meni povezuje sa metodom *menu.triggered.connect(action)*. Nakon toga se model osvježava.



Slika 16. Pomaknut blok

3.7. Provjera zatvorenosti mišem selektiranih blokova

DBBCommand() klasi je dodana akcija „*isClosed*?“ pomoću metode *menu.AddAction(„isClosed“),* te je povezana na metodu *onIsClosed* pomoću *menu.triggered.connect(action). OnIsClosed* provjerava postoji li trenutno selektirani blok i ako postoji, koristi algoritam opisan u poglavlju 4.1.16 i ispisuje njegov rezultat.

3.8. Izračun volumena mišem selektiranih blokova

DBBCommand() klasi je dodana akcija „*Mesh Volume*“ pomoću metode *menu.AddAction(„Mesh Volume“),* te je povezana na metodu *onMeshVolume()* pomoću *menu.triggered.connect(action). OnMeshVolum()* provjerava imamo li trenutno neku mrežu selektiranu i ako imamo, koristi algoritam opisan u poglavlju 4.1.10 i ispisuje njegov rezultat.

4. OBJAŠNJENJA ALGORITAMA PISANIH U PYTHONU

U ovom poglavlju su objašnjeni izrađeni algoritmi potrebni za izradu, manipulaciju i računanje mrežama. Te funkcije se pozivaju za generiranje, manipuliranje i obradu blokova i drugih elemenata.

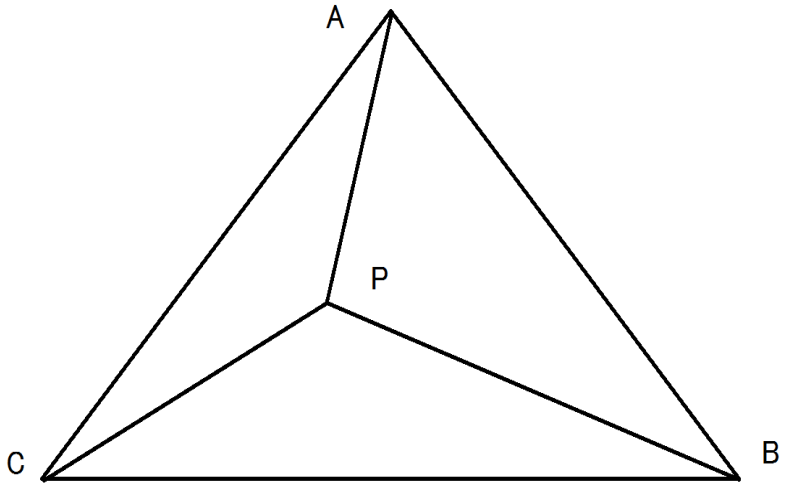
4.1.1. Računanje površine trokuta

Funkcija *triangle_surface()* računa površinu trokuta zadanu sa tri točke pomoću jednostavnog vektorskog množenja dvaju vektora koji povezuju točke pomoću *numpy.cross()* metode i množenjem tog produkta sa 0.5.

4.1.2. Provjera položaja točke u odnosu na trokut

Funkcija *is_inside_triangle()* je jedna od najosnovnijih internih funkcija koja se koristi u većini drugih funkcija; svrha joj je vrlo jednostavna: provjeriti leži li neka točka u dijelu ravnine koju dijele 3 točke. U slučaju da leži, funkcija vraća *True*, dok u suprotnom slučaju vraća *False* Booleovu varijablu.

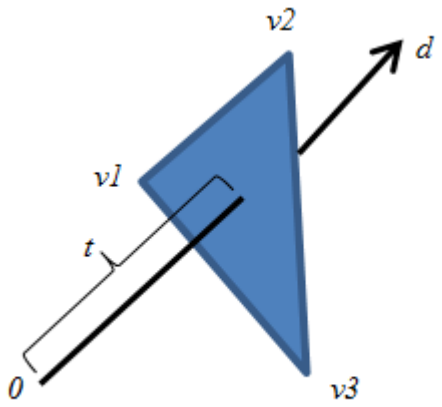
Istinitost se ispituje generacijom trokuta ABC od zadanih točaka i generacijom još 3 trokuta kod kojih je svaka stranica trokuta ABC povezana sa točkom koju ispitujemo P, tj. trokuti ABP, BCP, CAP. Tada se računaju površine tih četvero trokuta pomoću funkcije *triangle_surface()* i uspoređuje se vrijednost površine trokuta ABC sa sumom površina trokuta ABP, BCP, CAP. Ukoliko su te vrijednosti jednake, točka leži na trokutu i funkcija vraća *True*, dok u suprotnom slučaju točka leži izvan trokuta i funkcija vraća *False*.



Slika 17. Točka u trokutu

4.1.3. Pronalaženje presjecišta pravca sa licem

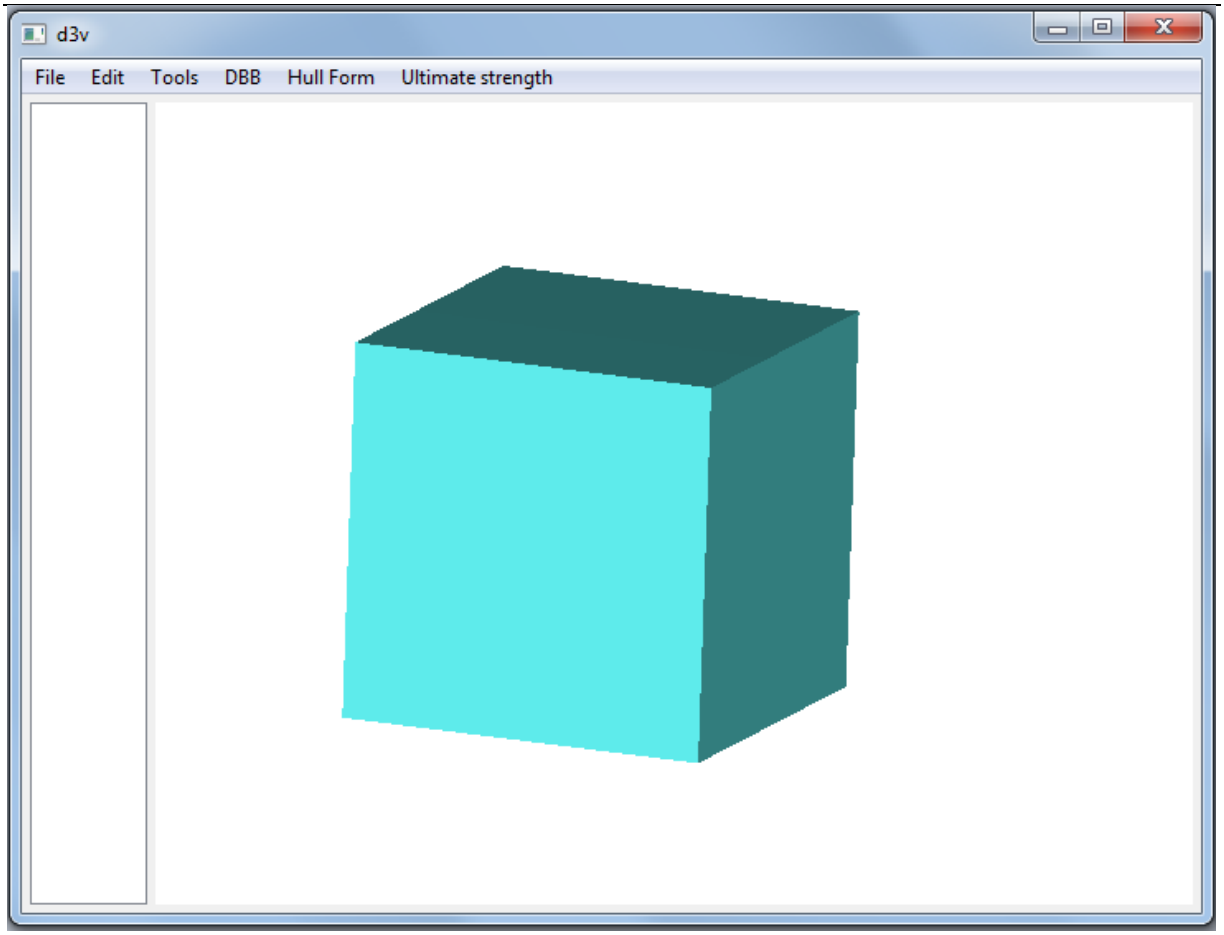
Funkcija *get_intersection()* je osnovna interna funkcija koju koriste mnoge druge; cilj funkcije je za zadanu točku i vektora smjera napraviti parametarski pravac i naći (ako postoji) točku presjecišta sa točkama trokuta. To se postiže generacijom dvaju vektora od zadanih točaka trokuta koji zajedno sa parametrizacijom čine ravninu. Jednadžba pravca i jednadžbe vektora tada čine linearni sustav jednadžbi, čija su rješenja vrijednosti parametara na sjecištu ravnine i pravca. Taj linearni sustav jednadžbi se računa pomoću *Numpy* funkcije *numpy.linalg.solve()*. Nakon dobivanja točke presjecišta uvrštavanjem dobivenog parametra u početnu jednadžbu pravca, potrebno je provjeriti leži li ta točka na zadanom trokutu pomoću funkcije *is_inside_triangle()*. Ukoliko leži, funkcija vraća koordinate točke presjecišta zajedno sa vrijednosti parametra pravca, dok u suprotnom vraća *None*.



Slika 18. Sjecište pravca sa trokutom ([19])

4.1.4. Generiranje mreže bloka

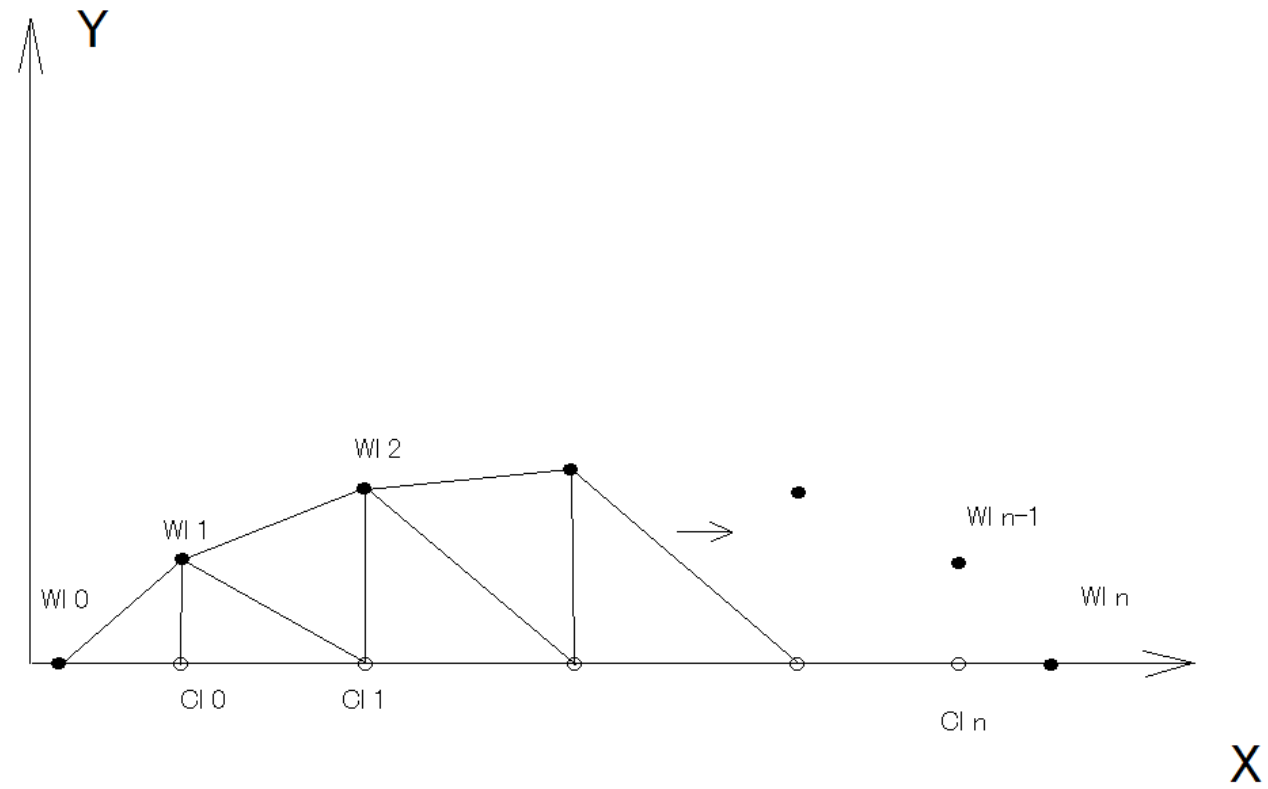
Funkcija *make_block()* stvara blok zadanih dimenzija i pozicije u trokutastoj mreži koji će kasnije predstavljati prostorije ili spremnike na brodu. Prvo se pomoću *openmesh.TriMesh()* funkcije generira prazna trokutasta mreža, te se zatim generiraju točke vrhova bloka pomoću funkcija *numpy.linspace()* i *numpy.meshgrid()*. *Numpy.linspace()* funkcija generira niz sa zadanim brojem točaka između dvije vrijednosti, dok se pomoću *numpy.meshgrid()* neki prostor dijeli sa mrežom točaka. Pomoću ove dvije funkcije moguće je neki prostor ili ravninu pravilno podijeliti sa proizvoljnom gustoćom točaka, no za ovaj konkretan slučaj, potrebni su nam samo vrhovi bloka, pa se prostor dijeli na samo 8 točaka. Potom se te točke zbrajaju sa vektorom pomaka, koji ih iz središta koordinatnog sustava pomiče na željenu poziciju. Od tih točaka se rade reference točaka sa metodom *mesh.add_verteks()*, te se zatim od tih referenca točaka stvaraju reference lica pomoću *mesh.add_face()* metode. Pritom se pazilo da prilikom unosa redoslijed unošenja referenci točaka bude konzistentan sa polurubnom strukturom.



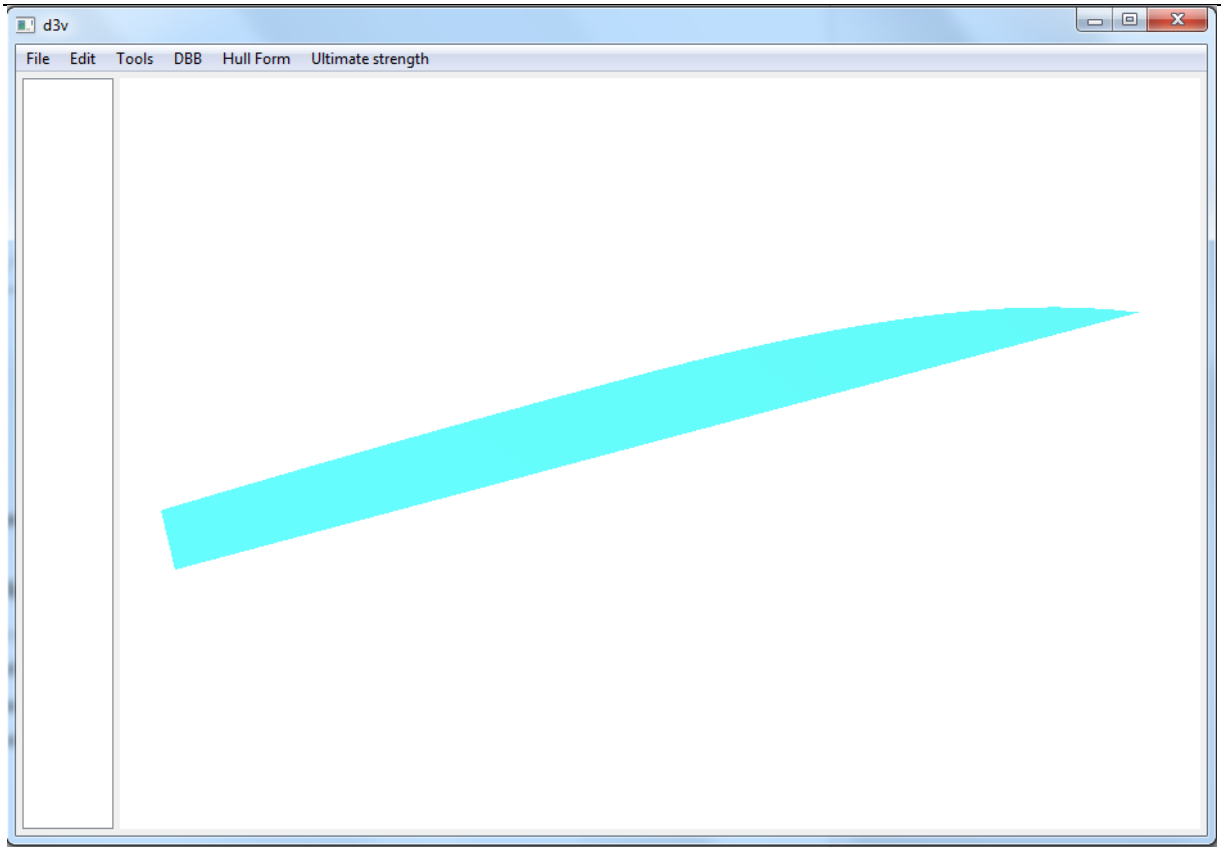
Slika 19. Mreža bloka

4.1.5. Izrada mreže paluba iz točaka vodnih linija

Funkcija *make_deck()* koristi točke vodnih linija od forme broda da generira mrežu palube na toj poziciji. Na slici, te točke su označene pomoću oznaka Wl 0 – Wl n. Potom se za svaku točku koja ne leži na x osi generira nova točka sa koordinatom y = 0, tako da leži na x osi, zvane Cl 0 – Cl n. Potom se u for petlji za svaki interval točaka na x osi, osim prvog i zadnjeg, generiraju dva lica pomoću Wl i Cl točaka na tom intervalu. Na prvom i zadnjem intervalu lica se u mrežu dodaju izvan petlje, jer je njima potrebno samo jedno lice.



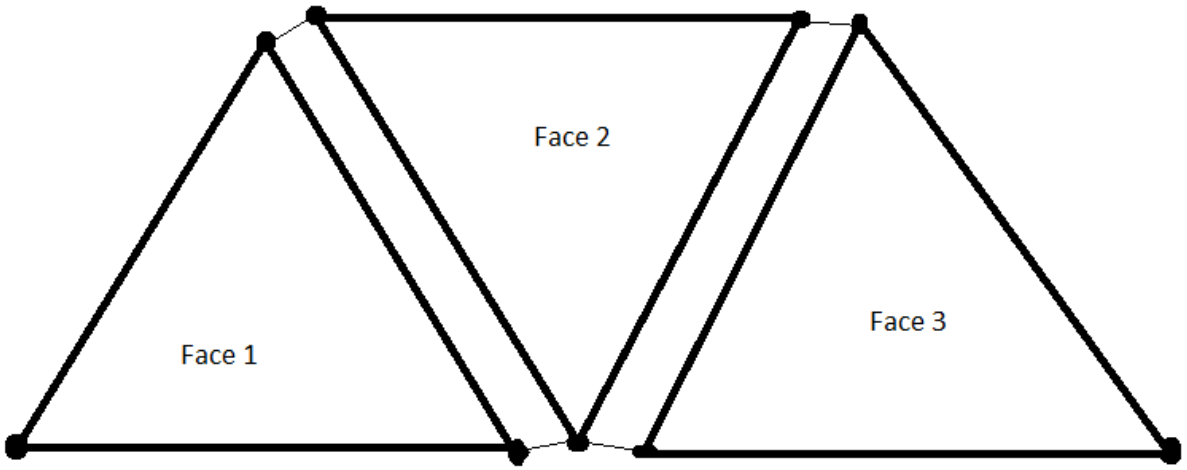
Slika 20. Skica algoritama za generiranje trokutne mreže palube



Slika 21. Mreža palube

4.1.6. Spajanje blokova

Funkcije *soft_merge_meshes()* i *hard_merge_meshes()* su funkcije kojima je cilj spojiti dva ili više zasebnih mrežnih objekta u jedan mrežni objekt, no to rade na dva različita načina. Zamislimo da imamo 3 zasebne mreže, svaka od kojih se sastoji od samo jednog lica i da im se neke od točaka nalaze na istim mjestima.



Slika 22. Merge_meshes primjer

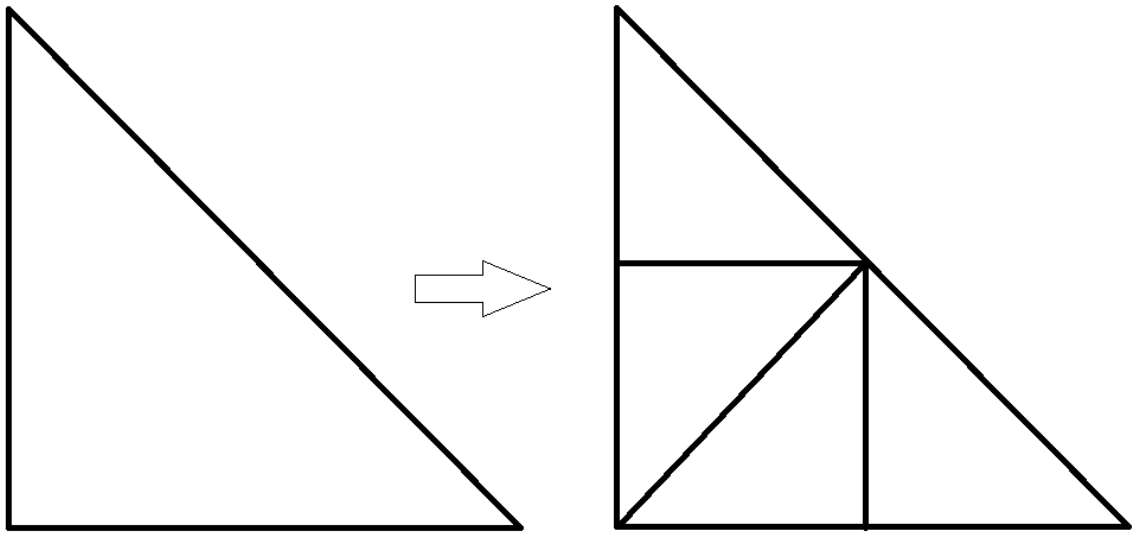
Tankim linijama su, zbog jasnoće, povezane točke koje su na istim mjestima u koordinatnom sustavu.

Soft_merge_meshes() jednostavno uzima od svih mreža podatke o točkama i indeksima lica, te ih zbraja u jedan niz, od kojega se radi nova mreža. Važno je za primijetiti da ovaj način sam po sebi, iako jako brz, ne stvara kvalitetnu mrežu za daljnju obradu, jer ne postoji međusobna povezanost između kombiniranih lica. U podacima za tu mrežu nalaze se duplikati točaka koje imaju istu poziciju i svaka od tih točaka ima istu povezanost kao i prije kombiniranja, tako da je nemoguće koristiti veliku većinu funkcionalnosti koju *Openmesh* daje, kao što je npr. kruženje oko susjednih lica. *Soft_merge_meshes()* se koristi kada je potrebno brzo spojiti mreže u jednu cjelinu za slučajeve kada nije dalje potrebno raditi sa njima, npr. vizualizacija.

Za razliku od *Soft_merge_meshes()*, *hard_merge_meshes()* vodi računa o duplikatima točaka i povezanosti spojenih lica, tako da se funkcionalnosti *Openmesha* mogu pravilno koristiti kao da je mreža napravljena „pravilno“. *Hard_merge_meshes()* prvo spaja zadane mreže pomoću *soft_merge_meshes()* i zatim traži duplikate točaka, njihove indekse, reference i lica kojima pripadaju. Potom u svim licima koji imaju te točke mijenja sa novom, zajedničkom točkom koja povezuje sva lica. Iako ova funkcija stvara kvalitetniju mrežu za daljnje računanje od *soft_merge_meshes()*, za veći broj točaka je osjetno sporija, jer mora pretraživati i uspoređivati puno veći broj podataka.

4.1.7. Podjela mreža

Funkcija *subdivide_mesh()* je rekurzivna funkcija kojoj je cilj podijeliti neko lice na više manjih lica, koji zajedno zadržavaju oblik početnog lica i time povećati gustoću te mreže. Algoritam za svako lice u zadanoj mreži pronalazi točke polovišta svakog ruba i zatim od njih i početnih točaka stvara 4 nova lica.



Slika 23. Podjela lica

Nakon što je postignuta željena gustoća mreže dobivena lica se spajaju pomoću funkcije *hard_merge_meshes()* u jednu cjelinu. Ova funkcija se koristi kada neka mreža nije dovoljno gusta za neku operaciju. Važno je primijetiti da vrijeme izvršavanja eksponencijalno raste sa brojem ponavljanja algoritma.

4.1.8. Stvaranje datoteka jediničnih blokova

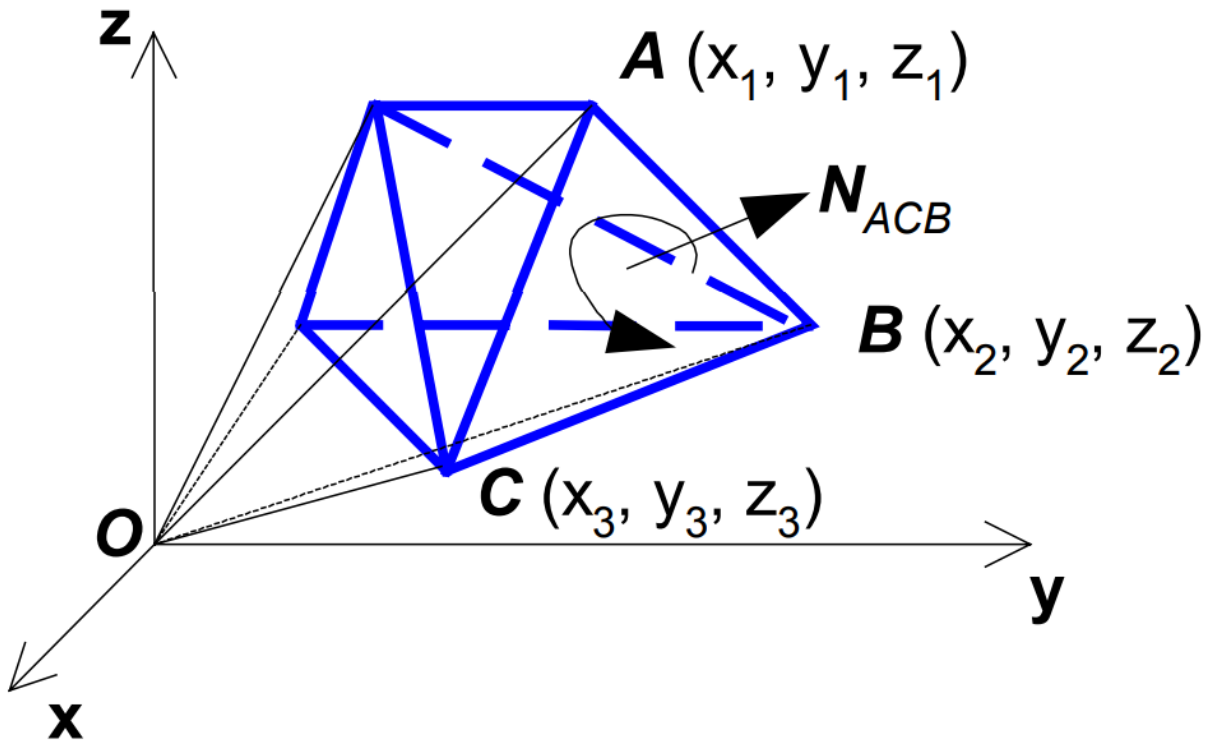
Koristi se funkcija *make_block_csv_file()*. S obzirom da generacija podijeljenih blokova klasičnim putem (funkcijom *make_block()* i *subdivide_mesh()*) traje dugo, pogotovo kada je potrebno generirati mnogo podijeljenih blokova kao što je slučaj za spremišta na brodu, stvara se potreba za ubrzavanjem generiranja. S obzirom da najviše vremena pri generiranju odlazi na samu podjelu, stvorena je funkcija *make_block_csv_file*(). Cilj je napraviti samo jedan, do željene gustoće podijeljen blok, kojemu su sve dimenzije 1, i njegove osobnosti (točke i indeksi točaka koji su povezuju sa licima) zapisati u *csv* datoteku. Tada se taj jedinični blok može koristiti kao predložak za sve druge blokove i potreba za uzastopnom podjelom je zaobiđena.

4.1.9. Izrada bloka iz datoteke jediničnog bloka

Funkcija *make_block_from_csv()* čita *csv* datoteku jediničnih blokova, te prije generacije tog bloka u pravu mrežu, skalira sve točke na željene dimenzije i pomiče ih na željenu poziciju. Ovaj postupak je neusporedivo brži od klasične generacije puno podijeljenih blokova.

4.1.10. Izračun volumena mreže

Funkcija *calc_volume()* je funkcija koja računa volumen neke zatvorene mreže. Iako na prvi pogled to izgleda kao zahtjevan problem, algoritam za računanje volumena je zapravo vrlo jednostavan. Algoritam zasebno za svako lice računa tetraedar od točaka trokuta lica i središta koordinatnog sustava(koji se nalazi u 0,0,0, pa je formula još kraća), te se taj volumen množi sa predznakom lica. Predznak lica označava orijentaciju tog lica, tj. gleda li to lice prema „unutra“ ili „van“ mreže. Predznak se računa pomoću skalarnog umnoška vektora normale lica i njegovog središta. Prilikom sumiranja tih volumena „viškovi“ volumena sa pozitivnim predznakom će biti poništeni sa volumenima negativnih predznaka. [20]



Slika 24. Izračun volumena lica ([20])

4.1.11. Pomicanje mreže

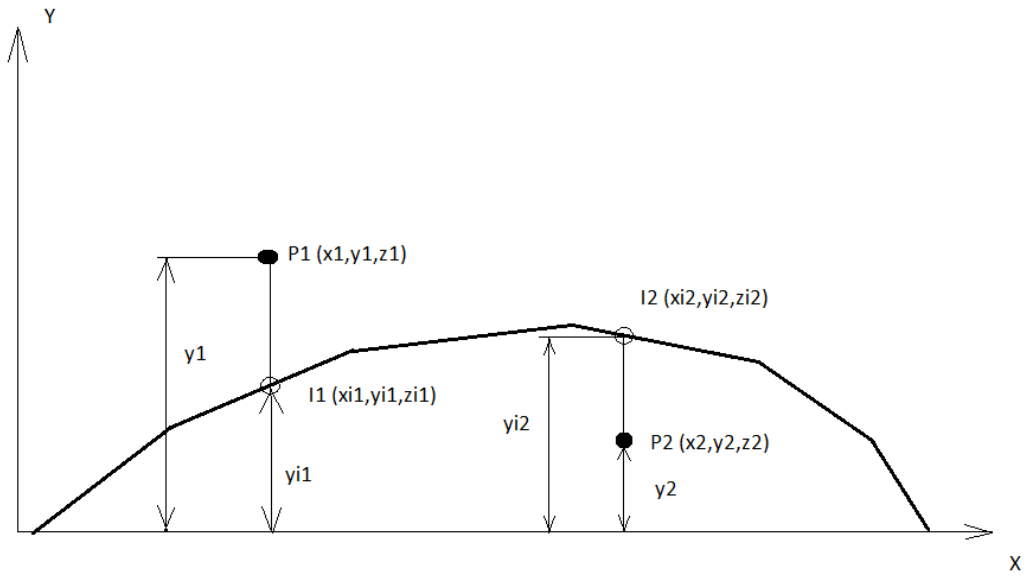
Funkcija *move_mesh()* je funkcija čija je svrha pomaknuti sve točke neke mreže pomoću metode *mesh.set_point()* na novu poziciju, koja je udaljena od početne za neki vektor.

4.1.12. Stvaranje mreže suprotne orijentacije lica

Funkcija *flip_mesh_face_orientation()* je funkcija koja vraća novu mrežu koja ima iste točke kao i unesena mreža, ali je redoslijed unosa točaka promijenjen unatrag(o redoslijedu unosa točaka u mrežu ovisi njegova orijentacija i time, normala) mijenjajući orijentacije svih lica u mreži. To se postiže *Numpy* funkcijom *numpy.flip()*, koja mijenja redoslijed indeksa točaka koji su vezani za indekse lica zadane mreže. Koristi se za ispravljanje mreža koje nemaju pravilnu orijentaciju.

4.1.13. Provjera pozicije točke u odnosu na formu po osi y

Funkcija *is_point_inside_form_mesh_y()* provjerava je li neka točka unutar mreže forme broda po y-osi. S obzirom da je forma broda otvorena mreža potrebne su dvije funkcije prepoznavanja(jedna po x osi, a druga po y osi) , koje su posebno napravljene za formu broda.

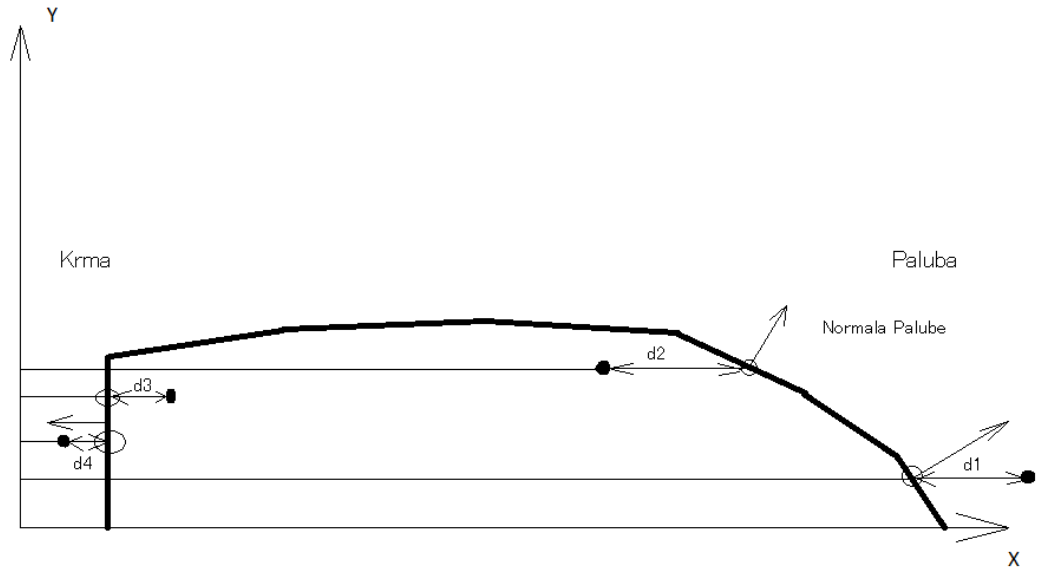


Slika 25. Provjera pozicije točke u odnosu na formu po y osi

Prvo se traži presjecište forme broda sa pravcem iz točke P i vektorom smjera prema ravnini xz. Ima mogućnost pretraživanja samo određenih lica forme koji su u blizini bloka, tako da se bitno skrati vrijeme pretraživanja. Ukoliko takvog presjecišta nema; točka je automatski izvan forme i funkcija vraća *None*. Ako presjecište postoji, funkcija uspoređuje apsolutne vrijednosti njihovih y koordinata(dovoljna je samo provjera apsolutnih vrijednosti jer je forma simetrična s obzirom na xz ravninu). Ako je y1 (y koordinata točke) veća od yi1 (y koordinate presjecišta sa formom), točka je izvan forme i funkcija vraća *False*, zajedno sa točkom presjecišta I1, dok ako je y1 manji od yi1, točka leži unutar forme i funkcija vraća *True* i točku presjecišta. Ukoliko točka leži baš na trokutu lica, funkcija vraća *True*.

4.1.14. Provjera pozicije točke u odnosu na formu po osi x

Funkcija *is_point_inside_form_mesh_x()* provjerava nalazi li se točka unutar forme broda po x osi. Kao i *is_point_inside_form_mesh_y()* funkcija može ograničiti svoju pretragu na lica blizu bloka. No za razliku od xz osi, forma broda nije simetrična s obzirom na yz os. Funkcija, osim točke presjecišta, računa i normalu tog lica. Ako je x komponenta pozitivna to lice leži na palubi, dok u suprotnom slučaju leži na krmi.



Slika 26. Provjera pozicije točke u odnosu na formu po x osi

U slušaju da je lice na palubi uspoređuju se razlike x koordinata točke koju ispituju se i točke presjecišta. U slučaju da je razlika pozitivna točka leži izvan forme i funkcija vraća *True*, dok u suprotnom slučaju vraća *False* i točku presjecišta. Ukoliko je lice na krmi, pozitivna razlika

znači da je točka u formi broda i funkcija vraća *True*, a ako je negativna, vraća *False* i točku presjecišta. Ukoliko je točka baš na trokutu lica, funkcija vraća *True*.

4.1.15. Aproksimacija mreže forme mrežom bloka

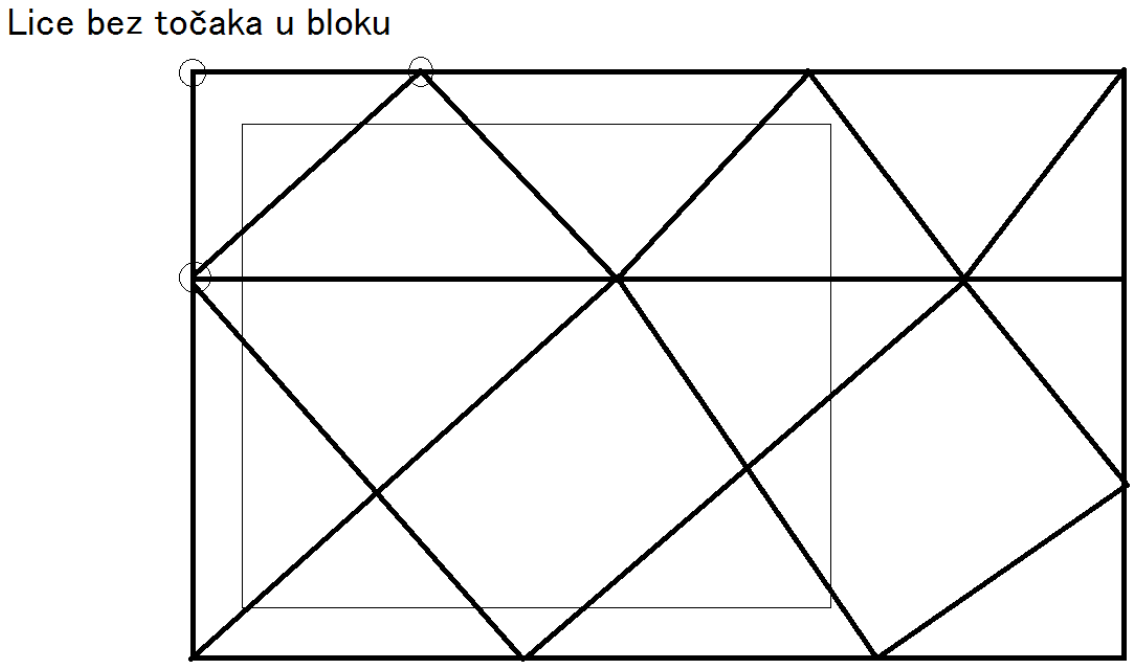
Funkcija *fit_block()* je kompliciranija funkcija kojoj je cilj deformirati podijeljeni blok, napravljen funkcijom *make_block_from_csv()*, na taj način da što bolje aproksimira formu broda u svrhu vizualizacije i daljnjih izračuna, kao npr. izračun volumena pomoću funkcije *calc_volume()*.

Postoje tri glavne pozicije, po dužini broda, gdje se blok može nalaziti*:*

- Na krmi
- Na palubi
- Između krme i palube

Ovo je važna razdjela, jer osobnosti brodske forme na tim pozicijama zahtijevaju drugačije algoritme kako bi se forma pravilno aproksimirala.

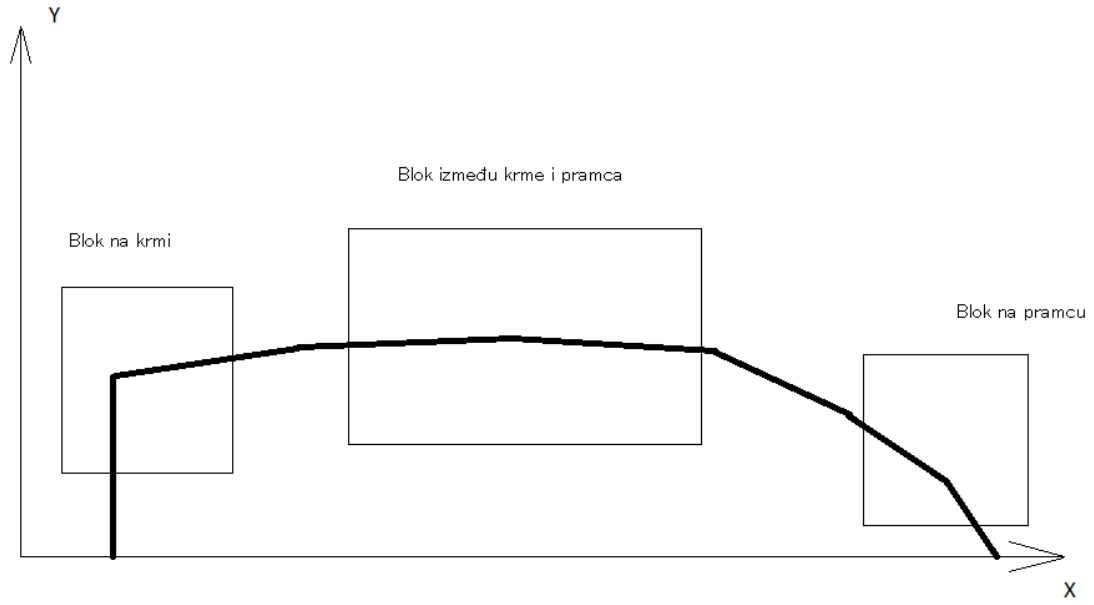
No, prije svega, potrebno je vidjeti koja lica se nalaze u blizini bloka. Ukoliko se ova pretraga ne napravi i algoritam provede za sve lica forme, vrijeme rada se znatno produžuje, jer ponekad se forme (ovisno o detaljnosti mreže) mogu sastojati od desetaka tisuća lica, dok ovom pretragom se taj broj znatno smanjuje. Ova pretraga se radi na sljedeći način: za svako lice u mreži se provjerava sadrži li to lice točke u prostoru koji blok zauzima. Ta se provjera radi jednostavnom usporedbom koordinata točke sa položajem i dimenzijama bloka; ako su zasebne koordinate točke između maksimalne i minimalne vrijednosti dimenzije bloka (kojoj je pridodan položaj bloka) onda je točka unutar prostora bloka. Nakon što se dobiju ta lica, algoritam traži njima susjedna lica koja nisu već ubrojeni. Ovo se provodi jer se nekada zna dogoditi da je samo dio ruba unutar bloka, bez točaka, kao što je vidljivo na slici.



Slika 27. Lice bez točaka unutar bloka

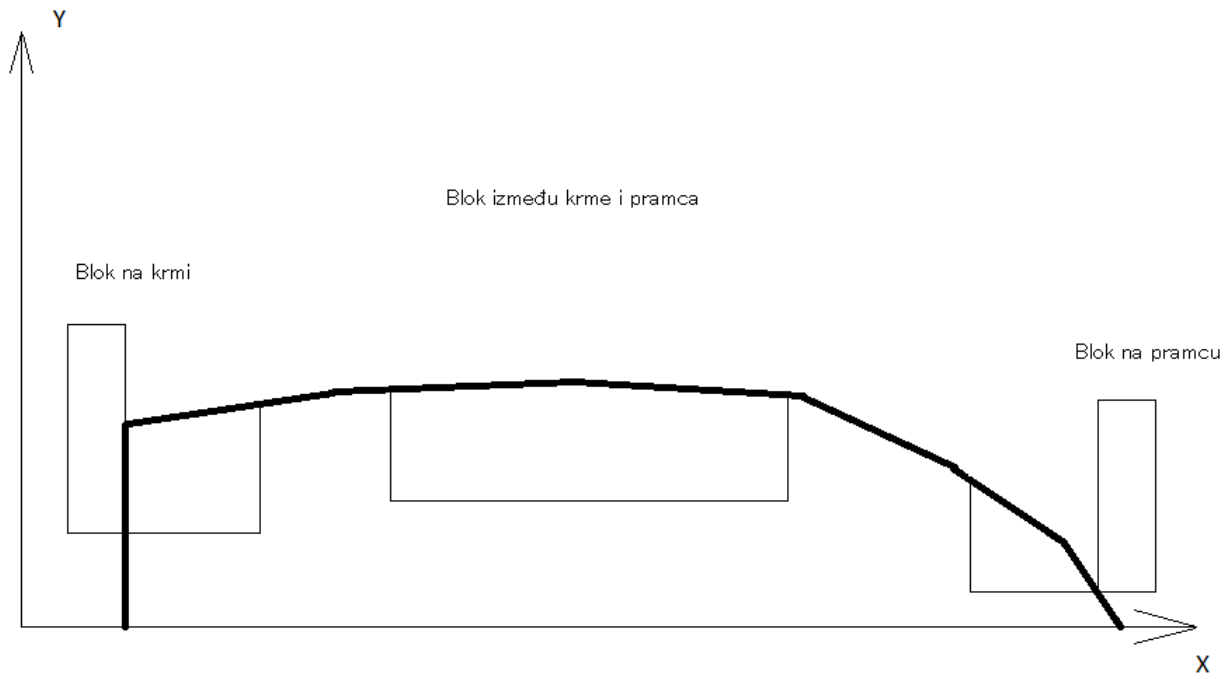
Ovaj se postupak ponavlja još jedan put, zbog sigurnosti.

Nakon što su lica za pretragu identificirana, točke bloka se dijele na točke koje imaju presjecišta na formi broda po y osi i na one koje nemaju pomoću funkcije *is_point_inside_form_mesh_y()*, koja vraća *None*, ukoliko se presjek nije dogodio sa formom.



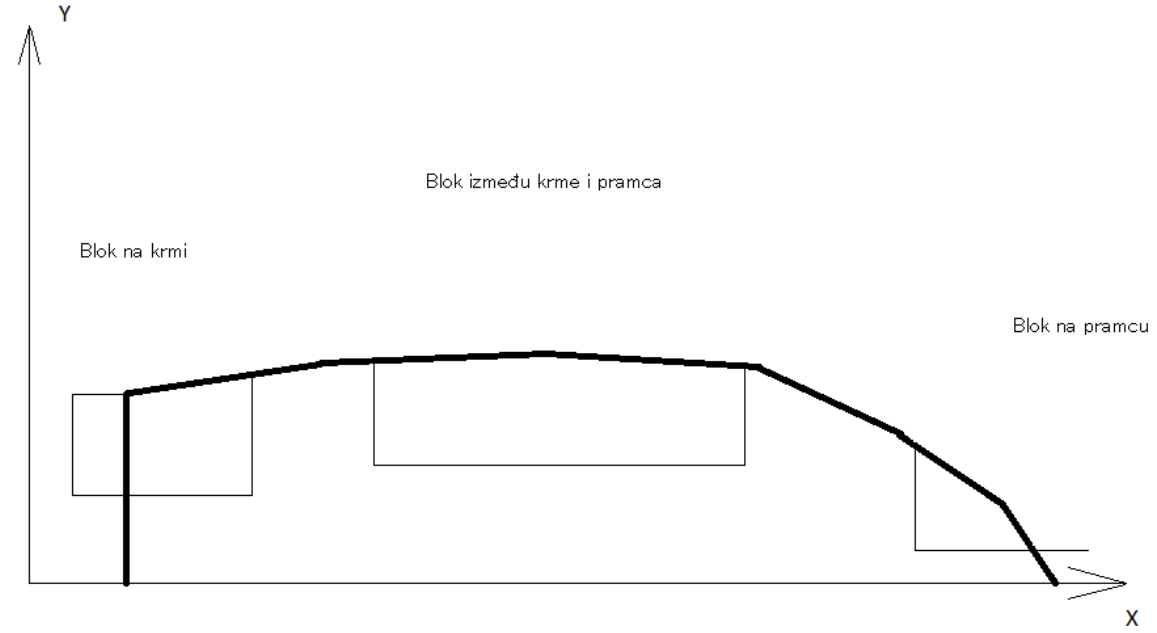
Slika 28. Pozicije blokova na krmi

Ukoliko takvih točaka nema, blok se nalazi između krme i palube, pa je potrebno jednostavno svaku točku izvan forme koja ima presjecište pomoću *mesh.set_point()* metode pomaknuti na točku presjecišta sa formom.



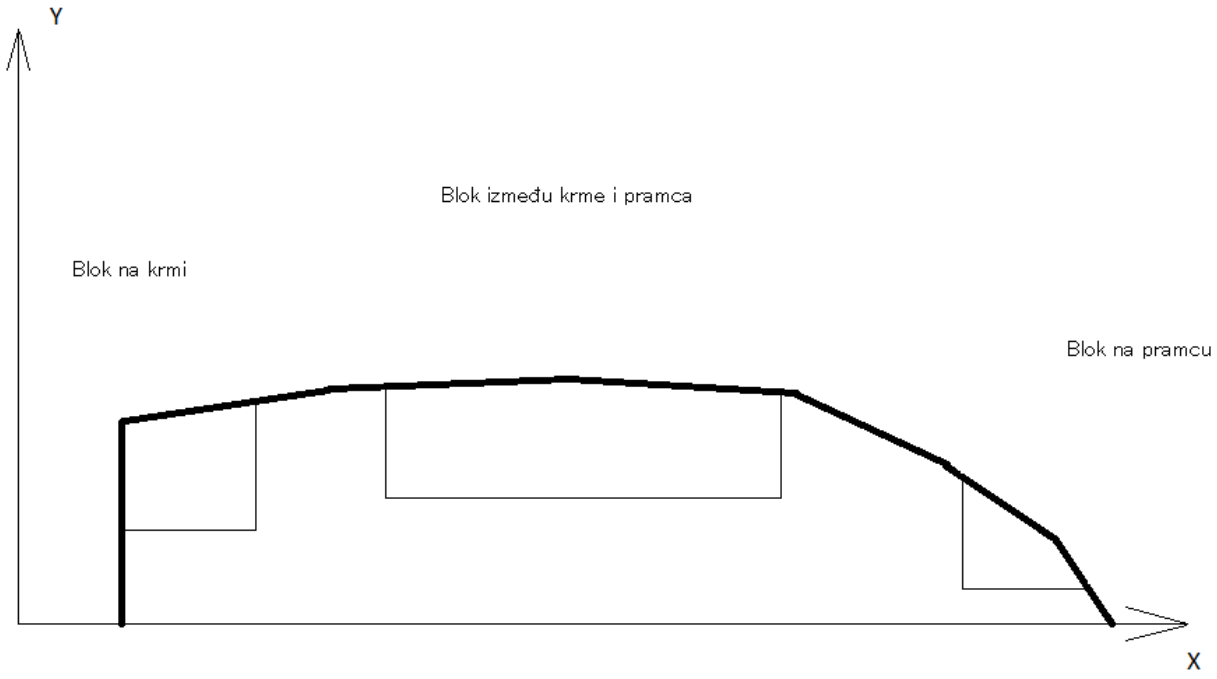
Slika 29. 1. Korak aproksimacije forme blokom

Vidimo da je ovim postupkom blok između krme i pramca potpuno sklopio sa formom, dok je za blokove na krmi i pramcu potrebno još manipulacije. Sljedeći korak je drugačiji za blokove na krmi od blokova na palubi. Za blokove na palubi se sve točke pomiču u ravninu u kojoj su točke u bloku minimalne po y koordinati(vrijednosti pozicije bloka po y koordinati), dok se na krmi sve točke iznad vrha krme (koji je identificiran sa točkama koje imaju maksimalnu koordinatu y na minimalnoj koordinati x među točkama u formi, za svaku z koordinatu) pomiču u ravninu vrha krme.



Slika 30. 2. Korak aproksimacije forme blokom

Sljedeći korak je pomicanje točaka na formu broda po x osi. To se postiže funkcijom *is_point_inside_form_mesh_x()*, koja identificira pozicije točaka nakon pomicanja i njihova presjecišta sa formom broda, te se identificirane točke pomiču na točke presjecišta sa formom pomoću metode *mesh.set_point()*.



Slika 31. 3. Korak aproksimacije forme blokom

Važno je spomenuti da je točnost kojom blok aproksimira formu bitno ovisi o razlici između gustoće mreže broda i gustoće mreže forme, te da na određenim mjestima gdje forma naglo mijenja oblik potrebno dodatno povećati gustoću bloka. Također je važno spomenuti da što je veća gustoća mreža, *fit_block()* funkcija je sporija, jer treba raditi sa većim brojem točaka.

4.1.16. Provjera zatvorenosti mreže

Funkcija *is_mesh_closed()* kruži po svakoj referenci ruba, te ispituje njegovu povezanost sa metodom *mesh.is_boundary()*. Ukoliko je taj rub graničan, funkcija odmah vraća *False,* dok na kraju petlje, ukoliko su svi rubovi prošli ispit, vraća *True*.

5. ZAKLJUČAK

Linaetal-fsb/d3v je besplatan program otvorenog koda, koji se može, uz minimalno znanje o programiranju proširivati po potrebi korisnika. Temelji se na *Qt for Python* tehnologiji, koja omogućuje trodimenzijsku vizualizaciju geometrijskih entiteta primjenom *OpenGL* tehnologije i jednostavnu izradu grafičkog sučelja za specifične namjene. Koristi metodu projektnih blokova (*Design Building Blocks* ili DBB) koja projektantu omogućuje odvajanje brodskih funkcija u diskretne elemente koje je moguće pozicionirati na odgovarajuća mjesta na brodu. Nakon uvođenja novih funkcionalnosti pojavila se manja mana trenutnog koda: loša optimizacija algoritma za presijecanje lica pravcem. Iako algoritmi funkcioniraju u razumnom vremenu, kod kompleksnijih konfiguracija forme sa velikim brojem podijeljenih blokova, gubitak vremena je osjetan. U svrhu optimizacije potrebno bi bilo koristiti brži algoritam.

LITERATURA

[1] https://en.wikipedia.org/wiki/Computer_graphics

[2] https://hum3d.com/3d-models/curiosity-robotic-rover/

[3] https://en.wikipedia.org/wiki/Python_(programming_language)

[4] https://en.wikipedia.org/wiki/Interpreted_language

[5] https://wiki.qt.io/About_Qt

[6] https://en.wikipedia.org/wiki/Language_binding

[7] https://en.wikipedia.org/wiki/Polygon_mesh

[8] https://www.androidauthority.com/qualcomm-opengl-es-3-snapdragon-600-800-156906/

[9] https://www.graphics.rwth-aachen.de/media/openmesh_static/Documentations/OpenMesh-8.0-Documentation/index.html

[10] https://numpy.org/doc/stable/user/whatisnumpy.html

[11] https://www.graphics.rwth-aachen.de/media/openmesh_static/Documentations/OpenMesh-6.3-Documentation/a00010.html

[12] Andrews, D.J., Dicks, C.A., 1997. The Building Block Design Methodology Applied to Advanced Naval Ship Design, in: Proceedings of the International Marine Design Conference (IMDC) 1997. Newcastle, UK

[13] Andrews, D.J., Pawling, R.J., 2009. The Impact of Simulation on Preliminary Ship Design, in: Proceedings of the International Marine Design Conference (IMDC) May 2009. Trondheim, Norway.

[14] Andrews, D.J., Pawling, R.J., 2003. SURFCON - A 21st Century Ship Design Tool, in: Proceedings of the International Marine Design Conference (IMDC) May 2003. Athens, Greece.

[15] Andrews, D.J., Pawling, R.J., 2008. A case study in preliminary ship design. RINA Int. J. Marit. Eng. 150

[16] Buconić, Marko. "Vizualizacija OOFEM modela brodske konstrukcije primjenom programa otvorenog koda linaetal-fsb/d3v." Završni rad, Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje, 2020. https://urn.nsk.hr/urn:nbn:hr:235:559756

[17] https://docs.python.org/3/library/csv.html

[18] Luka Josip Erhardt : Vizualizacija i izračun hidrostatičkih karakteristika triangularizirane forme broda primjenom programa otvorenog koda linaetal-fsb/d3v

[19] https://www.researchgate.net/figure/slicing-line-with-ray-triangle-intersection-method17_fig3_318741068

[20] http://chenlab.ece.cornell.edu/Publication/Cha/icip01_Cha.pdf

PRILOZI

I. Programski kod

Prilog I

DBB.py modul

```
from iohandlers import IOHandler
from signals import Signals
from geometry import Geometry
import openmesh as om
import os
import numpy as np
import copy
import myfunctions as mf
from hullform import HullForm
import csv

class DBBGeometry (Geometry):
      def __init__(self):
            super().__init__()
            self.subdivboxes=[]
            self.nsubdivbox=0
            self.minsubdivbox = 0


class DBBProblem ():
      def __init__(self,fileName):
            self.filename=fileName
            self.hull = 0
            self.decks = []
            self.dbbs = []


            if (fileName != ""):
                  self.readProblem()


      def readProblem(self):
            with open(self.filename, newline='') as csvfile:
                  hfr = csv.reader(csvfile, delimiter='\t', quotechar='|')
                  data = []
                  for row in hfr:
                        rown = []
                        for x in row:
                              rown.append(x)
                        data.append(rown)


            abspath1 = '\\'.join(self.filename.split('\\')[0:-1])
            abspath2 = '/'.join(self.filename.split('/')[0:-1])
            if len(abspath2) > len(abspath1):
                  abspath = abspath2 + '/'
            else:
                  abspath = abspath1 + '\\'

            huf_path = abspath + data[0][1]
            block_data_path = abspath + data[1][1]
            #make hull from huf file:
            self.hull= DBBHullForm(huf_path)
            deckz_list= self.hull.pdecks[:-1] #bez keela
            #deckz_list = []
            #deckz_list.append(self.hull.pdecks[2])
            #deckz_list.append(self.hull.pdecks[3])
            #deckz_list.append(self.hull.pdecks[4])
            for deckIndex in range(len(deckz_list)):
```

```
                  self.decks.append(DBBDeck(self.hull, deckz_list[deckIndex],
deckIndex))

            #make blocks
            with open(block_data_path, "r") as csv_file:
                  csv_reader = csv.DictReader(csv_file)
                  for row in csv_reader:  #each row contains 1 block data
                        deck  = int(row["deck"])
                        Ax = float(row["Ax"])
                        Ay = float(row["Ay"])
                        Az = self.decks[deck].z
                        x = float(row["x"])
                        y = float(row["y"])

                        try:
                              z = self.decks[deck - 1].z - self.decks[deck].z
                        except IndexError:
                              print("Invalid deck position")
                        else:
                              block_dims = np.array([x,y,z])
                              position_A = np.array([Ax,Ay])

                              block = DBB(self.hull, self.decks[deck],
block_dims, position_A, abspath)
                              self.dbbs.append(block)

            #print(self.hull.wlinesNeg)
            #print(self.hull.wlinesPos[1])
            #print(self.hull)
            #print(len(self.decks))
            #print(self.dbbs)
            #print(self.filename)
      def testProblem(self, scale, block_dims):
            self.hull= DBBHullForm("", scale)
            self.dbbs.append(DBB(self.hull,0))
            #self.dbbs[-1].setPosition(np.array([0,0,0]))
            self.dbbs[-1].genMesh(block_dims)

            #self.dbbs.append(DBB(self.hull, 0))
            #self.dbbs[-1].setPosition(0, -1, -4)
            #self.dbbs[-1].testMesh()




class DBBHullForm (HullForm):
      def __init__(self, fileName, scale = 1):
            super().__init__(fileName)         #vec u inicijalizaciji stvara
formu
            self.position = np.array([0.,0.,0.])
            self.centroid = np.array([0.,0.,0.])
            self.LOA = self.shipdata["loa_val"]
            self.centroid = np.array([self.LOA / 2, 0, 0])  # sredina samo za x
za sada
            #self.mesh = mf.hard_merge_meshes([self.mesh])

      #def readHullForm(self):
            #HullForm.__init__()


      def regenerateMesh(self):
            self.mesh = mf.make_form(scale = self.scale, move_vector =
self.position)
```

```
      def move(self, move_vector):
            self.position += move_vector
            self.mesh = mf.move_mesh(self.mesh, move_vector)
            #self.regenerateMesh()

      def setPosition(self, new_position):
            old_position = self.position
            self.position = new_position
            self.mesh = mf.move_mesh(self.mesh, new_position - old_position)

      def testMesh(self, scale):
            self.scale = scale
            self.mesh = mf.make_form(scale = self.scale, move_vector =
self.position)


class DBBDeck(Geometry):
      def __init__(self, hullform, z, deckIndex):
            super().__init__()
            self.hullform = hullform
            self.z = z
            self.deckIndex = deckIndex
            self.genMesh()


      def regenerateMesh(self):
            self.mesh = om.TriMesh()

      def genMesh(self):      #za keel koji je na 0 nema wl
            for wline in self.hullform.wlinesPos:
                  if np.isclose(self.z, wline[0][2]):
                        deck_points = np.asarray(wline)
                        break
            self.mesh = mf.make_deck(deck_points, subdivide = False)
            #for i in range(deck_points.shape[0]):
            #     bad_i = []
            #     point = deck_points[i]
            #     next_point = deck_points[(i+1) % deck_points.shape[0]]
            #     if np.allclose(point, next_point):
            #           bad_i.append(i)
            #deck_points = np.delete(deck_points, bad_i, 0)


            #new_points = []
            #for point in deck_points:
            #     if point[1] != 0: #ako point nije na osi x
            #           new_point = copy.copy(point)
            #           new_point[1] = 0
            #           new_points.append(new_point)

            #deck_points = np.append(deck_points, np.asarray(new_points), axis =
0)
            #deck_points = np.asarray(deck_points + new_points)
            #deck_points = np.unique(deck_points, axis = 0)      #duplikat na
tocki x=50? nakon uniqua, arrejevi su close; ne equal pa ih unique ne reze?



            #print(deck_points)

            #print(deck_points[18],deck_points[19],deck_points[20],)
            #print(np.allclose(deck_points[18],deck_points[19]))
            #self.mesh = mf.cut_meshes(self.mesh, self.hullform.mesh)  #predugo
traje
```

```
      def move(self, move_vector):
            self.z += move_vector[2]
            #self.regenerateMesh()
            self.mesh = mf.move_mesh(self.mesh, move_vector)

            #self.position[0] = self.position[0] + dx
            #self.position[1] = self.position[0] + dy
            #self.position[2] = self.position[0] + dz
            #self.regenerateMesh()

      def setPosition(self, new_position):
            old_position = self.position
            self.position = new_position
            self.mesh = mf.move_mesh(self.mesh, new_position - old_position)









class DBB(Geometry):
      def __init__(self, hullform, deck:DBBDeck,block_dims, position, abspath):
            super().__init__()
            self.folderpath = abspath
            self.hullform= hullform
            self.deck=deck
            self.position = np.array([position[0],position[1],deck.z])
            self.block_dims=block_dims
            self.genMesh()
            self.cutMesh()
            #print(self.hullform.filename)

      def regenerateMesh(self):
            self.mesh= mf.make_block(block_dims = self.block_dims, move_vector =
self.position)

      def move(self, move_vector):
            self.position += move_vector
            #self.regenerateMesh()
            self.mesh = mf.move_mesh(self.mesh, move_vector)

            #self.position[0] = self.position[0] + dx
            #self.position[1] = self.position[0] + dy
            #self.position[2] = self.position[0] + dz
            #self.regenerateMesh()

      def setPosition(self, new_position):

            self.position = new_position
            self.genMesh()

            old_position = self.position
            self.position = new_position
            self.mesh = mf.move_mesh(self.mesh, new_position - old_position)

      def genMesh(self):
            self.mesh = mf.make_block_from_unit_csv(self.block_dims,
self.position, self.folderpath)
            #self.mesh= mf.make_block(block_dims = self.block_dims, move_vector
```

```
= self.position)

      def cutMesh(self):
            mf.fit_block_to_form(self.mesh, self.block_dims, self.position,
self.hullform.mesh)
            #mf.cut_meshes(self.mesh, self.hullform.mesh)
            pass

      def calcVolume(self):
            print(mf.calc_mesh_volume(self.mesh))

      def IsClosed(self):
            print(mf.is_mesh_closed(self.mesh))
```

dbbcommand.py modul

```
from PySide2.QtWidgets import QApplication, QMenu, QMessageBox
from PySide2.QtWidgets import QDialog, QPushButton,QGridLayout
from commands import Command
from iohandlers import IOHandler
import openmesh as om
import numpy as np
from signals import Signals
from geometry import Geometry
from dbb import DBBProblem,DBB,DBBHullForm, DBBDeck
import os
from PySide2.QtCore import Slot
import dbbmenus as mm


class DBBCommand(Command):
      def __init__(self):
            super().__init__()
            app = QApplication.instance()
            importer=DBBImporter()
            importer.fsetproblem=self.setProblem
            app.registerIOHandler(importer)
            self.mainwin = app.mainFrame
            self.dbbprop = DialogDBBProps(self.mainwin)
            self.dbbproblem=0
            self.si=0

            #tools = app.mainFrame.menuTools
            mb = app.mainFrame.menuBar()

            self.menuMain = QMenu("DBB")

            self.menuImportFromCsv = self.menuMain.addAction("Import From Csv")
            self.menuImportFromCsv.triggered.connect(self.onImportFromCsv)

            self.menuInitTestProblem = self.menuMain.addAction("TestProblem")

      self.menuInitTestProblem.triggered.connect(self.onGenerateTestProblerm)

            self.menuModifyBlock = QMenu("&Modify block")
            self.menu2 = QMenu("&Menu2")
            self.menuMain.addMenu(self.menuModifyBlock)
            self.menuMain.addMenu(self.menu2)



            menuMove = self.menuModifyBlock.addAction("Move")
            menuMove.triggered.connect(self.onMoveDBB)

            menuSetPosition = self.menuModifyBlock.addAction("Set Position")
            menuSetPosition.triggered.connect(self.onSetPosition)

            self.menuDeleteSelected = self.menuModifyBlock.addAction("Delete
Selected")
            self.menuDeleteSelected.triggered.connect(self.onDeleteSelected)

            menuCutBlock = self.menuModifyBlock.addAction("Cut Block")
            menuCutBlock.triggered.connect(self.onCutBlock)

            menuMeshVolume = self.menu2.addAction("Mesh Volume")
            menuMeshVolume.triggered.connect(self.onMeshVolume)
```

```
            menuIsClosed = self.menu2.addAction("Is Closed?")
            menuIsClosed.triggered.connect(self.onIsClosed)
            #tools.addMenu(menu)
            mb.addMenu(self.menuMain)
            #self.menuMain.setEnabled(False)


            Signals.get().geometryAdded.connect(self.registerDBB)
            Signals.get().selectionChanged.connect(self.registerSelection)
            self.dbb = 0

      def setProblem(self,dbbproblem):
            self.dbbproblem=dbbproblem

      @Slot()
      def registerDBB(self, dbbproblem):
            if isinstance(dbbproblem,DBBProblem):
                  self.dbbproblem=dbbproblem
                  self.menuMain.setEnabled(True)

      @Slot()
      def registerSelection(self, si):
            self.si=si

      def onGenerateTestProblerm(self):
            FormMenu = mm.AddForm_Dialog()
            scale = FormMenu.run()

            BlockMenu = mm.AddBlock_Dialog()
            block_dims = BlockMenu.run()

            if block_dims is not None and scale is not None:     #ako niti jedan
od menia nije cancelan
                  self.dbbproblem = DBBProblem("")
                  self.dbbproblem.testProblem(scale, block_dims)
                  Signals.get().geometryImported.emit(self.dbbproblem.hull)
                  for deck in self.dbbproblem.decks:
                        Signals.get().geometryImported.emit(deck)
                  for dbb in self.dbbproblem.dbbs:
                        Signals.get().geometryImported.emit(dbb)
                  self.menuInitTestProblem.setEnabled(False)

      def onDeleteSelected(self):
            if self.si.haveSelection():
                  currDBB=self.si.getGeometry()
                  print(self.dbbproblem)
                  if isinstance(currDBB, DBB) or isinstance(currDBB,
DBBHullForm) or isinstance(currDBB, DBBDeck):
                        self.dbbproblem.dbbs.remove(currDBB)
                        Signals.get().geometryRemoved.emit(currDBB)
      #refresha!!!!!!

      def onMoveDBB(self):
            if self.si.haveSelection():
                  currDBB=self.si.getGeometry()
                  if isinstance(currDBB, DBB) or isinstance(currDBB,
DBBHullForm) or isinstance(currDBB, DBBDeck):
                        MoveMenu = mm.Move_Dialog()
                        move_vector = MoveMenu.run()
                        if move_vector is not None:
                              currDBB.move(move_vector)
                              Signals.get().geometryRebuild.emit(currDBB)
      #refresha!!!!!!
                              #self.dbbprop.setCurrentDBB(currDBB)
```

```
                              #self.dbbprop.moveCurrentDBB()
                              ##

      def onSetPosition(self):
            if self.si.haveSelection():
                  currDBB=self.si.getGeometry()
                  if isinstance(currDBB,DBB) or isinstance(currDBB,DBBHullForm)
or isinstance(currDBB,DBBDeck):
                        SetPositionMenu = mm.SetPosition_Dialog()
                        new_position = SetPositionMenu.run()
                        if new_position is not None:
                              currDBB.setPosition(new_position)
                              Signals.get().geometryRebuild.emit(currDBB)
      #refresha!!!!!!

      def onCutBlock(self):
            if self.si.haveSelection():
                  currDBB=self.si.getGeometry()
                  if isinstance(currDBB,DBB):
                        currDBB.cutMesh()
                        Signals.get().geometryRebuild.emit(currDBB)
      #refresha!!!!!!

      def onMeshVolume(self):
            if self.si.haveSelection():
                  currDBB=self.si.getGeometry()
                  if isinstance(currDBB,DBB):
                        currDBB.calcVolume()


      def onIsClosed(self):
            if self.si.haveSelection():
                  currDBB=self.si.getGeometry()
                  if isinstance(currDBB,DBB):
                        currDBB.IsClosed()

      def onImportFromCsv(self):
            ImportFromCsvMenu = mm.ImportFromCsvMenu_Dialog()
            folder_path = ImportFromCsvMenu.run()    #dodaj jos uvijet da se u
folderu nalaze prave datoteke


            self.dbbproblem = DBBProblem("")
            self.dbbproblem.readProblem(folder_path)
            #make form from huf
            #with open(huf_path, "r") as csv:

            #tu ih emmita u vizualizaciju
            Signals.get().geometryImported.emit(self.dbbproblem.hull)
            for deck in self.dbbproblem.decks:
                  Signals.get().geometryImported.emit(deck)
            for dbb in self.dbbproblem.dbbs:
                  Signals.get().geometryImported.emit(dbb)
            #self.menuInitTestProblem.setEnabled(False)
```

```
class DBBImporter(IOHandler):
      def __init__(self):
            super().__init__()
            self.fsetproblem=0

      def importGeometry(self, fileName):
            if len(fileName) < 1:
                  return
            filename, file_extension = os.path.splitext(fileName)
            if file_extension != ".dbb":
                  return
            self.dbbproblem = DBBProblem(fileName)
            self.fsetproblem(self.dbbproblem)
            print(self.dbbproblem)
            Signals.get().geometryImported.emit(self.dbbproblem.hull)
            for deck in self.dbbproblem.decks:
                  Signals.get().geometryImported.emit(deck)
            for dbb in self.dbbproblem.dbbs:
                  Signals.get().geometryImported.emit(dbb)

      def getImportFormats(self):
            return [".dbb"]

class DialogDBBProps(QDialog):
      def __init__(self, parent):
            super().__init__(parent)
            self.mainwin = parent
            self.btnMove = self.createButton("&Move", self.moveCurrentDBB)

            mainLayout = QGridLayout()
            mainLayout.addWidget(self.btnMove, 0, 0)
            self.setLayout(mainLayout)
            self.currentDBB=0


      def createButton(self, text, member):
            button = QPushButton(text)
            button.clicked.connect(member)
            return button

      def moveCurrentDBB(self):
            self.currentDBB.move(1, 0, 0)
            Signals.get().geometryRebuild.emit(self.currentDBB)
      #refresha?

      def setCurrentDBB(self, currentDBB):
            self.currentDBB = currentDBB
            self.setWindowTitle("Move DBB")

def createCommand():
      return DBBCommand()
```

myfunctions.py modul

```
import openmesh as om
import numpy as np
import copy
import sys
import csv

def array_where_equal(a,b, bool = True):       #ako hocemo stupce samo stavi a.T
, ako hocemo i gdje je razlicito stavimo bool = False
      i_array = np.empty(0, dtype = "int64")
      for i in range(a.shape[0]):
            if np.array_equal(a[i], b) == bool:
                  i_array = np.append(i_array, i)

      return i_array


def flip_mesh_face_orientation(mesh):
      points = mesh.points()
      flipped_fvi = np.flip(mesh.face_vertex_indices(), axis = 1)
      return om.TriMesh(points, flipped_fvi)







def soft_merge_meshes(meshes):     #meshes je lista sa meshevima
      points = np.empty((0,3))
      face_vertex_indices = np.empty((0,3))

      for mesh in meshes:
            if len(mesh.face_vertex_indices().shape) == 1:       #za slucaj da
nema fvi u mesh
                  add_fvi = np.empty((0,3))
            else:
                  add_fvi = mesh.face_vertex_indices()
            face_vertex_indices = np.append(face_vertex_indices, add_fvi +
points.shape[0], axis = 0)                     #+points.chape[0] je tu da poreda
face_vertex_indices sa njihovim indexom u novom arrayu
            points = np.append(points, mesh.points(), axis = 0)

      combined_mesh = om.TriMesh(points, face_vertex_indices)

      return combined_mesh




def hard_merge_meshes(meshes):   #meshes je lista sa meshevima
      merged_mesh = soft_merge_meshes(meshes)
      data = np.unique(merged_mesh.points(), return_counts = True, axis = 0)
      unique_points = data[0]
      duplicate_counter = data[1]
      points_with_duplicates = unique_points[np.where(duplicate_counter > 1)]
      new_vh = []
      for dpoint in points_with_duplicates:
            new_vh.append(merged_mesh.add_vertex(dpoint))

      bad_fh_list = []
```

```
      new_fvi_list = []


      merged_mesh_fvi = merged_mesh.face_vertex_indices().tolist()
      merged_mesh_points = merged_mesh.points()
      for i in range(len(merged_mesh_fvi)):    #trazimo bad fh i mijenjamo
njihov fvi u novi
            fvi = np.asarray(merged_mesh_fvi[i])
            face_points = merged_mesh_points[fvi]
            new_fvi = copy.copy(fvi)
            for nvh in new_vh:           #trazi jeli novi vh i face imaju istu
tocku
                  new_point = merged_mesh_points[nvh.idx()]

                  new_fvi[array_where_equal(face_points, new_point)] = nvh.idx()
            if np.array_equal(fvi, new_fvi) == False:      #ako originalni i
novi fvi nisu isti dodaje novi fvi u listu
                  fh = merged_mesh.face_handle(i)
                  bad_fh_list.append(fh)
                  new_fvi_list.append(new_fvi)


      for bad_fh in bad_fh_list:                        #delete bad faces:
            merged_mesh.delete_face(bad_fh, False)   #false da ne deletea
izolirane vertexe

      merged_mesh.garbage_collection()


      for new_fvi in new_fvi_list:                  #retriangularizacija sa
novim fvi
            new_face_vhandles = []
            for vi in new_fvi:
                  new_vh = merged_mesh.vertex_handle(vi)
                  new_face_vhandles.append(new_vh)

            merged_mesh.add_face(new_face_vhandles)

      delete_isolated_vertices(merged_mesh)

      return merged_mesh

def delete_isolated_vertices(mesh):
      mesh_points = mesh.points().tolist()
      mesh_vertex_face_indices = mesh.vertex_face_indices().tolist() #kod
vertex_face_indices izolirani su oni kojima je svima -1 (arrajevi moraju biti
svi istevelicine pa je null -1)
      for vh_idx in range(len(mesh_points)):

            neighbouring_faces_fh_idx =
np.asarray(mesh_vertex_face_indices[vh_idx])
            if np.all(neighbouring_faces_fh_idx == -1):
                  vh = mesh.vertex_handle(vh_idx)
                  mesh.delete_vertex(vh)
      mesh.garbage_collection()


def triangle_surface(triangle_points):
      AB = triangle_points[1]-triangle_points[0]
      AC = triangle_points[2]-triangle_points[0]
      surface = np.linalg.norm(np.cross(AB, AC))/2
      return surface
```

```
def is_inside_triangle(point, triangle_points):
      ABC = triangle_surface(triangle_points)
      Ai = np.empty((0))
      for i in range(triangle_points.shape[0]):
            points = copy.copy(triangle_points)
            points[i] = point
            Ai = np.append(Ai, triangle_surface(points))

      if np.isclose(np.sum(Ai), ABC) == True:
            return True
      else:
            return False



def make_block(block_dims = np.array([20,6,3]), move_vector =
np.array([0,0,0])):
      mesh = om.TriMesh()
      axes = []
      #stvara 2 tocke na svakoj osi
      for dim in block_dims:
            axes.append(np.linspace(0, dim, 2))

      block_corners = np.asarray(np.meshgrid(*axes)).T.reshape(8,3)
      block_corners += move_vector

      corner_vertices = []
      for corner in block_corners:
            corner_vertices.append(mesh.add_vertex(corner))



      #x+face
      mesh.add_face(corner_vertices[2],corner_vertices[3],corner_vertices[6])
      mesh.add_face(corner_vertices[3],corner_vertices[7],corner_vertices[6])

      #x-face
      mesh.add_face(corner_vertices[0],corner_vertices[4],corner_vertices[1])
      mesh.add_face(corner_vertices[1],corner_vertices[4],corner_vertices[5])

      #y+face
      mesh.add_face(corner_vertices[3],corner_vertices[1],corner_vertices[5])
      mesh.add_face(corner_vertices[3],corner_vertices[5],corner_vertices[7])

      #y-face
      mesh.add_face(corner_vertices[0],corner_vertices[2],corner_vertices[4])
      mesh.add_face(corner_vertices[2],corner_vertices[6],corner_vertices[4])

      #z+face
      mesh.add_face(corner_vertices[4],corner_vertices[6],corner_vertices[5])
      mesh.add_face(corner_vertices[6],corner_vertices[7],corner_vertices[5])

      #z-face
      mesh.add_face(corner_vertices[2],corner_vertices[0],corner_vertices[1])
      mesh.add_face(corner_vertices[2],corner_vertices[1],corner_vertices[3])



      return mesh

def make_deck(wline_points, subdivide = False):
      #clean duplicates
      wline_points = np.unique(wline_points, axis = 0)
```

```
      central_points = np.empty((0,3))
      for point in wline_points:
            if np.isclose(point[1], 0) ==  False:
                  central_point = copy.copy(point)
                  central_point[1] = 0
                  central_points = np.append(central_points,
np.expand_dims(central_point, 0), axis = 0)

      deck_points = np.append(wline_points, central_points, axis = 0)


      w_max_index = wline_points.shape[0]
      c_max_index = central_points.shape[0]

      #pocetni i zadnji fvi koji nemogu u petlju
      deck_fvi = np.array([[0,0+w_max_index,1],[deck_points.shape[0]-
1,w_max_index-1,w_max_index-2]])


      for interval_index in range(len(central_points-1)):
            fvi = np.array([[w_max_index,w_max_index+1,1],[w_max_index+1,2,1]])
+ interval_index
            deck_fvi = np.append(deck_fvi, fvi, axis = 0)

      if subdivide == False:
            return om.TriMesh(deck_points, deck_fvi)
      elif subdivide == True:
            return subdivide_mesh([om.TriMesh(deck_points, deck_fvi)])

def move_mesh(mesh, move_vector):
      for vh in mesh.vertices():
            new_point = mesh.points()[vh.idx()] + move_vector
            mesh.set_point(vh, new_point)
      return mesh


def get_intersection(radius, vector, face_points):
      face_radius = face_points[0]
      face_vector1 = face_points[1] - face_points[0]
      face_vector2 = face_points[2] - face_points[0]

      #face_radius, face_vector1, face_vector2,

      radius_matrix = (radius - face_radius).T
      vector_matrix = np.empty((3,3))
      vector_matrix[:,0] = face_vector1
      vector_matrix[:,1] = face_vector2
      vector_matrix[:,2] = -vector

      try:
            edge_parameter = np.linalg.solve(vector_matrix, radius_matrix)[2]
      except:
            return (None, None)

      intersection_point = radius + (vector * edge_parameter)
      if is_inside_triangle(intersection_point, face_points) == True:
            return (intersection_point, edge_parameter)
      else:
            return (None, edge_parameter)



def is_mesh_closed(mesh):
      for eh in mesh.edges():       #check if mesh has any boundary edges if not
```

```
mesh is closed, if yes mesh is open
            if mesh.is_boundary(eh) == True:
                  return False
      else:
            return True


def subdivide_mesh(mesh_list, c = 0 ,n = 1): #face po face subdividamo n puta,c
je counter
      if c < n:
            new_meshes = []
            for mesh in mesh_list:
                  mesh_points =  mesh.points()
                  mesh_fvi = mesh.face_vertex_indices().tolist()
                  mesh_hei = mesh.face_halfedge_indices().tolist() #lista sa 3
vrijednosti unutra
                  face_hevi = mesh.halfedge_vertex_indices().tolist() #heh idx -
> vertex indices  #lista [.........] velika sa slistama od 2 point =
points[vindices]
                  for i in range(len(mesh_fvi)): #i je idx od fh
                        face_points = np.empty((0,3))
                        midpoints = np.empty((0,3))
                        for j in mesh_hei[i]: #j je idx od heh za halfedgeve na
tom faceu /za svaki halfedge handleidx u faceu

                              hevi = (face_hevi[j])   #tu se vrti
                              halfedge_points = mesh_points[hevi] #array
                              face_points = np.append(face_points,
np.expand_dims(halfedge_points[0], axis = 0), axis = 0)    # da se zadrzi
orijentacija
                              midpoint = halfedge_points[0] +
(halfedge_points[1] - halfedge_points[0]) * 0.5
                              midpoints =  np.append(midpoints,
np.expand_dims(midpoint, axis = 0), axis = 0)
                        new_mesh = om.TriMesh()
                        vhandles = []
                        fhandles = []
                        for point in np.append(face_points, midpoints, axis =
0):
                              vhandles.append(new_mesh.add_vertex(point))


                        fhandles.append(new_mesh.add_face(vhandles[0],
vhandles[3], vhandles[5]))
                        fhandles.append(new_mesh.add_face(vhandles[3],
vhandles[1], vhandles[4]))
                        fhandles.append(new_mesh.add_face(vhandles[5],
vhandles[3], vhandles[4]))
                        fhandles.append(new_mesh.add_face(vhandles[5],
vhandles[4], vhandles[2]))
                        new_meshes.append(new_mesh)

            return subdivide_mesh(new_meshes, c = c + 1, n = n)

      else:
            return hard_merge_meshes(mesh_list)




def plot3D(points_list):
      scatter_points = []
      colors =
["red","green","yellow","blue","orange","purple","black","cyan","magneta"]
```

```
	fig=plt.figure()
	axes = plt.axes(projection='3d')
	axes.set_xlabel("x")
	axes.set_ylabel("y")
	axes.set_zlabel("z")

	for i in range(len(points_list)):
		points = points_list[i]
		color = colors[i % (len(colors)-1)]
		x = points[:,0]
		y = points[:,1]
		z = points[:,2]
		scatter_points.append(get_scatter(axes, x, y, z, color))

	#points1 = axes.scatter3D(points[:,0],points[:,1],points[:,2], color =
"green");
	#points2 =
axes.scatter3D(supertr_points[:,0],supertr_points[:,1],supertr_points[:,2],
color = "red");
	#point_center =
axes.scatter3D(points_centroid[0],points_centroid[1],points_centroid[2], color =
"blue");
	#supertr_center =
axes.scatter3D(supertr_centroid[0],supertr_centroid[1],supertr_centroid[2],
color = "black");
	plt.show()



def calc_face_volume(face_points):
	a = face_points[0]
	b = face_points[1]
	c = face_points[2]
	volume = np.abs(np.dot(a, np.cross(b,c))) / 6
	return volume

def calc_mesh_volume(mesh):
	mesh_volume = 0
	for fh in mesh.faces():
		face_normal = mesh.calc_face_normal(fh)
		face_centroid = mesh.calc_face_centroid(fh)
		fvi = mesh.face_vertex_indices()[fh.idx()]
		face_points = mesh.points()[fvi]
		face_volume = calc_face_volume(face_points)
		face_sign = np.sign(np.dot(face_centroid, face_normal))
		mesh_volume += face_volume * face_sign
	return mesh_volume

def make_block_csv_file():
	block_dims = np.array([1,1,1])
	mesh = make_block(block_dims)
	mesh = subdivide_mesh([mesh], n = 1)
	points = mesh.points().tolist()
	fvi = mesh.face_vertex_indices().tolist()

	with open("unit_block_points.csv", "w", newline = "") as csv_file:
		csv_writer = csv.writer(csv_file)
		for point in points:
			csv_writer.writerow([point[0],point[1],point[2]])

	with open("unit_block_fvi.csv", "w", newline = "") as csv_file:
		csv_writer = csv.writer(csv_file)
		for f in fvi:
			csv_writer.writerow([f[0],f[1],f[2]])
```

```
def   make_block_from_unit_csv(block_dims = np.array([1,1,1]), move_vector =
np.array([0,0,0]), path = ""):
      with open(path + "unit_block_points.csv", "r", newline ="") as csv_file:
            csv_reader = csv.reader(csv_file)
            points = np.asarray([line for line in csv_reader]).astype(float)

      with open(path + "unit_block_fvi.csv", "r", newline ="") as csv_file:
            csv_reader = csv.reader(csv_file)
            fvi = np.asarray([line for line in csv_reader]).astype(int)


      return om.TriMesh(points * block_dims + move_vector, fvi)

def is_point_inside_form_mesh_y(point, form_fh_idx_to_check, form_mesh): #ovdje
je moguc samo jedan intersection sa formom( ako idemo od xz ravnine)
      point_vector_j = np.array([0,1,0])
      form_mesh_points = form_mesh.points()
      form_mesh_fvi = form_mesh.face_vertex_indices().tolist()

      for fh_idx in form_fh_idx_to_check:
            fvi = form_mesh_fvi[fh_idx]
            face_points = form_mesh_points[fvi]
            if is_inside_triangle(point, face_points):     #jeli point na nekom
faceu
                  return (True, point)

            #print(point, face_points)
            intersection_point = get_intersection(point, point_vector_j,
face_points)[0]
            #print(intersection_point)
            if intersection_point is not None:
                  if abs(point[1]) <= abs(intersection_point[1]):
      #ako je abs(y) manji ili jednak od abs(IPy) : point lezi u meshu
                        return (True, intersection_point)
                  else:
                        return (False, intersection_point)

      else:
            return (None, point)


def is_point_inside_form_mesh_x(point, form_fh_idx_to_check, form_mesh): #ovdje
je moguc samo jedan intersection sa formom( ako idemo od xz ravnine)
      point_vector_x = np.array([1,0,0])
      form_mesh_points = form_mesh.points()
      form_mesh_fvi = form_mesh.face_vertex_indices().tolist()

      for fh_idx in form_fh_idx_to_check:
            fvi = form_mesh_fvi[fh_idx]
            face_points = form_mesh_points[fvi]
            if is_inside_triangle(point, face_points):     #jeli point na nekom
faceu
                  return (True, point)

            intersection_point = get_intersection(point, point_vector_x,
face_points)[0]
            if intersection_point is not None:
                  xi = intersection_point[0]
                  xp = point[0]
                  delta = xi - xp
                  fh = form_mesh.face_handle(fh_idx)
                  f_normal = form_mesh.calc_face_normal(fh)
                  if f_normal[0] < 0:     #face je na krmi
```

```
                        if delta < 0:
                              return (True, point)
                        else:
                              return (False, intersection_point)

                  elif f_normal[0] > 0: #face je na palubi
                        if delta > 0:
                              return (True, point)
                        else:
                              return (False, intersection_point)

      return (True, point)


#ekskluzivno za formu broda
def fit_block_to_form(block_mesh, block_dims, block_position, form_mesh):
      #1) koji facevi su blizu blocka
      form_mesh_fvi = form_mesh.face_vertex_indices().tolist()
      form_mesh_vfi = form_mesh.vertex_face_indices().tolist()
      form_mesh_ffi = form_mesh.face_face_indices().tolist()
      form_mesh_points = form_mesh.points()
      xmin = block_position[0]
      xmax = block_position[0] + block_dims[0]
      ymin = block_position[1]
      ymax = block_position[1] + block_dims[1]
      zmin = block_position[2]
      zmax = block_position[2] + block_dims[2]

      near_fh_idx_list = []

      #trazi fh_idx od svih faceva koji imaju tocku u projekciji sa blokm na xz
ravninu
      for vh_idx in range(form_mesh_points.shape[0]):
            point = form_mesh_points[vh_idx]
            if (xmin <= point[0] <= xmax) and (ymin <= point[1] <= ymax) and
(zmin <= point[2] <= zmax): #ako je point izmedju projekcije
                  neighbouring_fh_idx = form_mesh_vfi[vh_idx]
                  near_fh_idx_list += neighbouring_fh_idx

      #micanje duplikata i -1 u listi
      near_fh_idx_list = set(near_fh_idx_list)
      near_fh_idx_list = list(near_fh_idx_list)
      try:
            near_fh_idx_list.remove(-1)
      except:
            pass
      #trazenje susjednih facea susjednim facevima
      extra_fh_idx = []
      for fh_idx in near_fh_idx_list:
            neighbouring_fh_idx = form_mesh_ffi[fh_idx]
            extra_fh_idx += neighbouring_fh_idx

      near_fh_idx_list += extra_fh_idx
      #micanje duplikata i -1 u listi
      near_fh_idx_list = set(near_fh_idx_list)
      near_fh_idx_list = list(near_fh_idx_list)
      try:
            near_fh_idx_list.remove(-1)
      except:
            pass

      #trazenje jos susjednih faceva
      extra_fh_idx = []
      for fh_idx in near_fh_idx_list:
```

```
            neighbouring_fh_idx = form_mesh_ffi[fh_idx]
            extra_fh_idx += neighbouring_fh_idx

      near_fh_idx_list += extra_fh_idx
      #micanje duplikata i -1 u listi
      near_fh_idx_list = set(near_fh_idx_list)
      near_fh_idx_list = list(near_fh_idx_list)
      try:
            near_fh_idx_list.remove(-1)
      except:
            pass

      #jesu li facevi na krmi ili palubi?
      fh = form_mesh.face_handle(near_fh_idx_list[0])
      normal = form_mesh.calc_face_normal(fh)
      normal = np.sign(normal[0])


      #micanje tocaka na formu po y:
      block_mesh_points = block_mesh.points()
      x_outside_points_vh_idx = []
      x_inside_points_vh_idx = []
      for vh_idx in range(block_mesh_points.shape[0]):
            block_point = block_mesh_points[vh_idx]
            data = is_point_inside_form_mesh_y(block_point, near_fh_idx_list,
form_mesh)  #ispitivanje po y osi

            if data[0] == False:
                  intersection_point = data[1]
                  vh = block_mesh.vertex_handle(vh_idx)
                  block_mesh.set_point(vh, intersection_point)
                  x_inside_points_vh_idx.append(vh_idx)
            elif data[0] == None:
                  x_outside_points_vh_idx.append(vh_idx)

      if len(x_outside_points_vh_idx) != 0:    #ako ima outside tocaka
            block_mesh_points = block_mesh.points()
            #block_inside_points = block_mesh_points[x_inside_points_vh_idx]
            block_outside_points = block_mesh_points[x_outside_points_vh_idx]

            if normal > 0:    #pramac     min y na max x
                  for vh_idx in x_outside_points_vh_idx:
                        outside_point =  block_mesh_points[vh_idx]
                        vh = block_mesh.vertex_handle(vh_idx)
                        point_to_set_to =
np.array([outside_point[0],block_position[1],outside_point[2]])
                        block_mesh.set_point(vh, point_to_set_to)

            elif normal < 0: #krma  max y na min x
                  block_inside_points =
block_mesh_points[x_inside_points_vh_idx]
                  block_zs = np.unique(block_mesh_points[:,2]) #trebam li ovo
refreshat?
                  for block_z in block_zs:
                        block_inside_points_with_same_z =
np.unique(block_inside_points[np.where(block_inside_points[:,2] == block_z)],
axis = 0)
                        block_outside_points_with_same_z =
np.unique(block_outside_points[np.where(block_outside_points[:,2] == block_z)],
axis = 0)

                        min_x_points =
block_inside_points_with_same_z[np.where(block_inside_points_with_same_z[:,0] ==
np.min(block_inside_points_with_same_z[:,0]))]
```

```
                        max_y_point = min_x_points[np.where(min_x_points[:,1] ==
np.max(min_x_points[:,1]))][0]     #0 na kraju za slucaj da ih je vise na istoj
tocki

                        for vh_idx in x_outside_points_vh_idx:
                              outside_point = block_mesh_points[vh_idx]
                              if abs(outside_point[1]) > abs(max_y_point[1]):
                                    vh = block_mesh.vertex_handle(vh_idx)
                                    point_to_set_to =
np.array([outside_point[0],max_y_point[1],outside_point[2]])
                                    block_mesh.set_point(vh, point_to_set_to)


            block_mesh_points = block_mesh.points()
            for vh_idx in x_outside_points_vh_idx:
                  block_point = block_mesh_points[vh_idx]
                  data = is_point_inside_form_mesh_x(block_point,
near_fh_idx_list, form_mesh)  #ispitivanje po x osi
                  if data[0] == False:
                        intersection_point = data[1]
                        vh = block_mesh.vertex_handle(vh_idx)
                        block_mesh.set_point(vh, intersection_point)
```

dbbmenus.py modul

```
# -*- coding: utf-8 -*-

################################################################################
## Form generated from reading UI file 'AddBlock.ui'
##
## Created by: Qt User Interface Compiler version 5.14.0
##
## WARNING! All changes made in this file will be lost when recompiling UI file!
################################################################################

from PySide2.QtCore import (QCoreApplication, QMetaObject, QObject, QPoint,
      QRect, QSize, QUrl, Qt, SIGNAL)
from PySide2.QtGui import (QBrush, QColor, QConicalGradient, QFont,
      QFontDatabase, QIcon, QLinearGradient, QPalette, QPainter, QPixmap,
      QRadialGradient)
from PySide2.QtWidgets import *
import numpy as np
import sys
import os

class AddBlock_Dialog(object):
      def setupUi(self, Dialog):
            if Dialog.objectName():
                  Dialog.setObjectName(u"Dialog")
            Dialog.resize(356, 126)
            self.buttonBox = QDialogButtonBox(Dialog)
            self.buttonBox.setObjectName(u"buttonBox")
            self.buttonBox.setGeometry(QRect(250, 60, 81, 61))
            self.buttonBox.setOrientation(Qt.Vertical)
            self.buttonBox.setStandardButtons(QDialogButtonBox.Cancel|
QDialogButtonBox.Ok)
            self.verticalLayoutWidget = QWidget(Dialog)
            self.verticalLayoutWidget.setObjectName(u"verticalLayoutWidget")
            self.verticalLayoutWidget.setGeometry(QRect(10, 40, 41, 71))
            self.verticalLayout = QVBoxLayout(self.verticalLayoutWidget)
            self.verticalLayout.setObjectName(u"verticalLayout")
            self.verticalLayout.setContentsMargins(0, 0, 0, 0)
            self.XLabel = QLabel(self.verticalLayoutWidget)
            self.XLabel.setObjectName(u"XLabel")

            self.verticalLayout.addWidget(self.XLabel)

            self.YLabel = QLabel(self.verticalLayoutWidget)
            self.YLabel.setObjectName(u"YLabel")

            self.verticalLayout.addWidget(self.YLabel)

            self.ZLabel = QLabel(self.verticalLayoutWidget)
            self.ZLabel.setObjectName(u"ZLabel")

            self.verticalLayout.addWidget(self.ZLabel)

            self.verticalLayoutWidget_2 = QWidget(Dialog)
            self.verticalLayoutWidget_2.setObjectName(u"verticalLayoutWidget_2")
            self.verticalLayoutWidget_2.setGeometry(QRect(70, 40, 160, 71))
            self.verticalLayout_2 = QVBoxLayout(self.verticalLayoutWidget_2)
            self.verticalLayout_2.setObjectName(u"verticalLayout_2")
            self.verticalLayout_2.setContentsMargins(0, 0, 0, 0)
            self.XInputLine = QLineEdit(self.verticalLayoutWidget_2)
            self.XInputLine.setObjectName(u"XInputLine")
```

```
            self.verticalLayout_2.addWidget(self.XInputLine)

            self.YInputLine = QLineEdit(self.verticalLayoutWidget_2)
            self.YInputLine.setObjectName(u"YInputLine")

            self.verticalLayout_2.addWidget(self.YInputLine)

            self.ZInputLine = QLineEdit(self.verticalLayoutWidget_2)
            self.ZInputLine.setObjectName(u"ZInputLine")

            self.verticalLayout_2.addWidget(self.ZInputLine)

            self.DimensionNameLabel = QLabel(Dialog)
            self.DimensionNameLabel.setObjectName(u"DimensionNameLabel")
            self.DimensionNameLabel.setGeometry(QRect(10, 10, 89, 23))
            #buttonbox connections
            self.retranslateUi(Dialog)

            self.buttonBox.accepted.connect(Dialog.accept)       #accept and
reject are slots
            self.buttonBox.rejected.connect(Dialog.reject)

            #Dialog.accept.connect(print("hahaah"))


            QMetaObject.connectSlotsByName(Dialog)
      # setupUi



      def retranslateUi(self, Dialog):
            Dialog.setWindowTitle(QCoreApplication.translate("Dialog", u"Add
Block", None))
            self.XLabel.setText(QCoreApplication.translate("Dialog", u"X:",
None))
            self.YLabel.setText(QCoreApplication.translate("Dialog", u"Y:",
None))
            self.ZLabel.setText(QCoreApplication.translate("Dialog", u"Z:",
None))
            self.XInputLine.setText(QCoreApplication.translate("Dialog", u"15",
None))
            self.YInputLine.setText(QCoreApplication.translate("Dialog", u"15",
None))
            self.ZInputLine.setText(QCoreApplication.translate("Dialog", u"15",
None))
            self.DimensionNameLabel.setText(QCoreApplication.translate("Dialog",
u"Block Dimensions:", None))
      # retranslateUi

      def getInput(self):
            return np.array([float(self.XInputLine.text()),
float(self.YInputLine.text()), float(self.ZInputLine.text())])   #uzima text iz
svakog linea i pretvara ih u float i stavlja u array


      def run(self):                #app = QApplication
            Form = QDialog()        #Form je oblik ; Qwidget je emptybox a
Qdialogue je menu sa ok i cancel
            #self = AddBlock_Dialog()          #ui je sta se sve nalazi u menu
            self.setupUi(Form)            #setappa ui (ocito)
            Form.exec()                   #show je preview pogledaj modal
dialogue u dokumentaciji (modalni blokiraju acsess ostatku aplikacije dok nije
zavrsena)   #pokrece novi menu
            if Form.result() == True:  #ako je pritisnut ok
```

```
                while True:
                        try:
                                block_dims = self.getInput()
                        except:
                                print("numbers only")
                                Form.exec()
                                if Form.result() == False:
                                        break
                        else:
                                print(block_dims)
                                return block_dims
                                break

class AddForm_Dialog(object):
        def setupUi(self, Dialog):
                if Dialog.objectName():
                        Dialog.setObjectName(u"Dialog")
                Dialog.resize(240, 74)
                self.buttonBox = QDialogButtonBox(Dialog)
                self.buttonBox.setObjectName(u"buttonBox")
                self.buttonBox.setGeometry(QRect(150, 10, 81, 301))
                self.buttonBox.setOrientation(Qt.Vertical)
                self.buttonBox.setStandardButtons(QDialogButtonBox.Cancel|
QDialogButtonBox.Ok)
                self.ScaleLabel = QLabel(Dialog)
                self.ScaleLabel.setObjectName(u"ScaleLabel")
                self.ScaleLabel.setGeometry(QRect(10, 10, 61, 21))
                self.ScaleInputLine = QLineEdit(Dialog)
                self.ScaleInputLine.setObjectName(u"ScaleInputLine")
                self.ScaleInputLine.setGeometry(QRect(10, 40, 113, 20))

                self.retranslateUi(Dialog)
                self.buttonBox.accepted.connect(Dialog.accept)
                self.buttonBox.rejected.connect(Dialog.reject)

                QMetaObject.connectSlotsByName(Dialog)
        # setupUi

        def retranslateUi(self, Dialog):
                Dialog.setWindowTitle(QCoreApplication.translate("Dialog", u"Add
Form", None))
                self.ScaleLabel.setText(QCoreApplication.translate("Dialog", u"Form
Scale:", None))
                self.ScaleInputLine.setText(QCoreApplication.translate("Dialog",
u"5", None))
        # retranslateUi

        def getInput(self):
                return float(self.ScaleInputLine.text())

        def run(self):
                Form = QDialog()        #Form je oblik ; Qwidget je emptybox a
Qdialogue je menu sa ok i cancel
                #self = AddForm_Dialog()            #ui je sta se sve nalazi u menu
                self.setupUi(Form)              #setappa ui (ocito)
                Form.exec()                     #show je preview pogledaj modal
dialogue u dokumentaciji (modalni blokiraju acsess ostatku aplikacije dok nije
zavrsena)   #pokrece novi menu
                if Form.result() == True:  #ako je pritisnut ok
                        while True:
                                try:
                                        scale = self.getInput()
                                except:
                                        print("numbers only")
```

```
                              Form.exec()
                              if Form.result() == False:
                                    break
                        else:
                              print(scale)
                              return scale
                              break

class Move_Dialog(object):
      def setupUi(self, Dialog):
            if Dialog.objectName():
                  Dialog.setObjectName(u"Dialog")
            Dialog.resize(236, 140)
            self.buttonBox = QDialogButtonBox(Dialog)
            self.buttonBox.setObjectName(u"buttonBox")
            self.buttonBox.setGeometry(QRect(10, 100, 221, 41))
            self.buttonBox.setOrientation(Qt.Horizontal)
            self.buttonBox.setStandardButtons(QDialogButtonBox.Cancel|
QDialogButtonBox.Ok)
            self.verticalLayoutWidget = QWidget(Dialog)
            self.verticalLayoutWidget.setObjectName(u"verticalLayoutWidget")
            self.verticalLayoutWidget.setGeometry(QRect(20, 10, 41, 80))
            self.verticalLayout = QVBoxLayout(self.verticalLayoutWidget)
            self.verticalLayout.setObjectName(u"verticalLayout")
            self.verticalLayout.setContentsMargins(0, 0, 0, 0)
            self.MoveLabelX = QLabel(self.verticalLayoutWidget)
            self.MoveLabelX.setObjectName(u"MoveLabelX")

            self.verticalLayout.addWidget(self.MoveLabelX)

            self.MoveLabelY = QLabel(self.verticalLayoutWidget)
            self.MoveLabelY.setObjectName(u"MoveLabelY")

            self.verticalLayout.addWidget(self.MoveLabelY)

            self.MoveLabelZ = QLabel(self.verticalLayoutWidget)
            self.MoveLabelZ.setObjectName(u"MoveLabelZ")

            self.verticalLayout.addWidget(self.MoveLabelZ)

            self.verticalLayoutWidget_2 = QWidget(Dialog)
            self.verticalLayoutWidget_2.setObjectName(u"verticalLayoutWidget_2")
            self.verticalLayoutWidget_2.setGeometry(QRect(70, 10, 160, 80))
            self.verticalLayout_2 = QVBoxLayout(self.verticalLayoutWidget_2)
            self.verticalLayout_2.setObjectName(u"verticalLayout_2")
            self.verticalLayout_2.setContentsMargins(0, 0, 0, 0)
            self.DeltaXInputLine = QLineEdit(self.verticalLayoutWidget_2)
            self.DeltaXInputLine.setObjectName(u"DeltaXInputLine")

            self.verticalLayout_2.addWidget(self.DeltaXInputLine)

            self.DeltaYInputLine = QLineEdit(self.verticalLayoutWidget_2)
            self.DeltaYInputLine.setObjectName(u"DeltaYInputLine")

            self.verticalLayout_2.addWidget(self.DeltaYInputLine)

            self.DeltaZInputLine = QLineEdit(self.verticalLayoutWidget_2)
            self.DeltaZInputLine.setObjectName(u"DeltaZInputLine")

            self.verticalLayout_2.addWidget(self.DeltaZInputLine)


            self.retranslateUi(Dialog)
            self.buttonBox.accepted.connect(Dialog.accept)
```

```
            self.buttonBox.rejected.connect(Dialog.reject)

            QMetaObject.connectSlotsByName(Dialog)
      # setupUi

      def retranslateUi(self, Dialog):
            Dialog.setWindowTitle(QCoreApplication.translate("Dialog", u"Move",
None))
            self.MoveLabelX.setText(QCoreApplication.translate("Dialog", u"Delta
X:", None))
            self.MoveLabelY.setText(QCoreApplication.translate("Dialog", u"Delta
Y:", None))
            self.MoveLabelZ.setText(QCoreApplication.translate("Dialog", u"Delta
Z:", None))
            self.DeltaXInputLine.setText(QCoreApplication.translate("Dialog",
u"1", None))
            self.DeltaYInputLine.setText(QCoreApplication.translate("Dialog",
u"1", None))
            self.DeltaZInputLine.setText(QCoreApplication.translate("Dialog",
u"1", None))
      # retranslateUi

      def getInput(self):
            return np.array([float(self.DeltaXInputLine.text()),
float(self.DeltaYInputLine.text()), float(self.DeltaZInputLine.text())])

      def run(self):
            Form = QDialog()        #Form je oblik ; Qwidget je emptybox a
Qdialogue je menu sa ok i cancel
            #self = AddForm_Dialog()           #ui je sta se sve nalazi u menu
            self.setupUi(Form)            #setappa ui (ocito)
            Form.exec()                   #show je preview pogledaj modal
dialogue u dokumentaciji (modalni blokiraju acsess ostatku aplikacije dok nije
zavrsena)   #pokrece novi menu
            if Form.result() == True:  #ako je pritisnut ok
                  while True:
                        try:
                              move_vector = self.getInput()
                        except:
                              print("numbers only")
                              Form.exec()
                              if Form.result() == False:
                                    break
                        else:
                              print(move_vector)
                              return move_vector
                              break




class SetPosition_Dialog(object):
      def setupUi(self, Dialog):
            if Dialog.objectName():
                  Dialog.setObjectName(u"Dialog")
            Dialog.resize(248, 148)
            self.buttonBox = QDialogButtonBox(Dialog)
            self.buttonBox.setObjectName(u"buttonBox")
            self.buttonBox.setGeometry(QRect(50, 110, 171, 41))
            self.buttonBox.setOrientation(Qt.Horizontal)
            self.buttonBox.setStandardButtons(QDialogButtonBox.Cancel|
QDialogButtonBox.Ok)
            self.verticalLayoutWidget = QWidget(Dialog)
            self.verticalLayoutWidget.setObjectName(u"verticalLayoutWidget")
```

```
            self.verticalLayoutWidget.setGeometry(QRect(10, 20, 41, 80))
            self.verticalLayout = QVBoxLayout(self.verticalLayoutWidget)
            self.verticalLayout.setObjectName(u"verticalLayout")
            self.verticalLayout.setContentsMargins(0, 0, 0, 0)
            self.SetPositionXLabel = QLabel(self.verticalLayoutWidget)
            self.SetPositionXLabel.setObjectName(u"SetPositionXLabel")

            self.verticalLayout.addWidget(self.SetPositionXLabel)

            self.SetPositionYLabel = QLabel(self.verticalLayoutWidget)
            self.SetPositionYLabel.setObjectName(u"SetPositionYLabel")

            self.verticalLayout.addWidget(self.SetPositionYLabel)

            self.SetPositionZLabel = QLabel(self.verticalLayoutWidget)
            self.SetPositionZLabel.setObjectName(u"SetPositionZLabel")

            self.verticalLayout.addWidget(self.SetPositionZLabel)

            self.verticalLayoutWidget_2 = QWidget(Dialog)
            self.verticalLayoutWidget_2.setObjectName(u"verticalLayoutWidget_2")
            self.verticalLayoutWidget_2.setGeometry(QRect(60, 20, 160, 80))
            self.verticalLayout_2 = QVBoxLayout(self.verticalLayoutWidget_2)
            self.verticalLayout_2.setObjectName(u"verticalLayout_2")
            self.verticalLayout_2.setContentsMargins(0, 0, 0, 0)
            self.SetPositionXInputLine = QLineEdit(self.verticalLayoutWidget_2)
            self.SetPositionXInputLine.setObjectName(u"SetPositionXInputLine")

            self.verticalLayout_2.addWidget(self.SetPositionXInputLine)

            self.SetPositionYInputLine = QLineEdit(self.verticalLayoutWidget_2)
            self.SetPositionYInputLine.setObjectName(u"SetPositionYInputLine")

            self.verticalLayout_2.addWidget(self.SetPositionYInputLine)

            self.SetPositionZInputLine = QLineEdit(self.verticalLayoutWidget_2)
            self.SetPositionZInputLine.setObjectName(u"SetPositionZInputLine")

            self.verticalLayout_2.addWidget(self.SetPositionZInputLine)


            self.retranslateUi(Dialog)
            self.buttonBox.accepted.connect(Dialog.accept)
            self.buttonBox.rejected.connect(Dialog.reject)

            QMetaObject.connectSlotsByName(Dialog)
      # setupUi

      def retranslateUi(self, Dialog):
            Dialog.setWindowTitle(QCoreApplication.translate("Dialog", u"Set
Position", None))
            self.SetPositionXLabel.setText(QCoreApplication.translate("Dialog",
u"Set X:", None))
            self.SetPositionYLabel.setText(QCoreApplication.translate("Dialog",
u"Set Y:", None))
            self.SetPositionZLabel.setText(QCoreApplication.translate("Dialog",
u"Set Z:", None))

      self.SetPositionXInputLine.setText(QCoreApplication.translate("Dialog",
u"0", None))

      self.SetPositionYInputLine.setText(QCoreApplication.translate("Dialog",
u"0", None))
```

```
      self.SetPositionZInputLine.setText(QCoreApplication.translate("Dialog",
u"0", None))
      # retranslateUi


      def getInput(self):
            return np.array([float(self.SetPositionXInputLine.text()),
float(self.SetPositionYInputLine.text()),
float(self.SetPositionZInputLine.text())])

      def run(self):
            Form = QDialog()       #Form je oblik ; Qwidget je emptybox a
Qdialogue je menu sa ok i cancel
            #self = AddForm_Dialog()           #ui je sta se sve nalazi u menu
            self.setupUi(Form)           #setappa ui (ocito)
            Form.exec()                  #show je preview pogledaj modal
dialogue u dokumentaciji (modalni blokiraju acsess ostatku aplikacije dok nije
zavrsena)   #pokrece novi menu
            if Form.result() == True:  #ako je pritisnut ok
                  while True:
                        try:
                              new_position = self.getInput()
                        except:
                              print("numbers only")
                              Form.exec()
                              if Form.result() == False:
                                    break
                        else:
                              print(new_position)
                              return new_position
                              break



class ImportFromCsvMenu_Dialog(object):
      def setupUi(self, Dialog):
            if Dialog.objectName():
                  Dialog.setObjectName(u"Dialog")
            Dialog.resize(240, 102)
            self.buttonBox = QDialogButtonBox(Dialog)
            self.buttonBox.setObjectName(u"buttonBox")
            self.buttonBox.setGeometry(QRect(20, 60, 221, 41))
            self.buttonBox.setOrientation(Qt.Horizontal)
            self.buttonBox.setStandardButtons(QDialogButtonBox.Cancel|
QDialogButtonBox.Ok)
            self.PathLabel = QLabel(Dialog)
            self.PathLabel.setObjectName(u"PathLabel")
            self.PathLabel.setGeometry(QRect(10, 10, 121, 16))
            self.PathLine = QLineEdit(Dialog)
            self.PathLine.setObjectName(u"PathLine")
            self.PathLine.setGeometry(QRect(10, 30, 200, 20))

            self.retranslateUi(Dialog)
            self.buttonBox.accepted.connect(Dialog.accept)
            self.buttonBox.rejected.connect(Dialog.reject)

            QMetaObject.connectSlotsByName(Dialog)
      # setupUi

      def retranslateUi(self, Dialog):
            Dialog.setWindowTitle(QCoreApplication.translate("Dialog", u"Import
from csv", None))
            self.PathLabel.setText(QCoreApplication.translate("Dialog", u"Path
to Folder:", None))
```

```
            self.PathLine.setText("")
      # retranslateUi

      def getInput(self):
            return self.PathLine.text()

      def run(self):
            Form = QDialog()       #Form je oblik ; Qwidget je emptybox a
Qdialogue je menu sa ok i cancel
            #self = AddForm_Dialog()           #ui je sta se sve nalazi u menu
            self.setupUi(Form)           #setappa ui (ocito)
            Form.exec()                  #show je preview pogledaj modal
dialogue u dokumentaciji (modalni blokiraju acsess ostatku aplikacije dok nije
zavrsena)   #pokrece novi menu
            if Form.result() == True:  #ako je pritisnut ok
                  while True:
                        path = self.getInput()
                        if os.path.isdir(path) == False:
                              print("Path not valid.")
                              Form.exec()
                              if Form.result() == False:
                                    break
                        else:
                              print(path)
                              return path
                              break




#iz techinfo videa:
if __name__ == "__main__":
      app = QApplication(sys.argv) #app moramo importat
      a = SetPosition_Dialog()
      #a = Move_Dialog()
      a.run()
      #pass
      #     Form = QDialog()       #Form je oblik ; Qwidget je emptybox a
Qdialogue je menu sa ok i cancel
#     ui = AddBlock_Dialog()       #ui je sta se sve nalazi u menu
#     ui.setupUi(Form)       #setappa ui (ocito)
#     Form.exec()                  #show je preview pogledaj modal dialogue u
dokumentaciji (modalni blokiraju acsess ostatku aplikacije dok nije zavrsena)
#     #app.run
#     #result = Form.result()
#     while Form.result() == True:
#           Form.exec()
#           print("array koji ide u dims")


#     Form = QDialog()       #Form je oblik ; Qwidget je emptybox a Qdialogue
je menu sa ok i cancel
#     ui = AddBlock_Dialog()       #ui je sta se sve nalazi u menu
#     ui.setupUi(Form)       #setappa ui (ocito)
#     Form.exec()                  #show je preview pogledaj modal dialogue u
dokumentaciji (modalni blokiraju acsess ostatku aplikacije dok nije zavrsena)
      #pokrece novi menu
#     if Form.result() == True:  #ako je pritisnut ok
#           while True:
#                 try:
#                       block_dims = ui.getInput()
```

```
#                 except:
#                       print("numbers only")
#                       Form.exec()
#                 else:
#                       break
#           print(block_dims)
```