SVEUČILIŠTE U SPLITU
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Izrada aplikacije za obradu rezultata 3D skeniranja iz oblaka točaka i priprema modela za 3D ispis

Mijo Jurić-Pešić

Split, rujan 2024.

SVEUČILIŠTE U SPLITU
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

Sveučilišni prijediplomski studij: **Računarstvo**
Smjer/Usmjerenje: **/**
Oznaka programa: 120
Akademska godina: 2023./2024.

Ime i prezime: **Mijo Jurić-Pešić**
JMBAG: 0023143979

ZADATAK ZAVRŠNOG RADA

Naslov: **Izrada aplikacije za obradu rezultata 3D skeniranja iz oblaka točaka i priprema modela za 3D ispis**

Zadatak: Objasniti postupak 3D skeniranja strukturiranim svjetlom. Objasniti postupak 3D ispisa FFF tehnologijom te oblik potrebne datoteke (stl,...) za postojeće aplikacije za generiranje g-koda. Za jedan primjer napraviti postupak 3D skeniranja i dobivanje 3D geometrije kao oblaka točaka. Napisati skriptu za pripremu oblaka točaka te izvoz u prikladni geometrijski format za 3D ispis. Napraviti postupak 3D ispisa i usporediti rezultate s originalnim objektom.

Datum obrane: 17.09.2024.

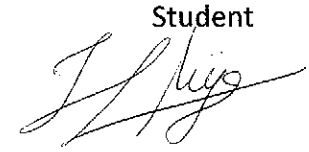
Mentor:

doc. dr. sc. Ivo Marinić-Kragić

IZJAVA

Ovom izjavom potvrđujem da sam završni rad s naslovom „Izrada aplikacije za obradu rezultata 3D skeniranja iz oblaka točaka i priprema modela za 3D ispis“ pod mentorstvom doc. dr. sc. Ive Marinića-Kragića pisao samostalno, primijenivši znanja i vještine stečene tijekom studiranja na Fakultetu elektrotehnike, strojarstva i brodogradnje, kao i metodologiju znanstveno-istraživačkog rada te uz korištenje literature koja je navedena u radu. Spoznaje, stavove, zaključke, teorije i zakonitosti drugih autora koje sam izravno ili parafrazirajući naveo u završnom radu citirao sam i povezao s korištenim bibliografskim jedinicama.

Student



Mijo Jurić-Pešić

SADRŽAJ

1 UVOD 1

2 OPĆENITO O 3D SKENIRANJU ¸ 2
2.1 Korištena tehnologija za obavljanje skeniranja 2
2.2 Primjena 3D skeniranja i ispis 3

3 PROCES 3D SKENIRANJA 4
3.1 Priprema predmeta za skeniranje 4
3.2 Postupak samog skeniranja 5
3.3 Proces poligonizacije 6

4 STL FORMAT DATOTEKE 8

5 NEDOSTATCI PRILIKOM 3D SKENIRANJA 10
5.1 Refleksija predmeta 10
5.2 Nemogućnost preciznog skeniranja svakog ruba predmeta 11

6 OBRADA 3D STL DATOTEKE I PRIPREMA ZA ISPIS 12
6.1 Vizualni prikaz skeniranog predmeta 12
6.2 Izrada softvera za detekciju šupljina 14
6.3 Krpanje šupljina koristeći gotova softverska rješenja 18

7 ZAKLJUČAK 19

LITERATURA 20

PRILOZI 22
Kazalo slika, tablica i kodova 22
Kazalo slika 22

SAŽETAK I KLJUČNE RIJEČI 23

SUMMARY AND KEYWORDS 24

1 UVOD

Rad je podijeljen na sedam poglavlja. U drugom poglavlju dan je općeniti pregled samog 3D skeniranja. Treće poglavlje govori o procesu 3D skeniranja, a četvrto poglavlje pobliže opisuje STL format datoteke koji se dobije kao rezultat 3D skeniranja. U petom poglavlju ukratko su izneseni problemi, poteškoće i nedostatci 3D skeniranja, dok u šestom poglavlju govorimo o obradi STL datoteke i pripremi iste za ispis. Zaključke i neka zapažanje donosim u istoimenom poglavlju.

3D skeniranje i ispis je jedna od najbrže razvijanih IT branša današnjice te svoju primjenu pronalazi u sve više sfera ljudskog života.

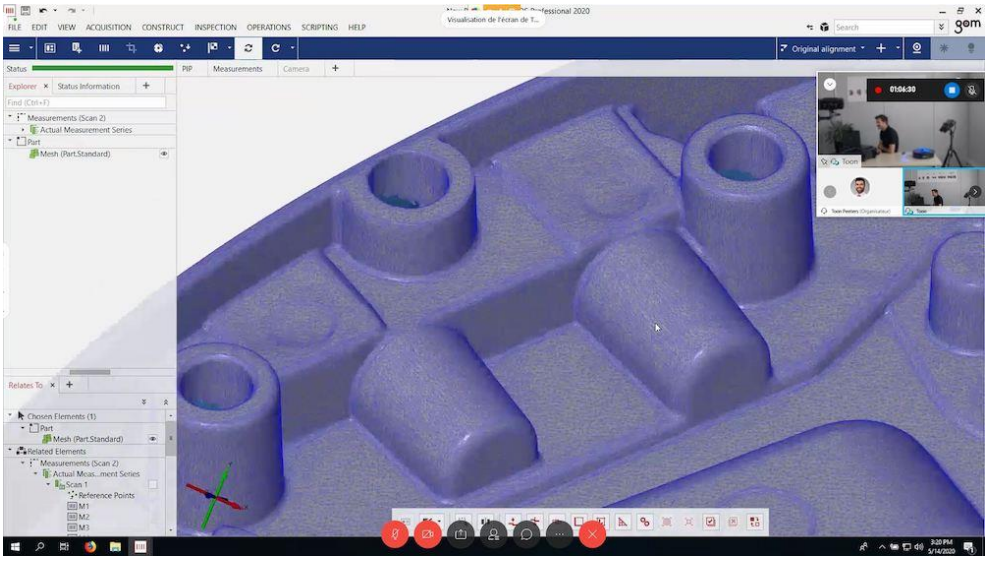
3D skeniranje je proces dohvaćanja 3D geometrija danog predmeta te pohrana točaka sa njihovim trodimenzionalnim koordinatama u STL datoteci koji se daljnjom obradom usavršava i priprema za ispis. Tehnologije koje se koriste su 3D skener sa plavim strukturiranim svjetlom i dvije kamere, ATOS tehnologije pomoću kojih će se predmet skenirati te 3D printer za ispis danog predmeta.

2 OPĆENITO O 3D SKENIRANJU ¸

3D skeniranje izvodi se istoimenim uređajem, 3D skenerom koji pomoću strukturalnog svijetla. Skener prikuplja informacije o predmetu u sve tri dimenzije (visina, širina i dubljina) te pomoću tih podataka kreira 3D prikaz skeniranog objekta. Strukturirano svjetlo ima uzorak pruga, mreže ili paralelnih linija koje na takav način pružaju uvid u dubinu i oblik samog skeniranog predmeta. Jedna od jednostavnijih implementacija SL-a (Structure Light) obuhvaća projiciranje uskog tamnog (svijetlog) vertikalnog »prozora« na svijetloj (tamnoj) pozadini pomoću video projektora.[1] Kada to svjetlo osvijetli dani predmet, 3D kamere koje su često IR (Infrared Camera tj. infracrvena kamera) i RGB (Red Green Blue Camera tj. crveno zeleno plava kamera) snimaju od desetak do tisuću slika koje se tada šalju softveru za obradu 3D skena.

2.1 Korištena tehnologija za obavljanje skeniranja

Za skeniranje prilikom izrade ovog završnog rada, koristimo 3D skener koji koristi tehnologiju strukturiranog svjetla zajedno sa ATOS tehnologijom. 3D skener koji koristimo ima dvije kamere te projektor koji projicira strukturirano svjetlo plave boje, a za obradu podataka ćemo se oslanjati na ATOS tehnologiju. Na koji način ćemo skenirati dani predmet ćemo se dotaknuti u slijedećem poglavlju. Slika 2-1 pokazuje izgled samog ATOS sučelja.



Slika 2-1 ATOS sučelje

[1] Pribanić et al., 'Pojednostavljenje Određivanja Položaja Ravnine Svjetla Za Vrijeme Skeniranja Strukturiranim Svjetlom'.

2.2 Primjena 3D skeniranja i ispis

3D skeniranje i ispis ima široku upotrebu u raznim sferama znanosti i industrije te uvelike olakšava demonstraciju gotovog proizvoda ili daje gotovo rješenje za određeni problem. Neke od primjena su:

- Industrija – brza izrada prototipova određenog proizvoda
- Medicina – izrada proteza ili ortopedskih pomagala
- Arhitektura – modeliranje prostora ili izrada već gotovog proizvoda (zgrade, kuće i slično)
- Arheologija – stvaranje vizualnih modela određenih artefakta
- Automobilska industrija – izrada prototipa vozila ili pojedinih komponenti
- I brojne druge…

Uz umjetnu inteligenciju, 3D sekiranje i ispis ima sve veću upotrebu te je jedna od najviše i najbrže razvijanih IT industrija.

3 PROCES 3D SKENIRANJA

Samo 3D skeniranje je dosta jednostavan proces koji se može podijeliti u tri etape:

1. Priprema predmeta za skeniranje
2. Postupak samog skeniranja
3. Proces poligonizacije

Ovisno o samom odabranom predmetu, proces može trajat od 20 minuta za jednostavne predmete do nekoliko sati za zahtjevne predmete. Predmeti koji su zahtjevni za skeniranje su predmeti koji imaju veći broj neravnina, udubina, šupljina te ako su predmeti refleksivni. Kako pravilno skenirati takve predmete, objasnit ćemo na primjeru metalnog vijka koji je zbog samog materijala refleksivan predmet te sadrži navoje koji upadaju u kategoriju neravnina i udubina koje se teže skeniraju.

3.1 Priprema predmeta za skeniranje

Kao i u svemu, pravilna priprema je većina posla koju sami operater 3D skenera mora obaviti te što se predmet kvalitetnije pripremi, skeniranje će biti preciznije, lakše i točnije. Prvo, razmatramo predmet koji skeniramo, a u našem slučaju to je metalni vijak tj. željezni vijak. Kako bi 3D skener znao koji dio samog predmeta skeniramo, na vijak ćemo prvo nanijeti niz referentnih točaka. Referentne točke su crno-bijele točke čije dimenzije ovise o skeniranom predmetu, veće referentne točke za veće predmete i obratno. Bitno je da pri svakom pojedinom skeniranju 3D skener vidi najmanje tri referentne točke koje će povezati u trokut. Vijak je refleksivan materijal što znači da pod utjecajem svjetla, sunčevog zračenja ili bilo kojeg drugog izvora svjetla, vijak reflektira određenu količinu svjetla koja onemogućava skeniranje. Taj 'višak' svjetla remeti strukturirano svjetlo koje odašilje projektor 3D skenera te se ne mogu pravilo očitati geometrije predmeta. Kako bismo to spriječili, na predmet se nanosi sprej protiv refleksije. Posljednji je korak čišćenje referentnih točaka kako bi ih skener mogao prepoznati. Nakon toga, sam predmet je spreman za skeniranje. Slika 3-1 pokazuje kako izgleda pravilno pripremljen predmet za skeniranje, a u našem slučaju radi se o metalnom vijku.

Slika 3-1 Primjer predmeta pripremljenog za skeniranje

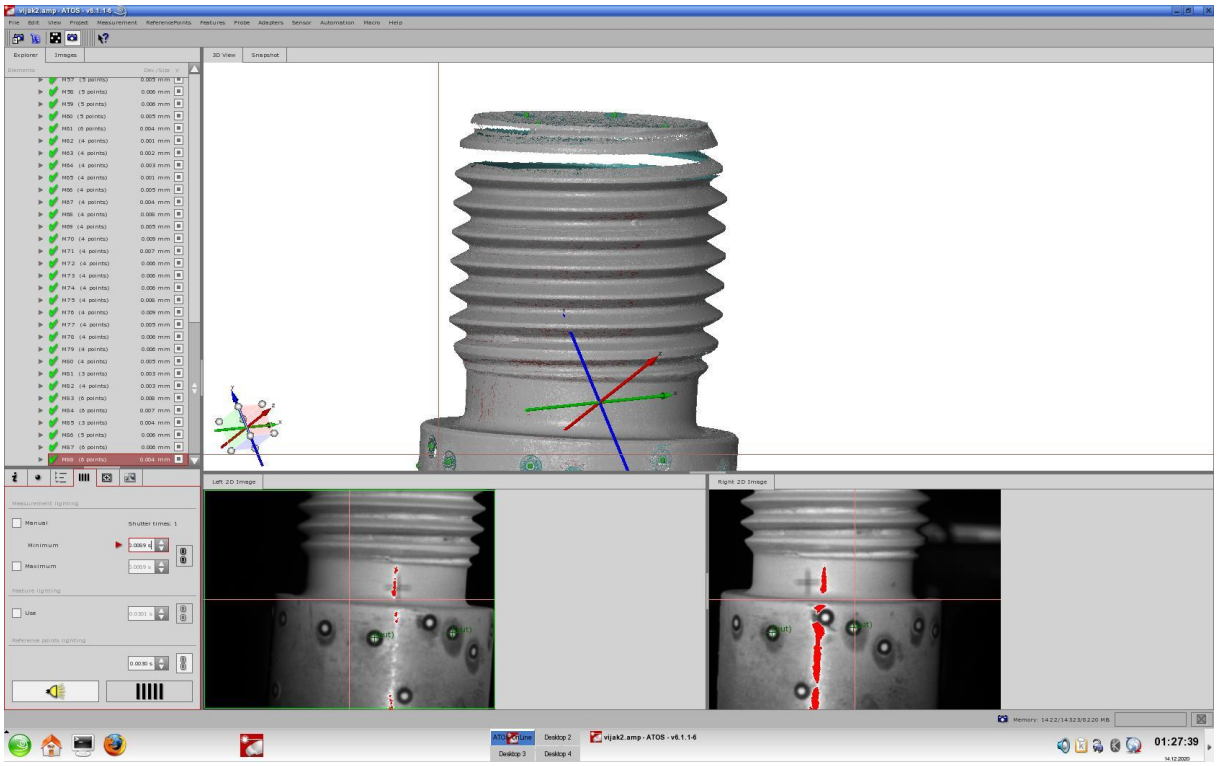
3.2 Postupak samog skeniranja

Nakon pripreme predmeta, slijedi sam proces skeniranja predmeta. 3D skener se postavi tako da su sve površine predmeta vidljive te se predmet postavi na tamnu površinu kako bi se izbjegle bilo kakve refleksije. Kao što je već spomenuto, skener može skenirati samo površinu određene veličine te u svakom trenutku pomoću referentnih točaka zna koji dio danog predmeta skenira. Kako bismo uspješno skenirali, svaka skenirana površina mora:

1. Imati najmanje tri referentne točke u svakom danom trenutku koje povezuje u trokute. Ako točaka ima više, tada su mjerenja točnija jer je samih trokuta referentnih točaka više.
2. Refleksija predmeta mora biti nula ili minimalna za dani predmet

Kada smo zadovoljni pozicijom te površinom kojom skeniramo, klikom na razmak tj. SPACE skener obavlja skeniranje. Trajanje jednog skeniranja je oko par sekundi. Nakon toga predmet se rotira te se pazi da su obje gore navedene točke ispunjene te se proces ponavlja sve dok u

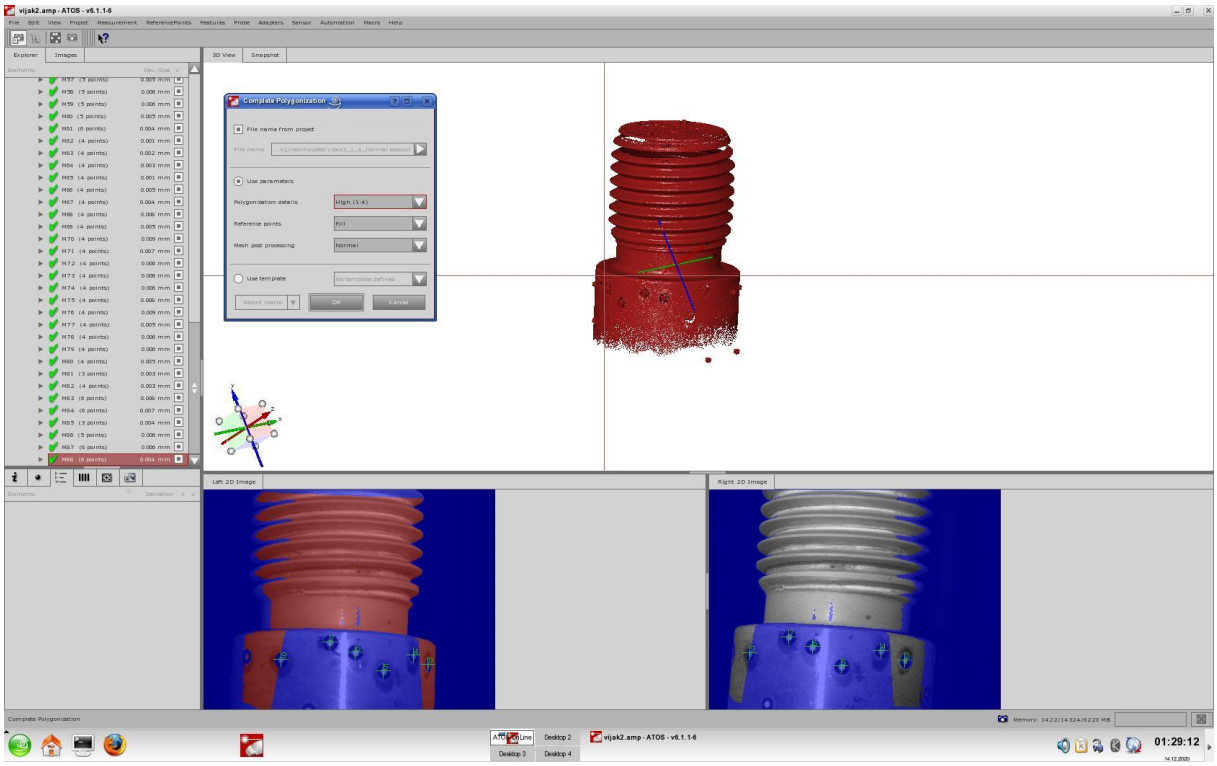
potpunosti skeniramo dani predmet. Slika 3-2 daje detaljan uvid u dio skeniranog vijka zajedno sa brojem obavljenih skeniranja, brojem referentnih točaka koji je obuhvaćen svakim skeniranjem te slikom jedne i druge kamere u trenutku skeniranja.



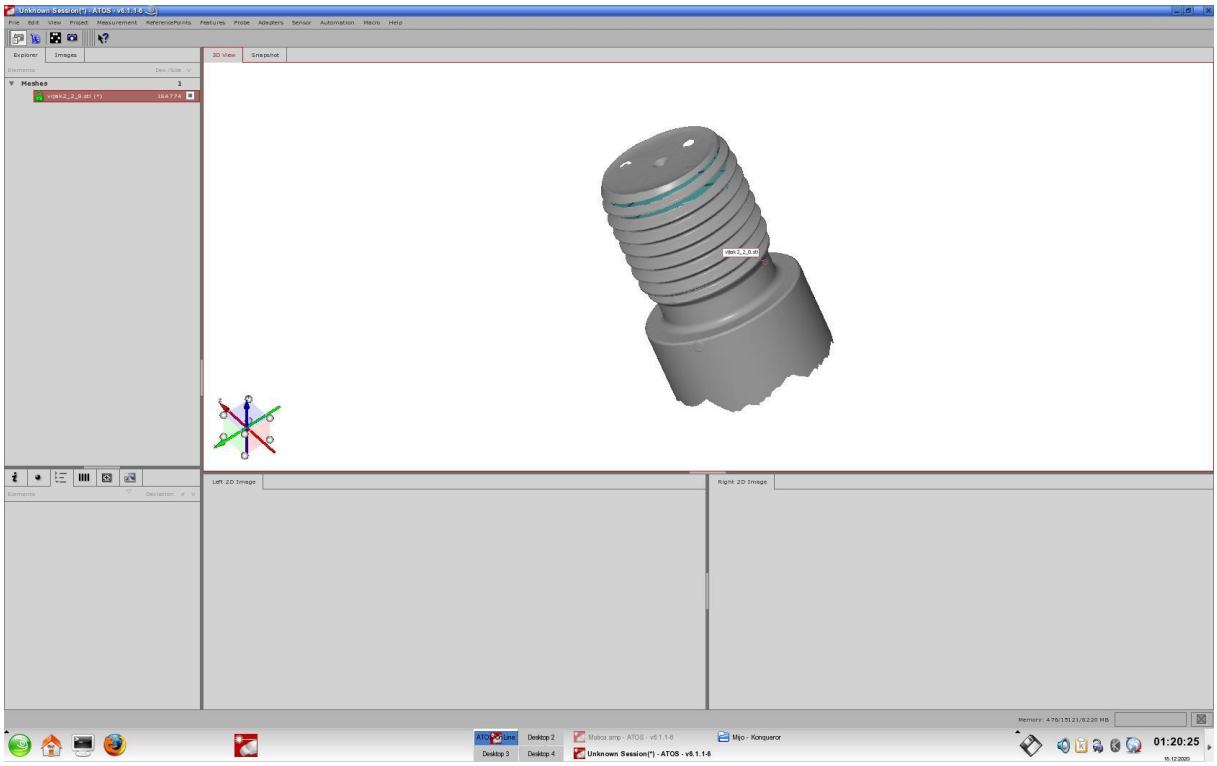
Slika 3-2 Primjer skeniranog predmeta

3.3 Proces poligonizacije

Proces poligonizacije je treći i ujedno posljednji korak skeniranja predmeta. Kada se predmet skenira, svaki sken se sastoji od oblaka točaka tj. predmet se skenira te se informacije o poziciji točaka pohranjuje o obliku STL formata kojeg ćemo obraditi u idućem poglavlju. Ono što je bitno za poligonizaciju jest da svaki sken sadrži mala odstupanja te se takve greške kroz poligonizaciju uklanjaju. Nakon završenog proces poligonizacije, dobijemo uredno skeniran predmet. Slika 3-3 prikazuje početak procesa poligonizacije, a Slika 3-4 prikazuje gotov proces poligonizacije tj. predmet spreman za daljnju obradu.



Slika 3-3 Početak proces poligonizacije



Slika 3-4 Predmet nakon poligonizacije

4 STL FORMAT DATOTEKE

STL (Standard Triangle Language ili STereoLithography) je format datoteke kojim opisujemo površinu u skeniranog predmeta u tri dimenzije. STL datoteke reproduciraju 3D geometriju objekta pohranjujući određeni broj aspekata ili 3D trokuta u složeni digitalni model.[2] STL format se sastoji od trokutastih poligona tj. trokuta gdje se svaki trokut sastoji od tri vrha sa svojim trodimenzionalnim koordinatama te svaki taj trokutasti polinom ima vektor normale koji pokazuje na vanjsku stranu pojedinog poligona. Razlikujemo dva formata STL datoteke a to su ASCII i binarni formirat koje ćemo ukratko objasniti koji kao jedinu razliku imaju veličinu samog formata. ASCII format datoteke je napisan u 'plain text-u' tj. tekstu razumljivom nama ljudima, a kao format za uštedu memorije se koristi binarni format.

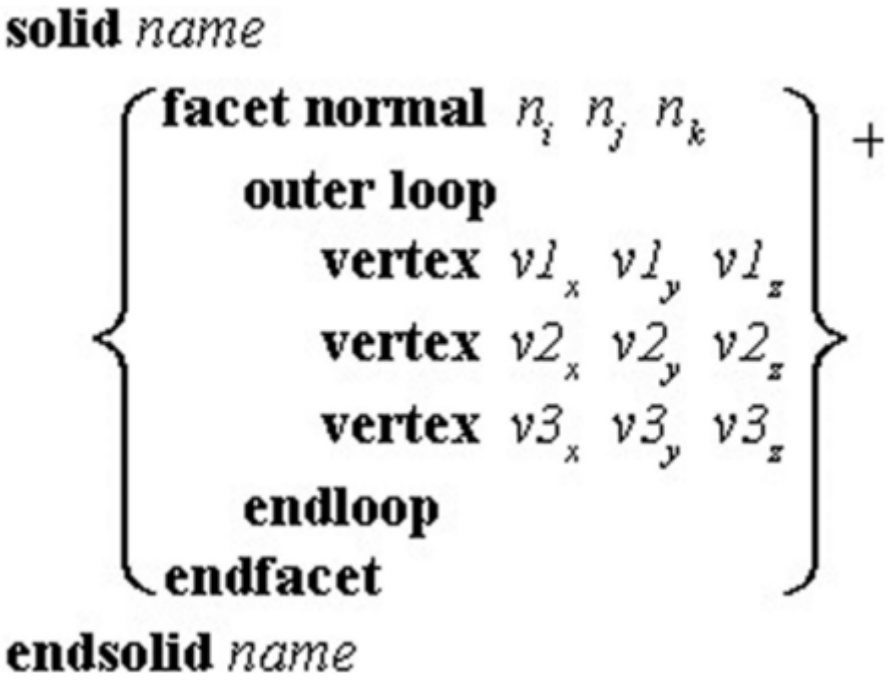
Glavna ideja STL formata jest prikazati svaku površinu preko niza trokuta, a taj se postupak naziva tessellacija. Tessellacija nije nužno popunjavanje neke površine trokutima, već popunjavanje neke površine određenim brojem jednostavnih geometrijskih oblika na način da između njih ne postoje praznine.

I ASCII i binarni format pohranjuju iste podatke a to su:

- Koordinate 3 vrha trokuta sa njihovim trodimenzionalnim koordinatama
- Vektor normale

Svaki vrh trokuta sadrži x, y i z prostornu koordinatu kojim jedinstveno opisuje svako točku u prostoru, dok je normalni vektor je koji pokazuje na vanjsku stranu tog trokuta te je također i okomit na taj trokut. Slika 4-1 pokazuje izgled strukture u programskom jeziku Pythonu ASCII formata STL datoteke.

[2] 'A 3D Visualization Algorithm with STL File Parser'.



Slika 4-1 Izgled formata ASCII STL-a

5 NEDOSTATCI PRILIKOM 3D SKENIRANJA

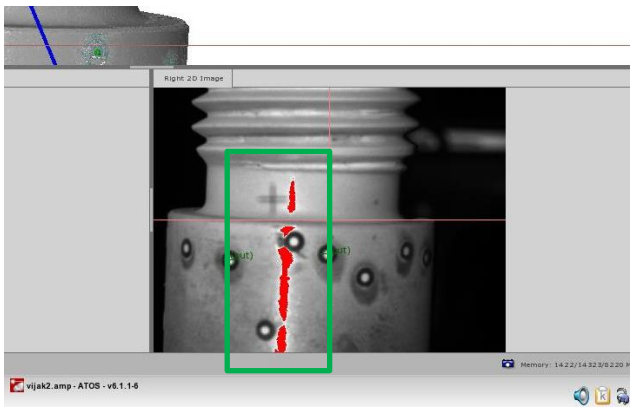
Kao i sve ostalo, postoje neke poteškoće i nedostatci koji nas ograničavaju prilikom skeniranja i ispisa 3D tijela. Očigledne razloge kao što je veličina u ovome poglavlju nećemo obrađivati, već ćemo se usredotočiti na neka koji su stvarali probleme prilikom izrade ovog rada te će jedan od ovih problema biti glavna zadaća ovog rada. Neki od problema s kojima smo se susreli su:

1. Refleksija predmeta
2. Nemogućnost preciznog skeniranja svakog ruba predmeta
3. Šupljine koje se pojavljuju kao rezultat nemogućnosti potpunog skeniranja

Prve dvije točke imaju jednostavna rješenja dok ćemo treću točku obraditi u šestom poglavlju gdje ćemo šupljine pronaći te ih gotovim softverskim rješenjima 'zakrpati'.

5.1 Refleksija predmeta

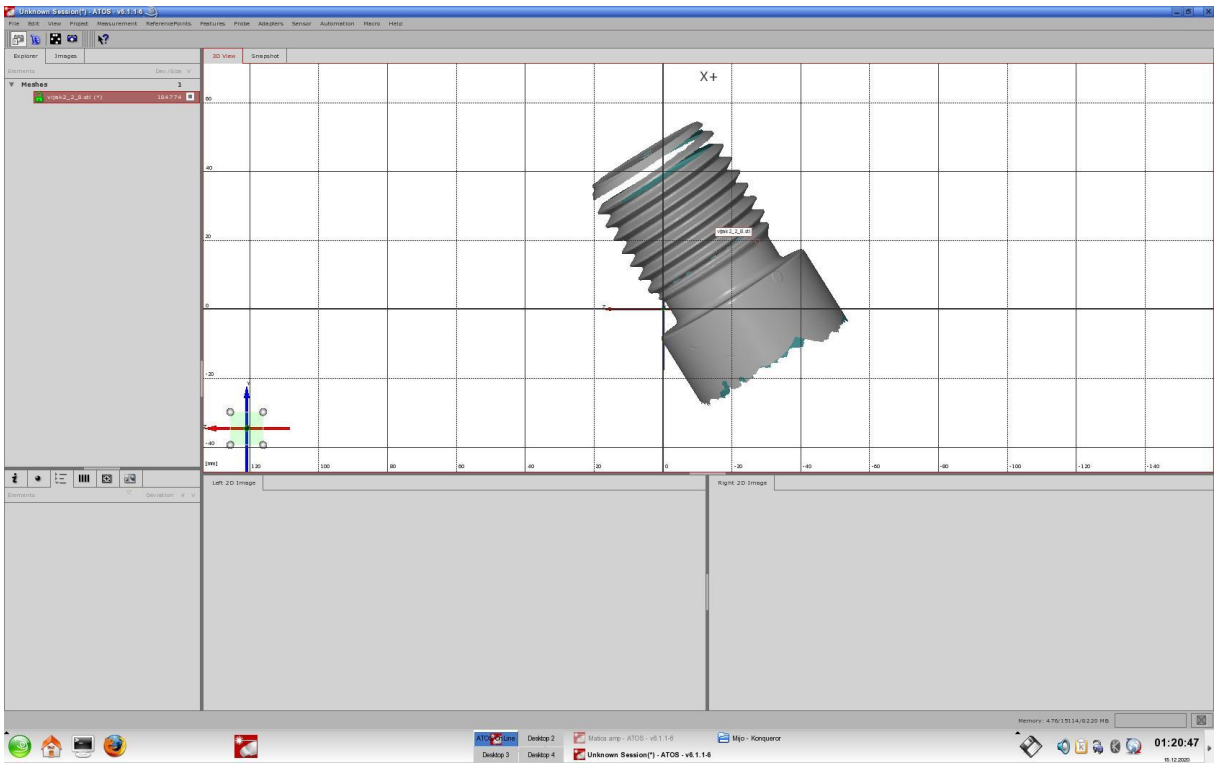
Ukratko smo se dotakli ovog problema u prethodnim poglavljima, ali situacija je slijedeća; svako tijelo zbog vrste svog materijala posjeduje određen stupanj refleksije zbog kojeg nije moguće skenirati dani predmet. Kako bismo taj problem riješili, na predmet nanosimo sprej protiv refleksije te tada reflektor koji obasjava predmet strukturiranim svjetlom može precizno očitati svaki rub predmeta. Slika 5-1 pokazuje crvene tragove koji označavaju veliku refleksiju koju predmet emitira.



Slika 5-1 Tragovi refleksije na metalnom vijku (crveno)

5.2 Nemogućnost preciznog skeniranja svakog ruba predmeta

Koristeći dani skener nije bilo moguće precizno skenirati vijak tj. njegove vršne navoje. Kako god predmet namjestili, nije bilo moguće u istom trenutku uhvatiti tri referentne točke i navoje koje je bilo potrebno skenirati. Kao rezultat javljaju se šupljine u skenu predmeta koje kasnije trebaju na dodatnu obradu. Slika 5-2 pokazuje jasno kako nemogućnost potpunog skeniranja ostavlja šupljene u skeniranoj STL datoteci.



Slika 5-2 Šupljine u navojima vijka

6 OBRADA 3D STL DATOTEKE I PRIPREMA ZA ISPIS

Sada kada imamo skenirani predmet, potrebno je predmet obraditi kako bismo dobili zadovoljavajuće rezultate ispisa. Obradu STL datoteke ćemo izvršiti kroz tri koraka:

1. Vizualni prikaz skeniranog predmeta
2. Izrada softvera za detekciju šupljina
3. Krpanje šupljina koristeći gotova softverska rješenja

Kroz ova tri koraka ćemo imati STL datoteku koja će biti spreman za ispis. Za obradu, ispis i vizualizaciju našeg predmeta, koristit ćemo Python programski jezik zbog njegove jednostavnosti te već gotovih biblioteka koje će nam znatno olakšati rad.

6.1 Vizualni prikaz skeniranog predmeta

Kako bismo bili sigurni da je predmet ispravno skeniran, napisat ćemo kratku Python skriptu kojom ćemo plotirati tj. nacrtati skenirani predmet, u našem slučaju to je metalni vijak. Za vizualizaciju uključit ćemo ukupno 3 biblioteke; „STL“, „MPL_TOOLKITS“ i „MATPLOTLIB“ te neke njihove module kojih ćemo se ukratko dotaknuti kako bismo razumjeli njihovu funkciju i zadaću. Biblioteka „STL“ se koristi za rad sa STL datotekama, a njene klasa „mesh“ se koristi za učitavanje, modificiranje te izvoz STL datoteka. Zatim, „MPL_TOOLKITS“ i modul „mplot3d“ koristimo za vizualizaciju podataka preko 3D ili drugih grafova, a za naše potrebe upravo će nam biti potreban 3D graf. „MATPLOTLIB“ i modul „pyplot“ ćemo koristit za jednostavno i brzo kreiranje grafika.

```
from stl import mesh
from mpl_toolkits import mplot3d
from matplotlib import pyplot
```

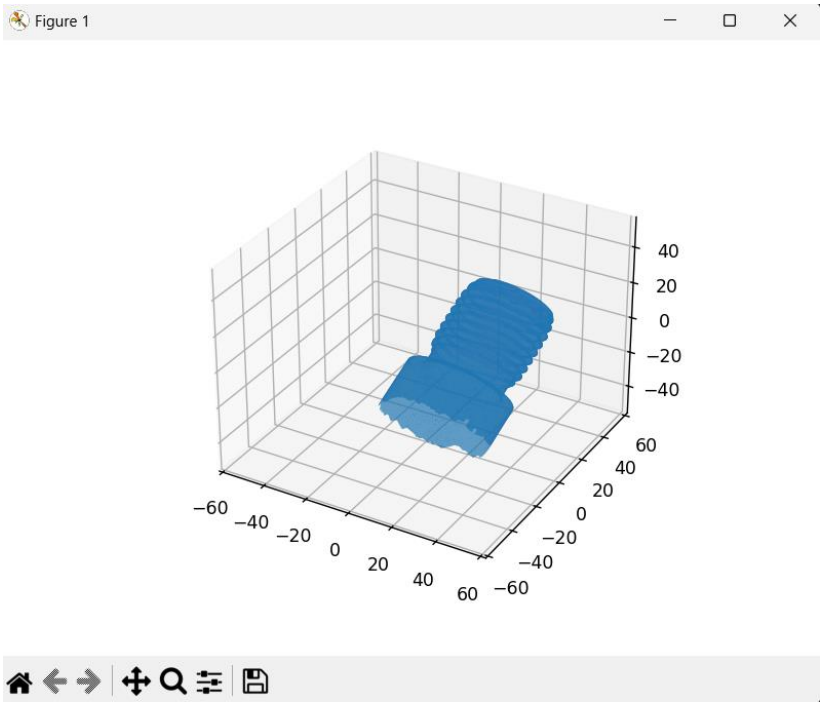
Slika 6-1 Korištene biblioteke za vizualizaciju

Nakon uključenih biblioteka možemo kreirati prvo novi plot u kojega ćemo grafički nacrtati naš vijak. Prvo, kreiramo novi plot korištenjem pyplot.figure() funkcije te mu dodajemo x, y i z os sa funkcijom figure.add_subplot(projection='3d'). Nakon toga kreiramo varijabli

vijak_mesh kojoj u koju ćemo pohraniti mesh iz datoteke vijak2_2_8_zakrpano.stl koju smo, radi urednosti, pohranili u direktorij STL files. Također ćemo u ovome koraku dodati i vektore u plot. Nakon toga, radimo skaliranje kako bismo čitav predmet uspjeli prikazati u njegovoj normalnoj veličini te na kraju funkcijom pyplot.show() prikazujemo čitav plot korisniku. Slika 6-2 daje uvid u čitav korišten kod, dok Slika 6-3 daje rezultat izvršavanja koda odnosno grafički prikazan vijak korištenjem Pythona.

```
5   # Kreiranje novog plot-a
6   figure = pyplot.figure()
7   axes = figure.add_subplot(projection='3d')
8
9   # Učitavanje STL filea i dodavanje vektora u plot
10  vijak_mesh = mesh.Mesh.from_file('STL files/vijak2_2_8_zakrpano.stl')
11  axes.add_collection3d(mplot3d.art3d.Poly3DCollection(vijak_mesh.vectors))
12
13  # Skalranje predmeta sa osima
14  scale = vijak_mesh.points.flatten()
15  axes.auto_scale_xyz(scale, scale, scale)
16
17  # Prikaz plota na ekran
18  pyplot.show()
```

Slika 6-2 Kôd za grafičku vizualizaciju vijka



Slika 6-3 Grafički prikazan vijak korištenjem Python skripte

6.2 Izrada softvera za detekciju šupljina

U ovom poglavlju ćemo obraditi detekciju šupljina tj. praznih prostora koji su ostali prilikom skeniranja, što je učestalost kod složenijih predmeta. Kada smo skenirali vijak, skenirali smo samo njegov vrh sa navojima, što znači da neke donje bridove nismo skenirali jer se radi samo o ravnim površinama koje nam trenutno nisu interesantne. Prilikom skeniranja su se potkrale neke manje i veće šupljine. Napisat ćemo Python skriptu kojom ćemo iterirati po svim rubovima te na taj način otkriti gdje se nalaze šupljine.

Logika čitave detekcije je slijedeća; prolazimo sve rubove učitane STL datoteke i brojimo koliko se puta pojavljuje pojedini brid, ako se brid pojavio dva puta, to znači da se taj brid nalazi između dva trokuta, a ako se pojavljuje jednom, to znači da je on rubni brid tj. nakon njega se nalazi šupljina.

Počnimo od uključenih biblioteka, ukupno uključujemo 4 biblioteke, „NUMPY“ za rad sa matricama i složenijim matematičkim operacijama, „STL“ za rad sa STL datotekama, „MATPLOTLIB.PYPLOT“ za crtanje grafova te „MPL_TOOLKITS.MPLOT3D.ART3D“ za manipuliranje grupama poligona u 3D grafu. Slika 6-4 pokazuje sve biblioteke te korištene module.

```
1   import numpy as np
2   from stl import mesh
3   import matplotlib.pyplot as plt
4   from mpl_toolkits.mplot3d.art3d import Poly3DCollection
```

Slika 6-4 Korištene biblioteke i moduli za detekciju šupljina

Nakon toga, učitavamo STL datoteku našeg vijka kojeg ćemo obrađivati, tj. obaviti detekciju svih šupljina. Definiramo varijablu file_path kojoj ćemo dati destinaciju naše STL datoteke. Ovdje ćemo koristiti originalno skenirani vijak sa svim manjim šupljinama kako bi demonstrirali da softver radi i za manje šupljine. Naziv datoteke je vijak2_2_8.stl. Zatim, učitamo mesheve iz čitave datoteke i pohranimo ih u varijablu vijak_mesh. Na Slika 6-5 možemo vidjeti korištene naredbe.

```
6   # Učitavanje STL datoteke
7   file_path = 'STL files/vijak2_2_8.stl'
8   vijak_mesh = mesh.Mesh.from_file(file_path)
```

Slika 6-5 Naredbe za učitavanje STL datoteke

Nakon toga, pohranimo sve vektore iz STL datoteke u varijablu triangles, koju ćemo proslijediti kao argument funkciji find_holes(). U toj funkciji ćemo obaviti traženje svih rubnih rubova tj. rubova koji se nalaze u samo jednom trokutu.

Prvo u toj funkciji definiramo rječnik edges u kojeg ćemo pohraniti sve rubove naših trokuta. Zatim, kroz dvostruku for petlju ćemo iterirati po svim trokutima te po svim njihovim vrhovima i sortirane ih pohraniti u varijablu edge. Ovdje moramo paziti na jednu stvar, naime, svaki trokut je definiran sa tri vrha i ti su vrhovi označeni npr. vrh[0], vrh[1] i vrh[2]. U unutarnjoj for petlji imamo iterator i koji se kreće od 0 do 3 te u drugom izrazu tj. drugi vrh tog trokuta ako bismo ostavili samo triangle(i + 1) faktor i + 1 bi narastao na 4 što nije ispravan vrh. Iz tog razloga obavljamo cjelobrojno dijeljenje sa 3 kako bismo uvijek ostali unutar istog trokuta. Nakon toga, provjeravamo da li se taj naš definirani rub nalazi u rječniku rubova te ako se nalazi njegovu broj puta ponavljanja povećamo za 1, a ako se ne nalazi tada ga dodajemo u rječnik. To ponavljamo za sve rubove te na kraju funkcije vraćamo kao povratnu vrijednost funkcije sve rubne rubove tj. rubove koji se javljaju samo jednom. Slika 6-6 daje kompletan kôd funkcije find_holes().

```
13  # Funkcija za detekciju šupljina
14  def find_holes(triangles):
15      # Pronalazak svih rubova sa jednim "rubom"
16      edges = {}
17      for triangle in triangles:
18          for i in range(3):
19              edge = tuple(sorted((tuple(triangle[i]), tuple(triangle[(i + 1) % 3]))))
20              if edge in edges:
21                  edges[edge] += 1
22              else:
23                  edges[edge] = 1
24
25      # Pronalaženje rubova koje se pojavljuju samo jednom, tj. šupljina
26      bound_edges = [edge for edge, count in edges.items() if count == 1]
27
28      return bound_edges
```

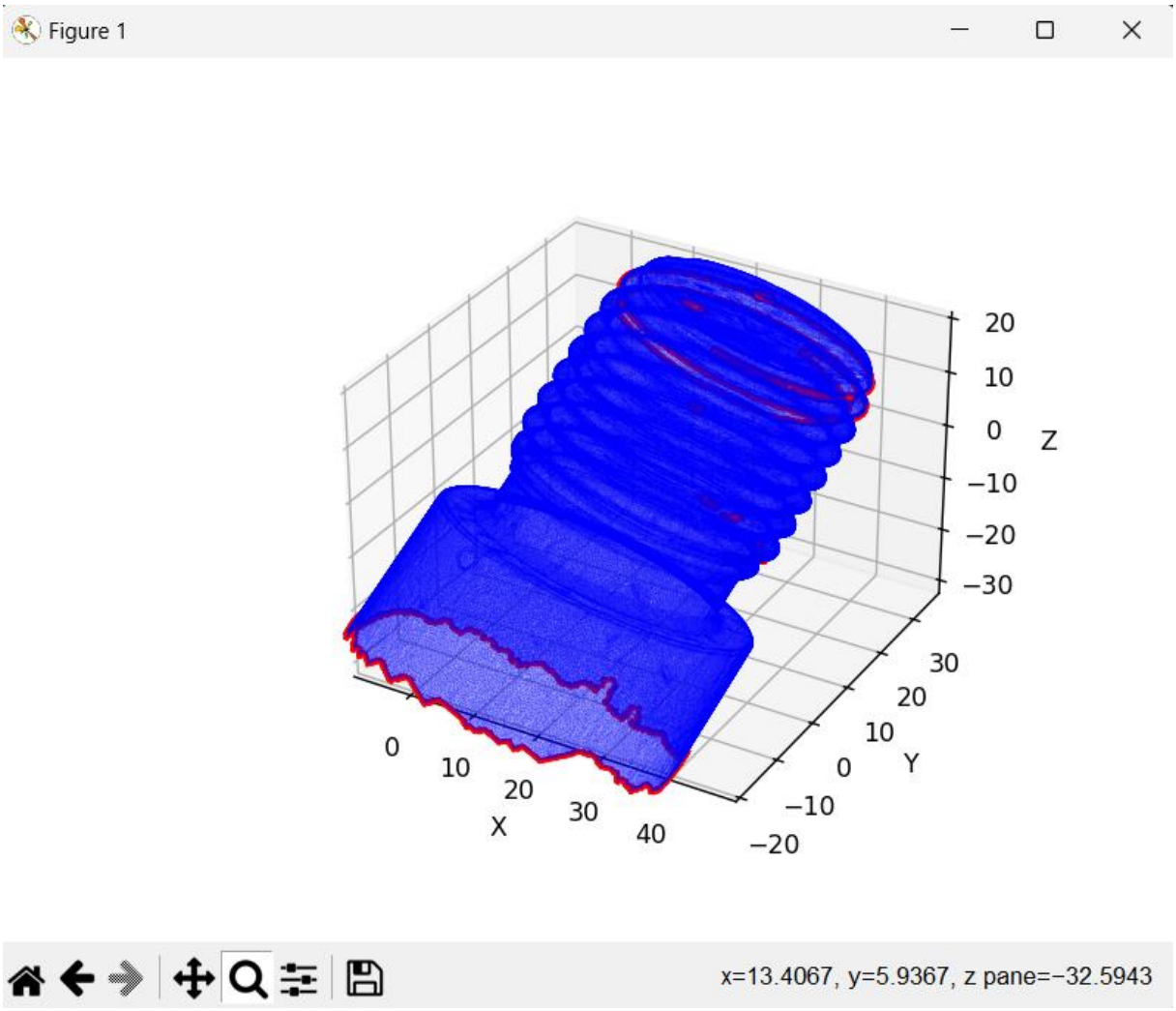
Slika 6-6 Kôd funkcije za pronalazak šupljina

Nakon pronalaska svih šupljina tj. njezinih rubnih rubova, potrebno je napisati funkciju u kojoj ćemo nacrtati naš originalan vijak te sa crvenim rubovima obojati sve rubne rubove. To ćemo obaviti kroz funkciju plot_screw_and_holes() kojoj ćemo proslijediti dva argumenta, sve trokute iz STL datoteke te šupljine koje smo pohranili kao rezultat funkcije find_holes(). Crtanje obavljamo slično kao u paragrafu 6.1, samo što ovdje imamo dva tipa podataka koje ćemo prikazati u drugoj boji, trokute originalne STL datoteke u plavoj boji i rubne rubove u crvenoj boji.

Prvi korak jest definiranje nove figure te definiranje njenog obilježja kao 3D grafa. Nakon toga, crtamo sve trokute kojemu definiramo slijedeće parametre:

- Boja ispune: plava
- Prozirnost trokuta (alpha): 0.3, kako bismo vidjeli „kroz“ vijak
- Širinu ruba: 0.2
- Boja ruba: plava

Zatim obavljamo crtanje rubnih rubova u crvenoj boji i veće širine ruba kako bi se vidjela razlika, na način da iteriramo kroz sve rubne rubove te ih „podebljavamo“ u crvenu boju jednog po jednog. Nakon toga, postavimo trodimenzionalne koordinatne osi na vrijednosti X, Y i Z i skaliramo graf kako bismo mogli prikazati cijeli vijak unutar istog. Naposljetku, pozovemo funkciju plot_screw_and_holes() sa argumentima triangles (trokutima STL datoteke) i holes (rubnim rubovima šupljina). Rezultat crtanja vidimo na Slika 6-7, a kôd funkcije za crtanje je dan na slici.



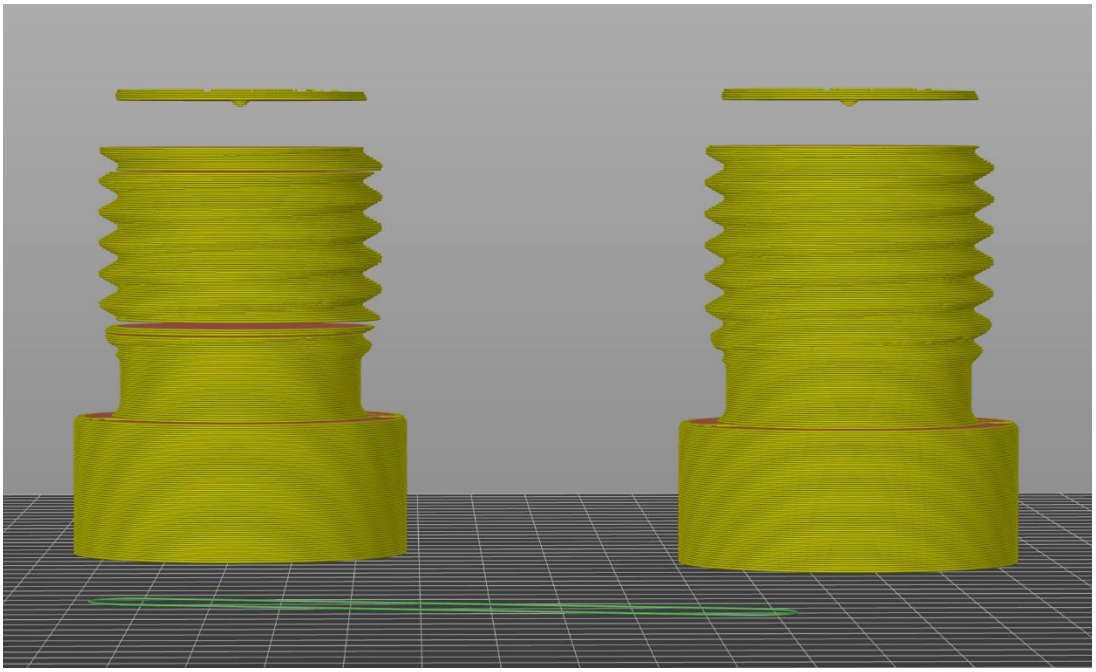
Slika 6-7 Vijak (plavo) sa podebljanim šupljinama (crveno)

```python
# Funkcija za crtanje STL modela i šupljina
def plot_screw_and_holes(triangles, holes):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Crtanje svih trokuta u plavoj boji tako da vijak bude plave boje
    ax.add_collection3d(Poly3DCollection(triangles, color='blue', alpha=0.3, linewidths=0.2, edgecolor='b'))

    # Crtanje šupljina tj. rubnih rubova sa crvenim rubovima kako bi vidjeli razliku
    for edge in holes:
        edge = np.array(edge)
        ax.plot(edge[:, 0], edge[:, 1], edge[:, 2], color='red', linewidth=2)

    (variable) ax: Any  natnih osiju
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    # Skaliranje
    scale = triangles.flatten(order='C')
    ax.auto_scale_xyz(scale, scale, scale)

    plt.show()

# Prikazivanje modela sa vijka sa šupljinama
plot_screw_and_holes(triangles, holes)
```

Slika 6-8 Kôd funkcije plot_screw_and_holes()

6.3 Krpanje šupljina koristeći gotova softverska rješenja

U okviru ovog rada nećemo se baviti na koji način krpati šupljine u 3D skenu, već ćemo za to koristiti gotova softverska rješenja u vidu softvera GOM Inspect. GOM Inspect je softver koji omogućava obradu STL datoteka, a neke od njegovih funkcija su: zaglađivanje nepravilnosti nastalih prilikom skeniranja, uklanjanje ili dodavanje novih poligona u STL datoteku te krpanje šupljina koje su nastale prilikom skeniranja. Slika 6-9 daje uvid u metalni vijak nakon krpanja manjih šupljina koje softver obavlja automatski detekcijom šupljine te njezinim krpanjem.



Slika 6-9 Vijak nakon krpanja manjih šupljina

Na kraju nam ostaju veće šupljine koje je potrebno zakrpati ručno. Naime, softver nije dovoljno razvijen da prepozna o kojem se obliku ili predmetu radi te samim time veće šupljine ne može zakrpati. Kako bismo to riješili, ručno dodajemo poligone kojima ćemo smanjiti veličinu same šupljine. Nakon toga, kao što je prethodno opisano, softver će automatski prepoznati šupljinu kada na nju kliknemo te će je moći precizno zakrpati.

7 ZAKLJUČAK

Ovaj završni rad pruža uvid u osnove 3D skeniranja, objašnjava što je to STL datoteka i daje uvid u manipulaciju istih te kratku usporedbu između skeniranog predmeta i na kraju dobivenog rezultata. 3D skeniranje i ispis je jedna od najvećih rastućih industrija i tehnologija današnjice te njihove mogućnosti, tehnike i preciznost raste iz dana u dan.

Pokazano je kako jednostavnim skriptama možemo obaviti detekciju šupljine, što može imati raznih primjena u bilo kojoj industriji, kako za manje tako i za veće projekte. Također, pokazano je kako se kroz Python može jednostavno manipulirati i grafički prikazati bilo kakva STL datoteka.

STL datoteke imaju široku upotrebu ne samo u industriji 3d skeniranja i ispisa, već i u računalnoj grafici. Njihovo razumijevanje omogućava upotrebu u velikom broju područja znanosti, informatike i informacijskih tehnologija, arhitekturi, medicini i brojnim drugim područjima.

Iako su detaljno opisani određeni procesi, tehnike i alati za 3D skeniranje i ispis, postoji mnoštvo drugih tema više razine koje se mogu obrađivati unutar ove teme. Daljnjim razvojem ovdje navedenih alata za detekciju šupljina može se postići da isti softver obavlja uređivanje STL datoteke te krpanje šupljina.

LITERATURA

[1] Pribanić, Tomislav, Ivan Cifrić, Mario Cifrek, Goran Goldberger, and Stanislav Peharec. 'Pojednostavljenje Određivanja Položaja Ravnine Svjetla Za Vrijeme Skeniranja Strukturiranim Svjetlom'. *Automatika : Časopis Za Automatiku, Mjerenje, Elektroniku, Računarstvo i Komunikacije* 47, no. 3–4 (6 December 2006): 141–47.

[2] 'A 3D Visualization Algorithm with STL File Parser', n.d. https://www.researchgate.net/profile/Rajani-Kamath/publication/270337821_A_3D_Visualization_Algorithm_with_STL_File_Parser/links/54a7b5920cf256bf8bb6d2d7/A-3D-Visualization-Algorithm-with-STL-File-Parser.

POPIS I OZNAKA KRATICA

3D – Three dimensiomal

SL – Structure Light

ATOS – Advanced Topometric Sensor

GOM – Gesellschaft für Optische Messtechnik

STL – Standard Triangle Language ili Standard Tessellation Language

ASCII – American Standard Code for Information Interchange

PRILOZI

Kazalo slika, tablica i kodova

Kazalo slika

Slika 2-1 ATOS sučelje 2
Slika 3-1 Primjer predmeta pripremljenog za skeniranje 5
Slika 3-2 Primjer skeniranog predmeta 6
Slika 3-3 Početak proces poligonizacije 7
Slika 3-4 Predmet nakon poligonizacije 7
Slika 4-1 Izgled formata ASCII STL-a 9
Slika 5-1 Tragovi refleksije na metalnom vijku (crveno) 10
Slika 5-2 Šupljine u navojima vijka 11
Slika 6-1 Korištene biblioteke za vizualizaciju 12
Slika 6-2 Kôd za grafičku vizualizaciju vijka 13
Slika 6-3 Grafički prikazan vijak korištenjem Python skripte 13
Slika 6-4 Korištene biblioteke i moduli za detekciju šupljina 14
Slika 6-5 Naredbe za učitavanje STL datoteke 15
Slika 6-6 Kôd funkcije za pronalazak šupljina 15
Slika 6-7 Vijak (plavo) sa podebljanim šupljinama (crveno) 17
Slika 6-8 Kôd funkcije plot_screw_and_holes() 17
Slika 6-9 Vijak nakon krpanja manjih šupljina 18

SAŽETAK I KLJUČNE RIJEČI

3D skeniranjem moguće je obaviti skeniranje bilo kojeg predmeta koristeći skener sa strukturnim svjetlom i dvije kamere. Na taj način je moguće skenirani predmet pohraniti u binarni ili ASCII STL datoteku koju daljnjim procesima možemo dalje obrađivati i pripremiti za ispis.

Temeljni dio ovog završnog rada jest izrada softvera za detekciju šupljina gdje smo koristeći programski jezik Python iskoristili svojstvo zatvorenosti STL datoteke za pronalazak rubnih rubova tj. rubova koji okružuju šupljinu. Ovakav softver je samo dio već izrazito naprednih softverskih rješenja koje nudi ATOS tehnologija sa GOM softverima.

KLJUČNE RIJEČI: 3D skeniranje, softver, Python, STL

SUMMARY AND KEYWORDS

With 3D scanning, it is possible to scan any object using scanner with structure light and two cameras. That way, it is possible to store scanned item into binary or ASCII STL file which uwnig further processes we can process and prepare file for print.

The fundamental part of this final work is the creation of cavity detection software where, using the Python programming language, we used the closed property of the STL file to find the edge edges, i.e. the edges surrounding the cavity. This kind of software is only a part of already extremely advanced software solutions that offer ATOS technology with GOM software.

KEYWORDS: 3D scanning, software, Python, STL