Vizualizacija OOFEM modela brodske konstrukcije primjenom programa otvorenog koda linaetal-fsb/d3v

Buconić, Marko

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Marko Buconić

Zagreb, 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:

Izv. prof. dr. sc. Pero Prebeg

Student:

Marko Buconić

Zagreb, 2020.

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

Zahvaljujem se Izv. prof. dr. sc. Peri Prebegu na potpori tijekom izrade završnog rada te na stručnoj pomoći i savjetima pruženim tijekom izrade ovog rada.

Marko Buconić

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: **Marko Buconić** Mat. br.: 0035204296

Naslov rada na hrvatskom jeziku: **Vizualizacija OOFEM modela brodske konstrukcije primjenom programa otvorenog koda linaetal-fsb/d3v**

Naslov rada na engleskom jeziku: **Use of the open source program linaetal-fsb/d3v for the visualization of OOFEM ship structural model**

Opis zadatka:

Pri projektiranju brodske konstrukcije neophodno je provesti analizu odziva metodom adekvatne točnosti. U skorije vrijeme u tu svrhu, umjesto analitičkih metoda, sve se više koristi metoda konačnih elemenata (MKE). Pri tome se najčešće koriste komercijalni programski alati, dok je primjena besplatnih MKE alata otvorenog koda, poput programa OOFEM, izuzetno rijetka. OOFEM posjeduje i bogato programsko sučelje (API) pomoću kojeg je iz programskih jezika poput Pythona moguće izraditi module koji proširuju osnovne funkcionalnosti. 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, uz primjenu OOFEM Python API-a, omogućiti vizualizaciju OOFEM modela i rezultata proračuna odziva brodske konstrukcije.

Zadatak obuhvaća sljedeće:

- upoznavanje s programom otvorenog koda OOFEM
- upoznavanje s programom otvorenog koda linaetal-fsb/d3v
- izradu Python modula koji omogućuje 3D vizualizaciju OOFEM modela konstrukcije broda u programu otvorenog koda linaetal-fsb/d3v
- izradu Python modula koji omogućuje 3D vizualizaciju rezultata proračuna dobivenih primjenom programa OOFEM u programu otvorenog koda linaetal-fsb/d3v
- izradu Python modula koji omogućuje interaktivnu promjenu jednostavnih parametara poput koncentriranog čvornog opterećenja konstrukcije, pokretanje proračuna te analizu osjetljivosti konstrukcije na promjenu tog parametra.

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 razvijena je vizualizacija osnovnih karakteristika OOFEM modela modeliranih sa konačnim elementima ljuske i grede što omogućuje vizualizaciju uobičajenih modela brodske konstrukcije. Vizualizacija je implementirana proširenjem funkcionalnosti programa otvorenog koda linaetal-fsb/d3v u programskom jeziku Python. Program otvorenog koda linaetal-fsb/d3v temeljen je na Qt for Python tehnologiji, koja omogućuje trodimenzijsku vizualizaciju geometrijskih entiteta primjenom OpenGL tehnologije te jednostavno definiranje grafičkog sučelja za pojedine specifične namjene. Izrađeni su moduli u programu Python koji omogućavaju 3D vizualizaciju karakteristika OOFEM modela konstrukcije broda i rezultata proračuna. Razvijene mogućnosti vizualizacije prikazane su na primjeru konstrukcije dvodna broda za rasuti teret.

Ključne riječi: OOFEM, Python, linaetal-fsb/d3v, OpenGl, Qt

SUMMARY

In this paper, a visualization of the basic characteristics of OOFEM models modeled with finite shell and beam elements is developed, which enables the visualization of conventional models of ship construction. The visualization was implemented by extending the functionality of the open source linaetal-fsb / d3v program in Python programming language. The open source linaetal-fsb / d3v program is based on Qt for Python technology, which enables three-dimensional visualization of geometric entities using OpenGL technology and easy defining of a graphical interface for specific purposes. Modules have been created in Python that allow 3D visualization of the features of the OOFEM model of ship construction and calculation results. The visualization possibilities developed are exemplified by the construction of a double bottom bulk carrier.

Key words: OOFEM, Python, linaetal-fsb/d3v, OpenGl, Qt

1. UVOD

Pri projektiranju brodske konstrukcije neophodno je provesti analizu odziva metodom adekvatne točnosti. U skorije vrijeme u tu svrhu, umjesto analitičkih metoda, sve se više koristi metoda konačnih elemenata (MKE) [1]. Pri tome se najčešće koriste komercijalni programski alati, dok je primjena besplatnih MKE alata otvorenog koda, poput programa OOFEM [2], izuzetno rijetka. OOFEM posjeduje i bogato programsko sučelje (API) pomoću kojeg je iz programskih jezika poput Pythona [7] moguće izraditi module koji proširuju osnovne funkcionalnosti. 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 proširenjem funkcionalnosti programa otvorenog koda linaetal-fsb/d3v u programskom jeziku Python, uz primjenu OOFEM Python API-a, omogućena vizualizacija OOFEM modela i rezultata proračuna odziva brodske konstrukcije.

1.1. Metoda konačnih elemenata

Klasične metode rješavanja problema kontinuiranih sustava temelje se na rješavanju diferencijalnih jednadžbi čije je točno analitičko rješenje moguće dobiti samo za jednostavnije proračunske modele. U općem slučaju vrlo je teško dobiti rješenje koje zadovoljava diferencijalnu jednadžbu u cijelom području razmatranog modela. Stoga se rabe približne numeričke metode koje se temelje na diskretizaciji kontinuiranog sustava gdje se diferencijalne jednadžbe zamjenjuju sustavom algebarskih jednadžbi.

Metoda konačnih elemenata numerička je metoda koja je nezaobilazna u inženjerskim proračunima. Danas postoji veliki broj računalnih programa temeljenih na toj metodi, koji omogućuju analizu konstrukcija bez razmatranja složene teorije koja opisuje fizikalno ponašanje konstrukcije. Njihovo korištenje nerijetko se svodi na zadavanje ulaznih podataka prema propisanim uputama, a dobivena rješenja, koja svojim grafičkim prikazom često fasciniraju korisnike, prihvaćaju se bez dovoljno kritičnosti. Računalni program shvaća se kao crna kutija (eng. *Black-Box*) u kojoj je skrivena složena teorija koja se smatra nepotrebnom za rješavanje inženjerskih problema. Takav pristup može dovesti do pogrešne procjene stanja naprezanja i deformacije u konstrukciji, a to može ugroziti njezinu čvrstoću i stabilnost.

Metoda konačnih elemenata približna je numerička metoda. Svaki njezin korisnik treba imati na umu da su dobivena rješenja približna, a realnim vrijednostima mogu se približiti samo uz pravilan izbor proračunskog modela i uz pravilno odabrane konačne elemente koji su u mogućnosti opisati realni proces deformiranja. Kako bi to bilo moguće, potrebno je razumjeti fizikalno ponašanje konstrukcije koja se analizira te poznavati teorijske osnove konačnih elemenata, a na taj način i ograničenja njihove primjene. Osim toga, korisnik mora biti u stanju kritički analizirati dobivene rezultate. [1]

2. Object Oriented Finite Element Solver (OOFEM)

U današnje vrijeme, programi otvorenog koda koriste se sve više, osobito na akademskoj razini. Otvoreni kod odnosi se na računalni program čiji je izvorni kod dostupan javnosti za korištenje i modifikaciju. Izvorno je osmišljen s ciljem da okupi zajednicu programera kako bi razvijali kod i međusobno ga dijelili. Na taj način, pojedinac može više naučiti, ali raste i kvaliteta programa. Danas postoji veliki broj licenci koje definiraju prava i obveze kako autora, tako i korisnika programa otvorenog koda.

Object Oriented Finite Element Solver (OOFEM) je besplatni program za analizu metodom konačnih elemenata (MKE) s objektno orijentiranom arhitekturom, koji se koristi za rješavanje mehaničkih, transportnih i problema dinamike fluida, koji radi na različitim operativnim sustavima. [2]

Program je razvijen zbog ograničenja postojećih komercijalnih programa. Naime, oni često pružaju set korisnički definiranih podrutina koje se mogu koristiti kako bi se proširile mogućnosti paketa u određenom području. No, implementacija u potpunosti novog konstitutivnog modela je u principu jako teška ili nemoguća. S druge strane, programi otvorenog koda često pate od nejasne strukture podataka, loše strukture programa ili nedovoljne dokumentacije. OOFEM je razvijen kao program koji ima sposobnosti kontinuiranog razvoja, neograničene mogućnosti nadogradnje, podrške za timski rad, robusnosti i upotrebe na različitim sustavima. Glavni ciljevi su sljedeći:

- otvoreni kernel - dizajn kernela i implementacija mora biti predviđena za široki stupanj nadogradnje. Ovo mora vrijediti i za dizajn pojedinog modula.
- struktura programa mora biti u skladu s modularnim dizajnom. Ovo je jako važno za podršku timskog rada.
- kod se mora moći lako održavati i mijenjati.
- upotrebljivost na različitim platformama je važan i prirodan zahtjev.
- računalne performanse slične programima napisanim u Fortran-u ili C-u. **Pogreška! Izvor reference nije pronađen.**

2.1. Osnovne mogućnosti programa

OOFEM nudi razne mogućnosti upotrebe zbog tri vrste problema koji se njime mogu rješavati, mnoštva analiza, elemenata i materijala koji se njime mogu opisati.

2.1.1. Analize

Trenutno, analize koje su podržane su sljedeće:

- linearno-statička analiza
- analiza linearne stabilnosti
- analiza vlastitih vrijednosti dinamičkih sustava
- direktna eksplicitna nelinearna analiza dinamičkih problema
- linearna eksplicitna analiza dinamičkih problema
- direktna implicitna analiza linearnih dinamičkih problema
- inkrementalna linearno statička analiza
- nelinearno-statička analiza
- adaptivna linearno-statička analiza
- adaptivna nelinearno-statička analiza
- stacionarni transportni problem
- nelinearni tranzijentni transportni problem
- tranzijentni nekompresibilni tok-CBS algoritam
- tranzijentni nekompresibilni tok-SUPG/PSPG algoritam
- *staggered* analiza. [3]

Prikazane analize se odnose na sve tri mogućnosti primjene programa, dakle probleme mehanike fluida, transportne i mehaničke probleme. U ovom radu bitni su mehanički problemi, stoga su zanimljive analize koje se na njih primjenjuju, a to su linearno-statička i nelinearno-statička analiza.

2.1.2. Vrste elemenata

Kao i u ostalim slučajevima, i elementi u programu OOFEM se dijele u 3 skupine:

- Elementi za analizu konstrukcije
- Elementi za transportne probleme
- Elementi za probleme dinamike fluida.

U ovom radu razmatrani su elementi za analizu konstrukcije, koji se opet dijele u više skupina od kojih ćemo izdvojiti neke koje će biti korištene u okviru ovog rada:

- elementi za ravninsko stanje naprezanja
  - *TrPlaneStress2d* element
  - *PlaneStress2d* element
  - *TrPlaneStrRot* element
  - *TrPlaneStressRotAllman* element
- elementi ploče
  - *DKT* element
  - *QDKTPlate* element
- elementi ljuske
  - *mitc4shell* element. [5]

2.1.3. Vrste opterećenja

- *NodalLoad* - koncentrirano čvorno opterećenje
- *DeadWeight* - težinsko opterećenje elementa po volumenu
- *StructTemperatureLoad* - temperaturno opterećenje po volumenu
- *StructEigenstrainLoad* - deformacije konstrukcijskog elementa bez pojave naprezanja
- *ConstantEdgeLoad* - konstantno opterećenje po stranici elementa
- *ConstantSurfaceLoad* - konstantno opterećenje po površini elementa
- *LinearEdgeLoad* - linearno opterećenje po stranici elementa. [6]

3. Python

Python je interpreterski, interaktivni, objektno orijentirani programski jezik. Ne donosi neke nove revolucionarne značajke u programiranju, već na optimalan način ujedinjuje sve najbolje ideje i načela rada drugih programskih jezika. On je jednostavan i snažan istodobno. Više nego drugi jezici on omogućuje programeru više razmišljanja o problemu nego o jeziku. Smatra se hibridom: nalazi se između tradicionalnih skriptnih jezika (kao što su Tcl, Schema i Perl) i sistemskih jezika (kao što su C, C++ i Java). To znači da nudi jednostavnost i lako korištenje skriptnih jezika, uz napredne programske alate koji se tipično nalaze u sistemskim razvojnim jezicima. Python je besplatan (za akademske ustanove i neprofitnu upotrebu), *open-source* softver, s izuzetno dobrom potporom, literaturom i dokumentacijom. Osim standardnih tipova podataka (brojevi, nizovi znakova i sl.) Python ima ugrađene tipove podataka visoke razine kao što su liste, n-terci i rječnici. Python nudi sve značajke očekivane u modernom programskom jeziku: objektno orijentirano programiranje s višestrukim nasljeđivanjem, dohvaćanje izuzetaka, redefiniranje standardnih operatora, pretpostavljene argumente, prostore imena i pakete. Može se lako proširivati novi značajkama ili API-ima. (eng. *application programming interface*). Pythonova knjižnica (eng. *library*), koja uključuje standardnu instalaciju, uključuje preko 200 modula, što pokriva sve od funkcija operacijskog sustava do struktura podataka potrebnih za gradnju web-servera. Glavna Python-ova web stranica nudi sažeti index mnogih Python projekata i različitih drugih knjižnica. [7]

3.1. Definiranje varijabli

U programskom jeziku Python nije potrebno deklarirati varijable prije prve upotrebe, tj. odrediti koji će se tip podataka u njima spremati, već ih je dovoljno inicijalizirati, tj. pridružiti im neku vrijednost. Time se određuje tip podataka spremljen u pojedinoj varijabli.

Kako bi se inicijalizirala varijabla, koristi se znak ”=”. S lijeve strane znaka nalazi se ime varijable, a s desne strane vrijednost varijable. Python ima pet tipova podataka: brojevi (eng. *numbers*), niz znakova (eng. *string*), lista (eng. *list*), n-terac (eng. *tuple*) i rječnik (eng. *dictionary*).

3.2. Pisanje komentara

Pisanje komentara unutar programa je ponekad vrlo korisno. Programi koji su složeni često se sastoje od mnogo dijelova te je korisno napisati komentar na što se koji dio odnosi. Komentari započinju sa znakom ”#” te se protežu do kraja retka. Oni se mogu nalaziti na početku retka ili na kraju, nakon naredbi, ali ne i unutar naredbi.

3.3. Ulazni i izlazni podaci

Svaki algoritam na početku uzima ulazne podatke, a na kraju vraća izlazne podatke koji predstavljaju rješenje problema. Osim što se varijablama može pridružiti konstanta ili rezultat nekih računskih operacija, može joj se pridružiti i ulazni podatak koristeći se naredbom *input()*. Varijabli se može pridružiti i neki slučajno generirani realni broj iz intervala [0.0, 1.0] pomoću naredbe *random.random()* ili neki slučajno generirani cijeli broj iz intervala [a, b] pomoću naredbe *random.randint(a,b)*. Za korištenje navedenih naredbi potrebno je uključiti modul *random*. Naredba je *import random*.

Ukoliko se želi ispisati određeni tekst potrebno ga je staviti u navodnike, dok ispisivanje vrijednosti varijabli se ne stavljaju u navodnike. Unutar jedne naredbe *print* se može ispisati više izlaznih podataka, pri čemu ih je potrebno odvojiti zarezima.

3.4. Brojevi

Programski jezik Python može koristiti kao kalkulator i pomoću njega se može zbrajati, oduzimati, množiti, dijeliti, potencirati, korjenovati,... Dijeljenje ovisi o tipu argumenata. Ako je barem jedan od argumenata *x* ili *y* realni broj, rezultat dijeljenja *x*/*y* će biti realni broj. Ako su *x* i *y* cijeli brojevi, dijeljenje *x*/*y* će biti cjelobrojno. Ukoliko se želi da rezultat bude realni broj, potrebno je to naznačiti dodavanjem točke jednom od argumenata: *x*/*y*. ili *x*./*y*

3.5. Niz znakova

Niz znakova (eng. *string*) u Pythonu se identificiraju kao skup znakova unutar jednostrukih ’...’ ili dvostrukih navodnika "...". Ukoliko niz znakova sadrži jednostruke navodnike, za njegovo identificiranje potrebno je korištenje dvostrukih navodnika, i obrnuto. Svaki niz znakova može se indeksirati na način da prvi znak u nizu ima indeks 0, drugi indeks 1, itd. Nizove se može indeksirati i negativnim brojevima na način da zadnji znak u nizu ima indeks −1, predzadnji −2, itd. Indeksiranje nizova znakova omogućava ispis podskupa nekog niza pomoću operatora [ ], ako se želi ispisati samo jedan znak niza, ili pomoću operatora [:], ako

se želi ispisati više znakova niza. Nizovi se mogu povezivati pomoću operatora + , ponavljati pomoću operatora * , a duljina niza može se izračunati koristeći naredbu *len()*.

3.6. Liste

Liste (eng. *lists*) u Pythonu se identificiraju koristeći uglate zagrade [ ]. Lista može sadržavati brojeve i nizove znakova istovremeno. Elemente liste odvajamo zarezima. Slično kao i kod nizova znakova, prvi element liste može se indeksirati indeksom 0, drugi indeksom 1, i tako dalje; ili zadnji indeksom −1, predzadnji indeksom −2, i tako dalje. Također, za ispisivanje određenih elemenata liste koriste se operatori [ ], ukoliko se želi ispisati samo jedan element liste, i [:], ukoliko se želi ispisati više elemenata liste. Liste se mogu spajati pomoću operatora + , ponavljati pomoću operatora * , a duljina liste može se izračunati koristeći naredbu *len()*. U neku listu se mogu dodati i pojedinačni elementi pomoću naredbe *imeliste.append()*.

3.7. Rječnici

Rječnici (eng. *dictionary*) su, uz liste, najfleksibilniji tip podataka u Pythonu. Oni su izmjenjivi objekti (kao i liste) pa ih je moguće izmjenjivati na licu mjesta, bez stvaranje kopije postojeće varijable. Visoko su optimizirani za nasumičan pristup (eng. *random access*) što znači da je brzina dohvata bilo kojeg elementa u rječniku jednaka. Ovo se može činiti nebitnim, no recimo, ako postoji lista od 1000 elemenata i dohvaća se petstoti element, računalo mora "proći" kroz sve članove do petstotog kako bi dohvatilo traženi element. U slučaju rječnika, za dohvaćanje elementa potrebna je jedna operacija, nema iteriranja po svim članovima do traženog. Brzina rječnika osobito dolazi do izražaja ukoliko se radi s većom količinom podataka. Nadalje, u slučaju liste postoji dohvaćanje elementa pomoću indeksa. Ako se želi dohvatiti element koji sadrži određenu vrijednost, mora se ili znati na kojem se on točno mjestu nalazi u listi, ili pretražiti listu kako bi našli element. Kod rječnika je svaki element zapravo uređeni par ključ-vrijednost (eng. *key-value*) te je rječnik optimiziran za dohvaćanje vrijednosti pomoću ključa. Rječnik, kao i lista, omogućuje gniježđenje proizvoljne dubine i to bilo kojih vrsta elemenata. Na primjer, može prvi element biti broj, drugi element rječnik, a treći lista rječnika. Podržava i funkciju *len()*, pomoću koje se saznaje broj elemenata rječnika (naravno, broj elemenata prve razine ugniježđenosti).

3.8. Petlje *for* i *while*

Petlje (eng. *loops*) u programiranju se koriste kada se neku naredbu želi ponoviti više puta. Ukoliko se neka naredba želi ponoviti točno određeni broj puta, koristi se petlja *for*. Sintaksa naredbe *for* glasi:

for i in range(a,b,k):

naredba ili skup naredbi

pri čemu *i* označava brojač petlje, *a* je početna vrijednost brojača, *b* je završna vrijednost brojača, a *k* označava korak brojača. Naredba ili skup naredbi će se izvršavati sve dok brojač ne dostigne vrijednost *b*.

Ukoliko se želi da se neka naredba ponovi *n* puta, pri čemu je *n* neki prirodan broj, dovoljno je napisati:

for i in range(n):

naredba ili skup naredbi

Ukoliko se neka naredba ili skup naredbi želi ponavljati sve dok je ispunjen neki uvjet, koristi se petlja *while*. Sintaksa naredbe *while* glasi:

while uvjet:

naredba ili skup naredbi

Razlika između *for* i *while* petlje je što petlja *while* ne sadrži brojač, već joj izvršavanje neke naredbe više puta omogućuje dani uvjet. Uvjet u petlji *while* može biti određen aritmetičkim, logičkim i/ili relacijskim (usporednim) operatorima.[8]

4. Primjena programa Linaetal-fsb/d3v za vizualizaciju konstrukcije

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. Implementacije je bazirana na bibliotekama QtForPython (PySide2) i OpenMesh.

4.1. Qt za Python

Qt je razvojni okvir koji omogućuje dodatne funkcionalnosti za C++, korištenjem tzv. "*Meta – Object*" prevoditelja (eng. *compiler*) nazvanog "MOC". Qt razvojni okvir napisan je u programskom jeziku C++ i moguće ga je koristiti na više platformi/operacijskih sustava, što je testirano pomoću alata Oracle VM za instalaciju virtualnih uređaja. MOC također omogućuje najznačajniju funkcionalnost specifičnu za razvojni okvir Qt, signale i prijemnike, odnosno mehanizam međukomunikacije objekata na način da pokretanje nekog događaja (signal) kao što su klik, nailazak miša preko objekta, povlačenje i otpuštanje objekata, dvostruki klik i sl. okidaju određenu funkciju (prijemnik), npr. odbrojavanje, kreiranje novoga objekta, promjenu fonta itd. Qt ima vlastiti izvršni (eng. *make*) alat, zvani *qmake*, za stvaranje izvršnih datoteka vezanih za platformu na kojoj se pokreće, čitajući datoteke s nastavkom „pro". Ako je u pro datoteci nešto promijenjeno, potrebno je pokrenuti *qmake* za ažuriranje izvršnih datoteka, ako nije, prevoditelj ignorira pro datoteku. Qt razvojni okvir obuhvaća nekoliko biblioteka: Core framework, Gui framework, SQL framework, Xml framework, Networking framework, OpenGL framework, Multimedia framework, WebKit framework itd. Postoje komercijalna verzija i slobodni softver Qt-a, ukoliko se koristi komercijalna verzija, nije potrebno dijeliti napisani kôd, a za verziju slobodnog softvera je potrebno. [9]

Programi potrebni za pokretanje vizualizacije i koji su korišteni u ovom radu su sljedeći: Visual Studio 2015, Python 3.7, Qt 5.12, CMake 3.16.

Nakon što se preuzmu i instaliraju svi potrebni programi, instalira se PySide2 tako da se utipka *pip install PySide2* u naredbenom retku.

Problem prilikom instalacije *PySide2* se javio problem zbog nepodržavanja verzije Python 3.8 te je riješen promjenom Python-a na verziju 3.7.

4.1.1. QtCreator

Okruženje za kreiranje Qt programa je jednostavno za korištenje. Sadrži uređivač kôda, dizajner i program za pronalaženje grešaka (eng. *debugger*). Pri pisanju kôda ima mogućnosti formatiranja, nadopunjavanje koda zvano "*intellisense*", upozoravanje na pogreške i u nekim slučajevima automatsko ispravljanje kôda (. i ->). QtCreator podržava više programa za pronalaženje grešaka i omogućuje opciju točke zaustavljanja (eng. *breakpoint*). Također ima mnogo mogućnosti u grafičkome sučelju. Za grafičke aplikacije omogućeno je brzo prebacivanje iz dizajnera u uređivač pomoću trake s lijeve strane ekrana, a za bilo koji izraz sintakse napisana je dokumentacija i pristupa joj se pomoću tipke F1. Qt ima i mogućnost praćenja curenja memorije (eng. *memory leak*). Instalacijom QtCreatora dobiva se i mnoštvo aplikacija – primjera napisanih u Qt-u, najviše grafičkih aplikacija. Podržava razne formate datoteka i moguće je otvoriti slike, xml, JSON dokumente itd. Nudi i mogućnost interakcije s Git sustavom.

4.1.2. PySide2

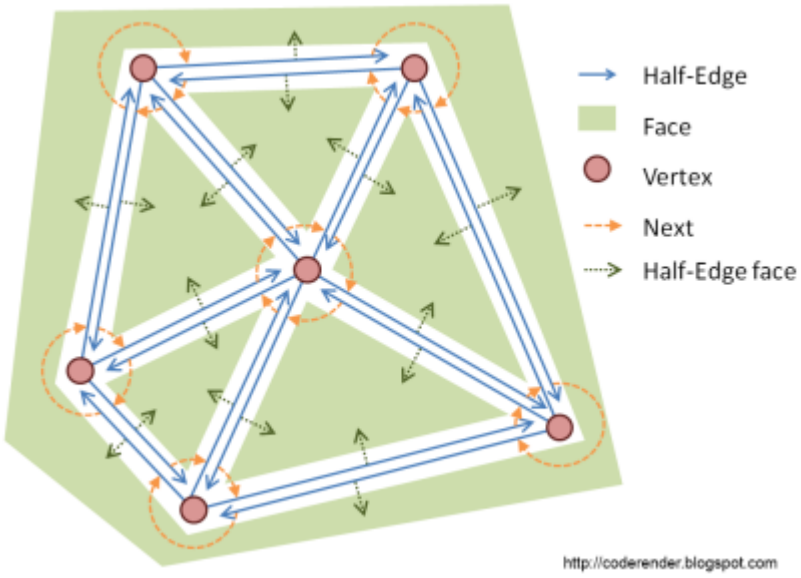
PySide2 je jezična poveznica Pythona s Qt višeplatformskim toolkitom za izradu sučelja. Jedna je od mogućih alternativa standardnoj Tkinter biblioteci. Tkinter, čije ime je skraćenica za *Tk interface*, poveznica je Pythona s toolkitom Tk i smatra se *de facto* standardom za izradu sučelja aplikacija u Pythonu. Kao i Qt, PySide spada u besplatni softver. Projekt razvoja PySide biblioteke nastao je kao posljedica neuspjeha u pregovorima Nokije s britanskom tvrtkom Riverbank Computing oko licenciranja PyQt biblioteke, također poveznice Qt-a s Pythonom, pri čemu je zahtjev Nokije bio da PyQt bude licenciran i pod LGPL licencom pored postojeće GPLv3 i komercijalne licence. LGPL licenca omogućuje primjenu PySide biblioteke u razvoju besplatnih programa otvorenog koda, ali i komercijalnog softvera. [11]

4.2. OpenMesh

OpenMesh je generička i učinkovita struktura podataka za predstavljanje i manipuliranje poligonalnim mrežama koji je knjižnica C ++. Dostupne su i Python veze. Razvijen je u *Computer Graphics Group*, RWTH Aachen. Financiralo ga je njemačko Ministarstvo za istraživanje i obrazovanje.

4.2.1. Polu-bridna struktura zapisa mreže [13]

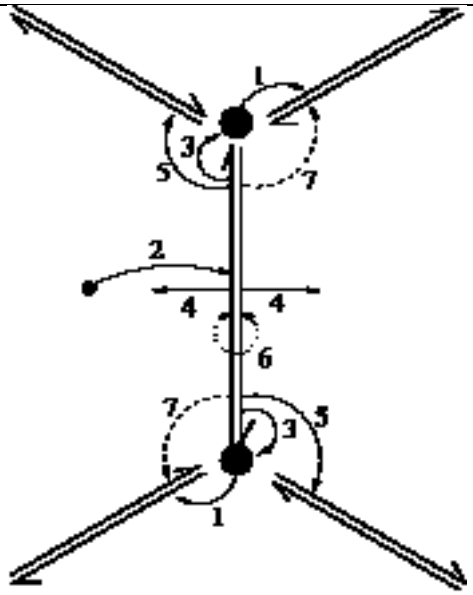
Ovaj odjeljak opisuje temeljnu strukturu podataka koja se koristi za pohranjivanje mrežnih entiteta (stavki) odnosno vrhova (eng. *vertex*), bridova (eng. *edge*), poligonskih ploha (eng. *face* ili *facet*) i njihovih podataka o povezivanju.



Slika 1 Osnovni entiteti polu-bridne strukture

Postoje mnoge popularne strukture podataka koje se koriste za predstavljanje poligonalnih mreža.

Struktura podataka koja se koristi u ovom projektu je tzv. podatkovna struktura polu-bridova . Dok strukture temeljene na poligonskim plohama pohranjuju svoju povezanost u plohe koje upućuju na njihove vrhove i susjede, strukture koje se temelje na rubovima postavljaju informacije o povezanosti na rubove. Svaki rub referencira svoja dva vrha, poligonskim plohama kojima pripada i dva sljedeća ruba na tim poligonskim plohama. Ako jedan sada dijeli rubove (tj. rub koji povezuje vrh *A* i vrh *B* postaju dva usmjerena polu-brida od *A* do *B* i obrnuto) jedan dobiva strukturu podataka polu-brida. Sljedeća slika prikazuje način na koji je povezanost pohranjena u ovu strukturu:



Slika 2 Pohranjivanje povezanosti

- Svaki vrh se referencira na jedan odlazni polu-brid, tj. polu-brid koji počinje na tom vrhu(1).
- Svaka ploha se referencira jedan polu-brid ograničavajući ga (2).
- Svaki polu-brid daje pokazivač (eng. *handle*) za:
    - vrh na koji upućuje (3),
    - plohi kojoj pripada (4)
    - sljedećem polu-bridu unutar plohe (suprotno od kazaljke na satu) (5),
    - suprotnom polu-bridu (6),
    - (izborno: prethodnom polu-bridu u face-u (7)).

Imajući ove veze između stavki, moguće je kružiti oko ploha kako bi se pristupilo svim njegovim vrhovima, polu-bridovima ili susjednim plohama. Kada počinjemo s polu-bridom vrha i iteriramo na suprotno od prethodnog, lako se može kružiti oko tog vrha i dobiti sve svoje susjede, dolazne/odlazne polu-bridove ili susjedne plohe.

Iako strukture temeljene na polu-bridovima troše više memorije nego njihovi dijelovi na plohi, imaju sljedeće važne prednosti:

- moguće je miješati plohe proizvoljnog broja vrhova u mreži
- direktan pristup vrhovima, plohama i polu-bridovima.
- kružno pristupanje entitetima oko vrhova važna je operacija za mnoge vrste algoritama na poligonalnim mrežama, što kod struktura zasnovanih na plohi to dovodi do mnogih *if-then* ogranaka, dok je kod polu-bridne struktura ta funkcionalnost prirodno prisutna.

4.2.2. Značajke i ciljevi OpenMesh-a [12]

Glavne značajke temeljne strukture podataka su:

- moguće su poligonske, a ne samo trokutaste mreže
- nezavisan pristup vrhovima, polu-bridovima i poligonskim plohama (eng. f*acets*)
- učinkovit pristup jednostrukoj kružnoj odnosno zajedničkoj točki više poligonskih ploha
- sposobnost da se obrađuju vrhovi koji nisu višestruki (poput dvije poligonske plohe koje se sastaju u samo jednom vrhu).

4.3. Vizualizacija konstrukcije broda u programu Linaetal-fsb/d3v

Za potrebu vizualizacije konstrukcije u programu Linaetal-fsb/d3v unutar mape *commands* izrađene su dvije python skripte: oofem.py i oofemcommands.py

U oofem.py dijelu koda se najprije iz učitane datoteke učitavaju podaci o čvorovima, zatim se učitavaju podaci o elementima od kojih se sastoji model (npr. *dktplate*, *mitc4shell, planestress2d*). Koordinate čvorova i podaci o elementima se spremaju u posebne rječnike koji se naknadno koriste. S obzirom da program linaetal-fsb/d3v podatke za vizualizaciju zapisuje u obliku trokutne openmesh mreže, neophodno je generirati trokutne plohe (eng. *facets*) koji će na odgovarajući način reprezentirati MKE model konstrukcije Pomoću koordinata čvorova se generiraju vrhovi koji se spremaju u listu, te se uz pomoć te liste generiraju trokutne plohe. Pokazivači na te trokutne plohe spremaju se u rječnik. Ako se konačni element, koji se vizualizira, sastoji od 3 vrha, tada se generira jedna trokutna ploha, a ukoliko se sastoji od 4 vrha (četverokutni konačni element), tada se generiraju dvije trokutne plohe. Prilikom generiranja elemenata s 4 vrha treba paziti na smjer dodavanja vrhova i njihovog povezivanja. U ovom radu je korišten sljedeći smjer dodavanja vrhova i povezivanja istih:

- generiranje nova 3 vrha (v1,v2, v3) za svaki trokutni konačni element
- ukoliko se radi o elementu s 4 vrha onda se generira 4 vrha (v1, v2, v3, v4)
- povezivanjem vrhova (v1, v2, v3) generira se trokutna ploha
- kod četverokutnih konačnih elemenata povezivanjem sljedeća 3 vrha (v3, v4, v1) generira se nova trokutna ploha

Na taj način se dobiju elementi s 3 ili 4 vrha. Trokutne plohe generirane za jedan četverokutni element imaju smjer povezivanja vrhova suprotan od smjera gibanja kazaljke na satu.

U oofemcommand.py nalazi se dio koda vezan za stvaranje izbornika (eng. *menu*). Oofemcommand.py nudi opciju preusmjeravanja otvaranja generalne datoteke na otvaranje oofem datoteke. Kada korisnik pokrene program otvori se prozor u kojemu se može učitati model koji se analizira pritiskom na tipku *File*, zatim na *Import geometry*. Nakon što se učita korišteni model za njega je moguće pritiskom na *View* prikazati 3 vrste rezultata:

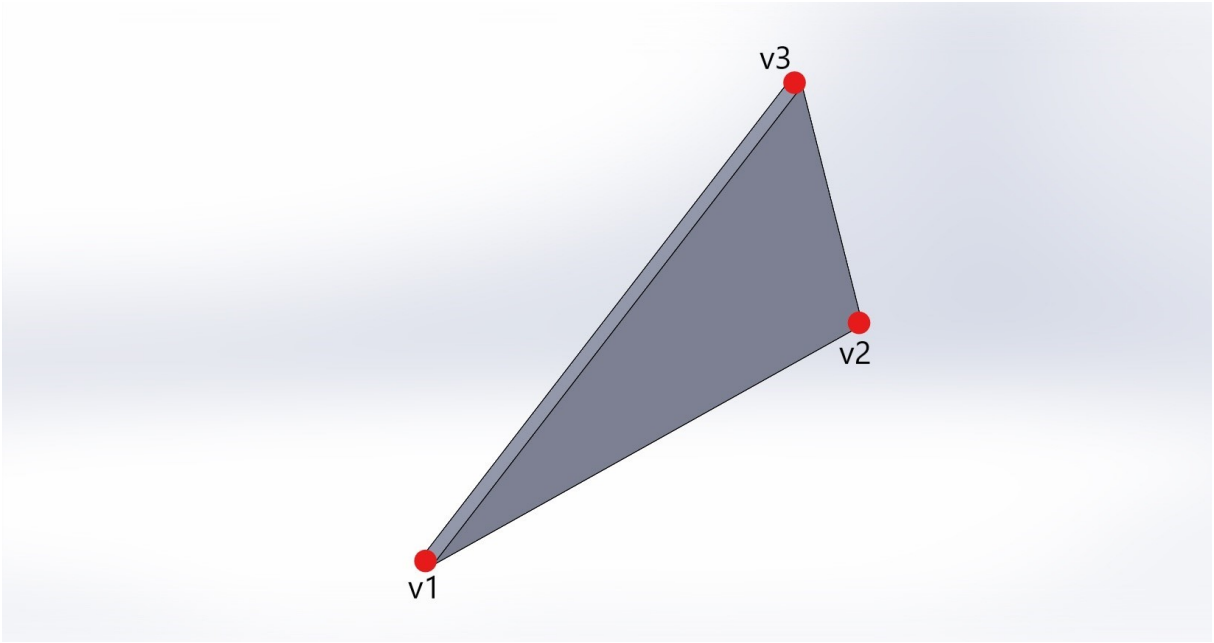
- presjek konstrukcije (CSID)
- materijal (MatID)
- debljina materijala (Thickness)

Također, moguće je i pritiskom na *Analyse* prikazati rezultate kojima sila djeluje na model.

Obje skripte nalaze se u prilogu 2 ovog rada.

4.3.1. Vizualizacija trokutnih 2D elementa

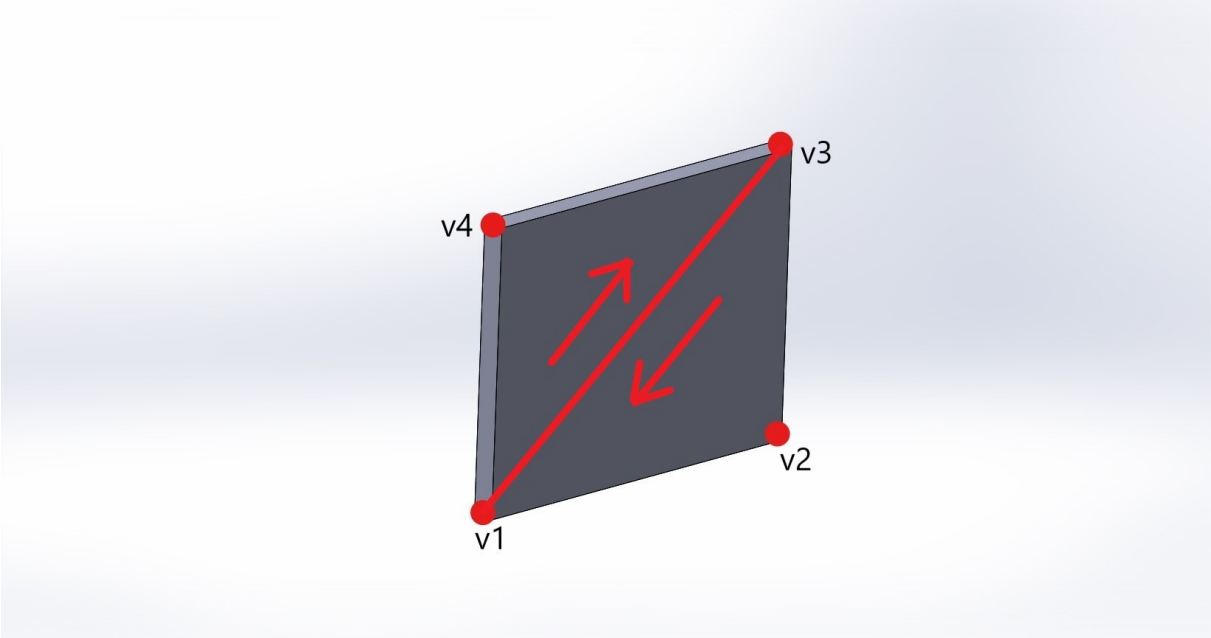
Trokutni 2D element je element koji definiraju 3 vrha (eng. *vertex*) odnosno točke. Vizualizira se na način da se pomoću koordinata čvorova (eng. *node*) koja su zadana u nekoj vrsti datoteke generiraju vrhovi pomoću funkcije *mesh.add_vertex()*. Zatim se funkcijom *mesh.add_face()* generira trokutni element kojeg definiraju ta tri vrha, a spajaju se redom u smjeru suprotnom od kazaljke na satu (v1, v2, v3).



Slika 3 Trokutni 2D element

4.3.2. Vizualizacija četverokutnih 2D elementa

Četverokutni 2D element je element koji se sastoji od 4 vrha. Generira se spajanjem dva trokutna 2D elementa s time da se mora paziti da orijentacija ta dva trokuta bude suprotnog smjera. Ukoliko je orijentacija dva trokuta ista tada se neće generirati trokutni element. Navedeno ograničenje je vezano uz način zapisa *openmesh* polu-brid strukture zapisa mreže.

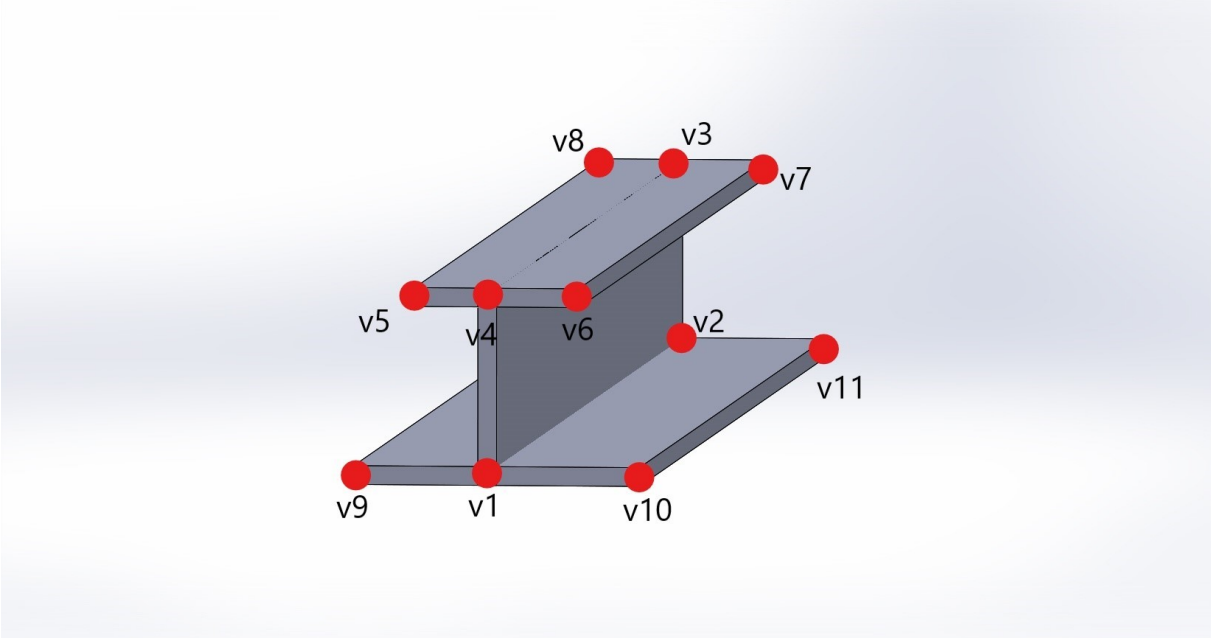


Slika 4 Četverokutni 2D element

4.3.3. Vizualizacija grednih elementa s T poprečnim presjekom i sunosivom širinom

Viziualizacija grednog elementa s T poprečnim presjekom i sunosivom širinom izvedena je pomoću 6 trokutnih elemenata odnosno 12 vrhova. Generira se zadavanjem dvaju čvorova , usmjerenjem struka i karakteristikama grednog elementa (visina struka, širina prirubnice, širina pokrova dna). Koordinate prvih dvaju vrhova su koordinate čvorova grednog elementa.. Uz pomoć usmjerenja i visine struka se dobiju još dva čvora i to na način da se zbroji vektor struka s početnim točkama. Preostalih 8 vrhova odredi se pomoću vektora prirubnice/pokrova dna, koje je okomito na struk i to uz pomoću vektorskog produkta vektora struka i vektora koji je dobiven kao razlika dviju početnih točaka. Poznavajući vektor prirubnice i pokrova dna dobiju se preostale točke grednog modela dodavanjem ili oduzimanjem pola širine prirubnice, odnosno pokrova dna . Nakon što se izračunaju sve potrebne koordinate vrhova, vrhovi se generiraju s funkcijom *mesh.add_vertex*, a pomoću tih vrhova generiraju se svi potrebni trokutni elementi funkcijom *mesh.add_face* spajanjem vrhova pazeći na orijentaciju zbog navedenog openmesh ograničenja. Struk, prirubnica i pokrov dna grednog elementa se

sastoje svaki od 2 trokutna elementa. Prvi trokutni element struka se generira pomoću 3 vrha (v1, v2, v3), dok se drugi sastoji od vrhova v3, v4 i v1. Trokutni elementi prirubnice se sastoje od vrhova v5, v6, v7, odnosno v7, v8, v9. Istu logiku slijede i trokuti pokrova dna koji se sastoje od vrhova v9, v10, v11, odnosno vrhova v11, v12 i v19.



Slika 5 Gredni element

4.4. Vizualizacija karakteristika modela konstrukcije bojama

Za potrebe vizualizacije značajki MKE modela primjenjuju se dva načina određivanja boja:

1. mapa sa konačnim brojem različitih boja

2. kontinuirana skala boja.

4.4.1. Mapa sa konačnim brojem različitih boja

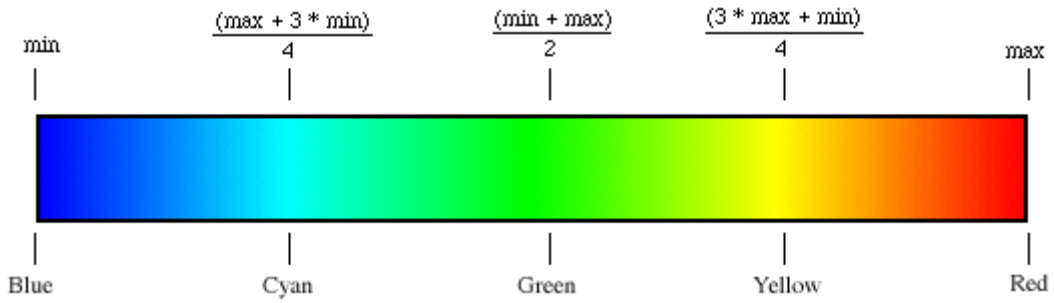
Mapa sa konačnim brojem različitih boja se koristi za prikazivanje diskretnih, konačnih setova npr. *property ID*, *material ID*. Svaka boja ima svoj r,g,b kanal (npr. crvena boja [255,0,0]) u kojemu prvi član (255) predstavlja udio crvene boje, drugi član udio zelene boje (0), a treći član (0) udio plave boje u konačnoj boji koju predstavlja taj kanal.

Na stranici [14] se nalaze r,g,b vrijednosti za različite preddefinirane boje od kojih je odabrano 20 koje su dovoljno različite da se njihove razlike jasno vide. Tih 20 boja je spremljeno u listu. Nakon toga se definira funkcija *GetDiscreteColor* koja za integerski ulaz (ID) vraća jednu boju [r,g,b] iz tako pripremljene liste. Ukoliko postoje elementi koji sadrže isti ID tada se oni bojaju s istom bojom, a ukoliko su različiti tada im se dodijeljuje prva sljedeća slobodna boja iz pripremljene liste.

4.4.2. Kontinuirana skala boja

Kontinuirana skala boja se koristi za prikaz rezultata npr. pomaka ili naprezanja.

Postoje dva načina prikaza, jedan način je konstantna boja po elementu, a drugi način je da je boja određena po čvorovima. Da bih se dobila skala boja potrebno je definirati metodu *GetContinuousColor* koja uzima jedan float argument koji je zapravo normirana vrijednost neke veličine između 0.0 i 1.0, a vraća jednu boju [r, g, b]. Način određivanja te boje ovisi o tome kakva paleta boja se želi prikazati. U poglavlju *Colour Ramping for Data Visualisation* u [15] objašnjen je način generiranja palete boja koja je primijenjena u ovom radu, a prikazana je na sljedećoj slici:



Slika 6 Kontinuirana skala boja

5. Vizualizacija OOFEM modela konstrukcije dvodna

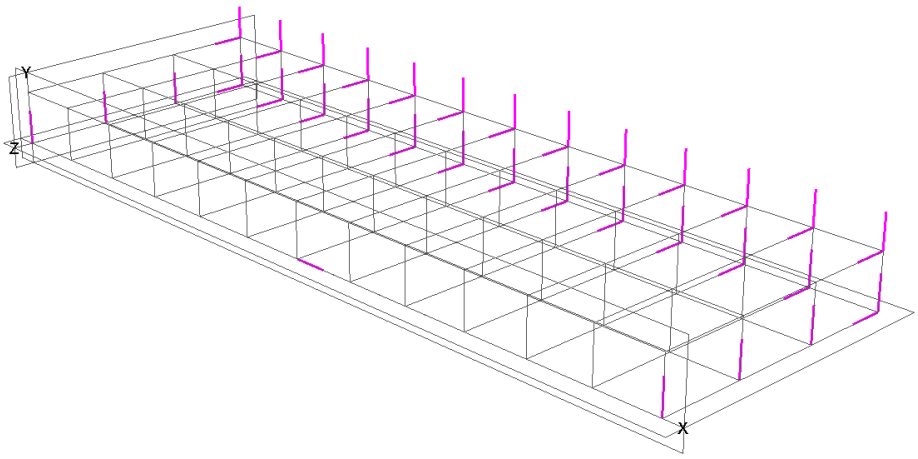
5.1. Model konstrukcije dvodna

Membranski model konstrukcije dvodna izrađen je u programskom paketu Maestro 11.0.0. Pri izradi modela ključno je diskretizirati poznatu strukturu na pravilan način. Struktura se diskretizira na način da se na ključnim mjestima strukture postavljaju čvorovi koji će biti rubne točke konačnih elemenata. Zbog toga je od velike važnosti ispravno postaviti početne čvorove, koji se postavljaju na mjestima spojeva jakih nosača s pripadnim opločenjem te na mjestima gdje se mijenja neko svojstvo strukture (debljina, materijal, vrste i razmak ukrepa).

Nakon što su određena mjesta na strukturi gdje će se nalaziti čvorovi, u Maestru se čvorovi unose s obzirom na koordinatni sustav u vidu 3 koordinate (x, y i z). Prije unosa koordinata čvorova u Maestro, potrebno je obratiti pozornost na koordinatni sustav kojeg programski paket koristi. Maestro ne koristi uvriježeni desni Kartezijev koordinatni sustav. Globalni koordinatni sustav u Maestru definiran je na sljedeći način:

- ishodište se nalazi u sjecištu uzdužne ravnine simetrije broda i osnovice
- X os se proteže u smjeru duljine broda, pozitivna je prema naprijed
- Y os se proteže u smjeru visine broda, pozitivna je prema gore
- Z os se proteže u smjeru širine broda, pozitivna je prema desno od uzdužne ravnine simetrije

Nakon definiranja položaja čvorova modela, definiraju se konačni elementi koji su omeđeni istima. Za određivanje konačnih elemenata su, osim čvorova, potrebna svojstva realne strukture koja će konačni elementi imati. Ovisno o materijalu, debljini i vrstama ukrepa koje se nalaze na realnoj konstrukciji, konačni elementi će međusobno imati drugačija svojstva te se stoga i drugačije ponašati što će preciznije opisati stvarno ponašanje. Nakon definiranih svojstava konačnih elemenata, potrebno je odrediti rubne uvjete modela. Rubni uvjeti se određuju poznavajući cjelokupnu konstrukciju koja okružuje model te način njihovog međudjelovanja. Za model roštilja dvodna određen je sljedeći rubni uvjeti:



Slika 7 Rubni uvjeti membranskog modela

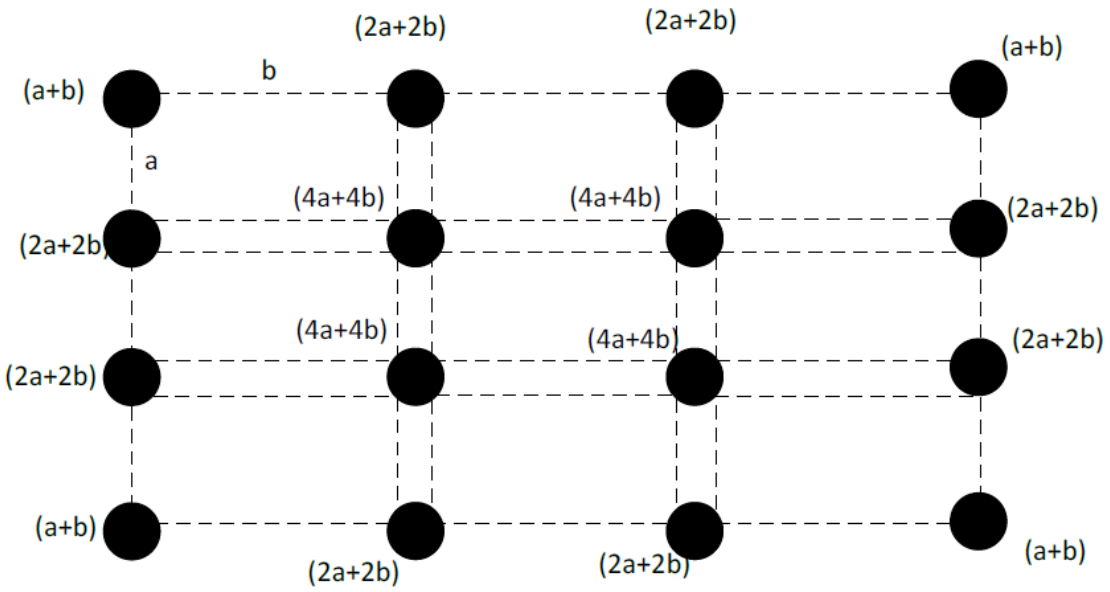
Ružičasta linija označava onemogućen pomak u smjeru osi koji odgovara smjeru linije. Modelirani dio strukture je roštilj dvodna broda za rasute terete u alternativnom stanju krcanja. Budući da se roštilj dvodna na boku nastavlja na uzvojni tank koji je vrlo kruta struktura, na tom rubu postavljeni su rubni uvjeti uklještenja . Na rubovima roštilja koji se vežu na susjedna skladišta postavljeni su zglobni rubni uvjeti jer su susjedna skladišta napunjena teretom, dok je skladište čiji se model izrađivao prazno (alternativno krcanje), zbog čega će doći do savijanja poprečne pregrade. Za membranski model bilo je dovoljno onemogućiti sve translacije na čvorovima pokrova dvodna i dna uzduž boka. Zbog simetrije stvarne konstrukcije, u Maestru je izrađena samo lijeva polovica konstrukcije dok je desna implementirana preko rubnih uvjeta na desnom kraju modela (ravnina XY). Rubni uvjeti simetrije su sljedeći:

- pomak u smjeru osi X je slobodan
- pomak u smjeru osi Y je slobodan
- pomak u smjeru osi Z je spriječen.

Membranski model je modeliran s grubom mrežom konačnih elemenata. Čvorovi su postavljeni na mjestima gdje se spajaju jaki uzdužni nosači s rebrenicama. Elementi opločenja dna i pokrova dvodna su tako definirani da im je duljina jednaka razmaku rebrenica, a širina razmak uzdužnih nosača. Vertikalni konačni elementi (elementi rebrenica i jakih uzdužnih

nosača) definirani su na način da im je visina jednaka visini dvodna, a širina odnosno duljina ograničena razmakom uzdužnih nosača odnosno rebrenica).

Membranski model ima istu diskretizaciju kao i makroelement model, ali se razlikuje po opterećenju i debljinama opločenja dna i pokrova dvodna. Debljine realne konstrukcije zamijenjene su ekvivalentnim debljinama u koje su uračunate i površine poprečnih presjeka uzdužnjaka dna i pokrova dvodna - na neki način „razmazani“ uzdužnjaci. Opterećenje konstantnog tlaka zamijenjeno je opterećenjem čvornih sila koje djeluju na mjestima spoja rebrenica i jakih uzdužnih nosača, a izračunate su prema sljedećoj shemi.



Slika 8 Shematski prikaz vrste čvorova s obzirom na ukupni iznos sile na čvoru

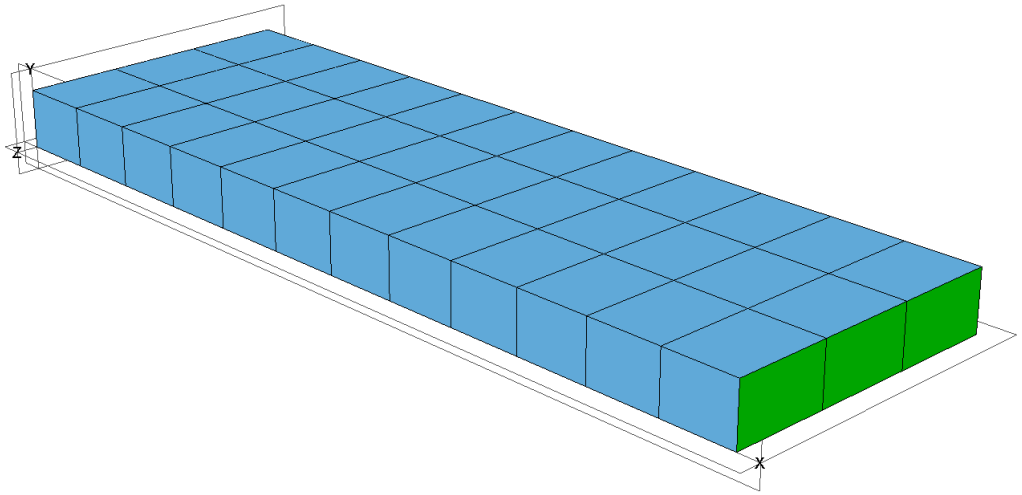
Prema tome čvorne sile se računaj na sljedeći način:

$$F_K = 0{,}15\frac{N}{mm^2} \cdot \quad \cdot$$

$$F_R = 0{,}15\frac{N}{mm^2} \cdot \quad \cdot$$

$$F_S = 0{,}15\frac{N}{mm^2} \cdot \quad \cdot$$

Pri tome je *a* razmak jakih uzdužnih nosača (*a*=3000 *mm*), a *b* razmak rebrenica (*b*=2300 *mm*). Sile se unose u Maestro kao ukupne sile na grupu čvorova (17 rubnih čvorova, 2 kutna čvora i 33 središnja čvora).

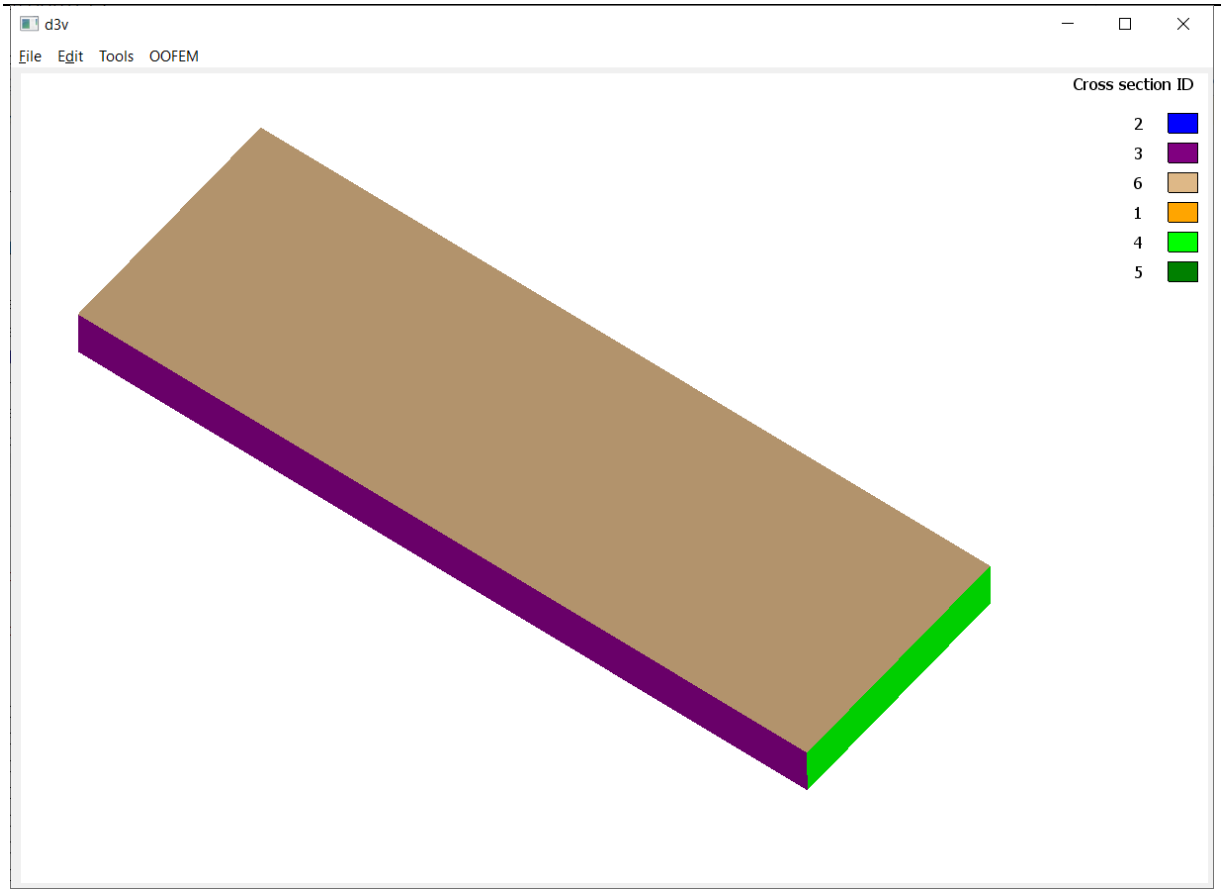


Slika 9 Membranski model

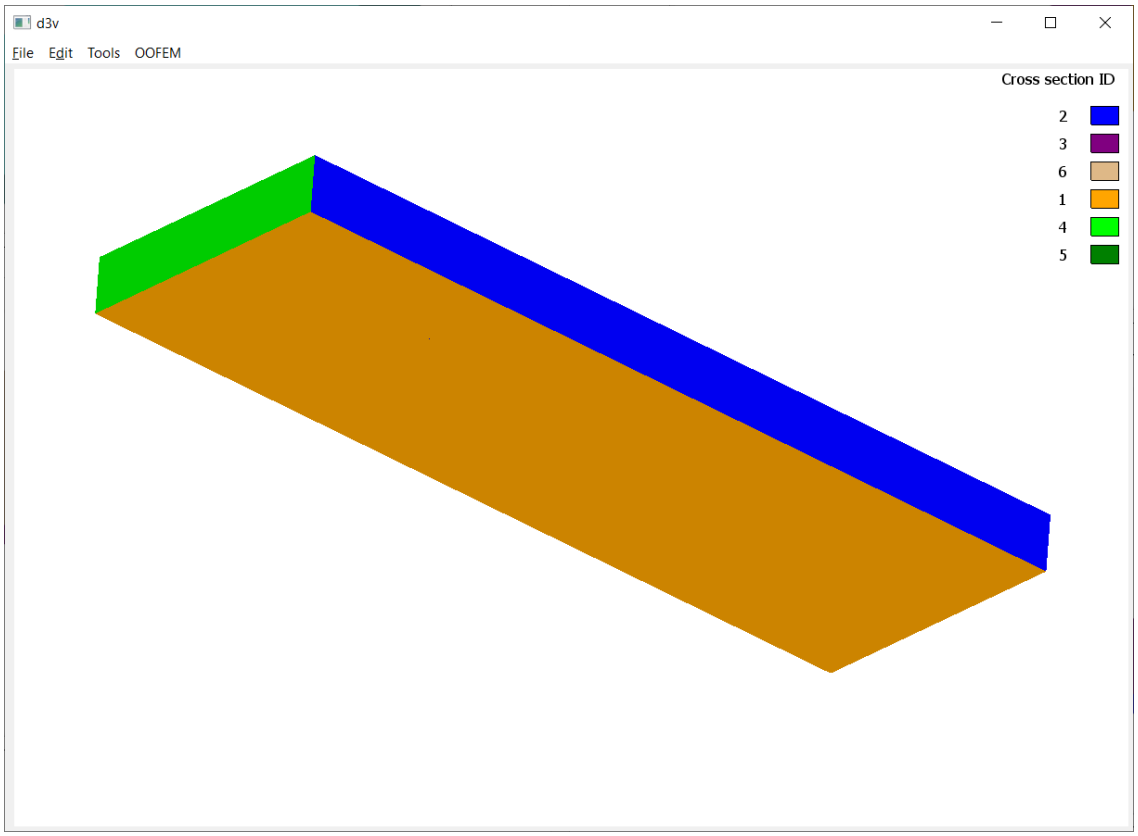
5.2. Vizualizacija konstrukcije u programu Linaetal-fsb/d3v

5.2.1. Vizualizacija karakteristika modela

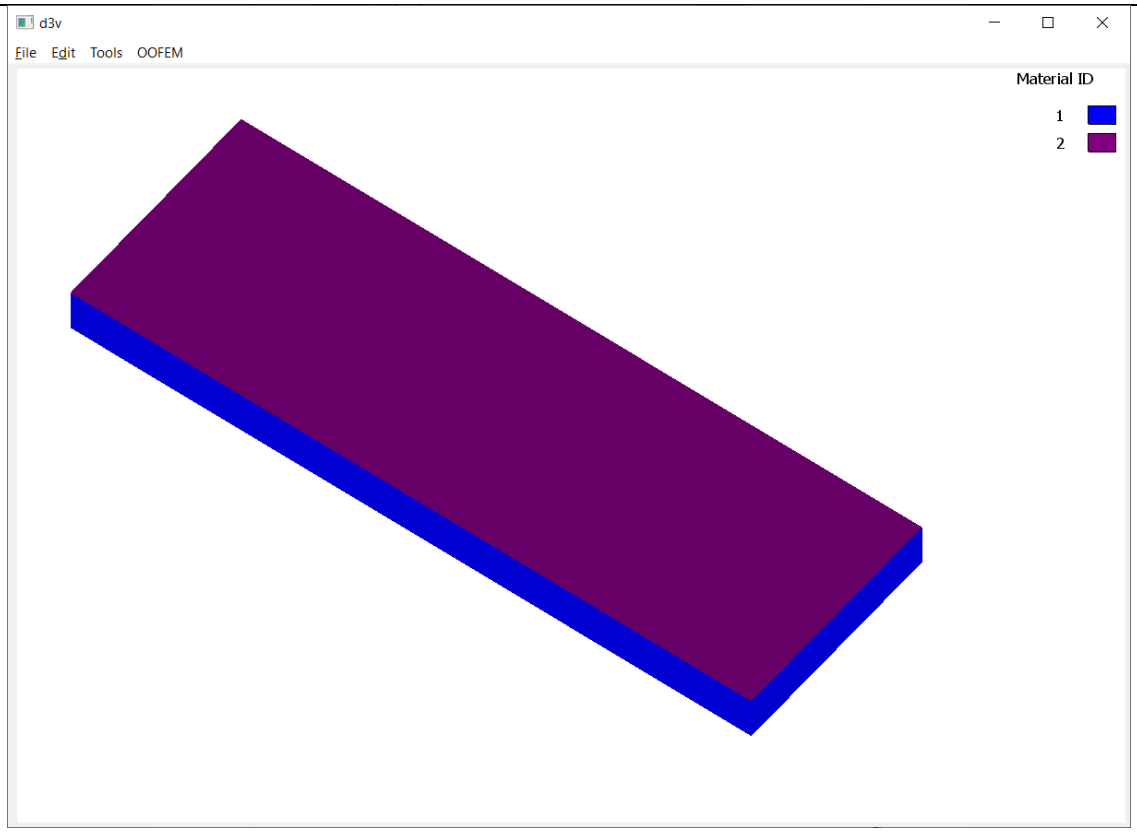
Na sljedećim slikama su prikazane karakteristike materijala kao što su materijal, poprečni presjek modela i debljine samih materijala. Za prikaz rezultata korištene su mape sa preddefiniranim bojama koje su odabrane kako bi se vidjela razlika između pojedinih karakteristika.



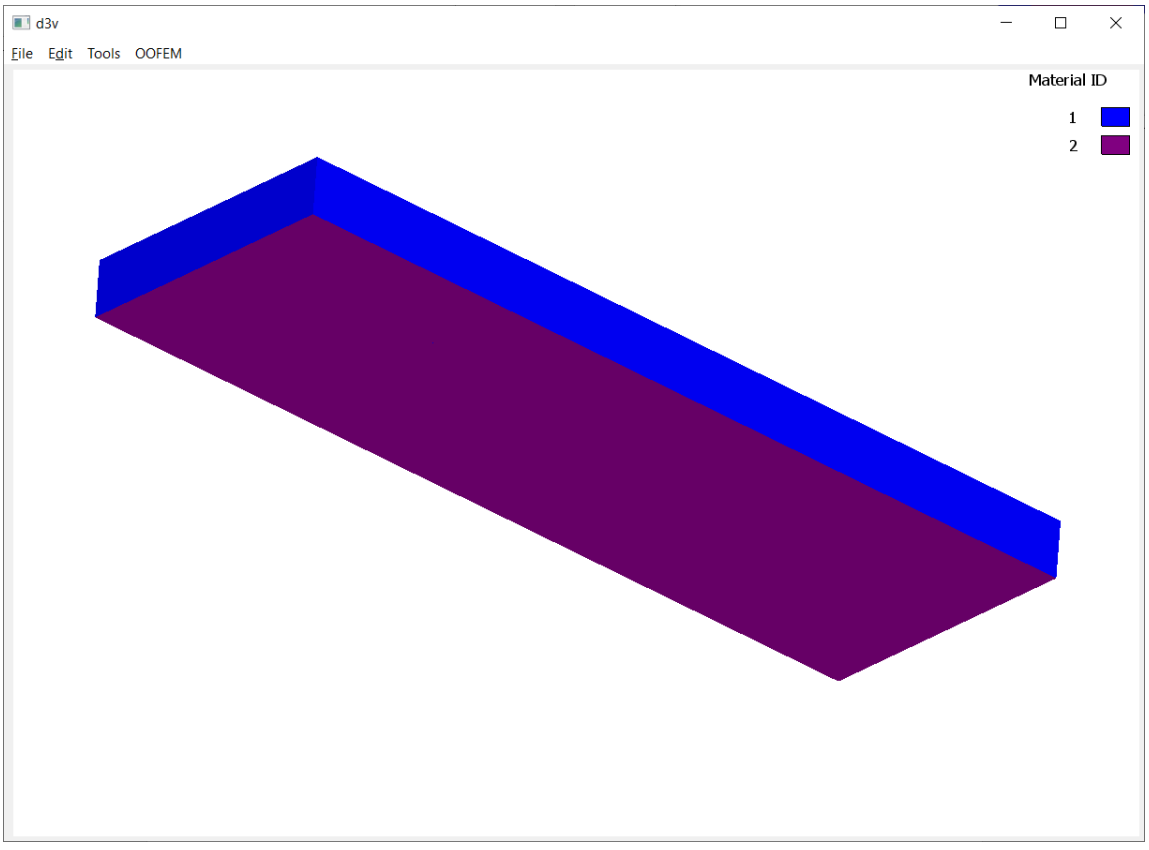
Slika 10 Poprečni presjek pokrova modela



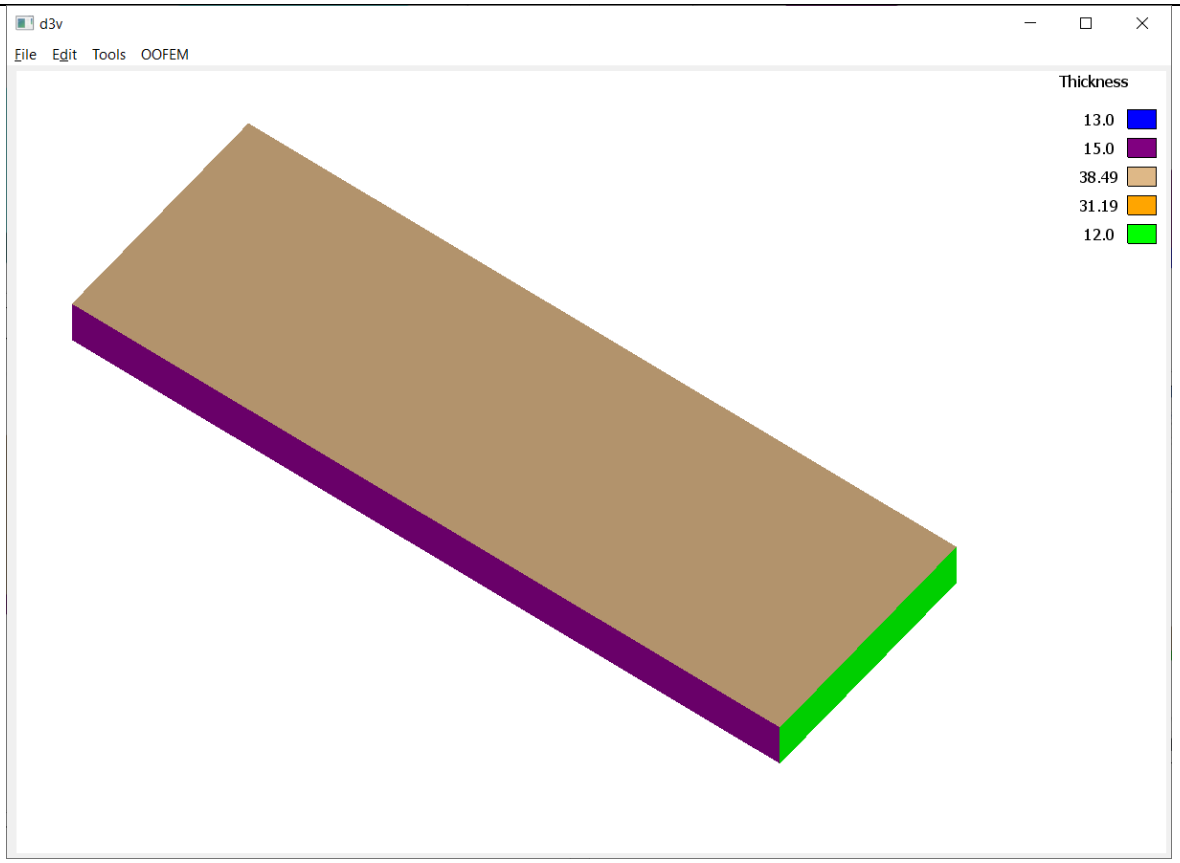
Slika 11 Poprečni presjek dna modela



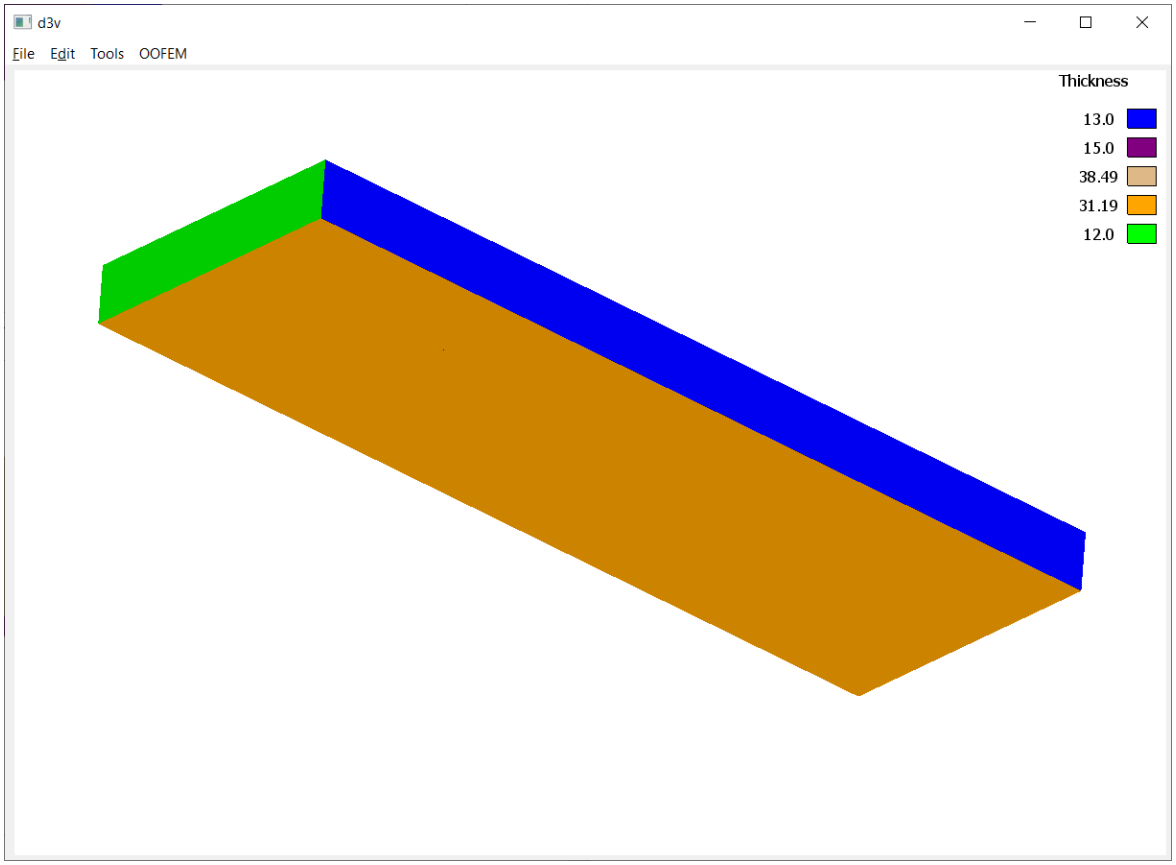
Slika 12 Materijal pokrova modela



Slika 13 Materijal dna modela

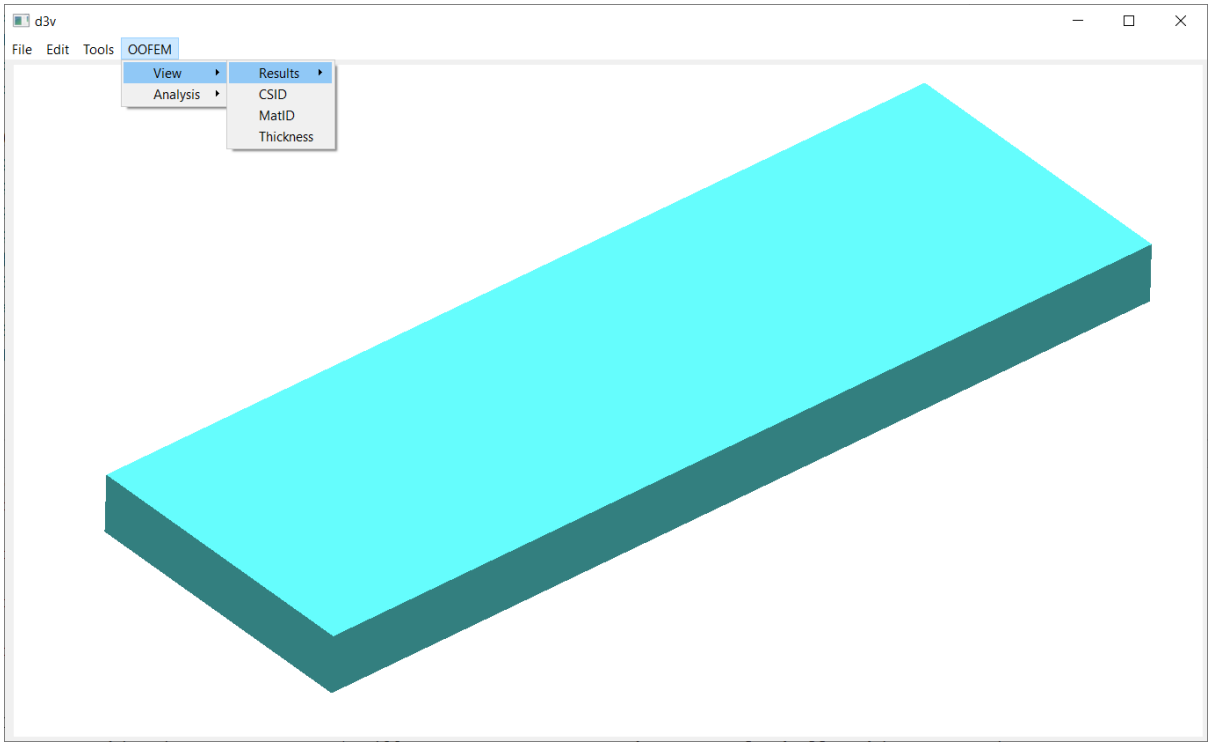


Slika 14 Debljina pokrova modela

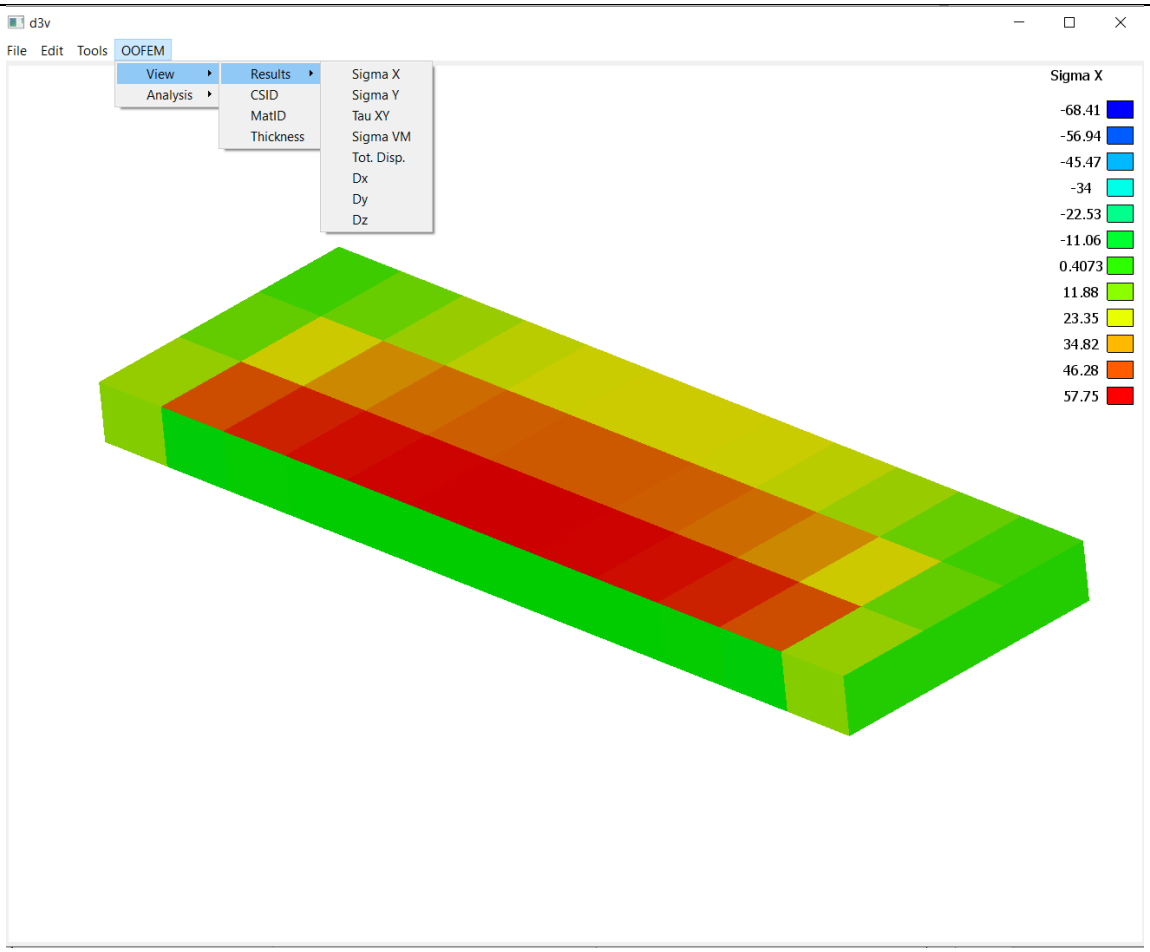


Slika 15 Debljina dna modela

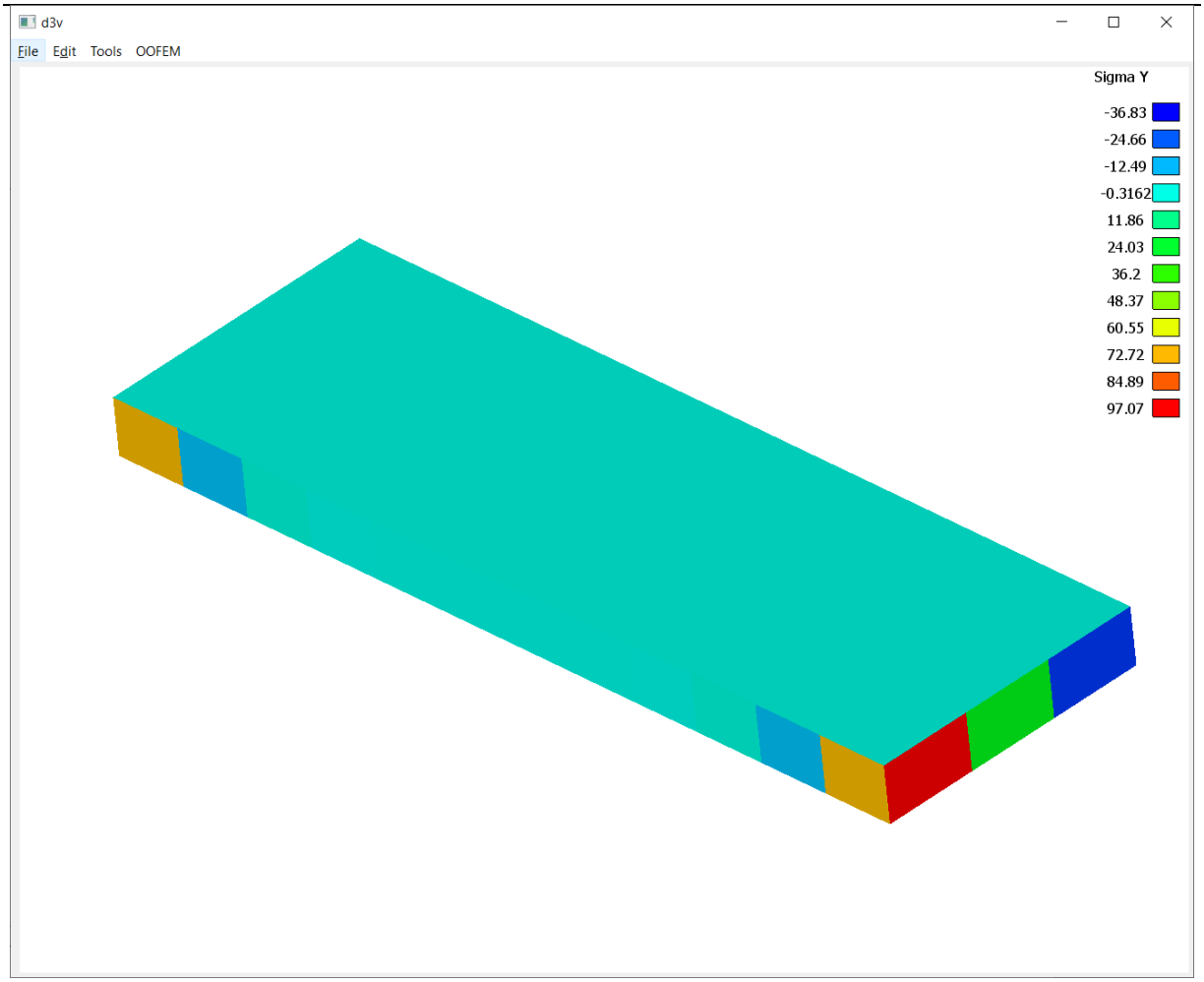
5.2.2. Vizualizacija rezultata analize



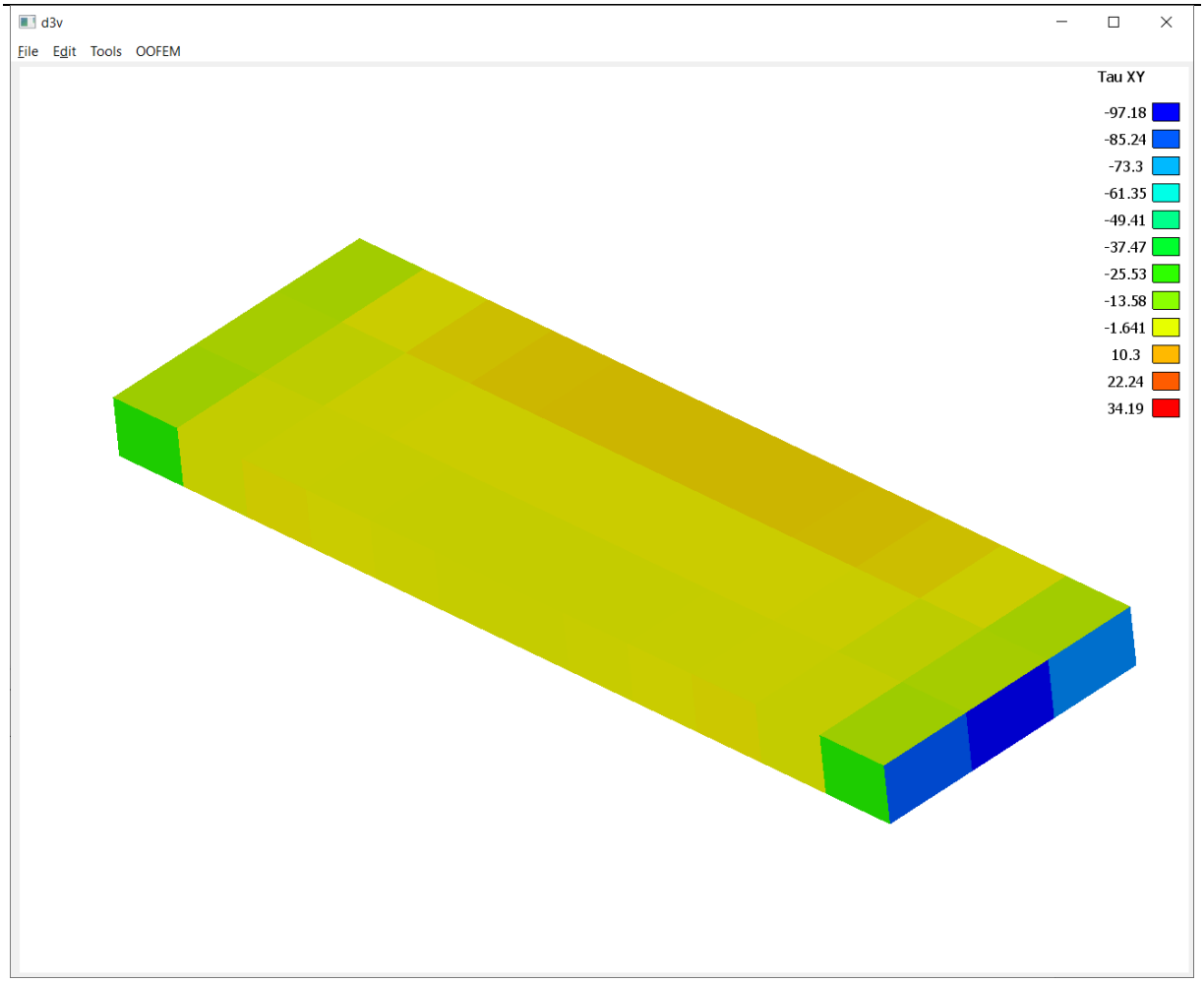
Slika 16 Vizualizacija rezultata



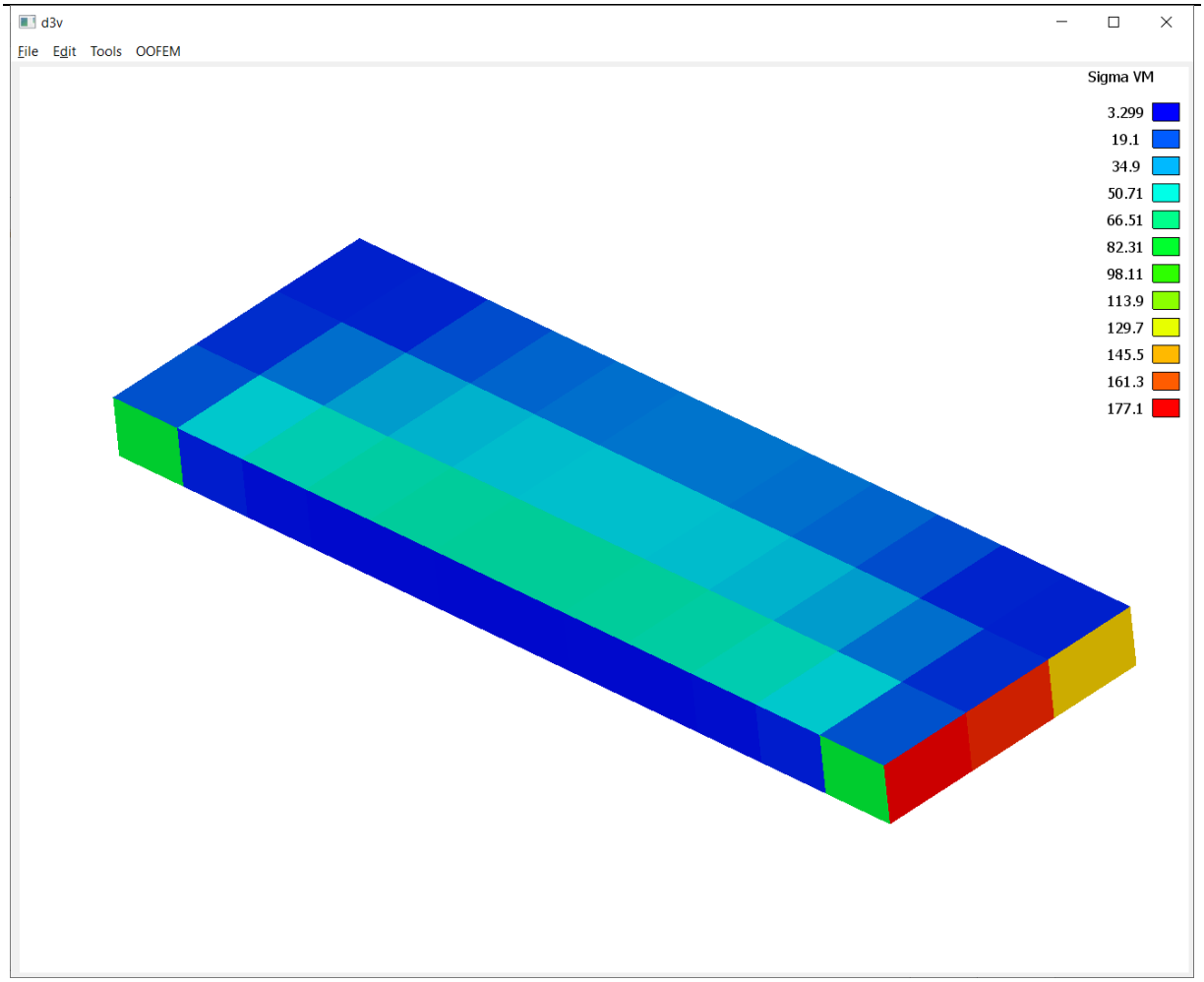
Slika 17 Prikaz normalnih naprezanja Sigma X



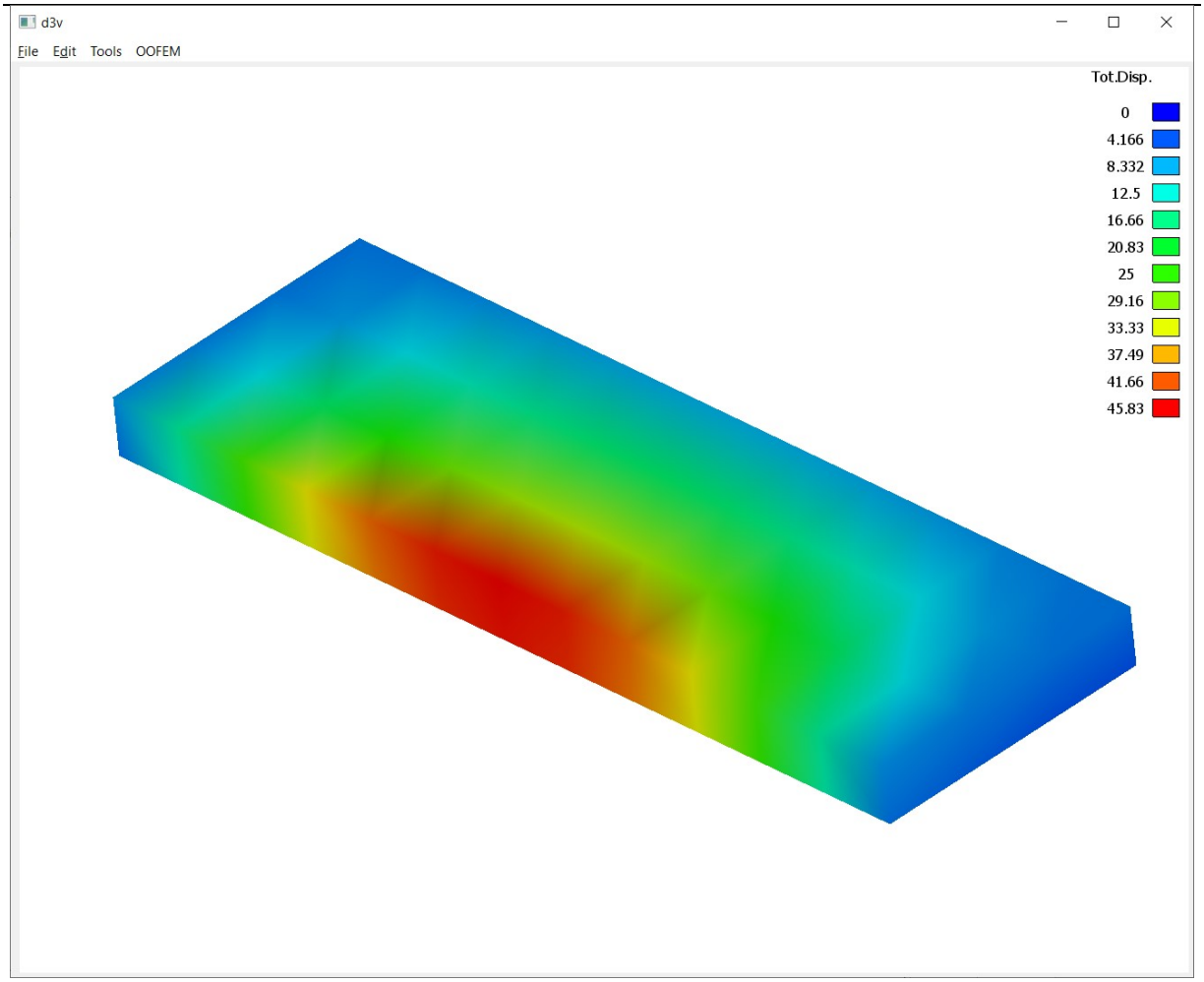
Slika 18 Prikaz normalnih naprezanja Sigma Y



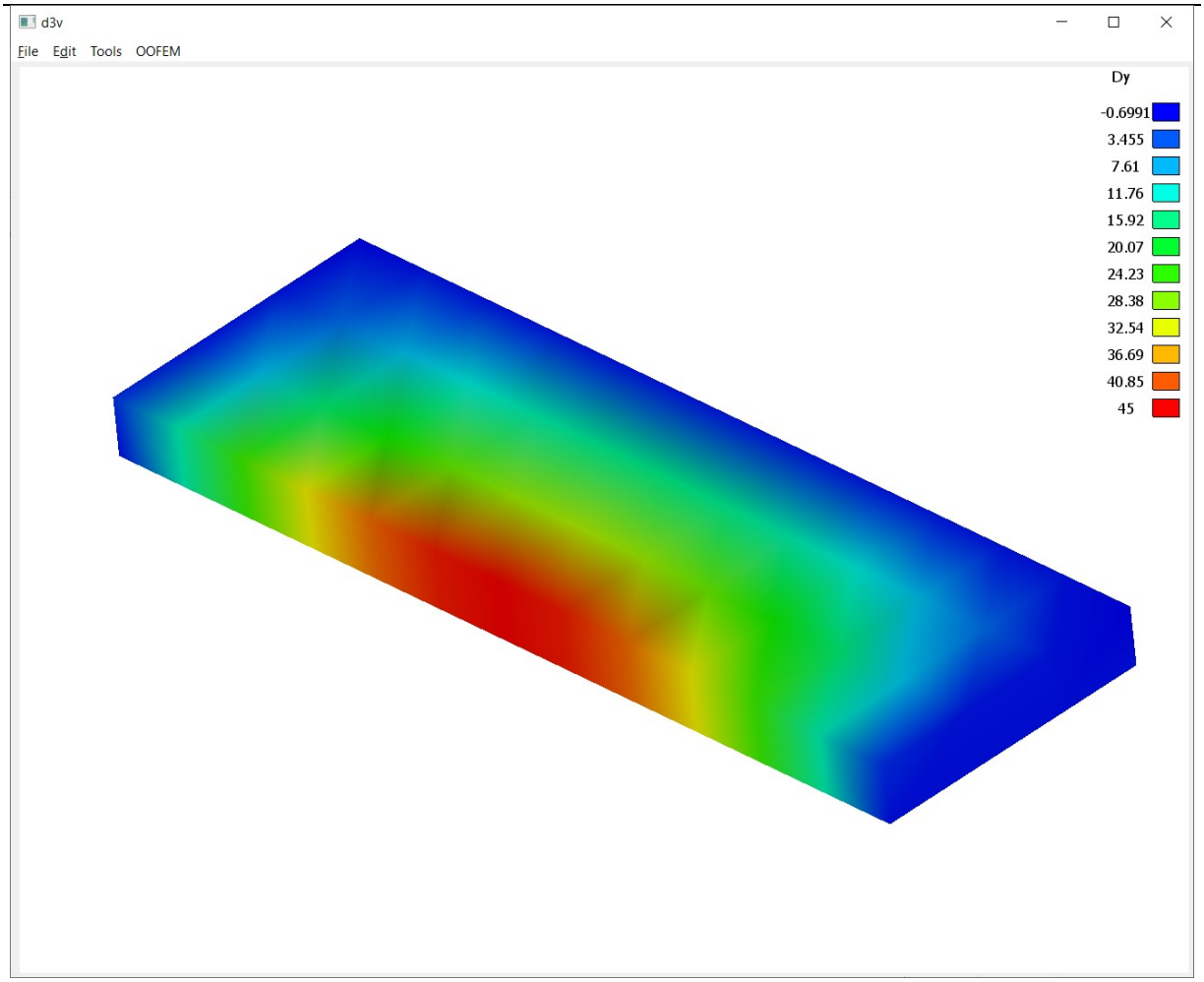
Slika 19 Prikaz smičnih naprezanja Tau XY



Slika 20 Prikaz naprezanja Sigma Von Mises



Slika 21 Prikaz ukupnih pomaka



Slika 22 Prikaz pomaka u smjeru osi y

ZAKLJUČAK

OOFEM program otvorenog koda je jednostavan za korištenje i proširivanje, a uz to je i besplatan. Karakteriziraju ga jednostavnost implementacije novih elemenata, vrste analiza koje su omogućene, problemi koji se s programom mogu rješavati. Program nema vlastitu okolinu za vizualizaciju modela i rezultata te korisnici stoga koriste različite dodatne softvere, ovisno o namjeni. U radu je razvijena vizualizacija osnovnih karakteristika OOFEM modela modeliranih sa konačnim elementima ljuske i grede što omogućuje vizualizaciju uobičajenih modela brodske konstrukcije. Vizualizacija je implementirana proširenjem funkcionalnosti programa otvorenog koda linaetal-fsb/d3v u programskom jeziku Python. Program linaetal-fsb/d3v je u ranoj fazi razvoja no već je moguća vizualizacija s ograničenim brojem funkcionalnosti. Trenutno nije moguć prikaz stranica konačnih elemenata s obzirom da ta funkcionalnost još uvijek ne postoji u linaetal-fsb/d3v. Izrađeni su moduli u programu Python koji omogućavaju 3D vizualizaciju karakteristika OOFEM modela konstrukcije broda i rezultata proračuna. Kod potreban za izradu Python modula je zahtijevao veliku količinu vremena, međutim zbog dostupnih materijala te kvalitetne podrške ga je moguće napisati. Instalacija programa i biblioteka potrebnih za rad programa linaetal-fsb/d3v je zahtjevna i potrebno je paziti koje se verzije programa koriste jer određene verzije pojedinih programa/biblioteka ne funkcioniraju sa ostalima. Tijekom rada ustanovljeno je da OOFEM Python interface trenutno radi isključivo na Linux operativnom sustavu, dok kod Windows sustava, na kojima je napravljen ovaj završni, to sučelje još uvijek ne funkcionira. U programu je omogućena vizualizacija karakteristika materijala i rezultata analiza odziva konstrukcije. Vizualizirane su 3 karakteristike vezane za materijal i to: materijal, poprečni presjek i debljina materijala. Pokretanjem programa može se odabrati što se želi prikazati na modelu koji je učitan i prikazati analizu rezultata.

LITERATURA

[1] Sorić, J., Metoda konačnih elemenata, Golden marketing - Tehnička knjiga, Zagreb, 2004.

[2] B. Patzák. OOFEM home page. http://www.oofem.org, 2000.

[3] B. Patzák and Z. Bittnar. Design of object oriented finite element code. Advances in Engineering Software,759–767, 2001.

[4] B. Patzák. Input data format specification. http://www.oofem.org, 2014.

[5] B. Patzák. Oofem element library manual. http://www.oofem.org, 2014.

[6] B. Patzák. Input data format specification. http://www.oofem.org, 2014.

[7] Python, url: https://www.python.org/

[8] The Python Tutorial: https://docs.python.org/3.5/tutorial/index.html

[9] About Qt., url: https://wiki.qt.io/About_Qt

[10] QtCreator, url: https://doc.qt.io/qtcreator/creator-overview.html

[11] About PySide, url: https://wiki.qt.io/About_PySide

[12] OpenMesh,url:https://www.graphics.rwth-aachen.de/media/openmesh_static/Documentations/OpenMesh-6.3-Documentation/a00008.html

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

[14] Colors, url: https://www.w3.org/TR/SVG11/types.html#ColorKeywords

[15] Continuous Colors, url: http://paulbourke.net/miscellaneous/colourspace/

PRILOZI

I. CD-R disc

II. Python skripte oofem.py i oofemcommands.py

Prilog II.

oofem.py skripta

```
import random

from geometry import Geometry
import openmesh as om
import numpy as np
import math


class OOFEM (Geometry):
    def __init__(self,fileName, guid = None):
        super().__init__(guid)
        self.drawLegend = False
        self.legendValues = []
        self.legendColors = []
        self.legendTitle=""
        self.filename=fileName
        self.all_face_handles = {} # key=faceHandle, value = element ID
        self.element2Face = {}     # key=element ID, value =
[faceHandle1,...faceHandlex]
        self.crosssectdict={}
        self.materialdict = {}
        self.elementcrosssectdict = {}
        self.resStress = {}
        self.resDisp = {}
        self.outFileName=""
        self.vertexNodeDict = {}

    def genMesh(self):
        self.mesh = self.oofemmesh()

    def pokusaj2(self):
        #prvaTocka=[0,0,0]
        #drugaTocka=[1,0,0]
        #orient=[0,1,0] # orijentacija struka
        #hw=0.3
        #tw=0.03
        #bf=0.2
        #tf=0.02
        #bp=0.25
        #tp=0.025

        prvaTocka = []
        for i in range(0, 3):
            ele = float(input("Unesite koordinatu " + str(i + 1) + " prve točke:
"))
            prvaTocka.append(ele)

        drugaTocka = []
        for i in range(0, 3):
            ele = float(input("Unesite koordinatu " + str(i + 1) + " druge točke:
"))
            drugaTocka.append(ele)

        orijent = []
```

```
        for i in range(0, 3):
            ele = float(input("Unesite koordinatu " + str(i + 1) + " orijentacije
struka: "))
            orijent.append(ele)

        hw = float(input("Unesite visinu struka u (m) : "))
        tw = float(input("Unesite debljinu struka u (m): "))
        bf = float(input("Unesite širinu flanže u (m): "))
        tf = float(input("Unesite debljinu flanže u (m): "))
        bp = float(input("Unesite širinu pokrova u (m): "))
        tp = float(input("Unesite debljinu pokrova u (m): "))


        return self.zadatak2(prvaTocka, drugaTocka,orijent,hw,tw,bf,tf,bp,tp)

        #return self.zadatak2(prvaTocka, drugaTocka, hw, tw, bf, tf, bp, tp)

    def zadatak2(self, prvaTocka, drugaTocka,orijent, hw,tw,bf,tf,bp,tp):
        mesh= om.TriMesh()

        vertexHandles = []
        faceHandles = []
        faceHandlesDict = {}

        Tocka1 = np.array(prvaTocka)
        vertexHandles.append(mesh.add_vertex(Tocka1))       #0
        Tocka2 = np.array(drugaTocka)
        vertexHandles.append(mesh.add_vertex(Tocka2))       #1
        Tocka1_ = Tocka1 + np.array([0, 0, 1])
        vertexHandles.append(mesh.add_vertex(Tocka1_))      #2
        Tocka2_ = Tocka2 + np.array([0, 0, 1])
        vertexHandles.append(mesh.add_vertex(Tocka2_))      #3
        Tocka1a = Tocka1 + np.array([0, 0, 2])
        vertexHandles.append(mesh.add_vertex(Tocka1a))      #4
        Tocka2a = Tocka2 + np.array([0, 0, 2])
        vertexHandles.append(mesh.add_vertex(Tocka2a))      #5

        x = np.array(Tocka2 - Tocka1)
        y = np.array(orijent)
        v = np.cross(x, y)
        v1 = v / np.linalg.norm(v)

        Tocka3 = Tocka2 + np.array(orijent)*hw
        vertexHandles.append(mesh.add_vertex(Tocka3))       #6
        Tocka4 = Tocka1 + np.array(orijent)*hw
        vertexHandles.append(mesh.add_vertex(Tocka4))       #7
        Tocka3_ = Tocka2_ + np.array(orijent)*hw
        vertexHandles.append(mesh.add_vertex(Tocka3_))      #8
        Tocka4_ = Tocka1_ + np.array(orijent)*hw
        vertexHandles.append(mesh.add_vertex(Tocka4_))      #9
        Tocka5_ = Tocka1_ + np.array(orijent)*hw - np.array(v1)*(bf * 0.5)
        vertexHandles.append(mesh.add_vertex(Tocka5_))      #10
        Tocka6_ = Tocka1_ + np.array(orijent)*hw + np.array(v1)*(bf * 0.5)
        vertexHandles.append(mesh.add_vertex(Tocka6_))      #11
        Tocka7_ = Tocka2_ + np.array(orijent)*hw + np.array(v1)*(bf * 0.5)
        vertexHandles.append(mesh.add_vertex(Tocka7_))      #12
        Tocka8_ = Tocka2_ + np.array(orijent)*hw - np.array(v1)*(bf * 0.5)
        vertexHandles.append(mesh.add_vertex(Tocka8_))      #13
        Tocka3a = Tocka2a + np.array(orijent)*hw
```

```
        vertexHandles.append(mesh.add_vertex(Tocka3a))      #14
        Tocka4a = Tocka1a + np.array(orijent)*hw
        vertexHandles.append(mesh.add_vertex(Tocka4a))      #15
        Tocka5a = Tocka1a + np.array(orijent)*hw - np.array(v1)*(bf * 0.5)
        vertexHandles.append(mesh.add_vertex(Tocka5a))      #16
        Tocka6a = Tocka1a + np.array(orijent)*hw + np.array(v1)*(bf * 0.5)
        vertexHandles.append(mesh.add_vertex(Tocka6a))      #17
        Tocka7a = Tocka2a + np.array(orijent)*hw + np.array(v1)*(bf * 0.5)
        vertexHandles.append(mesh.add_vertex(Tocka7a))      #18
        Tocka8a = Tocka2a + np.array(orijent)*hw - np.array(v1)*(bf * 0.5)
        vertexHandles.append(mesh.add_vertex(Tocka8a))      #19
        Tocka9a = Tocka1a - np.array(v1)*(bp * 0.5)
        vertexHandles.append(mesh.add_vertex(Tocka9a))      #20
        Tocka10a = Tocka1a + np.array(v1)*(bp * 0.5)
        vertexHandles.append(mesh.add_vertex(Tocka10a))     #21
        Tocka11a = Tocka2a + np.array(v1)*(bp * 0.5)
        vertexHandles.append(mesh.add_vertex(Tocka11a))     #22
        Tocka12a = Tocka2a - np.array(v1)*(bp * 0.5)
        vertexHandles.append(mesh.add_vertex(Tocka12a))     #23


#faceHandles.append(mesh.add_face(vertexHandles[0],vertexHandles[1],vertexHandles[
6]))

faceHandlesDict[mesh.add_face(vertexHandles[0],vertexHandles[1],vertexHandles[6])]
= 0

#faceHandles.append(mesh.add_face(vertexHandles[6],vertexHandles[7],vertexHandles[
0]))
        faceHandlesDict[mesh.add_face(vertexHandles[6], vertexHandles[7],
vertexHandles[0])] = 6

        #faceHandles.append(mesh.add_face(vertexHandles[2],
vertexHandles[3],vertexHandles[8]))
        faceHandlesDict[mesh.add_face(vertexHandles[2], vertexHandles[3],
vertexHandles[8])] = 2
        #faceHandles.append(mesh.add_face(vertexHandles[8],
vertexHandles[9],vertexHandles[2]))
        faceHandlesDict[mesh.add_face(vertexHandles[8], vertexHandles[9],
vertexHandles[2])] = 8

#faceHandles.append(mesh.add_face(vertexHandles[10],vertexHandles[11],vertexHandle
s[12]))
        faceHandlesDict[mesh.add_face(vertexHandles[10], vertexHandles[11],
vertexHandles[12])] = 10

#faceHandles.append(mesh.add_face(vertexHandles[12],vertexHandles[13],vertexHandle
s[10]))
        faceHandlesDict[mesh.add_face(vertexHandles[12], vertexHandles[13],
vertexHandles[10])] = 12


#faceHandles.append(mesh.add_face(vertexHandles[4],vertexHandles[5],vertexHandles[
14]))
        faceHandlesDict[mesh.add_face(vertexHandles[4], vertexHandles[5],
vertexHandles[14])] = 4
        #faceHandles.append(mesh.add_face(vertexHandles[14],vertexHandles[15],
vertexHandles[4]))
        faceHandlesDict[mesh.add_face(vertexHandles[14], vertexHandles[15],
```

```
vertexHandles[4])] = 14

#faceHandles.append(mesh.add_face(vertexHandles[16],vertexHandles[17],vertexHandle
s[18]))
        faceHandlesDict[mesh.add_face(vertexHandles[16], vertexHandles[17],
vertexHandles[18])] = 16

#faceHandles.append(mesh.add_face(vertexHandles[18],vertexHandles[19],vertexHandle
s[16]))
        faceHandlesDict[mesh.add_face(vertexHandles[18], vertexHandles[19],
vertexHandles[16])] = 18

#faceHandles.append(mesh.add_face(vertexHandles[20],vertexHandles[21],vertexHandle
s[22]))
        faceHandlesDict[mesh.add_face(vertexHandles[20], vertexHandles[21],
vertexHandles[22])] = 20

#faceHandles.append(mesh.add_face(vertexHandles[22],vertexHandles[23],vertexHandle
s[20]))
        faceHandlesDict[mesh.add_face(vertexHandles[22], vertexHandles[23],
vertexHandles[20])] = 22



        #print(vertexHandles)
        #print(faceHandles)
        print(faceHandlesDict)

        return mesh
        pass


    def oofemmesh(self):

        mesh = om.TriMesh()
        self.mesh=mesh
        f = open(self.filename, newline='')
        all_vertices = {}
        all_face_handles = {}

plateElementTypes={"dktplate","mitc4shell","planestress2d","trplanestress2d","trpl
anestressrotallman","trplanestrrot"}

        for line in f:
            line = ' '.join(line.split())
            if line.startswith('#'):
                continue
            if self.outFileName=="":
                self.outFileName=line.strip()
                abspath = '\\'.join(self.filename.split('\\')[0:-1])
                if len(abspath) < 1:
                    abspath = '/'.join(self.filename.split('/')[0:-1])
                self.outFileName = abspath + '/' + self.outFileName
                continue
            sline = line.split(" ")
            if len(sline) < 3:
                continue
            #print(sline[0].lower())
            if sline[0].lower() == "node":
```

```
                d = [float(sline[4]), float(sline[5]), float(sline[6])]
                all_vertices[sline[1]] = d
            elif  sline[0].lower() in plateElementTypes:
                id=int(sline[1])
                elementFaceHandles=[]
                self.element2Face[id]=elementFaceHandles
                self.elementcrosssectdict[id]=int(sline[-1])
                numNodes=int(sline[3])
                if numNodes >= 3:
                    vh_list = [mesh.add_vertex(all_vertices.get(sline[4])),
mesh.add_vertex(all_vertices.get(sline[5])),
                               mesh.add_vertex(all_vertices.get(sline[6]))]
                    index = 4
                    # print(vh_list)
                    for vh in vh_list:
                        # print(vh)
                        self.vertexNodeDict[vh.idx()] = int(sline[index])
                        index = index + 1
                    fh=mesh.add_face(vh_list)
                    self.all_face_handles[fh] = int(sline[1])
                    elementFaceHandles.append(fh)
                    pass
                    if numNodes == 4:
                        vh_list = [mesh.add_vertex(all_vertices.get(sline[6])),
mesh.add_vertex(all_vertices.get(sline[7])),
                                   mesh.add_vertex(all_vertices.get(sline[4]))]
                        index = 6
                        for vh in vh_list:
                            # print(vh)
                            if index == 8:
                                index = 4
                            self.vertexNodeDict[vh.idx()] = int(sline[index])
                            index = index + 1
                        fh = mesh.add_face(vh_list)
                        self.all_face_handles[fh] = int(sline[1])
                        elementFaceHandles.append(fh)
                        pass
                else:
                    print ("unhandled type of the element")
                    pass
            elif sline[0].lower() == "simplecs":
                self.crosssectdict[int(sline[1])]=sline
            elif sline[0].lower() == "isole":
                self.materialdict[int(sline[1])] = sline
        f.close()
        #self.showFaceColorC()
        #self.showFaceColorP()
        #self.showVertexColor()
        return mesh
    def getPropIDforElID(self,elID):
        n=elID
        while n > 10:
            n=n-10
        propID = self.elementId_color_dict[n]
        return propID

    def getElResult(self,elID,numEl):
        retVal=0
        retVal=random.uniform(0, 1)
```

```
        return  retVal

    def showSolidColor(self):
        self.legendValues.clear()
        self.legendColors.clear()
        self.drawLegend = False

    def showCrossSectID(self):
        self.legendTitle="Cross section ID"
        self.showFaceColorP(self.elementcrosssectdict)

    def showMaterialID(self):
        self.legendTitle = "Material ID"
        propDict={}
        for el in self.elementcrosssectdict:
            csID= self.elementcrosssectdict[el]
            sline= self.crosssectdict[csID]
            propDict[el]=int(sline[-1])
        self.showFaceColorP(propDict)

    def showThicknessID(self):
        self.legendTitle = "Thickness"
        propDict={}
        for el in self.elementcrosssectdict:
            thID= self.elementcrosssectdict[el]
            sline= self.crosssectdict[thID]
            propDict[el]=float(sline[3])
        self.showFaceColorP(propDict)

    def showElementStress(self,sType):
        legTitles = ["Sigma X", "Sigma Y", "Tau XY"]
        if sType < 3:
            self.legendTitle = legTitles[sType]
        else:
            self.legendTitle = "Sigma VM"
        elResDict={}
        for elID in self.resStress:
            stress=self.resStress[elID]
            if sType < 3:
                val = stress[sType]
            else:
                val = math.sqrt(math.pow(stress[0], 2) + math.pow(stress[1], 2) +
3*math.pow(stress[2], 2)-stress[0]*stress[1])
            elResDict[elID]=val
        self.showFaceColorC(elResDict)

    def showFaceColorP(self,propDict):
        colors = [[0, 0, 255,255],[128, 0, 128,255],[222, 184, 135,255],[255, 165,
0,255],[0, 255, 0,255],[ 0, 128, 0,255],[128, 0, 0,255],[255, 0, 0,255],[255, 192,
203,255],[222, 184, 135,255],[255, 165, 0,255],[255, 127, 80,255],[128, 128,
0,255],[255, 255, 0,255],[245, 245, 220,255],[0, 255, 0,255],[ 0, 128,
0,255],[245, 255, 250,255],[0, 128, 128,255],[0, 255, 255,255],[0, 0,
128,255],[230, 230, 250,255],[255, 0, 255,255],[205, 133, 63,255]]
        floatColors = []
        for color in colors:
            floatColors.append([x / 255 for x in color])
        mesh= self.mesh
        mesh.request_face_colors()
        propColorDict={}
```

```
        self.legendValues.clear()
        self.legendColors.clear()
        self.drawLegend = False
        for el in self.element2Face:
            idProp=propDict[el]
            nuc=len(propColorDict)
            indexColor = 0
            if idProp in propColorDict:
                indexColor=propColorDict[idProp]
            else:
                propColorDict[idProp]=nuc
                indexColor=nuc
                self.legendValues.append(str(idProp))
                self.legendColors.append(floatColors[indexColor])
            for fh in self.element2Face[el]:
                mesh.set_color(fh, floatColors[indexColor])
            pass
        if  len(self.legendValues)> 0:
            self.drawLegend=True
        pass

    def prepContColorLegend(self,minVal, maxVal,nColor):
        self.legendValues.clear()
        self.legendColors.clear()
        self.drawLegend = True
        legendValues=np.linspace(minVal,maxVal,nColor)
        for x in legendValues:
            self.legendValues.append(f"{x:.4g}")
        for val in legendValues:
            color = self.getContinuousColor(val, minVal, maxVal)
            self.legendColors.append(color)


    def showFaceColorC(self,dictElVals):
        mesh= self.mesh
        mesh.release_vertex_colors()
        mesh.request_face_colors()
        minVal=float('inf')
        maxVal = float('-inf')
        for el in dictElVals:
            val=dictElVals[el]
            if val > maxVal:
                maxVal=val
            if val < minVal:
                minVal = val
        index = 0
        self.prepContColorLegend(minVal,maxVal,12)
        for el in self.element2Face:
            index = index + 1
            val = dictElVals[el]
            for fh in self.element2Face[el]:
                color = self.getContinuousColor(val, minVal, maxVal)
                mesh.set_color(fh, color)
            pass
        pass

    def showElementVertexColor(self):
        mesh = self.mesh
        mesh.release_face_colors()
```

```
        mesh.request_vertex_colors()
        for el in self.element2Face:
            for fh in self.element2Face[el]:
                iv=1
                for vh in mesh.fv(fh):
                    val = iv / len(mesh.vertices())
                    color = self.getContinuousColor(val, 0, 1)
                    mesh.set_color(vh, color)
                    print(vh.idx())
                    print("VHid", vh.idx())
                    print("VHval ",self.vertexNodeDict.get(vh.idx()))
                    iv=iv+1

    def showNodeVertexColor(self,dType):
        legTitles=["Dx","Dy","Dz","Dx","Dy","Dz"]
        if dType < 6:
            self.legendTitle = legTitles[dType]
        else:
            self.legendTitle = "Tot.Disp."

        minVal = float('inf')
        maxVal = float('-inf')
        for idNode in self.resDisp:
            disp=self.resDisp[idNode]
            if dType < 6:
                val = disp[dType]
            else:
                val =
math.sqrt(math.pow(disp[0],2)+math.pow(disp[1],2)+math.pow(disp[2],2))
            if val > maxVal:
                maxVal = val
            if val < minVal:
                minVal = val
        self.prepContColorLegend(minVal, maxVal, 12)
        mesh = self.mesh
        mesh.request_vertex_colors()
        for vh in mesh.vertices():
            idNod=self.vertexNodeDict.get(vh.idx())
            disp=self.resDisp[idNod]
            if dType < 6:
                val = disp[dType]
            else:
                val =
math.sqrt(math.pow(disp[0],2)+math.pow(disp[1],2)+math.pow(disp[2],2))
            color = self.getContinuousColor(val, minVal, maxVal)
            mesh.set_color(vh, color)


    def getContinuousColor(self, v, vmin, vmax):
        color = [1.0, 1.0, 1.0, 1.0]
        if v < vmin:
            v = vmin
        if v > vmax:
            v = vmax
        dv = vmax - vmin

        if (v < (vmin + 0.25 * dv)):
            color[0] = 0
            color[1] = 4 * (v - vmin) / dv
```

```
        elif (v < (vmin + 0.5 * dv)):
            color[0] = 0
            color[2] = 1 + 4 * (vmin + 0.25 * dv - v) / dv
        elif (v < (vmin + 0.75 * dv)):
            color[0] = 4 * (v - vmin - 0.5 * dv) / dv
            color[2] = 0
        else:
            color[1] = 1 + 4 * (vmin + 0.75 * dv - v) / dv
            color[2] = 0
        return color

    def getSplitLine(self,line):
        line=line.strip()
        line = ' '.join(line.split())
        split = line.split(' ')
        return split

    def ReadOutput(self):
        f = open(self.outFileName, newline='')
        self.resStress.clear()
        self.resDisp.clear()
        readValues = [0.0] * 6;
        isDMPassed=False
        isNodeResults = False
        isElementResults=False

        for line in f:
            line=line.lower()
            if line.startswith('#'):
                continue
            if "dofmanager output" in line:
                isDMPassed=True
                continue
            if isDMPassed and line.startswith('node'):
                isNodeResults=True
            elif isNodeResults and "element output:" in line:
                isElementResults = True
                isNodeResults=False
                continue
            if isNodeResults:
                splitData = self.getSplitLine(line)
                while splitData[0] =='node':
                    idNode = int(splitData[1])
                    dispres=[0.0]*6
                    line=next(f).lower()
                    splitData = self.getSplitLine(line)
                    while splitData[0] =='dof':
                        idDof=int(splitData[1])
                        dispres[idDof-1]=float(splitData[3])
                        line=next(f).lower()
                        splitData = self.getSplitLine(line)
                    self.resDisp[idNode]=dispres
            elif isElementResults:
                splitData = self.getSplitLine(line)
                while splitData[0] == 'element':
                    stresres=[0.0]*3
                    nsl=0
                    idEl = int(splitData[1])
                    line = next(f).lower()
```

```
                splitData = self.getSplitLine(line)
                while len(splitData) > 0 and splitData[0] != 'element':
                    if splitData[0] == 'stresses':
                        stresres[0]=stresres[0]+float(splitData[1])
                        stresres[1] = stresres[1] + float(splitData[2])
                        stresres[2] = stresres[2] + float(splitData[4])
                        nsl=nsl+1
                        pass
                    try:
                        line = next(f).lower()
                        splitData = self.getSplitLine(line)
                    except StopIteration:
                        splitData=[]
                stresres[0]=stresres[0]/nsl
                stresres[1] = stresres[1] / nsl
                stresres[2] = stresres[2] / nsl
                self.resStress[idEl]=stresres
                if len(splitData)==0:
                    break

        f.close()
```

oofemcommand.py skripta

```
from PySide2.QtWidgets import QApplication, QMenu, QMessageBox
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 oofem import OOFEM
import os


class OOFEMCommand(Command):
    def __init__(self):
        super().__init__()
        app = QApplication.instance()
        app.registerIOHandler(OOFEMImporter())

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

        self.menuOOFEM = QMenu("OOFEM")

        self.menuView = QMenu("&View")
        self.menuAnalysis = QMenu("&Analysis")
        self.menuOOFEM.addMenu(self.menuView)
        self.menuOOFEM.addMenu(self.menuAnalysis)

        self.menuResults = QMenu("&Results")
        self.menuView.addMenu(self.menuResults)
        self.menuResults.setEnabled(False)

        menuCSID = self.menuView.addAction("CSID")
        menuCSID.triggered.connect(self.onCSID)

        menuMatID = self.menuView.addAction("MatID")
        menuMatID.triggered.connect(self.onMatID)

        menuThID = self.menuView.addAction("Thickness")
        menuThID.triggered.connect(self.onThID)

        menuAnalyse = self.menuAnalysis.addAction("Analyse")
        menuAnalyse.triggered.connect(self.onAnalyse)

        menuStressX = self.menuResults.addAction("Sigma X")
        menuStressX.triggered.connect(self.onSx)

        menuStressY = self.menuResults.addAction("Sigma Y")
        menuStressY.triggered.connect(self.onSy)

        menuStressXY = self.menuResults.addAction("Tau XY")
        menuStressXY.triggered.connect(self.onTxy)

        menuStressVM = self.menuResults.addAction("Sigma VM")
        menuStressVM.triggered.connect(self.onSVM)
```

```
        menuDisp = self.menuResults.addAction("Tot. Disp.")
        menuDisp.triggered.connect(self.onDisp)

        menuDx = self.menuResults.addAction("Dx")
        menuDx.triggered.connect(self.onDx)

        menuDy = self.menuResults.addAction("Dy")
        menuDy.triggered.connect(self.onDy)

        menuDz = self.menuResults.addAction("Dz")
        menuDz.triggered.connect(self.onDz)

        #tools.addMenu(menu)
        mb.addMenu(self.menuOOFEM)
        self.menuOOFEM.setEnabled(False)
        Signals.get().geometryAdded.connect(self.registerOOFEM)
        self.oofem = 0

    def registerOOFEM(self, oofem):
        if isinstance(oofem,OOFEM):
            self.oofem=oofem
            self.menuOOFEM.setEnabled(True)

    def onSx(self):
        self.oofem.showElementStress(0)
        Signals.get().geometryAdded.emit(self.oofem)

    def onSy(self):
        self.oofem.showElementStress(1)
        Signals.get().geometryAdded.emit(self.oofem)

    def onTxy(self):
        self.oofem.showElementStress(2)
        Signals.get().geometryAdded.emit(self.oofem)


    def onSVM(self):
        self.oofem.showElementStress(10)
        Signals.get().geometryAdded.emit(self.oofem)


    def onDx(self):
        self.oofem.showNodeVertexColor(0)
        Signals.get().geometryAdded.emit(self.oofem)

    def onDy(self):
        self.oofem.showNodeVertexColor(1)
        Signals.get().geometryAdded.emit(self.oofem)

    def onDz(self):
        self.oofem.showNodeVertexColor(2)
        Signals.get().geometryAdded.emit(self.oofem)

    def onDisp(self):
        self.oofem.showNodeVertexColor(10)
        Signals.get().geometryAdded.emit(self.oofem)

    def onMatID(self):
```

```
        self.oofem.showMaterialID()
        Signals.get().geometryAdded.emit(self.oofem)

    def onCSID(self):

        self.oofem.showCrossSectID()
        Signals.get().geometryAdded.emit(self.oofem)
        pass

    def onThID(self):

        self.oofem.showThicknessID()
        Signals.get().geometryAdded.emit(self.oofem)
        pass

    def onAnalyse(self):
        self.oofem.ReadOutput()
        self.menuResults.setEnabled(True)


class OOFEMImporter(IOHandler):
    def __init__(self):
        super().__init__()
        Signals.get().importGeometry.connect(self.importGeometry)

    def importGeometry(self, fileName):
        if len(fileName) < 1:
            return
        filename, file_extension = os.path.splitext(fileName)
        if file_extension != ".in":
            return
        oofem=OOFEM(fileName)
        oofem.genMesh()
        Signals.get().geometryImported.emit(oofem)

    def getImportFormats(self):
        return []


def createCommand():
    return OOFEMCommand()
```