API otvorenog koda za proceduralno 3D modeliranje

Šimić, Luka

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij računarstva

API otvorenog koda za proceduralno 3D modeliranje

Diplomski rad

Luka Šimić

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 29.06.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Luka Šimić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1092R, 06.10.2019.
OIB studenta:	52147607177
Mentor:	Izv. prof. dr. sc. Irena Galić
Sumentor:	Dr. sc. Hrvoje Leventić
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	Izv. prof. dr. sc. Irena Galić
Član Povjerenstva 2:	Marija Habijan
Naslov diplomskog rada:	API otvorenog koda za proceduralno 3D modeliranje
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Zadatak diplomskog rada je istražiti i opisati postojeća dostupna programska sučelja (API) za proceduralno 3D modeliranje. Opisati njihove prednosti i nedostatke te primjere korištenja (koja im je primjena i gdje se koriste). Za praktični dio zadatak je implementirati API za 3D modeliranje, te prateću biblioteku za rad s matematičkim operacijama (matrice, vektori) koristeći programski jezik C++. API bi svojim funkcionalnostima bio relativno sličan Bmesh API-ju unutar Blendera, ali s dodatnim fokusom na funkcije koje su potrebne za proceduralni pristup 3D modeliranju, prvenstveno mogućnosti za filtriranje i selekciju elemenata modela za koje nije potrebna interakcija korisnika, te fokus na parametrizirani pristup svim operacijama
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	29.06.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 21.07.2021.

Ime i prezime studenta:	Luka Šimić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1092R, 06.10.2019.
Turnitin podudaranje [%]:	2%

Ovom izjavom izjavljujem da je rad pod nazivom**: API otvorenog koda za proceduralno 3D modeliranje**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Irena Galić

i sumentora Dr. sc. Hrvoje Leventić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD

Proceduralno 3D modeliranje je pojam koji obuhvaća brojne metode za generiranje 3D modela uz ograničeni korisnički unos podataka ili bez ikakve korisničke interakcije. U ovom radu razmatra se proceduralno 3D modeliranje zasnovano na poligonalnim mrežama (*engl. polygon mesh*).

Proceduralni pristup ima nekoliko prednosti u usporedbi s konvencionalnim tehnikama 3D modeliranja. Primarno, to je mogućnost kreiranja različitih varijacija modela koji su generirani koristeći ista pravila, ali uz značajne vizualne razlike između varijacija modela. Zatim, spremanje modela kao niz operatora visoke razine može pružiti smanjenje u veličini datoteke[1] u usporedbi s uobičajenim pristupom, gdje se modeli uglavnom spremaju kao liste točaka (*engl. vertex*), rubova (*engl. edge*) i poligona (*engl. polygon, face*). Zbog velikog rasta industrije video igara i vizualnih efekata (*engl. VFX*), te zbog pojave novih područja kao što su simulacija okoline i generiranje sintetičkih skupova podataka za strojno učenje, neki oblici proceduralnog generiranja moraju se koristiti kako bi se izbjegao linearni porast potrebnih ljudskih resursa[2] koji je prisutan koristeći isključivo konvencionalne metode 3D modeliranja.

U prvom poglavlju dan je uvod u diplomski rad. Drugo poglavlje daje definiciju proceduralnog 3D modeliranja, kratak pregled trenutno dostupnih i često korištenih metoda i načina rada (*engl. workflow*) za proceduralno 3D modeliranje. Zatim slijedi pregled i usporedba trenutno dostupnih, popularnih rješenja, biblioteka i programskih paketa za rad s 3D modelima i propituje se njihova prikladnost za uporabu u kontekstu proceduralnog 3D modeliranja. Treće poglavlje u glavnim crtama definira zahtjeve za programska sučelja fokusirana na proceduralno 3D modeliranje koristeći poligonalne mreže. Četvrto poglavlje detaljno opisuje implementaciju programskog sučelja i pruža detaljna objašnjenja strukture projekta, strukture podataka i implementiranih operatora, funkcija i klasa. Zatim je prezentirano nekoliko primjera korištenja koji demonstriraju sposobnosti implementiranog programskog sučelja. Na kraju se razmatraju moguće primjene i mogućnosti daljnjeg razvoja programa.

Glavni doprinos ovog rada je razvoj AobaAPI - nova biblioteka (*engl. library, mdoeling kernel*) i pripadajuće aplikacijsko programsko sučelje (*engl. Application Programming Interface, API*) za 3D modeliranje zasnovano na poligonalnim mrežama. Biblioteka pruža operatore visoke razine i set operatora niske razine (*engl. euler operators*) koji se mogu koristiti za razvoj vlastitih funkcija. Uz to, implementirano je nekoliko primjera različite razine kompleksnosti koji demonstriraju korištenje biblioteke. Biblioteka i njen izvorni kod, kao i primjeri korištenja su javno dostupni i objavljeni pod MIT licencom otvorenog koda.

2. PROCEDURALNO 3D MODELIRANJE

2.1. Techike proceduralnog 3D modeliranja

Proceduralno generiranje je širok pojam koji obuhvaća razne tehnike, algoritme i programe koji generiraju 3D model, teksturu, sliku ili vizualni efekt [3]. Proceduralno 3D modeliranje je dio proceduralnog generiranje koji je fokusiran na generiranje 3D modela. Programi za proceduralno generiranje generiraju modele na osnovu određenog pravila ili niza pravila koji mogu biti definirani putem koda, skripti, grafova, matrica, gramatika ili na druge načine. Glavne karakteristike proceduralnog generiranja su apstrakcija kompleksnih detalja u funkcije (procedure), parametrizirana kontrola i fleksibilnost. Proceduralni pristupi često pružaju značajno smanjenje u memorijskoj kompleksnosti prilikom pohrane podataka, jer je potrebno spremiti samo pravila za generiranje i parametre, a ne rezultirajući 3D model. Parametrizirana kontrola može se koristiti za brzo stvaranje različitih iteracija modela varijacijom ulaznih parametara, bez promjene pravila za generiranje. Proceduralni pristup je fleksibilan i omogućava nedestruktivne promjene svih dijelova i pravila generiranja.

Metode proceduralnog modeliranja mogu se podijeliti u nekoliko kategorija ovisno o pristupu generiranju 3D modela i načinu na koji su predstavljena i pohranjena pravila generiranja. U praksi se često koristi kombinacija različitih pristupa. Neki od pristupa su:

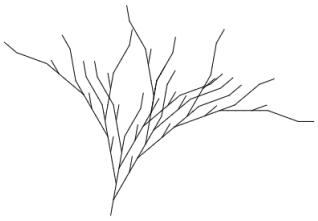
- Proceduralno modeliranje zasnovano na gramatikama
- Proceduralno modeliranje zasnovano na šumu
- Proceduralno modeliranje koristeći implicitno predstavljene plohe
- Proceduralno modeliranje zasnovano na poligonalnim mrežama

2.1.1 Proceduralmo modeliranje zasnovano na gramatikama

Proceduralno modeliranje zasnovano na gramatikama (*engl. grammar based procedural modeling*) razvijeni su iz koncepata formalnih gramatika i teorije jezika [4]. Nizovi znakova (*engl. string*) se generiraju iz pravila. Pravilo definira način na koji se ne-završni znakovi pretvaraju u završne znakove. Pravilo se primjenjuje sve dok postoje ne-završni simboli.

L-sustav (*engl. L-system, Lindenmayer system*) [5] je algoritam za generiranje nizova znakova koji primjenjuje pravilo paralelno na sve ne-završne simbole i može se zaustaviti u bilo kojem koraku, bez obzira postoje li ili ne još ne-završnih simbola. L-sustavi se koriste za modeliranje raznih organskih struktura, a u računalnoj grafici se često koriste za modeliranje bilja i drveća [6]. Primjer strukture generirane koristeći L-sustav dan je na slici 2.1.

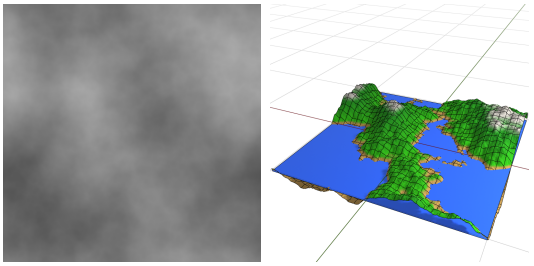
Shape Grammar je sustav koji radi s nizom osnovnih 2D i 3D objekata (linije, pravokutnici, kocke...) umjesto znakova. [7]. *Shape grammar* je kompleksan jer se svaki osnovni element može rastaviti na dodatne (npr. pravokutnik se može rastaviti na linije) i daje najbolje rezultate kada je ograničen na specifične probleme s ograničenim brojem osnovnih objekata.



Sl. 2.1: *Jednostavna struktura generirana koristeći L-sustav*

2.1.2 Proceduralno modeliranje koristeći šum

Šum (*eng. noise*), preciznije gradijentni šum često se koristi u računalnoj grafici. Većina algoritama za generiranje šuma zasnovani su na mreži (rešetka, *engl. lattice*) gdje se vrijednost za sredinu svake ćelije određuje nasumično, a ostale vrijednosti dobivene su interpolacijom. Šum se često koristi za kreiranje *displacement/height* mapa za definiranje površinskih detalja ili generiranje terena, kao na slici 2.2. Funkcije za generiranje šuma mogu se proširiti na tri ili četiri dimenzije i u tom slučaju omogućuju kreiranje nasumičnih volumena, koji se mogu pretvoriti u poligonalne mreže koristeći neki od algoritama (npr. *marching cubes* [8]). Takvi sustavi mogu generirati kompleksne terene i podržavaju prevjese i rupe u terenu.



(a) *Perlin šum* **(b)** *Model terena*

Sl. 2.2: *Modeliranje terena koristeći Perlin šum*

Pristup zasnovan na šumu ima brojne prednosti

- Smanjen prostor potreban za pohranu. Sprema se samo funkcija za generiranje i *seed*
- Algoritmi za generiranje mogu se izvršavati paralelno na GPU, što pruža značajno ubrzanje
- LOD (*engl. Level of detail*) se relativno jednostavno ostvaruje koristeći *teselaciju*.
- Dobra umjetnička kontrola koristeći više razina/slojeva šuma

Neke od negativnih strana su:

- Manje artističke kontrole nasuprot ručnog modeliranja
- Repetitivni izgled ako funkcije nisu dovoljno detaljne i kompleksne
- Teško ostvariti realistične i vizualno ugodne rezultate.

2.1.3 Modeliranje temeljeno na implicitno predstavljenim plohama

Sustavi zasnovani na implicitno predstavljenim plohama (*engl. implicit surface representation*) i konstruktivnoj stereometriji (*engl. Constructive Solid Geometry, CSG*) su po svojoj prirodi proceduralni. Oplošje modela se definira putem implicitnih funkcija i kombinacijom *boolean* operacija nad primitivnim objektima. Model se na kraju može pretvoriti u poligonalnu mrežu (najčešće za *rendering*). Ovakav je način najčešće korišten u CAD/CAM sustavima[9].

Glavne prednosti su:

- Visoka preciznost, plohe su definirane kao implicitne funkcije i nema gubitka podataka prilikom predstavljanja zakrivljenih ploha. Do gubitka preciznosti dolazi tek prilikom konverzije u poligonalnu mrežu, dok se svo modeliranje vrši putem implicitnih funkcija. Zbog toga su pogodni za CAD/CAM sustave
- Funkcije za modeliranje su intuitivne

Negativne strane ovog pristupa su

- Nije pogodno za opisivanje organskih 3D modela poput biljaka, životinja i ljudi. Implicitne funkcije koje bi opisivale takve plohe bile bi izrazito kompleksne. Neki CAD sustavi pružaju *boundary representation* koristeći *NURBS ili Bezier* krivulje koji rješavaju ovaj problem ali i dalje zadržavaju prednosti implicitne definicije ploha.
- Modeli moraju biti prebačeni u oblik zasnovan na poligonima (trokutima) kako bi bili prikazani (*render*). Taj postupak konverzije, ovisno o algoritmu, može rezultirati artefaktima u obliku dugačkih, tankih trokuta koji rezultiraju *shading* greškama prilikom prikaza. Također, ovisno o algoritmu, rezultirajući model može imati neprikladan broj poligona za opisivanje zakrivljenih ploha.

2.1.4 Modeliranje zasnovano na poligonalnim mrežama

Cilj proceduralnog modeliranja zasnovanog na poligonalnim mrežama je replicirati *workflow* koji se koristi u konvencionalnim aplikacijama za kreiranje 3D modela. Proceduralni sustavi su programi koji pozivaju generičke operatore za 3D modeliranje sličnim redosljedom i koristeći slične parametre kao kod ručnog modeliranja. Zatim se vrijednost parametara za svaki poziv operatora mogu izvesti u zaseban objekt i generirati nasumično. Na taj se način mogu dobiti raznoliki rezultati koristeći istu funkciju i operatore [10].

Zbog toga što se rezultirajući objekt već sastoji od poligona, može se prebaciti u objekt temeljen na trokutima koristeći relativno jednostavne algoritme. Model se zatim može poslati na GPU za *rendering*. *3D artist* za vrijeme pisanja funkcija i prilikom odabira parametara ima veliku razinu kontrole nad rezultirajućim modelom. Ovaj pristup se može kombinirati s *subdivision surface* pristupom.

Najpopularnija aplikacija koja podržava ovakav način rada je Houdini. Pravila za generiranje se definiraju koristeći pristup temeljen na grafovima gdje svaki operator odgovara jednom

čvoru grafa, a ulazi i izlazi se definiraju spajanjem čvorova grafa ili kao numeričke vrijednosti. Parametri pojedinih operatora također mogu biti povezani. Trenutno, Houdini se smatra standardnim alatom u industriji za proceduralno generiranje u području video igara, animacije i vizualnih efekata.

2.2. Pregled trenutnih rješenja za rad s 3D modelima

Postoji niz biblioteka i programskih paketa koji posjeduju različite značajke i razine prikladnosti za proceduralno 3D modeliranje.

OCCT[11] (*Open CASCADE technology*) je C++ biblioteka dizajnirana za razvoj CAD, CAM, CAE aplikacija. OCCT pruža klase za strukture podataka i napredne algoritme na 3D modeliranje i podršku za rad s učestalim CAD formatima (IGES, STEP). OCCT sadrži nekoliko *toolkita* (*Mesh, Express Mesh, OMF*) koji imaju mogućnost rada s poligonalnim mrežama. Ti su *toolkiti* razvijeni u svrhu vizualizacije - konverzija različitih 3D reprezentacija u format zasnovan na trokutima i indeksima pogodan za *GPU rendering* ili u format zasnovan na trokutima koji se koristi za aproksimaciju 3D modela i ubrzavanje izvođenja određenih algoritama za numeričku simulaciju.

OpenFlipper [12] je *framework* za procesiranje i vizualizaciju geometrije razvijen na *RWTH Aachen*. Glavni je cilj pružiti zajedničku platformu za razvoj softvera u za procesiranje geometrije i 3D modela. Cilj platforme je prebaciti fokus na razvoj novih algoritama umjesto na ponovni razvoj osnovnih funkcionalnosti čije su implementacije gotove, poznate i lako dostupne. OpenFlipper je zasnovan na *OpenMesh* i *OpenVolumeMesh* strukturama podataka koje pružaju mogućnost rada s *non-manifold* poligonalnim mrežama. Iako OpenFlipper pruža implementacije brojnih algoritama za procesiranje poligonalnih mreža kao što su *smoothing*, *decimation*, *remeshing* i skeletalna animacija, nije namijenjen kao alat za poligonalno modeliranje i samim time ne pruža funkcionalnosti potrebne za razvoj funkcija i naprednih operatora za 3D modeliranje. Moguće je kreirati jednostavne modele uz direktnu manipulaciju strukture podataka, ali implementacije operatora niske razine nisu dostupne.

CGAL [13] (Computer graphics and algorithms library) je projekt koji pruža velik broj efikasnih, pouzdanih i testiranih geometrijskih algoritama. CGAL paket *3D polyhedral surface*[14] pruža implementaciju *Half-edge* strukture podataka i operatore niske razine za dodavanje elemenata i modifikaciju modela/topologije. Modeli kreirani koristeći navedeni paket mogu se koristiti u ostalim CGAL paketima koji pružaju funkcije za procesiranje 3D modela kao što su *subdivision* i *decimation*. CGAL pruža ograničen broj operatora za geometrijsko modeliranje visoke razine i kreiranje osnovnih geometrijskih objekata, ali svi algoritmi koriste *half-edge* strukturu podataka i zbog toga nisu u mogućnosti modelirati *non-manifold* modele.

OpenSCAD[15] je softver za CAD modeliranje, implementiran koristeći nekoliko CGAL paketa. Modeli unutar OpenSCAD-a se ne modeliraju koristeći interaktivne alate za modeliranje već pisanjem i parsiranjem skriptnih datoteka koje opisuju način na koji treba kreirati pojedini objekt. Ovaj je koncept drugačiji od načina na koji rade uobičajeni CAD alati ili alati za poligonalno modeliranje. Glavne tehnike modeliranja u OpenSCAD su CSG (*Constructive*

solid geometry) i ekstrudiranje 2D oblika. Korisnici mogu implementirati kompleksne funkcije koristeći OpenSCAD skriptni jezik i uobičajene programske koncepte (petlje, matematički operatori, kontrola toka) koje bi bilo teško ostvariti koristeći konvencionalno interaktivno modeliranje. Iako je skriptni pristup fleksibilniji, značajno je kompleksniji od konvencionalnih pristupa.

LibIGL[16] je C++ biblioteka algoritama za procesiranje geometrije. Neke od funkcionalnosti su generiranje jednostavnih *facet* i *edge-based* struktura podataka i alati za vizualizaciju koristeći *OpenGL*. LibIGL je razvijena kao biblioteka za procesiranje 3D modela i samim tim nije pogodna za proceduralno 3D modeliranje.

Meshlab [17] je alat za procesiranje modela otvorenog koda temeljen na poligonalnim mrežama. Pruža alate za vizualizaciju i prezentaciju 3D modela, rekonstrukciju, mapiranje boja i teksturiranje, te popravak 3D modela i pripremu za 3D ispis. Iako se često koristi za procesiranje 3D modela, nije pogodan za 3D modeliranje.

Open3D[18] je biblioteka otvorenog koda čiji je cilj pružanje podrške za razvoj softvera koji radi s 3D podacima. Pruža strukture podataka za rad s oblacima točaka, RGB-D slikama i poligonalnim mrežama, te algoritme za procesiranje, vizualizaciju i konverziju između navedenih struktura podataka. Open3D koristi strukturu podataka zasnovanu na trokutima koja se može koristiti za *non-manifold* modele, ali mogućnosti modeliranja su ograničene jer je Open3D dizajniran kao biblioteka za procesiranje.

BMesh[19] je modul unutar Blendera koji pruža pristup internim strukturama podataka i operatorima za 3D modeliranje. *BMesh* struktura podataka podržava *non-manifold* geometriju. Python programsko sučelje pruža velik broj naprednih operatora za geometrijsko modeliranje i ima dobru integraciju s uobičajenim programskim konceptima i tako omogućava automatizaciju procesa modeliranja. Iako *BMesh* interno koristi operatore niske razine za razvoj naprednih operatora, operatori niske razine nisu izloženi krajnjem korisniku putem programskog sučelja. Zbog toga što je Blender objavljen pod GPL licencom, nije ga moguće uključiti kao dio/modul komercijalnih alata i proizvoda.

Ostale konvencionalne aplikacije za poligonalno 3D modeliranje, kao što su Autodesk Maya [20], Autodesk 3dsMax [21], Modo[22] i brojni drugi također pružaju pristup naprednim operatorima za 3D modeliranje i internim strukturama podataka putem raznih programskih sučelja i *SDK (Software development kit)*. Ta su programska sučelja namijenjena za razvoj raznih ekstenzija i alata koje mogu ubrzati, pojednostaviti ili na drugi način poboljšati proces 3D modeliranja. Takvi se alati često integriraju u postojeće korisničko sučelje unutar samog programa. Većina je popularnih alata (uz iznimku Blendera) objavljena pod komercijalnim licencama, što znači da ih nije moguće uključiti kao dio komercijalnih projekata. Uz to, čak i kada bi licenca to dopuštala, bilo bi potrebno uz razvijeni alat krajnjim korisnicima dostaviti i korištenu aplikaciju za 3D modeliranje (ili iz korištene aplikacije ukloniti sve funkcionalnosti koje nisu korištene) što značajno utječe na troškove razvoja, održivost i kvalitetu koda.

ACIS[23], Solids++[24], ParaSolid[25], SMLib[26] su samostalne (*engl. standalone*), komercijalne jezgre za CAD i 3D modeliranje i služe kao osnova za razvoj naprednih, specijali-

ziranih 3D grafičkih, CAD i simulacijskih alata. Neki od navedenih alata pružaju podršku za rad s poligonalnim mrežama.

SideFx Houdini[27] je značajna iznimka koja se po svojim funkcionalnostima i osnovnim konceptima drastično razlikuje od uobičajenih programa za 3D modeliranje. Houdini, skupa s *Houdini engine* pruža izradu i učitavanje 3D modela u druge 3D aplikacije koristeći sustav modeliranja zasnovan na grafovima i operatorima. Cijeli je proces ne-destruktivan i svaka je operacija parametrizirana. Zbog toga je Houdini *de-facto* Standardni alat u industriji za proceduralno 3D modeliranje.

Navedeni programi i biblioteke mogu se svrstati u 4 osnovne kategorije

- Biblioteka za procesiranje 3D modela - koriste poligonalne mreže, često podržavaju non-manifold modele ali ne pružaju funkcionalnosti potrebne za proceduralno 3D modeliranje
- Program za 3D modeliranje - mogu se koristiti za implementaciju alata za 3D modeliranje, ali su ti alati usko vezani uz korišteni softver i ne mogu se dalje koristiti (zbog licenci i praktičnih ograničenja)
- Biblioteke za CAD modeliranje otvorenog koda - pružaju operatore za modeliranje visoke razine, mogu se koristiti kao komponenta većih programa ali su mogućnosti poligonalnog modeliranja ograničene
- Komercijalne biblioteke za CAD modeliranje - pružaju operatore za modeliranje visoke razine. Određene biblioteke pružaju i podršku za poligonalno modeliranje. Skupe i nisu pogodne za manje projekte.

Neke od popularnih, trenutno dostupnih biblioteka za rad s 3D modelima, programi i biblioteke za 3D modeliranje su uspoređeni prema sljedećim značajkama. Rezultati su dani u obliku tablice u 2.1. Početni podaci su prikupljeni iz [28] i prošireni.

- Licenca definira je li dani program objavljen pod licencom otvorenog koda ili restriktivnom, komercijalnom licencom. Različite komercijalne licence mogu dolaziti s različitim uvjetima i ograničenjima dopuštenih načina korištenja. Neke licence otvorenog koda također mogu ograničiti način na koji se programi koriste i dijele.
- Zadnja verzija definira datum javno dostupne zadnje verzije programa. Prilikom izbora bilo kakve softverske biblioteke koja će biti korištena kao dio većeg projekta, obavezno je pratiti održava li se željena biblioteka redoviti, i postoji li mogućnost da potrebnu biblioteku autori održavaju sami. Za neke komercijalne programe nije poznat točan datum zadnje verzije.
- Podržava li program *non-manifold* geometriju i topologiju, npr. više od dva *facea* koji dijele *edge*, Više *faceva* spojenih isključivo putem jednog *vertexa* i sl.
- Je li program samostalna biblioteka (Zamišljena za korištenje kao dio većeg programa) ili je *API* koji pruža usko vezan uz određenu aplikaciju za 3D modeliranje.

- Pruža li *API* pristup operatorima niže razine (*engl. Euler operators*) za lokalnu manipulaciju topologijom ili implementaciju kompleksnih operatora više razine.
- Pruža li *API* implementacije kompleksnih operatora visoke razine, kao što su stvaranje primitivnih objekata, *extrude/sweep*, geometrijske transformacije, boolean operacije i sl.
- Je li *API* fokusiran na rad s poligonalnim mrežama ili koristi drugi način reprezentacije objekata (npr. CSG).
- Koji programski/skriptni jezici se koriste za pristup i rad s *API-jem*.

Tab. 2.1: *Pregled popularnih programa i biblioteka za rad s 3D modelima*

Naziv	Licenca	Zadnja ver.	Non-Manifold	Samostalna	low-level ops	High-level Ops	Poly Focused	Jazici
OCCT	GNU LGPL	7.5.1 (2 Feb 2021)	Yes	Yes	Yes	Yes	No	C++, python, Java CS
OpenFlipper	GNU LGPL	4.3 (13 Dec 2019)	Yes	Yes	No	No	Yes	C/C++
CGAL	GPL3+ / commercial	5.2.1 (17 Mar 2021)	No	Yes	Yes	No	Yes	C++
OpenSCAD	GPL v2	2021.01 (7. feb 2021)	Yes	Yes	No	Yes	No	custom
LiblGL	MPL2	2.3.0 (28 Feb 2021)	Yes	Yes	No	No	Yes	C++, python
Open3D	MIT	0.12.0 (24 dec 2020)	Yes	Yes	No	No	Yes	C++, Python
MeshLab	GNU GPL	2020.12 (1 dec 2020)	Yes	Yes	No	No	Yes	C++, Javascript
BMesh	GPL3+	2.92.0 (Feb 25 2021)	Yes	No	No	Yes	Yes	Python
Maya	Proprietary	2022 (24 Mar 2021)	Yes	No	*	Yes	Yes	MEL, Python
3dsMAX	Proprietary	2022 (24 Mar 2021)	Yes	No	*	Yes	Yes	MaxScript, Python, CS, C++
MODO	Proprietary	14.1v2 (3 sep 2020)	Yes	No	*	Yes	Yes	Macros, Python, Perl, Lua, C++
Houdini	Proprietary	18.5.351 (17 oct 2020)	Yes	Yes	*	Yes	Yes	Python, C++
ACIS	Proprietary	2021 1.0 (10 Nov 2020)	Yes	Yes	*	Yes	No	C++
SOLIDS++	Proprietary	*	Yes	Yes	Yes	Yes	Yes	C++
Parasolid	Proprietary	33.0 (9 Dec 2020)	Yes	Yes	Yes	Yes	No	C#
SMLib	Proprietary	2021*	Yes	Yes	Yes	Yes	Yes	C++

* Informacije nisu lako/javno dostupne

Iako postoji nekoliko samostalnih biblioteka koje pružaju brojne operatore za 3D modeliranje, uglavnom su fokusirane na druge pristupe (CSG, implicitno predstavljene plohe) i pružaju ograničene mogućnosti poligonalnog modeliranja (poligonalno modeliranje može biti ostvareno predstavljanjem svakog poligona kao *implicit surface* ali taj pristup nije efikasan). Biblioteke koje su fokusirane rad s poligonalnim mrežama uglavnom ne pružaju operatore za 3D modeliranje visoke razine, dok je podrška za operatore niske razine ograničena ili ne postoji. Takve su biblioteke uglavnom fokusirane na obradu postojećih poligonalnih mreža i pružaju algoritme poput *smoothinga*, *decimation* ili *remeshing* (*point cloud* i/ili *voxel*). Programska sučelja koja pružaju napredne operatore za 3D modeliranje su uglavnom dio postojećih alata za 3D modeliranje i često su objavljeni pod restriktivnim komercijalnim licencama (najveća iznimka je Blender). Prema trenutnim saznanjima, ne postoji popularna i široko poznata biblioteka zasnovana na poligonalnim mrežama za 3D modeliranje koja pruža operatore za geometrijsko modeliranje objavljena pod licencom otvorenog koda.

3. ZNAČAJKE I ZAHTJEVI NA PROGRAMSKO SUČELJE

Ovo poglavlje definira neke od zahtjeva i značajki koji su korisni i potrebni za implementaciju jednostavnih i efikasnih algoritama za proceduralno 3D modeliranje bez korisničkog unosa i definira svojstva sučelja koja omogućavaju jednostavno korištenje uz druge aplikacije.

3.1. Enkapsulacija kroz operatore niske razine

Strukture podataka za *manifold* i *non-manifold* reprezentacije topologije su načelno kompleksne. Uz očekivana svojstva, kao što su prostorne koordinate pojedinog *vertexa* i normale, osnovni elementi (*vertex, edge, face, loop*) moraju implementirati dodatna svojstva potrebna za jednoznačno spremanje i efikasno dohvaćanje informacija o susjednim elementima. Jedno takvo svojstvo jest *edge->vNext/vPrev*, koje se koristi za praćenje *edgeva* oko pojedinog *vertexa*. Ako su ta svojstva izložena prema krajnjem korisniku, postoji mogućnost da će vrijednosti tih svojstava biti postavljene na nevaljanu vrijednost, te će ostaviti strukturu podataka u nevaljanom stanju. Jedan takav (preuveličan) primjer dan je u primjeru koda 3.1, gdje se pokušava kreirati trokut koji se sastoji od tri *vertexa* i tri *wire edgea*.

```
1 Vert* v1 = new Vert(); // isto za v2, v3
2 Edge* e1 = new Edge(); // isto za e2, e3
3 e1->v1 = v1; // postavljanje prvog vertexa
4 e1->v2 = v2; // postavljanje drugog vertexa
5 v1->e = e1; // postavljanje edga e1 kao susjednog vertexu v1
6 v2->e = e1 // postavljanje edga e1 kao susjednog vertexu v2
7 // e2 se kreira izmedu v2, v3; e3 se kreira izmedu v3, v1 na isti nacin
8 auto edges = v1->Edges(); // vrijednost e->v1Next nije postavljena, dogada se greska
```

Primjer koda 3.1: *Direktna manipulacija strukture, ostavlja model u nevaljanom stanju*

Operatori niske razine, poznati kao i *Euler* operatori, ili u *non-manifold* slučaju NMT (*engl. Non-Manifold Topology*) operatori su funkcije koje vrše lokalne modifikacije nad strukturom podataka. Koncept operatora niske razine prvi je definirao Bruce G. Baumgart za *manifold winged-edge* strukturu podataka [29], a proširio ga je Kevin J. Weiler *non-manifold* slučajeve prilikom razvoja *radial-edge* strukture podataka [30, 31]. Implementirani operatori niske razine koje pruža AobaAPI sučelje slijede predložene specifikacije uz promjene koje su bile potrebne zbog razlika u strukturama podataka. Operatori niske razine su pažljivo implementirani i obvezni su uvijek postaviti svojstva za definiranje susjedstva na ispravne vrijednosti i ostaviti strukturu podataka u valjanom stanju. Korištenjem tih operatora umjesto direktne manipulacije strukture podataka možemo implementirati kompleksne funkcije za 3D modeliranje, znajući da će struktura podataka biti u valjanom stanju, kao što je prikazano primjerom koda 3.2.

```
Vert* v1 = new Vert(); // isto za v2, v3
Edge* e1 = new Edge(); // isto za e2, e3
MakeVert(mesh, v1); // isto za v2, v3
MakeEdge(v1, v2, e1); // e2, e3 se stvaraju na isti nacin, izmedu v2, v3 i v3, v1
auto edges = v1->Edges(); // rezultat su e1, e3, kao sto je ocekivano, nema greske
```

Primjer koda 3.2: *Manipulacija strukture podataka putem operatora*

Pojedine kompleksnije funkcije za 3D modeliranje moguće je implementirati direktnom manipulacijom strukture podataka uz poboljšanje performansi kroz provođenje raznih optimizacija algoritma, preskakanjem određenih sigurnosnih provjera i slično. Izlaganje cijele strukture podataka bi rezultiralo brojnim greškama koje bi bilo izrazito teško dijagnosticirati. Operatori niske razine poboljšavaju mogućnosti za proširenje programskog sučelja, jer nije potrebno znanje o internim svojstvima strukture podataka[32]. Uz to, operatori niske razine odvajaju korisničke aplikacije od strukture podataka i ostavljaju mogućnost njene izmjene. Ovakav pristup se može promatrati kao kompromis između brzine izvođenja i jednostavnosti implementacije kompleksnih funkcija.

3.2. Funkcije za odabir i filtriranje elemenata

Standardne aplikacije za 3D modeliranje oslanjaju se na korisnički unos - kombinaciju unosa mišem, tipkovnicom ili grafičkim tabletom kako bi odabirali elemente modela i unosili vrijednosti za pojedine geometrijske operacije. Kada se 3D model generira proceduralno, takav korisnički unos nije dostupan i zbog toga je potrebno koristiti druge načine za odabir i filtriranje elemenata[33]. Zbog toga što sve funkcije vraćaju objekte gdje su elementi spremljeni unutar *std::vector contaiera*, filtriranje se može ostvariti korištenjem funkcija koje su dio standardne C++ biblioteke, npr *std::copy_if, std::remove_if* i lambda izraza (*engl. Lambda expressions*). Također, može se i koristiti jednostavna petlja. Prilikom dohvaćanja susjednih elemenata ili dohvaćanja svih elemenata modela, može se primijeniti lambda funkcija direktno na pojedine elemente, a kao rezultat će biti vraćen filtrirani podskup elemenata (primjer koda 3.3).

Osim filtriranja elemenata, AobaAPI sučelje implementira funkcije koje transformiraju odabrane *vertexe, edgeve i faceve* u druge elemente. Kada se koristi *face* kao ulaz, vraćaju se *vertex, edge* koji čine njegov *boundary*. Ako je ulaz *edge*, kao izlaz se vraćaju *vertexi* koji čine *edge* i *face* čiji je cijeli *boundary* odabran. U slučaju *vertexa*, vraćaju se *edgevi i facevi* čiji su cijeli *boundary* odabrani.

```
// dohvati sve ne izolirane vertexe cija je z koordinata unutar raspona [min, max]
float min = 2.0f;
float max = 4.0f;
auto myFunc = [min, max](const Vert* const v) {
    if(v->Edges().size() > 0) {
        return v->co.z > bottom && v->co->z < max;
    } else {
        return false
    }
};
auto verts = mesh->Verts(myfunc);
// pronalazi edgeve i faceve koji imaju sve vertexe oznacene
SelectResult result = SelectFromVerts(mesh, verts);
auto edges = result.edges;
auto verts = result.verts;
```

Primjer koda 3.3: *Filtriranje elemenata*

3.3. Operatori za geometrijsko modeliranje visoke razine

Iako je moguće kreirati kompleksne 3D modele u konvencionalnim aplikacijama za 3D modeliranje dodavanjem pojedinih *vertexa* i njihovim spajanjem u nove *edgeve* i *faceve*, takav se pristup rijetko koristi jer nije brz niti fleksibilan. Uglavnom se koriste napredni operatori za 3D modeliranje, koji pojednostavljuju i ubrzavaju proces 3D modeliranja. Isto vrijedi i za proceduralni pristup. Iako se 3D modeli mogu generirati koristeći isključivo operatore niske razine (*make vert/edge/face, kill vert/edge/face*, takav pristup nije optimalan i zbog toga se koriste operatori visoke razine [32].

Operatori koje pruža AobaAPI programsko sučelje slični su onima koji se mogu pronaći u konvencionalnim programima za 3D modeliranje. Mogu se podijeliti u 4 kategorije, ovisno o tome kakve promjene i operacije vrše nad strukturom podataka

- *Create* - Dodaju nove elemente u model, bez izmjene trenutnih elemenata. Npr. kreiranje primitivnih objekata i 3D modela (kocke, sfere, rešetke i sl.).
- *Modify* - Izmjenjuju postojeće elemente, te dodaju ili brišu nove elemente po potrebi, pazeći na održavanje informacije o susjednim elementima u valjanom stanju. Npr. *mirror*, *extrude*, *delete*, *inset face*
- *Select* - Pretvaraju jedan tip odabranih elemenata (*vertex, edge, face*) u drugi tip, bez njihove izmjene.
- *Transform* - Transformiraju postojeću geometrije (*vertex coordinates*), bez izmjene susjedstva ili topologije. Npr. Translacija, skaliranje

Skup operatora koje pruža AobaAPI sučelje odabran je tako da bude koristan u različitim slučajevima. Korisnici također imaju mogućnost implementacije vlastitih operatora koristeći pri tome kombinaciju operatora visoke i niske razine. Na primjer, operator koji generira tlocrte/planove kuća jest koristan, ali takav operator nije primjenjiv u općem slučaju. Primjeri korištenja operatora za stvaranje kompleksnih modela dani su u poglavlju 4.6.

3.4. Struktura podataka zasnovana na indeksima

Poligonalna struktura podataka zasnovana na indeksima (*engl. index-based polygon mesh, polygon soup*) jedan je od najfleksibilnijih načina za pohranu 3D modela i prijenos između različitih aplikacija. U osnovi se sastoji od dva *buffera*, gdje prvi sadrži geometrijske koordinate točaka, a drugi sadrži indekse točaka koje tvore pojedini trokutasti poligon. Takav format je čest u raznim 3D aplikacijama, kao što su alati za 3D modeliranje, *game engine* i video igre, pa je zbog toga pogodan za prijenos podataka između njih. Uz to, takav format je pogodan za slanje podataka na GPU za *rendering*. Osim koordinata mogu se dodati i druga svojstva, poput normala ili *UV texture* koordinata. Jednostavan primjer takve strukture prikazan je u primjeru koda 3.4.

```
1 class IndexMesh {
2         std::array<Vec3> vertexCoords;
3         std::array<Vec3> vertexNormals;
4         std::array<unsigned int> indices;
5 };
```

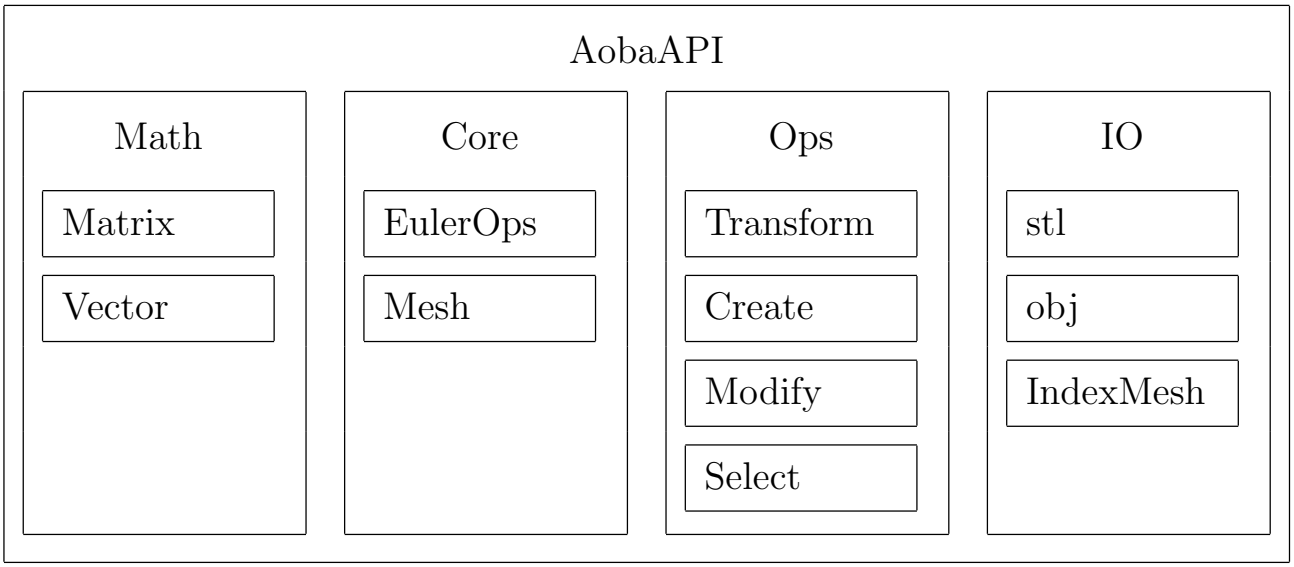
Primjer koda 3.4: *Jednostavna struktura podataka zasnovana na indeksima*

4. IMPLEMENTACIJA RJEŠENJA

AobaAPI projekt i programsko sučelje podijeljeni su u četiri imenska prostora (*engl. namespace*) i struktura direktorija i datoteka slijedi hijerarhiju imenskih prostora i klasa. Vlastita matematička biblioteka je razvijena kako bi se otklonila potreba za dodatnim *dependencyjem*. Struktura podataka i operatori niske razine (Euler operatori) nalaze se u "Core" imenskom prostoru dok se operatori visoke razine nalaze u "Ops" imenskom prostoru.

4.1. Struktura projekta

Tab. 4.1: *Struktura projekta*



"Math" imenski prostor sadržava klase i funkcije često korištene u računalnoj grafici - vektore i matrice. Implementirane su generičke funkcije i matematički operatori potrebni za efikasno 3D modeliranje, kao što su množenje matrica i vektora, operacije zbrajanja i oduzimanja i metode za kreiranje često korištenih matričnih reprezentacija geometrijskih transformacija, poput rotacije, skaliranja i translacije.

"Core" imenski prostor sadrži osnovne funkcionalnosti uključujući implementaciju strukture podataka, funkcije za dohvaćanje informacija o susjednim elementima i operatore niske razine koji mijenjaju strukturu podataka na najnižoj razini. Topološka svojstva strukture podataka (pokazivali koji se koriste za praćenje i pohranu informacija o susjednim elementima) nisu izloženi izvan ovog imenskog prostora, a pristup je dan operatorima niske razine koristeći koncept *friend funkcije*.

"IO" imenski prostor pruža funkcije za učitavanja i pohranu 3D modela u nekoliko formata, npr. ".stl" i ".obj". Također pruža i strukturu podataka zasnovanu na indeksima koja se koristi kao "most" prema drugim aplikacijama i za slanje podataka na GPU za *rendering*:

"Ops" imenski prostor daje često korištene operatore za geometrijsko modeliranje visoke razine, kao što su *duplicate, extrude, mirror*, geometrijske transformacije i slično.

4.2. Matematička biblioteka

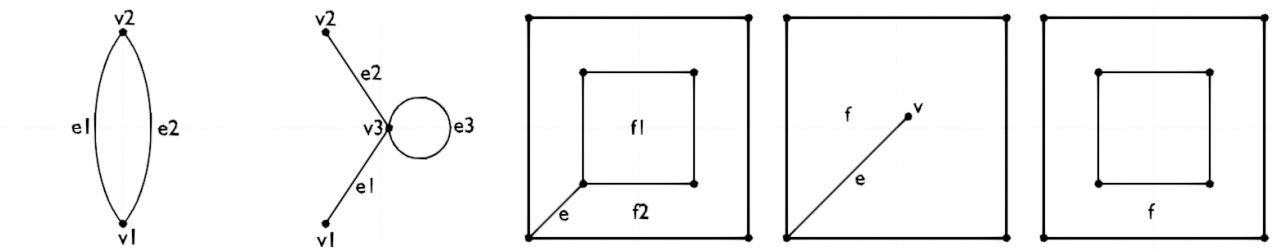
Jednostavna matematička biblioteka fokusirana na algoritme i klase korištene u računalnoj grafici razvijena je uz pomoć standardne C++ biblioteke, kako bi se uklonila potreba za vanjskom bibliotekom kao što su "Eigen" ili "glm". Biblioteka pruža implementacije 2D i 3D vektora i matrica, te 4D vektore i matrice za potrebe rada s homogenim koordinatama. Implementirane su uobičajene funkcije za rad s vektorima i matricama poput skalarnog i vektorskog produkta, inverza i transponiranja matrica, kao i matematički operatori za zbrajanje/oduzimanje vektora i matrica i njihovo množenje. Primjer korištenja matematičke biblioteke za kreiranje matrica i njihove primjene za 3D transformacije dan je u primjeru koda 4.1.

```
1 Vec3 rotAxis = Vec3(0, 1, 1); // novi 3D vektor, os rotacije
2 rotAxis.Normalize(); // normalizacija vektora za os rotacije
3 Mat3 rot = Mat3::Rotation(rotAxis, PI / 8); // matrica rotacije za kut od PI/8 rad
4 Mat3 scale = Mat3::Scale(Vec3(1, 0, 0), 2); // matrica skaliranja, 2x po X osi.
5 Mat3 transform = rot * scale; // transformacijska matrica, kombinacija rotacije i
    ↪ skaliranja
6 Vec3 co = Vec3(2, 3, 4); // 3D vektor, predstavlja tocku u prostoru
7 co = transform * co; // transformacija koordinate koristeci matricu
```

Primjer koda 4.1: *Korištenje matematičkih funkcija*

4.3. Struktura podataka

Struktura podataka zasnovana je na strukturi podataka korištenoj unutar Blendera - Bmesh [19] i na *Radial-edge* strukturi[31]. Određena ograničenja postavljena su na razini operatora koja onemogućuju određene posebne slučajeve (sl. 4.1) - *multigraph* i *self-loop*. Uz to, svaki poligon (*face*) ograničen je na točno jedan *loop*, a unutar *loopa* nije dozvoljeno korištenje jednog *vertexa* ili *edgea* više puta. Takvi slučajevi, iako se mogu koristiti za razvoj operatora, donose malo vrijednosti u programu fokusiranom isključivo na poligonalno modeliranje, ali značajno povećavaju kompleksnost strukture podataka i otežavaju implementaciju operatora i algoritama zbog većeg broja rubnih slučajeva.



(a) *Multigraph* **(b)** *Self-loop* **(c)** *Isthmus edge* **(d)** *Strut edge* **(e)** *Multi-loop face*

Sl. 4.1: *Razni nedozvoljeni slučajevi*

Struktura podataka sastoji se od četiri osnovna elementa - *vertex (vert)*, *edge*, *face* i *loop*. Uz to, definirana je i *mesh* klasa koja služi kao *parent* element svim ostalim elementima. U primjerima koda 4.2 - 4.5 slijede C++ implementacije svakog osnovnog elementa (svojstva za održavanje dvostruko povezane liste svih elemenata unutar *mesha* i svojstva koja sadržavaju zastavice nisu prikazana zbog jednostavnosti).

```
class Vert() {
    public:
        Math::Vec3 co;       // X,Y,Z vertex koordinate
        Math::Vec3 no;       // X,Y,Z vertex normale
    private:
        Edge* e;     // prvi element u listi edgeva oko vertexa
        Mesh* m;     // pokazivac na mesh
};
```

Primjer koda 4.2: *Vertex C++ implementacija*

```
class Loop() {
    private:
        Vert* v;     // vertex koji ovaj loop koristi, definira orijentaciju
        Edge* e;     // edge koji ovaj loop koristi
        Face* f;     // face koji ovaj loop tvori
        Loop* eNext; // lista loopova oko edga e
        Loop* ePrev; // lista loopova oko edga e
        Loop* fNext; // lista loopova koji tvore face f
        Loop* fPrev; // lista loopova koji tvore face f
        Mesh* m;     // pokazivac na mesh
};
```

Primjer koda 4.3: *Loop C++ implementacija*

```
class Face() {
    public:
        Math::Vec3 no;       // face normala
        short materialIdx;   // face intex materijala
    private:
        Loop* l;     // prvi element u listi loopova koji tvore ovaj face
        Mesh* m;     // pokazivac na mesh

};
```

Primjer koda 4.4: *Face C++ implementacija*

```
class Edge() {
        Vert* v1;     // neporedani vertexi ovog edga
        Vert* v2;     // neporedani vertexi ovog edga
        Loop* l;      // prvi element u listi loopova oko ovog edga
        Edge* v1Next; // lista edgeva oko v1
        Edge* v1Prev; // lista edgeva oko v1
        Edge* v2Next; // lista edgeva oko v2
        Edge* v2Prev; // lista edgeva oko v2
        Mesh* m;      // pokazivac na mesh
};
```

Primjer koda 4.5: *Edge C++ implementacija*

Osim prikazanih svojstava, svaki element posjeduje indeks i dva 32-bitna seta zastavica (*engl. flags*). Jedan set zastavica rezerviran je za interne operatore (operatori koje definira AobaAPI sučelje), dok je drugi set zastavica dostupan za korisničke operatore. Uz to, svaki element sadrži *mNext/mPrev* pokazivač koji se koristi za održavanje liste svih elemenata *mesha*.

4.3.1 Susjedstvo elemenata i potpunost strukture podataka

Osnovni načini praćenja informacija o susjedstvu koriste dvostruko povezane liste. To su:

- Lista *edgeva* oko pojedinog *vertexa*, koji se obilaze koristeći *edge->v1Next/v1Prev* i *edge->v2Next/v2Prev* (sl. 4.2)
- Lista *loopova* oko svakog *edgea*, obilaze se putem *loop->eNext* i *loop->ePrev* (sl. 4.3)
- Lista *loopova* koji tvore *face*, obilazi se koristeći *loop->fNext* i *loop->fPrev* (sl. 4.4)

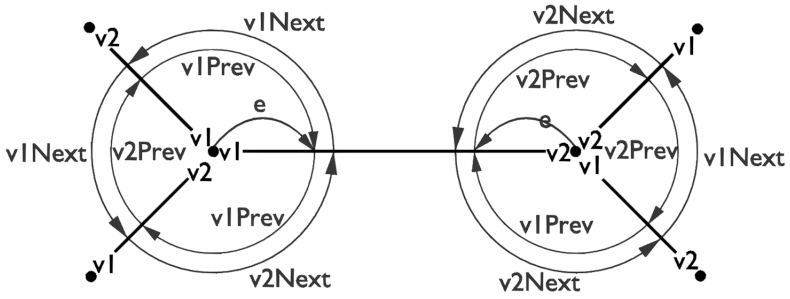
Uz to, slijedeće informacije o susjedstvu ne zahtjevaju liste:

- Svaki *edge* drži referencu na dva *vertexa* koji ga tvore
- Svaki *loop* drži referencu na *edge* koji ga tvori
- Svaki *loop* drži referencu na *vertex* koji definira orijentaciju
- Svaki *loop* drži referencu na *face* čiji *boudnary* tvori

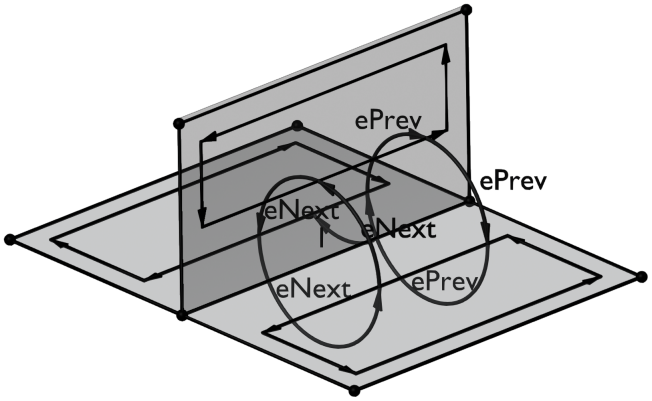
Sve korištene informacije o susjedstvu dane su u obliku matrice (*engl. Adjacency matrix* [30]) u tablici 4.2 i na slikama 4.2 - 4.5

Tab. 4.2: *Susjedstvo elemenata u obliku matrice*

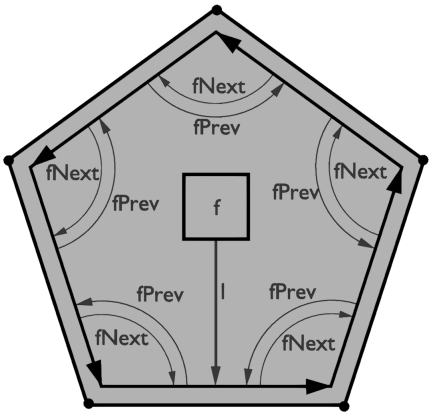
	V	E	L	F
V		V<E>		
E	E{v1, v2}		E<L>	
L	L v	L e		L f
F			F<L>	



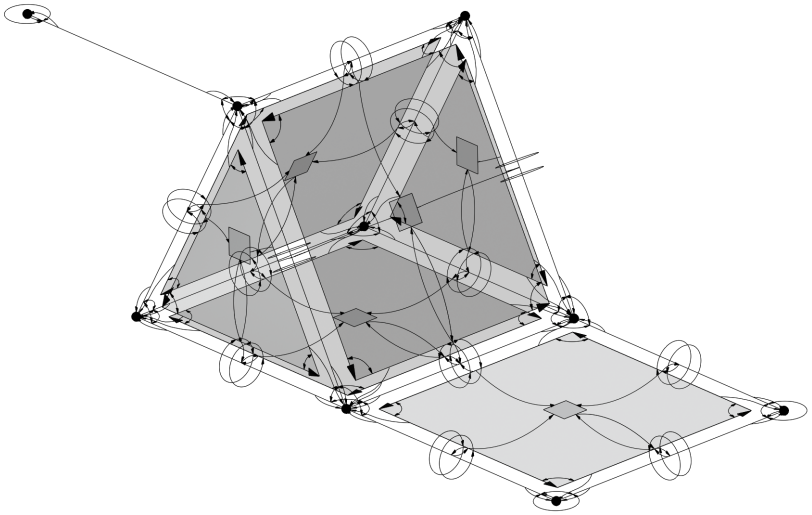
Sl. 4.2: *Lista edgeva oko pojedinog vertexa*



Sl. 4.3: *Lista loopova oko pojedinog edgea*



Sl. 4.4: *Lista loopova koji tvore face boundary*



Sl. 4.5: *Sve informacije o susjedstvu*

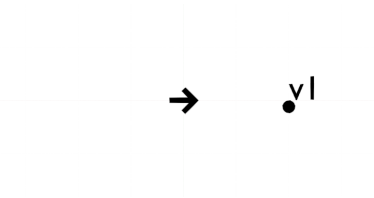
Struktura podataka je potpuna (*engl. complete*) ako se sve informacije o susjedstvu mogu jednoznačno odrediti iz postojećih informacija. `V<V>` susjedstvo se može odrediti prolaskom kroz svaki *edge* oko *vertexa*. `V<L> i V<F>` susjedstvo može se odrediti obilaženjem *loopa*, odnosno *facea* za svaki *edge* oko pojedinog *vertexa*. `E<E>` je moguće odrediti spajanjem lista *edgeva* oko *v1* i *v2*. `E<F>` informacije možemo dobiti koristeći *loop*. `L<L>` se može odrediti kroz *face boundary* ili kroz *edge*. `F<V> u F<E>` je moguće trivijalno odrediti prolaskom kroz *face boundary*. `F<F>` je moguće odrediti spajanjem lista *faceva* oko svakog *vertexa*, pazeći pri tome na ponavljanje elemenata. Ovime je potvrđena potpunost strukture podataka uz prethodno definirana ograničenja.

Svaka od dvostruko povezanih lista koja se koristi za informacije o susjedstvu može se zamijeniti jednostruko vezanom listom, što pruža kompromis između računske i memorijske kompleksnosti[31]. Ako se koriste jednostruke liste, potrebno je pretraživanje do prethodnog elementa kako bi se element mogao ukloniti iz liste. Na primjer, ako se *loop* uobičajeno briše kada se briše cijeli *face*, i obilazi koristeći *next* svojstvo, *prev* svojstvo je moguće ukloniti, no tada je potrebno pretraživanje kada se briše samo jedan *loop* koji tvori *face boundary*, kao što je slučaj kod *EdgeSqueeze* operatora.

4.4. Operatori niske razine

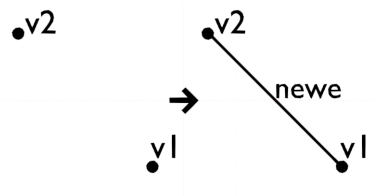
Operatori niske razine (*engl. low-level, euler operators*) [29, 30] su apstrakcija između strukture podataka i kompleksnijih operatora [34] za geometrijsko modeliranje. Pružaju sučelje za lokalne izmjene topologije, dok istovremeno provjeravaju sve ulazne podatke i osiguravaju da će struktura podataka biti u valjanom stanju nakon izvođenja. Svako svojstvo strukture podataka koji se koristi za pohranu informacija o susjedstvu označeno je kao privatano, a pristup je operatorima dan kroz koncept *friend* funkcija. Slijedi opis i definicija operatora niske razine implementiranih u trenutnoj verziji. Novi operatori mogu biti dodani po potrebi ako poboljšavaju performanse ili kvalitetu koda.

MakeVert(Mesh* m, Vert* newv) - Kreira novi *vertex* i dodaje ga u model. Novi *vertex* nema susjednih elemenata (sl. 4.6).



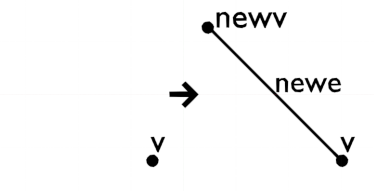
Sl. 4.6: *MakeVert*

MakeEdge(Vert* v1, Vert* v2, Edge* newe) - kreira novi *wire edge* između dva postojeća *vertexa*. Novi *edge* dodaje se u odgovarajuće liste susjednosti danih *vertexa*. Novi *edge* ne može se napraviti ukoliko *edge* koji spaja predane *vertexe* već postoji ili ako su *vertex v1* i *v2* jednaki (sl. 4.7).



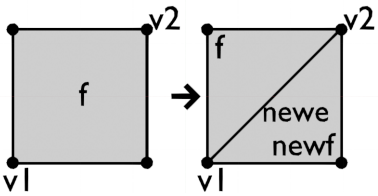
Sl. 4.7: *MakeEdge*

MakeEdgeVert(Vert* v, Edge* newe, Vert* newv) - Kreira novi *wire edge* između jednog postojećeg i jednog novog *vertexa* (sl. 4.8).



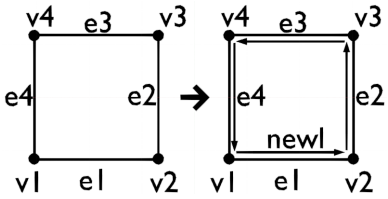
Sl. 4.8: *MakeEdgeVert*

ManifoldMakeEdge(Vert* v1, Vert* v2, Face* f, Edge* newe, Face* newf) - Kreira novi *edge* između *vertexa v1, v2*, gdje *v1, v2* pripadaju istom *faceu*, te dijeli postojeći *face* i njegov *loop* na dva različita (sl. 4.9).



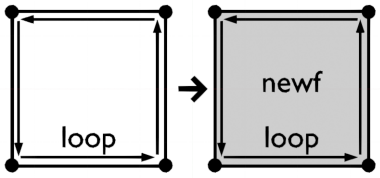
Sl. 4.9: *ManifoldMakeEdge*

MakeLoop(vector¡Edge*¿ edges, vector¡Vert*¿ verts, Loop* newl) - Kreira novi *loop* koristeći danu listu *edgeva* i *vertexa*. Uz to, postoji i funkcija *ValidateLoop* koja provjerava je li dani *loop* ispravno definiran (sl. 4.10).



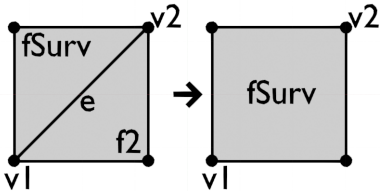
Sl. 4.10: *MakeLoop*

MakeFace(Loop* loop, Face* newf) - Kreira novi *face* koristeći *loop* koji je kreiran putem *MakeLoop* operatora (sl. 4.11).



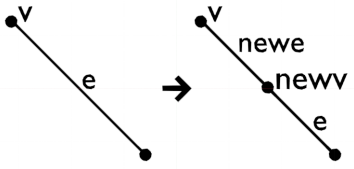
Sl. 4.11: *MakeFace*

DissolveEdge(Edge* e, Face* fSurvivor) - Spaja točno dva *facea* koji dijele *edge* u jedan *face*. *Loopovi* i *edge* se brišu, te se također briše i jedan *face* (sl. 4.12).



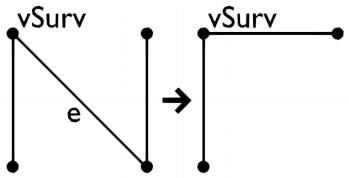
Sl. 4.12: *DissolveEdge*

EdgeSplit(Edge* e, Vert* v, Edge* newe, Vert* newv) - Dijeli *edge* na dva dijela tako što stvara novi *vertex* u sredini. Novi *loopovi* i informacije o susjedstvu su izmijenjeni po potrebi (sl. 4.13).



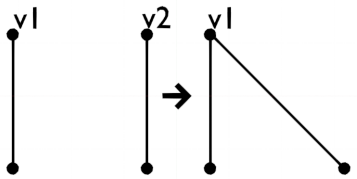
Sl. 4.13: *EdgeSplit*

EdgeSqueeze(Edge* e, Vert* vSurvivor) - Spaja krajeve danog *edgea* i briše ga. *Loopovi*, *edgevi* i *facevi* se brišu po potrebi (sl. 4.14).



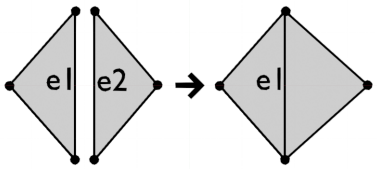
Sl. 4.14: *EdgeSqueeze*

GlueVert(Vert* v1, Vert* v2) - Spaja dva dana *vertexa* i čuva informacije o susjednim elementima. Ukoliko postoji *edge* između danih *vertexa*, operator se ponaša kao *EdgeSqueeze* operator (sl. 4.15).



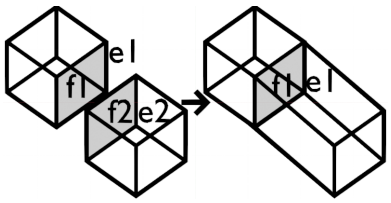
Sl. 4.15: *GlueVert*

GlueEdge(Edge* e1, Edge* e2) - Spaja *edge e1* i *edge e2*, i briše *vertex* ako nije dijeljen između danih *edgeva*. Operator je implementiran koristeći *GlueEdge* operator (sl. 4.16).



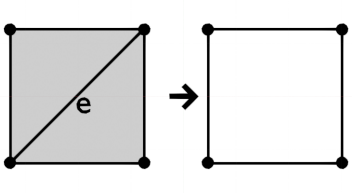
Sl. 4.16: *GlueEdge*

GlueFace(Face* f1, Edge* e1, Face* f2, Edge* e2) - Spaja *Face f1, f2*, te briše *face f2* i njegov *loop*. *Edge e1, e2* služe kako bi specificirali orijentaciju u kojoj je potrebno spojiti dani *face*. Ukoliko je potrebno, operator briše elemente koji se nalaze između danih *faceva*. Implementiran je koristeći *GlueVert* operator (sl. 4.17).



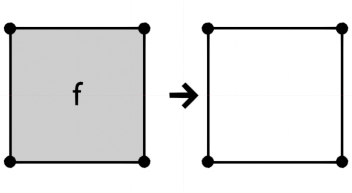
Sl. 4.17: *GlueFace*

KillEdge(Edge* e) - Briše dani *edge* i sve susjedne *faceve* i *loopove* (sl. 4.18).



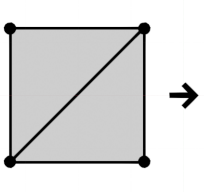
Sl. 4.18: *KillEdge*

KillFace(Face* f) - Briše dani *face* i njegov *loop* (sl. 4.19).



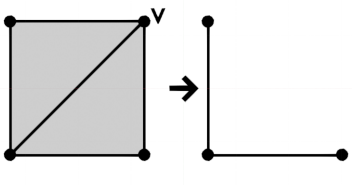
Sl. 4.19: *KillFace*

KillMesh(Mesh* m) - Briše sve elemente unutar modela i sam model (sl. 4.20).



Sl. 4.20: *KillMesh*

KillVert(Vert* v) - Briše dani *vertex* i sve susjedne *edgeve* i *faceove* (sl. 4.21).



Sl. 4.21: *KillVert*

Make vert/edge/face/loop i *kill vert/edge/face* tvore minimalni set operatora koji su potrebni za razvoj kompleksnijih operatora za geometrijsko modeliranje i mogu se koristiti za stvaranje bilo kakvog modela, ali korištenje samo navedenih operatora uvodi nepotrebnu kompleksnost i zato je poželjno implementirati dodatne operatore niske razine. Na primjer, alternativa *EdgeSplit* operatoru bila bi pozvati *KillEdge* operator, te zatim zvati *MakeVert, MakeEdge, MakeLoop i MakeFace* operatore[30].

4.5. Operatori za geometrijsko modeliranje visoke razine

Operatori visoke razine su funkcije koje su često korištene za 3D modeliranje. Operatori koje pruža programsko sučelje AobaAPI ekvivalentni su onima koji se koriste u konvencionalnim programima za 3D modeliranje. Svi operatori visoke razine implementirani su koristeći kombinaciju operatora niske razine i javnih metoda koje pružaju pojedini elementi strukture podataka. Iako bi neke od operatora bilo moguće implementirati uz direktnu manipulaciju strukture podataka i tako dodatno optimizirati algoritme, takav bi pristup uzrokovao veću povezanost (*engl. tight coupling*) sa samom strukturom podataka i kod bi bio teže razumljiv ostalim korisnicima. Ovaj se problem može ublažiti implementacijom dodatnih operatora niske razine.

4.5.1 Sustav zastavica i indeksa

Svaki element strukture podataka ima javno dostupne dvije 32-bitne zastavice i indeks čija veličina ovisi o platformi na kojoj se biblioteka koristi (*32/64 bit*). Jedan set zastavica rezerviran je za korištenje unutar internih operatora (operatori implementirani unutar biblioteke), dok je drugi set zastavica dostupan za korisnički-definirane funkcije. Ta su dva seta zastavica odvojeni kako bi se smanjila mogućnost konflikata ako se interni operatori koriste kao komponente kompleksnijih funkcija. Indeksi se mogu koristiti za praćenje parova elemenata, dok se zastavice, na primjer, mogu koristiti za označavanje prethodno posjećenih ili na neki način obrađenih elemenata. Primjer korištenja sustava zastavica i indeksa dan je u primjeru koda 4.6, koji prikazuje isječak iz *ExtrudeEdge* operatora. U ovom slučaju zastavica se koristi kako bi se označio pojedinačni *vertexi* koji su prethodno ekstrudirani, dok se indeks koristi za povezivanje starog s novim generiranim *vertexom*. Prije kraja izvođenja operatora, sve zastavice i indeksi se postavljaju na nulu.

```
1  const int32_t EXTRUDED_VERT = 1 << 0;
2  int vertIdx = 0;
3  for(Edge* edge : edges) {
4      for(Vert* vert : edge->Verts()) {
5          if(!(vert->flagsIntern & EXTRUDED_VERT)) {
6              Core::Vert* newv = new Core::Vert();
7              Core::Edge* newe = new Core::Edge();
8              Core::MakeEdgeVert(currentVerts.at(j), newe, newv);
9              currentVerts.at(j)->flagsIntern = EXTRUDED_VERT;
10             currentVerts.at(j)->index = vertIdx;
11             vertIdx++;
12         }
13     }
14 }
```

Primjer koda 4.6: *Korištenje sustava indeksa i zastavica*

4.5.2 Ulazni i povratni tipovi podataka

Kao ulazi u operator, elementi modela se predaju unutar *std::vector containera*, dok ostali parametri mogu biti vektori, matrice i osnovni C++ tipovi podataka, kao što su *float* ili *bool*. Svaki operator kao parametar prima i *mesh* objekt, koji se koristi za provjeru da svi ulazni elementi pripadaju odgovarajućem *meshu*, što olakšava i povećava sigurnost pri istovremenom radu s većim brojem *mesheva*. U slučaju kada operator ima kompleksan rezultat (rezultat je veći broj različitih elemenata), AobaAPI programsko sučelje implementira novu klasu koja služi za spremanje povratnih podataka. Koriste se *std::vector containeri* unutar kojih se spremaju pokazivači na pojedine elemente modela. Iako je moguće kao povratni tip koristiti *dictionary* (BMesh API koristi ovakav način), ključevi nisu poznati u trenutku prevođenja, te ih je potrebno potražiti unutar dokumentacije, a ako postoji greška, ona neće biti otkrivena do izvođenja programa, što iziskuje dodatno vrijeme za razvoj programa. Primjer definicije operatora i povratnog tipa dan je primjerom koda 4.7.

```
1 class ExtrudeVertsResult {
2   public:
3     vector<Vert*> verts;
4     vector<Edge*> edges;
5 };
6 const ExtrudeVertsResult ExtrudeVerts(Mesh* m, const vector<Vert*>& verts);
```

Primjer koda 4.7: *Definicija ExtrudeVerts operatora*

4.5.3 Primjer implementacije operatora visoke razine

Primjer implementacije jednog od operatora, *ExtrudeVerts* je dan u primjeru koda 4.8. Ovaj operator kao ulaz prima listu *vertexa*, i za svaki *vertex* kreira novi *vertex* na originalnim koordinatama. Zatim se taj par *vertexa* spaja u novi *wire edge*.

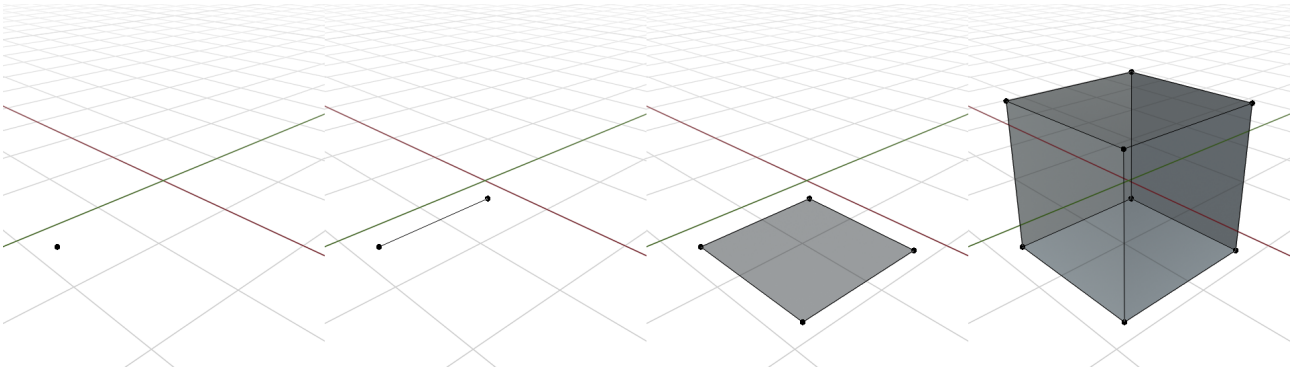
```
1 const ExtrudeVertsResult ExtrudeVerts(Mesh* m, const vector<Vert*>& verts) {
2     ExtrudeVertsResult result = ExtrudeVertsResult();
3     for(Vert* v : verts) {
4         Vert* newv = new Vert();
5         newv->co = v->co;
6         Edge* newe = new Edge();
7         MakeEdgeVert(v, newe, newv);
8         result.verts.push_back(newv);
9         result.verts.push_back(newe);
10    }
11    return result;
12 }
```

Primjer koda 4.8: *ExtrudeVerts operator*

4.6. Primjeri korištenja

Ovo poglavlje daje nekoliko primjera korištenja AobaAPI programskog sučelja za generiranje 3D poligonalnih modela. Detaljnija implementacija, izvorni kod i objašnjenja dani su u priloženom *GitHub* repozitoriju.

4.6.1 Generiranje jednostavne kocke



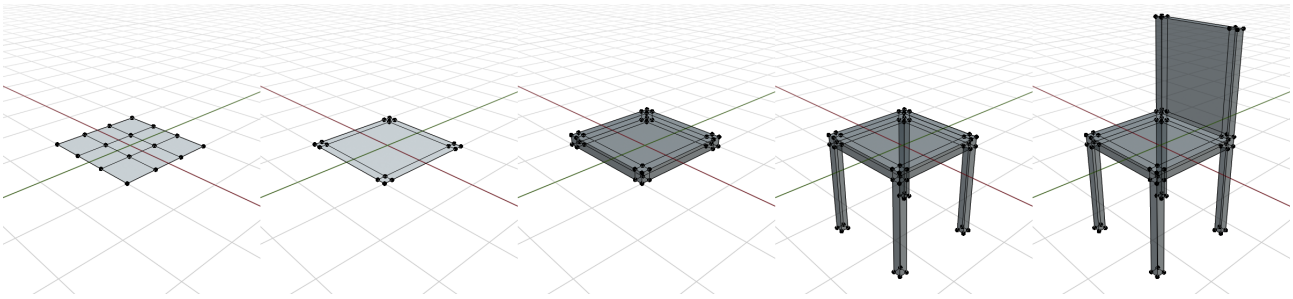
Sl. 4.22: *Postupak modeliranja kocke*

Osim trivijalnog rješenja koje koristi *CreateCube* operator za kreiranje kocke, moguće ju je napraviti ekstrudiranjem iz jednog *vertexa*. Prvo se *vertex extrudira* u *edge*. Zatim se novi *edge* ekstrudira u *face*. Na kraju se novi *Face* ekstrudira u volumen, odnosno kocku (slika 4.22, primjer koda 4.9). Također je moguće i kocku modelirati korištenjem operatora niske razine. Takav je primjer moguće pronaći u implementaciji *CreateCube* operatora unutar izvornog koda biblioteke u priloženom repozitoriju.

```
1 Vert* vert = CreateVert(mesh, Vec3(-1, -1, -1));
2 ExtrudeVertsResult evResult = ExtrudeVerts(mesh, {vert});
3 Translate(mesh, evResult.verts, Vec3(2, 0, 0));
4 ExtrudeEdgesResult eeResult = ExtrudeEdges(mesh, evResult.edges);
5 Translate(mesh, eeResult.verts, Vec3(0, 2, 0));
6 ExtrudeFacesResult efResult = ExtrudeFaces(mesh, eeResult.faces, true);
7 Translate(mesh, efResult.verts, Vec3(0, 0, 2));
```

Primjer koda 4.9: *Modeliranje kocke*

4.6.2 Modeliranje kompleksnog modela - jednostavna stolica



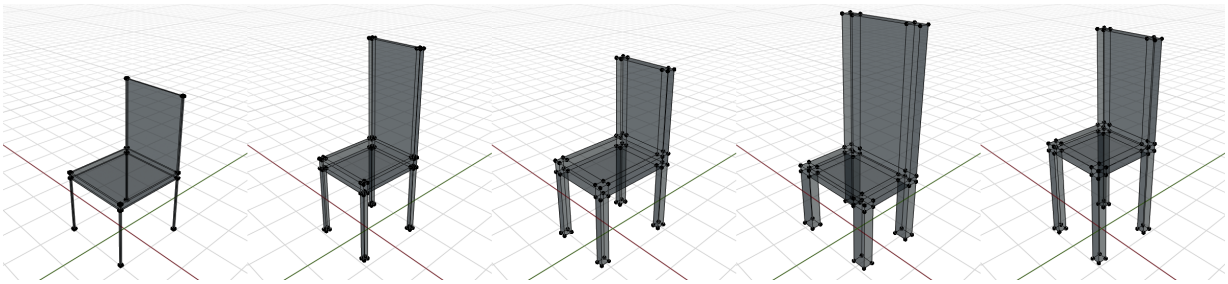
Sl. 4.23: *Modeliranje jednostavne stolice*

U ovom se primjeru *grid* podijeljen na 9 ravnomjernih dijelova koristi kao početna točka. Srednji *vertexi* se prvo skaliraju po x osi, a zatim i po y osi. Sjedalo stolca dobiveno je ekstrudiranjem *faceva* prema gore. Noge stolice modelirane su ekstrudiranjem faceva koji se nalaze u kutu prema dolje. Naslon stolice se također dobije ekstrudiranjem. U ovom se primjeru selekcija elemenata provodi putem lambda izraza i srednje vrijednosti koordinata pojedinih *faceva*.

```
1  CreateGridResult grid = CreateGrid(mesh, 2, 2, 0.5, 0.5);
2  vector<Vert*> vx = mesh->Verts([](const Vert* v) { return abs(v->co.x) < 0.5 / 3; });
3  Scale(mesh, vx, Vec3(0, 0, 0), Vec3(2.5, 1, 1));
4  vector<Vert*> vy = mesh->Verts([](const Vert* v) { return abs(v->co.y) < 0.5 / 3; });
5  Scale(mesh, vy, Vec3(0, 0, 0), Vec3(1, 2.5, 1));
6  vector<Face*> seatFaces = mesh->Faces();
7  vector<Face*> legFaces = mesh->Faces([](const Face* f) { return abs(f->CalcCenterAverage
       ↪ ().x) > 0.01 && abs(f->CalcCenterAverage().y) > 0.01; });
8  ExtrudeFaceRegionResult seat = ExtrudeFaceRegion(mesh, seatFaces, true);
9  Translate(mesh, seat.verts, Vec3(0, 0, 0.05));
10 ExtrudeFacesResult legs = ExtrudeFaces(mesh, legFaces, false);
11 Translate(mesh, legs.verts, Vec3(0, 0, -0.5));
12 vector<Face*> backFaces = mesh->Faces([](const Face* f) { return f->CalcCenterAverage().
       ↪ y < -0.5 / 3 && f->CalcCenterAverage().z > 0.05 / 1.5; });
13 ExtrudeFaceRegionResult back = ExtrudeFaceRegion(mesh, backFaces, false);
14 Translate(mesh, back.verts, Vec3(0, 0, 0.5));
```

Primjer koda 4.10: *Modeliranje jednostavne stolice*

4.6.3 Parametrizirano modeliranje - jednostavna stolica



Sl. 4.24: *Varijacije modela stolice, generirane koristeći istu funkciju uz drugačiji seed*

U prethodnom primjeru model stolice je generiran koristeći *hard-coded* vrijednosti. Neke od tih vrijednosti moguće je izdvojiti u zasebnu klasu i generirati pseudo-nasumično, gdje će svaka od izdvojenih vrijednosti biti generirana u određenom rasponu. Kod za generiranje stolice može se zatim izdvojiti u zasebnu funkciju koja kao parametar prima nasumični *seed* ili parametre za generiranje, a kao rezultat vraća pokazivač na generirani model.

```
class ChairParams {
  public:
    float width;
    float depth; //...
    ChairParams(int seed) {
        std::default_random_engine generator(seed);
        std::uniform_real_distribution<double> distribution(0.0, 1.0);
        width = 0.4 + distribution(generator) * 0.2;
        depth = 0.4 + distribution(generator) * 0.2; //...
    }
};
```

Primjer koda 4.11: *Implementacija ChairParameters klase*

```
Mesh* GenerateChair(ChairParams params) {
    Mesh* mesh = new Mesh();
    auto grid = CreateGrid(mesh, 2, 2, params.width, params.depth);
    ...
    return mesh;
}
```

Primjer koda 4.12: *Modeliranje stolice koristeći parametre*

Za modeliranje stolica prikazanih na slici 4.24, širina, dubina, visina stolice, visina naslona, debljina stolice i debljina nogu izdvojeni su kao parametri. Dodatni parametri mogu biti izdvojeni kako bi se povećala raznolikost generiranih modela. npr. može se dodati parametar "*imaNaslon*" koji će definirati ima li stolica naslon ili ne.

5. ZAKLJUČAK

Glavni doprinos je razvoj nove biblioteke za 3D modeliranje fokusirane na proceduralni pristup. Sposobnosti i prikladnost biblioteke za proceduralno modeliranje demonstrirani su kroz primjere u poglavlju 6.

Zanimljivo je promatrati različite kompromise koji se pojavljuju prilikom razvoja biblioteka za 3D modeliranje. Jedan od kompromisa je dizajn programskog sučelja (API) - treba li krajnjem korisniku dati pristup svim svojstvima strukture podataka prilikom razvoja operatora visoke razine kako bi se omogućilo korištenje efikasnijih algoritama, što dolazi sa značajno manjom sigurnosti i stabilnosti, ili enkapsulirati strukturu podataka iza operatora niske razine što pojednostavljuje implementaciju operatora visoke razine ali uz lošije performanse i manju mogućnost optimizacije algoritama. Još se kompromisa pojavljuje pri dizajnu strukture podataka. Načelno, strukture podataka koje mogu jednoznačno pohraniti kompliciranije slučajeve (npr. *multigraph, self-loop*) zahtijevaju više memorije za pohranu te kompleksnije algoritme, ali povećavaju fleksibilnost. Također, odabir između jednostruko i dvostruko povezanih lista utječe na memorijske i računske performanse - jednostruke liste imaju bolje memorijske performanse (jer zahtijevaju manje pokazivača), ali brisanje elemenata iz sredine jednostruke liste zahtjeva linearno pretraživanje i smanjuje računske performanse.

Daljnji razvoj biblioteke trebao bi se fokusirati na poboljšanje performansi, koje primarno može biti ostvareno kroz *memory pool* implementaciju za sve elemente. Takva bi implementacija rezultirala bržom alokacijom i dealokacijom memorije (ubrzanje operatora do 30%). Uz to, svojstva koja se koriste za održavanje liste svih elemenata unutar modela također bi se mogla ukloniti, što bi rezultiralo boljim memorijskim performansama. Uz to, moguće je implementirati još operatora visoke razine što bi omogućilo jednostavniju i bržu implementaciju kompleksnih funkcija za modeliranje. Ako je potrebno, moguće je implementirati i dodatne operatore niske razine kako bi se omogućila optimizacija algoritama u operatorima visoke razine. Podrška za proizvoljne atribute za svaki element može se implementirati, te bi bila značajno fleksibilnija od trenutnog (ograničenog) sustava koji koristi zastavice. Također, moguće je i dodati podršku za jedan od skriptnih jezika, gdje bi *Python* bio dobar izbor zbog svoje popularnosti i jednostavnosti.

Implementirana biblioteka može služiti kao osnova za razvoj novih aplikacija za 3D modeliranje ili drugih aplikacija koje zahtijevaju neku razinu podrške za generiranje 3D modela. Biblioteka također može biti korištena za proceduralno generiranje različitih 3D modela unutar *game-enginea* i video igara. Također može koristiti i za razvoj novih tehnika pohrane 3D modela, koji ne bi pohranjivali koordinate točaka, već operatore i njihove ulaze i izlaze u obliku grafa, stoga ili jednostavnog skriptnog jezika, što bi za rezultat imalo bolje memorijske performanse pri pohrani modela.

LITERATURA

[1] Sven Havemann and Dieter W Fellner. Generative mesh modeling Technical Report TUBS-CG-2003-01 Generative Mesh Modeling Institute of Computer Graphics University of Technology. (May), 2014.

[2] Telmo Adão, Luís Pádua, Pedro Marques, Joaquim João Sousa, Emanuel Peres, and Luís Magalhães. Procedural modeling of buildings composed of arbitrarily-shaped floor-plans: Background, progress, contributions and challenges of a methodology oriented to cultural heritage. *Computers*, 8(2), 2019.

[3] David S. Ebert. Texturing & modeling: a procedural approach. page 687, 2003.

[4] Stefan Lienhard. Visualization , Adaptation , and Transformation of Procedural Grammars par. 7627, 2017.

[5] Aristid Lindenmayer. Mathematical models for cellular interactions in development i. filaments with one-sided inputs. *Journal of theoretical biology*, 18(3):280–299, 1968.

[6] Miłosz Makowski, Torsten Hädrich, Jan Scheffczyk, Dominik L. Michels, Sören Pirk, and Wojtek Pałubicki. Synthetic silviculture: Multi-scale modeling of plant ecosystems. *ACM Transactions on Graphics*, 38(4), 2019.

[7] George Stiny and James Gips. Shape grammars and the generative specification of painting and sculpture. In *IFIP Congress (2)*, volume 2, pages 125–135, 1971.

[8] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.

[9] Nicholas M Patrikalakis and Takashi Maekawa. *Shape interrogation for computer aided design and manufacturing*, volume 15. Springer, 2002.

[10] Marie Saldaña. An Integrated Approach to the Procedural Modeling of Ancient Cities and Buildings. *Digital Scholarship in the Humanities*, 30:148–163, 06 2015.

[11] OPEN CASCADE SAS. Open cascade technology. https://dev.opencascade.org/, 2021. Pristup: 26.6.2021.

[12] Jan Mobius and Leif Kobbelt. Openflipper: An open source geometry processing and rendering framework. In Jean-Daniel Boissonnat, Patrick Chenin, Albert Cohen, Christian Gout, Tom Lyche, Marie-Laurence Mazure, and Larry Schumaker, editors, *Curves and Surfaces*, volume 6920 of *Lecture Notes in Computer Science*, pages 488–500. Springer Berlin / Heidelberg, 2012.

[13] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.2.2 edition, 2021.

[14] Lutz Kettner. 3D polyhedral surface. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.2.2 edition, 2021.

[15] Marius Kintel. Openscad. https://openscad.org/, 2021. Pristup: 26.6.2021.

[16] Alec Jacobson, Daniele Panozzo, et al. libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/.

[17] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008.

[18] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[19] Blender Online Community. *Source/Modeling/BMesh/Design - Blender Developer Wiki*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2021. Pristup: 26.6.2021.

[20] Autodesk Inc. Maya. https://www.autodesk.com/products/maya/overview/, 2021. Pristup: 26.6.2021.

[21] Autodesk Inc. 3dsmax. https://www.autodesk.com/products/3ds-max/overview/, 2021. Pristup: 26.6.2021.

[22] The Foundry Visionmongers Limited. Modo. https://www.foundry.com/products/modo/, 2021. Pristup: 26.6.2021.

[23] Spatial Corp. Dassault Systèmes. 3d acis modeler. https://www.spatial.com/products/3d-acis-modeling, 2021. Pristup: 26.6.2021.

[24] Inc. IntegrityWare. Solids++. http://www.integrityware.com/solids.html, 2021. Pristup: 26.6.2021.

[25] Siemens. Parasolid. https://www.plm.automation.siemens.com/global/en/products/plm-components/parasolid.html, 2021. Pristup: 26.6.2021.

[26] Solid Modeling Solutions. Smlib. https://smlib.com/, 2021. Pristup: 26.6.2021.

[27] SideFX. Houdini. https://www.sidefx.com/products/houdini/, 2021. Pristup: 26.6.2021.

[28] Aikaterini Chatzivasileiadi, Nicholas Mario Wardhana, Wassim Jabi, Robert Aish, and Simon Lannon. Characteristics of 3D solid modeling software libraries for non-manifold modeling. *Computer-Aided Design and Applications*, 16(3):496–518, 2019.

[29] Bruce G. Baumgart. Winged Edge Polyhedron Representation. *National Technical Information Service*, (October), 1972.

[30] Kevin Weiler. Boundary Graph Operators for Non-Manifold Geometric Modeling Topology Representations. In *Geometric Modeling for CAD Applications*, pages 37–66. 1988.

[31] Kevin Weiler. The radial edge structure: a topological representation for non-manifold geometric boundary modeling. *Geometric modeling for CAD applications*, pages 3–36, 1988.

[32] Sven Havemann. Generative mesh modeling, PhD Dissertation. 2005.

[33] Pedro Brandão Silva, António Coelho, Rui Rodrigues, and A. Augusto Sousa. A procedural geometry modeling API. *GRAPP 2012 IVAPP 2012 - Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications*, pages 129–134, 2012.

[34] Charles Eastman and Kevin Weiler. Geometric Modeling Using the Euler Operators. In *IEE Conference Publication*, pages 248–259, 1979.

SAŽETAK

Proceduralno 3D modeliranje je pojam koji obuhvaća ratne načine za generiranje 3D modela bez korisničkog unosa i pruža brojne prednosti naspram ručnog 3D modeliranja. Iako postoje komercijalni programi i programi otvorenog koda s mogućnostima poligonalnog proceduralnog 3D modeliranja, njihova primjena u aplikacijama otvorenog koda je ograničena zbog licenci i tehničkih ograničenja. U ovom radu predstavljena je nova biblioteka i pripadajuće programsko sučelje za 3D modeliranje temeljeno na poligonalnim mrežama, te su opisani proces razvoja, ciljevi i ključne značajke biblioteke. Fokus je na proceduralnom pristupu, mogućnosti proširivanja programskog sučelja i ugradnje u druge aplikacije. Biblioteka i njen izvorni kod su javno dostupni i objavljeni pod licencom otvorenog koda. Prikladnost implementiranog rješenja u kontekstu proceduralnog 3D modeliranja demonstrirana je kroz nekoliko jednostavnih primjera.

Ključne riječi: 3D modeliranje, Jezgra za poligonalno modeliranje, Poligonalno modeliranje, Proceduralno generiranje, Računalna geometrija.

ABSTRACT

Procedural 3D modeling is a term which encompasses multiple ways to generate 3D models without user interaction and provides several advantages compared to manual modeling. Altough there exist commercial and open-source software and libraries with polygonal procedural 3D modeling capabilities, their use in open-source applications is limited due to both technical and license limitations. In this paper a new library and the appropriate application programming interface for polygonal 3D modeling is developed and showcased. The development process, design goals and key features of the library are described. The library is focused on the procedural approach and the ability to extend the interface and embed the API into other applications. The library and it's source code are publicaly available and released under an open-source license. The suitability of the library in the context of procedural 3D modeling is demonstrated through several basic examples.

Keywords: 3D modeling, Computational geometry, Polygon mesh modeling, Polygonal modeling kernel, Procedural generation.

ŽIVOTOPIS

Luka Šimić rođen je 4. Rujna 1997. godine u Osijeku. Nakon završenog osnovnoškolskog i srednjoškolskog obrazovanja, 2016. godine upisao je preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Nakon završenog preddiplomskog studija 2019. godine, na istom fakultetu upisuje diplomski studij računarstva.

PRILOG 1 - DOSTUPNI MATERIJALI I IZVORNI KOD

Izvorni kod objavljen je pod MIT licencom i dostupan je u repozitoriju kojem se može pristupiti na sljedećoj poveznici:

https://github.com/lsimic/AobaAPI

Izvorni kod za primjere korištenja dostupan je u sljedećem repozitoriju:

https://github.com/lsimic/AobaExamples