



УНИВЕРЗИТЕТ У НОВОМ САДУ  
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У  
НОВОМ САДУ

---



Јован Срданов

# **Интерактивно графичко окружење за демонстрацију рендеровања тродимензионалних објеката**

ДИПЛОМСКИ РАД  
- Основне академске студије -


Нови Сад, 2023.

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, <b>РБР:</b>	
Идентификациони број, <b>ИБР:</b>	
Тип документације, <b>ТД:</b>	Монографска публикација
Тип записа, <b>ТЗ:</b>	Текстуални штампани документ/ЦД
Врста рада, <b>ВР:</b>	Завршни-bachelor рад
Аутор, <b>АУ:</b>	Јован Срданов
Ментор, <b>МН:</b>	Проф. др Драган Иветић
Наслов рада, <b>НР:</b>	Интерактивно графичко окружење за демонстрацију рендеровања тродимензионалних објеката
Језик публикације, <b>ЈП:</b>	Српски(ћирилица)/Српски (ћирилица)
Језик извода, <b>ЈИ:</b>	Српски/Енглески
Земља публикавања, <b>ЗП:</b>	Србија
Уже географско подручје, <b>УГП:</b>	Војводина
Година, <b>ГО:</b>	2023
Издавач, <b>ИЗ:</b>	Ауторски репринт
Место и адреса, <b>МА:</b>	Факултет Техничких Наука (ФТН), Д. Обрадовића 6, 21000 Нови Сад
Физички опис рада, <b>ФО:</b> (поглавља/страна/ цитата/табела/слика/графика/прилога)	8/42/9/1/26/0/1
Научна област, <b>НО:</b>	Електротехничко и рачунарско инжењерство
Научна дисциплина, <b>НД:</b>	Примењене рачунарске науке и информатика
Предметна одредница/Кључне речи, <b>ПО:</b>	Рачунарска графика, темена у тродимезионом простору, нормале темена, усредењивање нормала, сенчење, полигони, Фонг, Гуоро,
<b>УДК</b>	
Чува се, <b>ЧУ:</b>	Библиотека ФТН, Д. Обрадовића 6, 21000 Нови Сад
Важна напомена, <b>ВН:</b>	
Извод, <b>ИЗ:</b>	Рад се бави разматрањем начина како интерактивно и постепено корисницима приказати разлике у техникама сечења тродимензионалног простора у рачунарској графици. За решавање овог проблема имплементирано је коришћење <i>OpenGL</i> -а у програмском језику <i>C++</i> . Као крајњи резултат, развијена је функционална апликација која се користи за анализу и поређење различитих техника сенчења.
Датум прихватања теме, <b>ДП:</b>	
Датум одбране, <b>ДО:</b>	
Чланови комисије, <b>КО:</b>	Председник: Др Александар Купусинац, ред. проф.
	Члан: Др Дуња Врбашки, доцент
	Члан, ментор: Др Драган Иветић, ред. проф.
	Потпис ментора

## KEY WORDS DOCUMENTATION

Accession number, <b>ANO</b> :		
Identification number, <b>INO</b> :		
Document type, <b>DT</b> :		Monographic publication
Type of record, <b>TR</b> :		Textual material, printed/CD
Contents code, <b>CC</b> :		Bachelor thesis
Author, <b>AU</b> :		Jovan Srdanov
Mentor, <b>MN</b> :		Dragan Ivetić, PhD, full professor
Title, <b>TI</b> :		Interactive graphical environment for demonstrating the rendering of three-dimensional objects
Language of text, <b>LT</b> :		Serbian (cyrillic script)/Serbian (latin script)
Language of abstract, <b>LA</b> :		Serbian/English
Country of publication, <b>CP</b> :		Serbia
Locality of publication, <b>LP</b> :		Vojvodina
Publication year, <b>PY</b> :		2010
Publisher, <b>PB</b> :		Author reprint
Publication place, <b>PP</b> :		Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad
Physical description, <b>PD</b> : (chapters/pages/ref./tables/pictures/graphs/appendixes)		8/42/9/1/26/0/1
Scientific field, <b>SF</b> :		Electrical and computer engineering
Scientific discipline, <b>SD</b> :		Applied computer science and informatics
Subject/Key words, <b>S/KW</b> :		Computer graphics, vertices in 3D space, vertex normals, normal averaging, shading, polygons, Phong, Gouraud, OpenGL
<b>UC</b>		
Holding data, <b>HD</b> :		Library of the Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad
Note, <b>N</b> :		
Abstract, <b>AB</b> :		The work focuses on exploring ways to interactively and gradually present users with differences shading techniques of three-dimensional space in computer graphics. To address this problem, OpenGL was implemented using the C++ programming language. As a final result, a functional application was developed for analyzing and comparing various shading techniques.
Accepted by the Scientific Board on, <b>ASB</b> :		
Defended on, <b>DE</b> :		
Defended Board, <b>DB</b> :	President:	Dr Aleksandar Kupusinac, full professor
	Member:	Dr Dunja Vrbaški, PhD, assist. prof.
	Member, Mentor:	Dragan Ivetić, PhD, full professor
		Menthor's sign

	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	<b>ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА</b>	Датум:

(Податке уноси предметни наставник - ментор)

Врста студија:	<input type="checkbox"/> Основне академске студије
Студијски програм:	РАЧУНАРСТВО И АУТОМАТИКА
Руководилац студијског програма:	Проф. др Милан Рапаић

Студент:	Јован Срданов	Број индекса:	RA 145/2019
Област:	Рачунарска графика		
Ментор:	Проф. др Драган Иветић		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;
- литература

### НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:

**Интерактивно графичко окружење за демонстрацију рендеровања тродимензионалних објеката**

### ТЕКСТ ЗАДАТКА:

Имплементирати интерактивно графичко окружење које ће визуелизовати сваку фазу рендеровања тродимензионалних објекта са слободним подешавањем свих параметара сваког процеса.

Руководилац студијског програма:	Ментор рада:

Примерак за: ☐ - Студента; ☐ - Ментора

## Spisak korišćenih skraćenica

VAO	Vertex Array Object
VBO	Vertex Buffer Object
EBO	Element Buffer Object

## Sadržaj

1. Uvod .....	4
2. Teoretske osnove .....	5
2.1 Temena.....	5
2.2 Model .....	5
2.3 Normale temena .....	6
2.4 Usrednjene normale temena .....	6
2.5 Izvori svetlosti.....	7
2.5.1 Tačkasti izvor svetlosti.....	7
2.5.2 Reflektorski izvor svetlosti.....	7
2.5.3 Direkcionni izvor svetlosti.....	7
2.6 Svetlosne komponente.....	7
2.6.1 Ambijentalna svetlost.....	8
2.6.2 Difuzna svetlost .....	8
2.6.3 Spekularna svetlost .....	8
2.7 Fongova svetlosna jednačina .....	8
2.8 Senčenje .....	9
2.8.1 Konstantno senčenje .....	9
2.8.2 Guroovo senčenje.....	9
2.8.3 Fongovo senčenje.....	10
3. Implementacija.....	11
3.1 Tehnologije.....	11
3.1.2 C++.....	11
3.1.2 <i>OpenGL, GLEW, GLFW3</i> .....	11
3.1.4 <i>VAO, VBO, EBO</i> .....	12
3.1.5 <i>Dear ImGui</i> .....	12
3.2 Učitavanje i obrada modela i <i>mesh</i> -a .....	12
3.3 Šejderi.....	18
3.8 Rednerovanje modela u zavisnosti od odabranog moda .....	19
4. Primer korišćenja.....	21
4.1 Korisničke komande .....	21
4.2 Modovi .....	21
4.2.1 Mod 01 - Temena .....	22

4.2.2 Mod 02 - Trouglovi .....	23
4.2.3 Mod 03 - Popunjeni trouglovi bez vidljive mreže.....	23
4.2.4 Mod 04 - Popunjeni trouglovi sa vidljivom mrežom .....	24
4.2.5 Mod 05 - Sve normale .....	24
4.2.6 Mod 06 - Usrednjene normale .....	25
4.2.7 Mod 07- Senčenja.....	25
4.2.8 Mod 08 - Teksture .....	27
4.3 Baterijska lampa iz prvog lica .....	27
4.4 Materijal modela na osnovu parametara svetlosnih komponenti.....	28
4.5 Poređenje usrednjenih i svih normala .....	29
4.6 Poređenje različitih tehnika senčenja .....	29
5. Ograničenja i unapređenja .....	31
5.1 Raznovrsnost modela i tekstura .....	31
5.2 Specifičnost parametara.....	31
5.3 Algoritam za izračunavanje usrednjenih normala.....	31
5.4 Opcija za čuvanje stanja aplikacije .....	31
5.5 Prelazak na internet .....	31
6. Zaključak .....	32
7. Literatura .....	33
8. Dodatak A .....	34
9. Podaci o kandidatu .....	38

## 1. Uvod

Računarska grafika je disciplina koja se bavi stvaranjem, manipulacijom i prikazom vizuelnih elemenata na računarima. Ona ima široku primenu u različitim industrijama i svakodnevnom životu kao i značajan uticaj na način na koji komuniciramo i interagujemo sa digitalnim svetom. Glavna svrha računarske grafike je stvaranje i manipulacija modela i animacija putem računara. Ova tehnologija omogućava korisnicima da stvaraju realistične vizuelne efekte, simulacije, video igre, medicinske vizualizacije, arhitektonske planove i mnogo više.

Mnogi istraživački radovi koji se trude da unaprede razumevanje ključnih koncepata računarske grafike često su preplavljeni apstraktnim matematičkim i fizičkim formalizmima i zanemaruju aspekt korisničke interakcije. Takođe, često nedostaje dinamički prikaz tranzicije između različitih koncepata u stvarnom vremenu. Ovi radovi, iako vredni i duboko analitički, često propuštaju da pruže korisnicima intuitivno iskustvo učenja i istraživanja računarske grafike. Nedostatak interaktivnosti i vizualizacije u stvarnom vremenu može otežati razumevanje i primenu ovih složenih koncepata.

Cilj ovog rada jeste implementacija aplikacije koja će korisnicima pružiti interaktivno iskustvo sa visokom fleksibilnošću u kontrolisanju, omogućavajući im dublje razumevanje elemenata računarske grafike. Aplikacija će omogućiti korisnicima da postepeno istražuju složene aspekte računarske grafike, počevši od osnovnih koncepta i postepeno prelazeći na sve složenije. Poseban fokus aplikacije će biti na analizi različitih tehnika senčenja i njihovih efekata na vizuelni izgled renderovanih modela. Takođe, aplikacija će omogućiti korisnicima da eksperimentišu sa različitim parametrima osvetljenja i sagledaju kako ti parametri utiču na konačni izgled renderovanog modela.

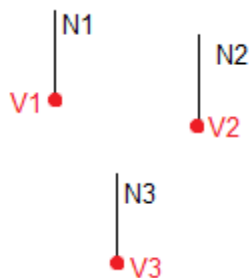
Rad je sačinjen od 5 celina. Prva celina će objašnjavati osnovne i složene koncepte računarske grafike. Fokus druge celine biće na opisu korišćenih tehnologija prilikom implementacije softverskog rešenja kao i razrađivanje detalja same implementacije. Treća celina se bavi pojašnjenjem upotrebe realizovane aplikacije kao i mogućim koracima sagledavanja i analiziranja koncepta računarske grafike implementiranih unutar softverskog rešenja. U četvrtoj celini akcenat je na mogućim ograničenjima i potencijalnim unapređenjima zadatka. U petoj celini će biti izloženi zaključci donešeni nakon izrade zadatka.



## 2. Teoretske osnove

### 2.1 Temena

U računarskoj grafici, temena (ili verteksi) su osnovne komponente koje definišu geometrijske oblike i strukturu trodimenzionalnog modela. Svako teme predstavlja tačku u trodimenzionalnom prostoru, a kombinacija ovih tačaka čini kompleksne objekte poput modela karaktera, pejzaža, ili objekata u simulacijama i video igrama. Primer temena sa svojim normalama je prikazan na slici 1.



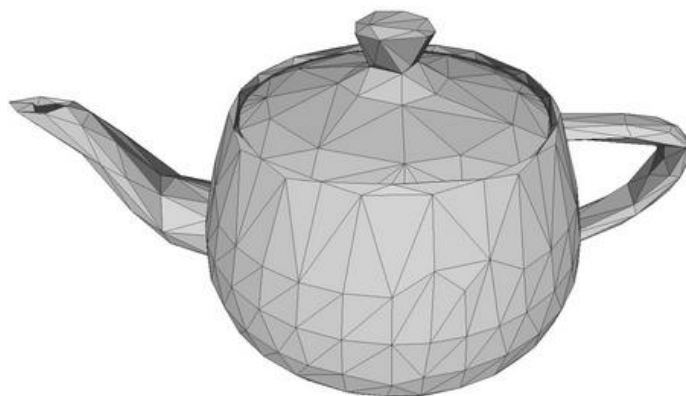
Slika 1 Temena sa normalama

Temena se koriste za opisivanje pozicije, boje, teksture i drugih svojstava svakog pojedinačnog elementa modela. Dodatno, temena mogu sadržati informacije o normalama, koje su vektori koji označavaju smer površine i koriste se za pravilno osvetljavanje i senčenje.

Ključna uloga temena je u stvaranju trouglova, osnovnih grafičkih elemenata koji se koriste za prikazivanje trodimenzionalnih objekata na ekranu. Transformacije temena, kao što su rotacija, translacija i skaliranje, imaju ključnu ulogu u pojednostavljenju prikaza objekata u različitim situacijama.

### 2.2 Model

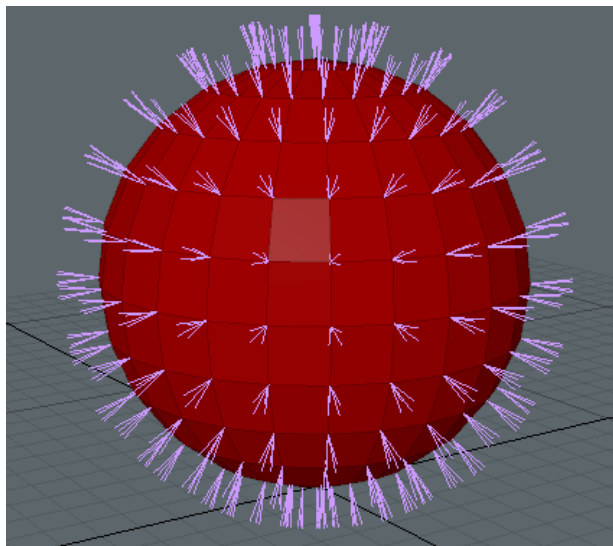
Model, u kontekstu računarstva i trodimenzionalne grafike, predstavlja digitalnu reprezentaciju trodimenzionalnog objekta ili scene. Ovi modeli se mogu sastojati od mreža (*mesh*) koje se sastoje od temena povezanih trouglovima. Trouglovi su trostrane geometrijske figure koje povezuju temena, čineći osnovnu građevnu jedinicu modela dok su ivice linije koje povezuju temena. Ovaj složeni skup temena, trouglova i ivica omogućava računarima da prikažu i manipuliraju trodimenzionalnim objektima u digitalnom okruženju. Na slici 2 prikazan je model čajnika [1].



Slika 2 Model čajnika

## 2.3 Normale temena

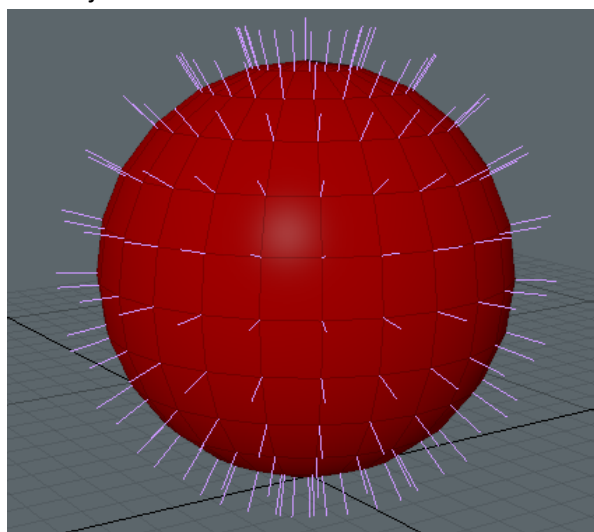
Normale temena su vektori koji se često koriste u računarskoj grafici kako bi se odredila orijentacija površine na svakom temenu unutar trodimenzionalnog modela. Ovi vektori pružaju informacije o tome kako svako teme reaguje na svetlost i kako će biti osenčeno na renderovanom objektu. Na slici 3 prikazan je model lopte sa svim normalama koje potiču iz temena te lopte.



Slika 3 Model lopte sa neusrednjenim normalama temena [2]

## 2.4 Usrednjene normale temena

Usrednjene normale se u računarskoj grafici koriste za simuliranje detalja na površinama objekata bez povećavanja geometrijske složenosti tih objekata. Usrednjene normale su posebno važne za efikasno računanje osvetljenja na površini objekta. Kada se objekti u trodimenzionalnoj sceni renderuju, usrednjene normale se koriste za izračunavanje intenziteta svetla koji pada na svaki piksel. To omogućava realističnije modeliranje refleksija svetlosti na površini objekta i stvara efekat glatkog senčenja. Bez usrednjenih normala postiže se samo efekat konstantnog senčenja. Proces usrednjavanja normala u temenima zahteva da se sve normale koje se povezuju s jednim temenom (temenom koje se deli između više trouglova) vektorski saberu, a zatim rezultujući vektor normalizuje. Model sa usrednjenim normalama prikazan je na slici 4.



Slika 4 Model lopte sa usrednjenim normalama temena [2]

## 2.5 Izvori svetlosti

Izvori svetlosti u računarskoj grafici su simulirani izvori svetlosti koji se koriste za modeliranje osvetljenja u trodimenzionalnim scenama i objektima. Oni igraju ključnu ulogu u stvaranju realističnih vizualnih efekata. Pravilno modeliranje osvetljenja pomaže u postizanju realizma i atmosfere. Na slici 5 su prikazani primeri tačkastog, reflektorskog i direkcionog izvora svetla.



Slika 5 Primeri različitih izvora svetlosti [3]

### 2.5.1 Tačkasti izvor svetlosti

Tačkasti izvor svetlosti simulira svetlost koja dolazi iz jedne tačke u prostoru i širi se u svim pravcima. Ovakav izvor svetlosti često se koristi kako bi se postigla određena atmosfera, stvarala kontrastna senčenja i dodavala dubina u scenama. Takođe, intenzitet svetlosti opada sa udaljenošću od tačkastog izvora, što znači da objekti dalji od izvora izgledaju manje osvetljeni.

### 2.5.2 Reflektorski izvor svetlosti

Reflektorski izvor svetlosti simulira svetlosne izvore s karakterističnom usmerenošću i intenzitetom. Za razliku od tačkastog izvora svetlosti koji emituje svetlost iz jedne tačke u svim pravcima, reflektorski izvor svetlosti fokusira svetlosne zrake u određenom smeru. Kroz ovaj proces, svetlost se fokusira i usmerava u određenom pravcu, stvarajući intenzivnije svetlosne snopove. Reflektorski izvori svetlosti često se koriste za naglašavanje određenih delova scene, postizanje dramatičnih senki i stvaranje visokokontrastnih efekata.

### 2.5.3 Direkciono izvor svetlosti

Direkciono izvor svetlosti karakteriše snažan i usmeren snop svetlosti koji se širi u određenom smeru. Za razliku od tačkastog izvora svetlosti koji emituje svetlost iz jedne tačke ili reflektorskog izvora sa koncentrisanim snopom svetlosti, direkciono izvor svetlosti je beskonačno udaljen i svetlost dolazi iz jednog neprekidnog pravca. Glavna karakteristika direkcionog izvora svetlosti je da svi svetlosni zraci dolaze iz istog smera i paralelni su jedni drugima. Intenzitet ovog svetla se ne menja u zavisnosti od udaljenosti.

## 2.6 Svetlosne komponente

Svetlosne komponente su ključni elementi u oblasti računarske grafike, igrajući značajnu ulogu u simulaciji osvetljenja. Ove komponente se koriste za modeliranje različitih aspekata svetlosnog izvora i njegovog uticaja na scenu. Tri osnovne svetlosne komponente su ambijentalna, difuzna i spekularna, prikazane na slici 6.



Slika 6 Svetlosne komponente [3]

### 2.6.1 Ambijentalna svetlost

Ambijentalna svetlost predstavlja konstantnu osvetljenost koja dolazi iz svih pravaca i ravnomerno obasjava scenu ili objekat. Ova komponenta simulira svetlost koja se reflektuje od okoline i omogućava da objekti budu vidljivi iako nisu direktno izloženi svetlosnom izvoru. Ambijentalna komponenta pomaže u izbegavanju potpuno crnih objekata i doprinosi ravnoteži osvetljenja u sceni.

### 2.6.2 Difuzna svetlost

Difuzna svetlost opisuje kako se svetlost ravnomerno raspršuje i osvetljava objekte kada svetlosni zraci padaju na površinu. Difuzna komponenta refleksije uzrokuje da svaka tačka površine emituje svetlost u svim pravcima pod istim uglom. Rezultat je meka i jednolična osvetljenost bez sjaja ili oštih kontrasta.

### 2.6.3 Spekularna svetlost

Spekularna svetlost simulira refleksiju svetlosti na glatkim površinama. Ova komponenta je odgovorna za sjajne i reflektujuće površine. Spekularna svetlost stvara blistave tačke na površini objekata koje se pomeraju sa promenom gledišta i stvaraju efekat sjaja.

## 2.7 Fongova svetlosna jednačina

Fongova svetlosna jednačina (1), takođe poznata kao Fongov model osvetljenja, predstavlja osnovni model koji se koristi u računarskoj grafici kako bi se simuliralo osvetljenje na trodimenzionalnim objektima. Fongova svetlosna jednačina pomaže računarima da generišu realistične slike tako što uzima u obzir kako svetlost interaguje sa površinama objekata.

$$L_0(v) = k_a I_a + \min\left(\frac{1}{c_1 + c_2 d + c_3 d^2}, 1\right) \sum_{i \in \text{izvori}} k_d I_{di} \cos\theta + k_s I_{si} \cos^n_s(\alpha) \quad (1)$$

$L_0$  – intenzitet reflektovane svetlosti,  $v$  – vektor gledanja kamere ili posmatrača

$k_a$  – refleksioni koeficijent materijala za ambijentalno svetlo

$k_d$  – refleksioni koeficijent materijala za difuzno svetlo

$k_s$  – refleksioni koeficijent materijala za spekularno svetlo

$I_a$  – intenzitet ambijentalne komponente svetla

$I_d$  – intenzitet difuzne komponente svetla

$I_s$  – intenzitet spekularne komponente svetla

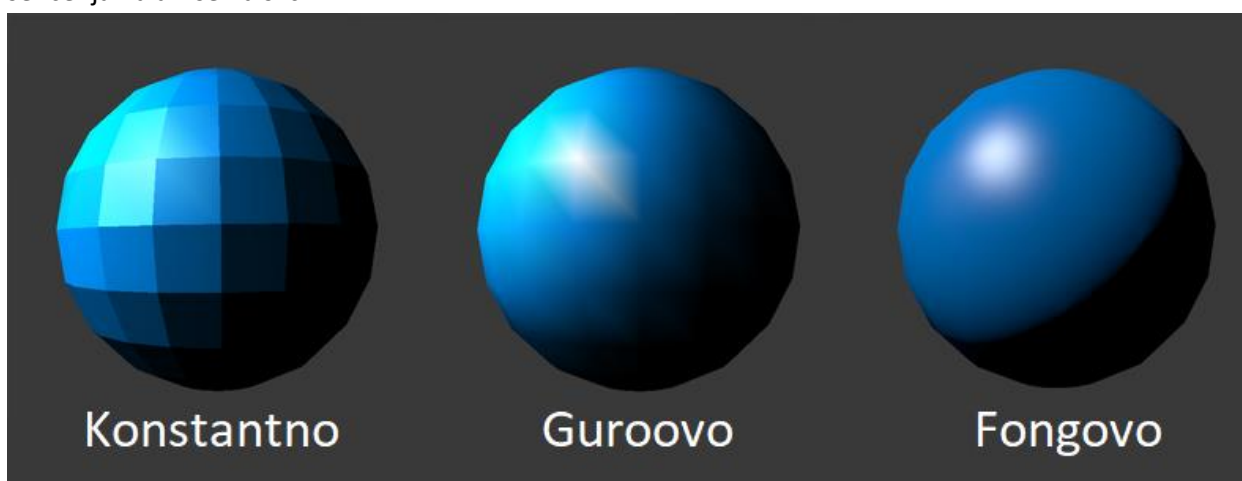
$c_1, c_2, c_3$  – koeficijenti konstantnog, linearnog i kvadratnog slabljenja svetla na osnovu udaljenosti od svetlosnog izvora

$\theta$  – upadni ugao,  $n_s$  – koeficijent sjajnosti

$\alpha$  – ugao između refleksionog vektora i vektora gledanja kamere ili posmatrača  $v$

## 2.8 Senčenje

Senčenje u računarskoj grafici je ključni proces koji omogućava prikaz trodimenzionalnih objekata na dvodimenzionalnom ekranu. Ovaj postupak se bavi određivanjem boja i tonova piksela na površinama objekata u simuliranom prostoru, kako bi se postigao efekat osvetljenja i dubine. Senčenje igra ključnu ulogu u realističnom prikazu grafičkih scena, doprinoseći percepciji oblika, materijala i svetlosti. Postoje različite tehnike senčenja kao što su konstantno senčenje, Guroovo senčenje i Fongovo senčenje. Svaka od ovih tehnika poseduje svoje prednosti i nedostatke. Prikaz lopte osenčene različitim tehnikama senčenja nalazi se na slici 7.



Slika 7 Uticaj različitih tehnika senčenja na model lopte [4]

### 2.8.1 Konstantno senčenje

Konstantno senčenje je tehnika koja se karakteriše odsustvom glatkih prelaza između površina. Svaki piksel na poligonu se oboji jednom bojom, bez obzira na njegovu orijentaciju u prostoru ili osvetljenje koje ga pogađa. Ova tehnika ističe nepovezanost između pojedinačnih poligona, što rezultira jasno vidljivim ivicama između njih. Konstantno senčenje se često koristi kada je brzina renderovanja ključna, jer zahteva manje računarske resurse u poređenju s tehnikama senčenja koje stvaraju glatke prelaze između poligona. Prilikom korišćenja ove tehnike ne koriste se usrednjene normale.

### 2.8.2 Guroovo senčenje

Princip rada Guroovog senčenja uključuje izračunavanje intenziteta svetlosti za svako teme u trodimenzionalnom prostoru. Potom, takvi intenziteti svetlosti se interpoliraju između temena kako bi se odredila boja svakog piksela unutar poligona. Ovaj proces rezultira glatkim prelazima boja na površini objekta, jer se boje postepeno menjaju ka unutrašnjosti, između temena. Guroovo senčenje može stvoriti realističan izgled objekata, ali ima svoje ograničenje. Ne uzima u obzir specifične refleksije na površini, što ga čini manje pogodnim za vrlo detaljne scene. Ipak, zbog svoje brzine i sposobnosti da

stvari glatke prelaze boja, Guroovo senčenje je i dalje široko primenjena tehnika. Prilikom korišćenja ove tehnike koriste se usrednjene normale.

### 2.8.3 Fongovo senčenje

Fongovo senčenje je još jedna napredna tehnika u renderovanju trodimenzionalnih objekata. Za razliku od Guroovog senčenja, Fongovo senčenje ne samo da uzima u obzir intenzitet svetlosti na temenima poligona, već i intenzitet svetlosti na svakom pikselu unutar poligona. Ovaj proces se postiže računanjem normale za svaki piksel i upoređivanjem te normale sa smerom svetlosnog izvora. Na osnovu ovih podataka, Fongovo senčenje određuje intenzitet svetlosti za svaki piksel, što rezultuje realističnijim prikazom refleksija na površini objekta. Fongovo senčenje ima prednost u stvaranju detaljnijih i realističnijih scena u poređenju sa Guroovim senčenjem. Ono uzima u obzir specifične karakteristike refleksija na površini, što ga čini idealnim za vrlo kompleksne i detaljne scene. Zbog veće računske složenosti, Fongovo senčenje može biti zahtevnije za izvođenje i zahteva više resursa. Prilikom korišćenja ove tehnike koriste se usrednjene normale.

## 3. Implementacija

U ovom poglavlju se detaljnije opisuju korišćene tehnologije u izradi softverskog rešenja, kao i sama implementacija rešenja. Ključne stavke koje je neophodno implementirati jesu renderovanje samog trodimenzionalnog modela, unos korisničkih komandi i prikaz grafičkog korisničkog interfejsa. Implementirano rešenje treba da se odlikuje visokom brzinom izvršavanja, efikasnom upotrebom resursa kao i preciznim i jasno vidljivim prikazom renderovane slike.

### 3.1 Tehnologije

#### 3.1.2 C++

C++ je popularan programski jezik koji se koristi za razvoj različitih vrsta softvera. Ovaj jezik je razvijen kao proširenje jezika C, čime su programerima pružene dodatne mogućnosti i apstrakcije u kodiranju.

Jedna od ključnih karakteristika jezika C++ je podrška za objektno-orijentisano programiranje koje omogućava programerima da organizuju svoj kod u objekte, koji sadrže podatke i funkcije koje rade s tim podacima. Ovo olakšava razvoj većih i kompleksnijih aplikacija, jer omogućava modularnost i ponovnu upotrebu koda. C++ je takođe poznat po svojoj efikasnosti i brzini izvršavanja. Ovaj jezik omogućava direktnu manipulaciju memorijom, što omogućava programerima da pišu efikasan kod za resursno-kritične aplikacije kao što su igre ili sistemski softver. *Standard Template Library* je još jedna ključna karakteristika jezika C++ jer sadrži mnoge korisne biblioteke i algoritme koji olakšavaju rad sa podacima, uključujući kontejnere poput vektora, liste i mape.

#### 3.1.2 OpenGL, GLEW, GLFW3

*OpenGL* (*Open Graphics Library*) je standardna specifikacija koja opisuje višepatformski programski interfejs za pisanje programa koji rade sa dvodimenzionalnom i trodimenzionalnom računarskom grafikom. Interfejs čini preko 250 različitih funkcija koje se mogu koristiti za izradu kompleksnih trodimenzionalnih scena od jednostavnih elemenata. Pored ovoga, mnogo češće se koristi u naučne svrhe, u projektima virtuelne stvarnosti kao i u raznim simulatorima. [5]

*OpenGL* se odlikuje platformskom nezavisnošću, što znači da se može koristiti na različitim operativnim sistemima, uključujući *Windows*, *Linux*, *macOS* i druge. Ovo ga čini popularnim izborom za razvoj aplikacija koje treba pokretati na različitim platformama.

*GLEW* (*OpenGL Extension Wrangler Library*) i *GLFW3* (*OpenGL Framework*) su dva često korišćena alata u razvoju grafičkih aplikacija i igara koje koriste *OpenGL* kao svoj grafički aplikativni programski interfejs. Ovi alati su posebno važni zbog svoje uloge u olakšavanju upravljanja ekstenzijama *OpenGL*-a i upravljanja prozorima i unosom u aplikacijama.

*GLEW* je biblioteka koja se koristi za lakše rukovanje *OpenGL* ekstenzijama. *OpenGL* ekstenzije pružaju dodatne funkcionalnosti koje idu izvan osnovnog *OpenGL* standarda. *GLEW* omogućava programerima da dinamički učitavaju ove ekstenzije, čime se olakšava pisanje prenosivog koda koji radi na različitim

platformama. Takođe, *GLEW* pruža lako korišćenje makroa za proveru dostupnosti ekstenzija i njihovih funkcija, čime se štedi vreme i resursi u razvoju.

Sa druge strane, *GLFW3* je biblioteka koja pomaže u upravljanju prozorima, unosom i događajima u aplikacijama koje koriste *OpenGL*. Ova biblioteka omogućava kreiranje prozora, postavljanje *OpenGL* konteksta i interakciju sa tastaturom, mišem i džojsticima. *GLFW3* takođe podržava više prozora, omogućava lak rad sa glavnom petljom i pruža podršku za različite platforme.

### 3.1.4 VAO, VBO, EBO

*VAO* je objekat u *OpenGL*-u koji se koristi za organizaciju i skladištenje informacija o temenima u grafičkom modelu. Svaki *VAO* sadrži konfiguracije atributa temena kao što su položaj, normale, boje i reference na *VBO* za svaki atribut. *VAO* omogućava efikasno organizovanje i upravljanje podacima temena za crtanje objekata.

*VBO* je *OpenGL* objekat koji služi za efikasno skladištenje i manipulaciju podacima temena. Umesto da se svako teme šalje zasebno prilikom crtanja objekta, koristi se *VBO* kako bi se podaci o temenima smestili u *OpenGL buffer* na grafičkoj kartici. Ovo značajno ubrzava proces crtanja objekata, jer se podaci mogu preneti na grafičku karticu samo jednom, a zatim se koristiti više puta.

*EBO* je takođe *OpenGL* objekat koji se koristi za optimizaciju crtanja objekata, posebno kada su u pitanju indeksi za temena. Umesto da se svako teme navede više puta u nizu indeksa kako bi se nacrtao objekat, *EBO* omogućava da se svaki vertex navede samo jednom. Zatim se koriste indeksi iz *EBO*-a kako bi se odredilo kako se verteksi povezuju prilikom formiranja trouglova. Ovo štedi memoriju i ubrzava crtanje kompleksnih objekata.

### 3.1.5 Dear ImGui

*Dear ImGui* je grafička biblioteka za korisnički interfejs za *C++*. Ona generiše optimizovane *buffer*-e sa temenima koji se mogu renderovati bilo kada u aplikaciji koja podržava trodimenzionalni *pipeline*. Ova biblioteka je brza, prenosiva, nezavisna od renderera i samostalna. [6]

*Dear ImGui* je dizajniran da osnaži programere da kreiraju alate za vizualizaciju i alate za kreiranje sadržaja. Ona favorizuje jednostavnost i produktivnost prema ovom cilju i nedostaju joj određene funkcionalnosti koje se često nalaze u višim nivoima biblioteka.

## 3.2 Učitavanje i obrada modela i *mesh*-a

Napravljene su klase *Model* i *Mesh* koje služe za obradu i manipulaciju učitanoj modela. Sastoje se iz odgovarajućih polja i metoda za učitavanje, obradu i renderovanje.

Za učitavanje se koristi biblioteka *Assimp* [7]. Učitavamo podatke koji se nalaze u učitanoj datoteci, obrađujemo ih i dodajemo u listu *mesh*-eva koja se nalazi u klasi *model*. Učitavanje je prikazano na listingu 1.



```

bool
Model::Load() {
    Assimp::Importer Importer;
    const aiScene *Scene = Importer.ReadFile(mFilename, POSTPROCESS_FLAGS);

    if (!Scene || Scene->mFlags & AI_SCENE_FLAGS_INCOMPLETE || !Scene->mRootNode) {
        std::cerr << "[Err] Failed to load model:" << std::endl <<
Importer.GetErrorString() << std::endl;
        return false;
    }
    mMeshes.reserve(Scene->mNumMeshes);
    for(unsigned MeshIdx = 0; MeshIdx < Scene->mNumMeshes; ++MeshIdx) {
        aiMesh* CurrAIMesh = Scene->mMeshes[MeshIdx];
        Mesh CurrMesh(CurrAIMesh, Scene->mMaterials[CurrAIMesh->mMaterialIndex],
mDirectory, MeshIdx+1);
        mMeshes.push_back(CurrMesh);
    }
    std::cout << mFilename << " Loaded " << mMeshes.size() << " meshes" << std::endl;
    return true;
}

```

#### Listing 1 Učitavanje modela

Proces obrade se vrši nad svakim *mesh*-om pojedinačno. Obrada je prikazana na listingu 2 i listingu 3.

```

Mesh::Mesh(const aiMesh* mesh, const aiMaterial* material, const std::string& resPath,
const int meshNumber) {
    processMesh(mesh, material, resPath, meshNumber);
}

```

#### Listing 2 Konstruktor *mesh-a*

```

void
Mesh::processMesh(const aiMesh* mesh, const aiMaterial* material, const std::string&
resPath, const int meshNumber) {
    const aiVector3D Zero3D(0.0f, 0.0f, 0.0f);
    processVertices(mesh, Zero3D);
    processIndices(mesh);
    processTextures(material, resPath);
    flatSetup();
    normalLinesSetup();
    std::string file_start;
    std::string numStr;
    std::string file_end;
    std::string filename;
    averagedNormalsSetup(meshNumber, file_start, numStr, file_end, filename);
    smoothSetup(file_start, numStr, file_end, filename);
}

```

#### Listing 3 Procesiranje *mesh-a*

Metoda za obradu *mesh-a* sastoji se od nekoliko pomoćnih funkcija koje igraju ključnu ulogu u pripremi trodimenzionalnih modela za dalju upotrebu. Metoda za procesiranje temena ima zadatak da popuni posebno polje u okviru *mesh-a* listom elemenata koji sadrže važne informacije o svakom temenu modela. Ovi elementi uključuju poziciju temena, informacije o normalama i teksturama koje su povezane sa svakim od tih temena.

Sledeća važna faza u procesiranju *mesh-a* je procesiranje indeksa. Ova metoda ima za cilj izračunavanje indeksa stranica u *mesh-u*, kao i njihov ukupan broj. Indeksi stranica definišu kako se temena povezuju kako bi se formirale površine *mesh-a*. Precizno izračunavanje ovih indeksa ključno je za ispravno prikazivanje i manipulaciju modelom. Iduća metoda vezana je za teksture i koristi se za učitavanje tekstura koje su eventualno priložene uz datoteku modela. Detalji ovih metoda su prikazani u listingu 4.

```

void Mesh::processVertices(const aiMesh* mesh, const aiVector3D Zero3D)
{
    for (unsigned VertexIndex = 0; VertexIndex < mesh->mNumVertices; ++VertexIndex)
    {
        std::vector<float> Position = { mesh->mVertices[VertexIndex].x, mesh->
mVertices[VertexIndex].y, mesh->mVertices[VertexIndex].z };
        mVertices_flat.insert(mVertices_flat.end(), Position.begin(),
Position.end());
        std::vector<float> Normals = { mesh->mNormals[VertexIndex].x, mesh->
mNormals[VertexIndex].y, mesh->mNormals[VertexIndex].z };
        mVertices_flat.insert(mVertices_flat.end(), Normals.begin(),
Normals.end());
        const aiVector3D* TexCoords = mesh->HasTextureCoords(0) ? &(mesh->
mTextureCoords[0][VertexIndex]) : &Zero3D;
        std::vector<float> UV = { TexCoords->x, TexCoords->y };
        mVertices_flat.insert(mVertices_flat.end(), UV.begin(), UV.end());
    }
}

void Mesh::processIndices(const aiMesh* mesh)
{
    for (unsigned FaceIndex = 0; FaceIndex < mesh->mNumFaces; ++FaceIndex) {
        const aiFace& Face = mesh->mFaces[FaceIndex];
        mIndices.push_back(Face.mIndices[0]);
        mIndices.push_back(Face.mIndices[1]);
        mIndices.push_back(Face.mIndices[2]);
    }
    mVertexCount = mVertices_flat.size() / 8;
    mIndexCount = mIndices.size();
}

void Mesh::processTextures(const aiMaterial* material, const std::string& resPath)
{
    mDiffuseTexture = loadMeshTexture(material, resPath, aiTextureType_DIFFUSE);
    mSpecularTexture = loadMeshTexture(material, resPath, aiTextureType_SPECULAR);
}

```

Listing 4 Metode za procesiranje temena, indeksa i teksture

Metoda `flatSetup()` popunjava odgovarajuće podatke neophodne za prikaz modela prilikom konstantnog senčenja. Metoda je prikazana u listingu 5.

```

void Mesh::flatSetup() {
    glGenVertexArrays(1, &mVAO_flat);
    glBindVertexArray(mVAO_flat);
    glGenBuffers(1, &mVBO_flat);
    glBindBuffer(GL_ARRAY_BUFFER, mVBO_flat);
    glBufferData(GL_ARRAY_BUFFER, mVertices_flat.size() * sizeof(float),
mVertices_flat.data(), GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 *
sizeof(float)));
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 *
sizeof(float)));
    glEnableVertexAttribArray(2);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    if (mIndexCount) {
        glGenBuffers(1, &mEBO_flat);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, mEBO_flat);
        glBufferData(GL_ELEMENT_ARRAY_BUFFER, mIndexCount * sizeof(float),
mIndices.data(), GL_STATIC_DRAW);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
    }
    glBindVertexArray(0);
}

```

Listing 5 Metode postavke modela za konstantno senčenje

Metoda `normalLinesSetup()` popunjava odgovarajuće podatke neophodne za prikaz normala temena. Metoda je prikazana u listingu 6.

```
void Mesh::normalLinesSetup()
{
    for (size_t i = 0; i < mVertices_flat.size(); i += 8) {
        float x = mVertices_flat[i];
        float y = mVertices_flat[i + 1];
        float z = mVertices_flat[i + 2];
        float nx = mVertices_flat[i + 3];
        float ny = mVertices_flat[i + 4];
        float nz = mVertices_flat[i + 5];
        glm::vec3 start_point(x, y, z);
        glm::vec3 direction(nx, ny, nz);
        direction = normalize(direction);
        glm::vec3 scaled_direction = 0.2f * direction;
        glm::vec3 end_point = start_point + scaled_direction;
        normal_line_vertices.push_back(start_point.x);
        normal_line_vertices.push_back(start_point.y);
        normal_line_vertices.push_back(start_point.z);
        normal_line_vertices.push_back(end_point.x);
        normal_line_vertices.push_back(end_point.y);
        normal_line_vertices.push_back(end_point.z);
    }
    glGenVertexArrays(1, &normal_lines_vao);
    glBindVertexArray(normal_lines_vao);
    glGenBuffers(1, &normal_lines_vbo);
    glBindBuffer(GL_ARRAY_BUFFER, normal_lines_vbo);
    glBufferData(GL_ARRAY_BUFFER, normal_line_vertices.size() * sizeof(float),
normal_line_vertices.data(), GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
static_cast<void*>(nullptr));
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
}
```

**Listing 6 Metoda za postavku crtanja normala**

Metoda `averagedNormalsSetup()` popunjava odgovarajuće podatke neophodne za prikaz usrednjenih normala temena. Kako je proces popunjavanja VAO i VBO sličan prethodnom primeru, taj deo koda se neće prikazivati jer je fokus na algoritmu za usrednjavanje normala koji predstavlja najkompleksniji deo ovog zadatka i njegovo izvršavanje direktno zavisi od broja temena učitano *mesh*-a. Rezultati izračunavanja se skladište u posebne datoteke i učitavaju pri svakom narednom pokretanju aplikacije. Na listingu 7 je prikazan algoritam za učitavanje i čuvanje vrednosti usrednjenih normala.

```
std::ifstream inputFile(filename);
if (inputFile.is_open()) {
    float value;
    while (inputFile >> value) {
        averaged_normal_vertices.push_back(value);
    }
    inputFile.close();
}
else { // Ostatak koda prikazan u listingu 8
}

std::ofstream output_file(filename);
if (output_file.is_open()) {
    for (const float& value : averaged_normal_vertices) {
        output_file << value << "\n";
    }
    output_file.close();
}
else std::cerr << "Unable to save data to file.\n";}
```

**Listing 7 Učitavanje i čuvanje vrednosti**

Algoritam za izračunavanje usrednjenih normala se nalazi na listingu 8.

```

std::vector<glm::vec3> added_to_vertices;

for (size_t i = 0; i < mVertices_flat.size(); i += 8) {

    glm::vec3 averaged_normal(0.0f);
    std::vector<glm::vec3> added_to_current_normal_calculation;
    float start_x = mVertices_flat[i];
    float start_y = mVertices_flat[i + 1];
    float start_z = mVertices_flat[i + 2];

    if (!contains_element(glm::vec3(start_x, start_y, start_z), added_to_vertices))
        added_to_vertices.emplace_back(start_x, start_y, start_z);
    else continue;

    for (size_t j = i; j < mVertices_flat.size(); j += 8) {
        float current_x = mVertices_flat[j];
        float current_y = mVertices_flat[j + 1];
        float current_z = mVertices_flat[j + 2];
        float current_nx = mVertices_flat[j + 3];
        float current_ny = mVertices_flat[j + 4];
        float current_nz = mVertices_flat[j + 5];
        if (start_x == current_x && start_y == current_y && start_z == current_z)
        {
            glm::vec3 current_normal(current_nx, current_ny, current_nz);
            if (!contains_element(current_normal,
added_to_current_normal_calculation)) {
                averaged_normal += current_normal;

                added_to_current_normal_calculation.push_back(current_normal);
            }
        }
        if (averaged_normal != glm::vec3(0.0f)) {

            averaged_normal = static_cast<float>(1.00 /
added_to_current_normal_calculation.size()) * averaged_normal;
            averaged_normal = glm::normalize(averaged_normal);
            glm::vec3 scaled_direction = 0.2f * averaged_normal;
            glm::vec3 end_point = glm::vec3(start_x, start_y, start_z) +
scaled_direction;

            averaged_normal_vertices.push_back(start_x);
            averaged_normal_vertices.push_back(start_y);
            averaged_normal_vertices.push_back(start_z);

            averaged_normal_vertices.push_back(end_point.x);
            averaged_normal_vertices.push_back(end_point.y);
            averaged_normal_vertices.push_back(end_point.z);
        }
    }
}

```

**Listing 8 Algoritam za izračunavanje usrednjenih normala**

Metoda smoothSetup () popunjava odgovarajuće podatke neophodne za prikaz modela prilikom Gurovog i Fongovog senčenja. Kako je proces popunjavanja VAO, VBO i EBO sličan primeru konstantnog senčenja, i kako je proces čuvanja i učitavanja datoteka sa izračunatim vrednostima sličan prethodnom primeru, ti delovi koda će biti izostavljeni jer je glavni fokus na postavljanju novih vrednosti za normale. Postupak postavljanja novih vrednosti prikazan je na listingu 9.

```

for (size_t i = 0; i < mVertices_flat.size(); i += 8) {
    glm::vec3 averaged_normal(0.0f);
    std::vector<glm::vec3> added_to_current_normal_calculation;
    float start_x = mVertices_flat[i];
    float start_y = mVertices_flat[i + 1];
    float start_z = mVertices_flat[i + 2];

```

```

for (size_t j = 0; j < mVertices_flat.size(); j += 8) {
    float current_x = mVertices_flat[j];
    float current_y = mVertices_flat[j + 1];
    float current_z = mVertices_flat[j + 2];
    float current_nx = mVertices_flat[j + 3];
    float current_ny = mVertices_flat[j + 4];
    float current_nz = mVertices_flat[j + 5];
    if (start_x == current_x && start_y == current_y && start_z == current_z)
    {
        glm::vec3 current_normal(current_nx, current_ny, current_nz);
        if (!contains_element(current_normal,
added_to_current_normal_calculation)) {
            averaged_normal += current_normal;

            added_to_current_normal_calculation.push_back(current_normal);
        }
        averaged_normal = static_cast<float>(1.00 /
added_to_current_normal_calculation.size()) * averaged_normal;
        mVertices_smooth.push_back(start_x);
        mVertices_smooth.push_back(start_y);
        mVertices_smooth.push_back(start_z);
        mVertices_smooth.push_back(averaged_normal.x);
        mVertices_smooth.push_back(averaged_normal.y);
        mVertices_smooth.push_back(averaged_normal.z);
        mVertices_smooth.push_back(mVertices_flat[i + 6]);
        mVertices_smooth.push_back(mVertices_flat[i + 7]);
    }
}

```

Listing 9 Algoritam za postavljanje usrednjenih normala

Napravljeno je nekoliko metoda za renderovanje u zavisnosti od toga koji slučaj treba da bude renderovan. U modelu se za svaki *mesh* poziva odgovarajuća metoda u zavisnosti od toga koja metoda je pozvana u klasi modela. Na listingu 10 su implementacije metoda za konstantno i glatko senčenje.

```

void
Mesh::RenderFlat() const {
    if (mIndexCount) {
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, mEBO_flat);
        glDrawElements(GL_TRIANGLES, mIndexCount, GL_UNSIGNED_INT, (void*)0);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
        return;
    }
    glDrawArrays(GL_TRIANGLES, 0, mVertexCount);
    glBindVertexArray(0);
}
void
Mesh::RenderSmooth() const {
    glBindVertexArray(mVAO_smooth);
    if (mIndexCount) {
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, mEBO_smooth);
        glDrawElements(GL_TRIANGLES, mIndexCount, GL_UNSIGNED_INT, (void*)0);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
        return;
    }
    glDrawArrays(GL_TRIANGLES, 0, mVertexCount);
    glBindVertexArray(0);
}

```

Listing 10 Metode za različite slučajeve renderovanja

### 3.3 Šejderi

Šejderi su programi koji se koriste u procesu renderovanja trodimenzionalne grafike kako bi se odredila boja, tekstura, osvetljenje i materijal objekata. Oni su od suštinskog značaja za postizanje realističnih vizuelnih efekata.

Uniforme u šejderima su promenljive koje se koriste za prenos konstantnih ili statičkih podataka iz aplikacije u šejdere. Ove vrednosti ostaju nepromenjene za sve piksele ili fragmente koji se obrađuju šejderom tokom renderovanja scene. Tipični primeri uniformi uključuju matrice transformacije (kao što su matrice za modeliranje, pogled i projekciju), boje svetla, texture i druge parametre koji utiču na izgled objekata i scene.

Ulazne vrednosti u šejderima su podaci koji se koriste kao informacije o trenutnoj situaciji i karakteristikama objekata ili scene koja se trenutno renderuje. Ove vrednosti se obično prenose iz aplikacije ili iz drugih delova grafičkog sistema šejderima kako bi se obavila obrada i generisala krajnja slika. Izlazne vrednosti u šejderima predstavljaju rezultate obrade podataka koji se koriste kao ulazne vrednosti za idući korak obrade.

Verteks šejder je programska komponenta u grafičkim aplikacijama. Koristi se za obradu pojedinačnih temena u trodimenzionalnom modelu. Ovaj šejder transformiše koordinate temena iz trodimenzionalnog prostora u dvodimenzionalne koordinate na ekranu, omogućavajući im da budu pravilno pozicionirani i orijentisani u sceni. Fragment šejder je programska komponenta u grafičkim aplikacijama koja se koristi za obradu pojedinačnih piksela ili fragmenata na ekranu. Ova vrsta šejdera utvrđuje kako će svaki piksel na ekranu biti obojen, primenjujući razne efekte poput texture i bojenja.

Klasa Shader služi za učitavanje verteks i fragment šejdera kao i za postavljanje odgovarajućih uniformi. Polja i metode klase su prikazane na listingu 11.

```
class Shader {
private:
    unsigned loadAndCompileShader(std::string filename, GLuint shaderType);
    unsigned createBasicProgram(unsigned vShader, unsigned fShader);
    unsigned mId;
    static const unsigned POSITION_LOCATION = 0;
    static const unsigned COLOR_LOCATION = 1;

public:
    Shader(const std::string& vShaderPath, const std::string& fShaderPath);
    unsigned GetId() const;

    void SetUniform1i(const std::string& uniform, int v) const;
    void SetUniform1f(const std::string& uniform, float v) const;
    void SetUniform3f(const std::string& uniform, const glm::vec3& v) const;
    void SetUniform4m(const std::string& uniform, const glm::mat4& m) const;

    void SetModel(const glm::mat4& m) const;
    void SetView(const glm::mat4& m) const;
    void SetProjection(const glm::mat4& m) const;
};
```

**Listing 11** Polja i metode klase Shader

U glavnoj funkciji programa, Guroov i Fongov šejder su kreirani na način prikazan na listingu 12.

```
Shader gouraud_shader_material("shaders/gouraud.vert", "shaders/gouraud.frag");
Shader phong_shader_material("shaders/phong.vert",
"shaders/phong_material.frag");
```

**Listing 12 Kreiranje Guroovog i Fongovog šejdera**

Sadržaji datoteka gouraud.vert, gouraud.frag, phong.vert i phong\_material.frag se nalazi u dodatku A.

### 3.8 Rednerovanje modela u zavisnosti od odabranog moda

U glavnoj funkciji dolazi do renderovanja učitano modela u željenom modu. Pored već napomenutih klasa implementirani su i delovi zadatka zaduženi za korisnički unos, kretanje, poziciju kamere kao i svi pozivi funkcija potrebni za kreiranje prozora, renderovanje trodimenzionalnog modela kao i prikaz grafičkog korisničkog interfejsa. Kako fokus ovog zadatka nije na rešavanju takvih problema, implementacija tog dela koda neće biti prikazana. Fokus je na odabiru modova i postavljanju uniformnih vrednosti različitih izvora svetlosti. Za trenutno izabrani šejder, vrednosti uniformi se za direkciono osvetljenje, baterijsku lampu, tačkasto osvetljenje i vrednosti svetlosnih komponenti materijala postavljaju na način prikazan u listingu 13.

```
current_shader->SetUniform3f("uSunLight.Position", point_light_position_sun);
current_shader->SetUniform1f("uSunLight.Kc", 0.001);
current_shader->SetUniform1f("uSunLight.Kq", 0.01);
current_shader->SetUniform1f("uSunLight.Kl", 0.11);
current_shader->SetUniform3f("uSunLight.Ka", glm::vec3(0.00, 0.0, 0.00));
current_shader->SetUniform3f("uSunLight.Kd", glm::vec3(1.00, 1.0, 0.00));
current_shader->SetUniform3f("uSunLight.Ks", glm::vec3(1.0));

current_shader->SetUniform3f("uFlashLight.Position",
glm::vec3(fps_camera.GetPosition()));
current_shader->SetUniform3f("uFlashLight.Direction", glm::vec3(0));
current_shader->SetUniform3f("uFlashLight.Ka", glm::vec3(0));
current_shader->SetUniform1f("uFlashLight.Kc", 0.6f);
current_shader->SetUniform1f("uFlashLight.Kl", 0.0002f);
current_shader->SetUniform1f("uFlashLight.Kq", 0.0002f);
current_shader->SetUniform1f("uFlashLight.InnerCutOff", glm::cos(glm::radians(1.0f)));
current_shader->SetUniform1f("uFlashLight.OuterCutOff",
glm::cos(glm::radians(30.0f)));

current_shader->SetUniform3f("uMaterial.Ka", material_ka); // *** Check what is it for
current_shader->SetUniform3f("uMaterial.Kd", material_kd);
current_shader->SetUniform3f("uMaterial.Ks", material_ks);
current_shader->SetUniform1f("uMaterial.Shininess", shininess * 128);

current_shader->SetUniform3f("uDirLight.Direction", glm::vec3(0, -0.1, 0));
current_shader->SetUniform3f("uDirLight.Ka", glm::vec3(0.6));
current_shader->SetUniform3f("uDirLight.Kd", glm::vec3(0.6));
current_shader->SetUniform3f("uDirLight.Ks", glm::vec3(1));
```

**Listing 13 Postavljanje uniformi**

Prvih 6 modova predstavljaju prikaz različitih nivoa napretka prilikom stvaranja osenčenog trodimenzionalnog modela. Za ovaj prikaz se koristi `color_only` šejder koji jednobojno boji sve fragmente. Prikaz dostupnih modova je na listingu 14.

```
switch (state.mode)
{
case 1:
    current_shader = &color_only;
    mode_render_vertices(model, current_shader, glm::vec3(points_and_lines_color),
2);
    break;
```

```

case 2:
    current_shader = &color_only;
    mode_render_triangles(model, current_shader,
glm::vec3(points_and_lines_color));
    break;
case 3:
    current_shader = &color_only;
    mode_render_filled_triangles(model, current_shader, glm::vec3(filled_color));
    break;
case 4:
    current_shader = &color_only;
    mode_render_filled_triangles(model, current_shader, glm::vec3(filled_color));
    mode_render_triangles(model, current_shader,
glm::vec3(points_and_lines_color));
    break;
case 5:
    current_shader = &color_only;
    mode_render_filled_triangles(model, current_shader, glm::vec3(filled_color));
    mode_render_triangles(model, current_shader,
glm::vec3(points_and_lines_color));
    mode_render_normals(model, current_shader, all_normals_color);
    break;
case 6:
    current_shader = &color_only;
    mode_render_filled_triangles(model, current_shader, glm::vec3(filled_color));
    mode_render_triangles(model, current_shader,
glm::vec3(points_and_lines_color));
    mode_averaged_normals(model, current_shader, averaged_normals_color);
    break;

```

**Listing 14 Prikaz dostupnih modova za prikaz različitih nivoa napretka modela**

Slučaj 7 pokriva tri moguća načina senčenja dok slučaj 8 pokriva prikaz Fongovog senčenja sa uticajem svetlosnih komponenti materijala i tekstura. Na listingu 15 su prikazani odabiri senčenja.

```

case 7:
    switch (state.shading_mode)
    {
        case flat:
            current_shader = &flat_shader_material;
            glUseProgram(current_shader->GetId());
            model.RenderFlat();
            break;
        case gouraud:
            current_shader = &gouraud_shader_material;
            glUseProgram(current_shader->GetId());
            model.RenderSmooth();
            break;
        case phong:
            current_shader = &phong_shader_material;
            glUseProgram(current_shader->GetId());
            model.RenderSmooth();
            break;
    }
    break;
case 8:
    current_shader = &phong_shader_material_texture;
    mode_render_with_texture(model, test_texture, test_specular_texture,
current_shader);
    break;
default:
    break;
}

```

**Listing 15 Prikaz dostupnih modova senčenja**



## 4. Primer korišćenja

Kada korisnik pokrene aplikaciju, dočekuje ga početni ekran koji sjedinjuje dvodimenzionalni grafički interfejs i trodimenzionalni model prikazan u modu 01. Učitani model koji se prikazuje korisniku je model motora [8].

### 4.1 Korisničke komande

Na gornjoj desnoj strani ekrana pozicioniran je prozor koji informiše korisnika kako da koristi aplikaciju:

- Pritiskom na taster Q korisnik sakriva i otkriva grafički korisnički interfejs.
- Pritiskom testera W, A, S, D korisnik se kreće napred, nazad, levo i desno u prostoru.
- Pritiskom tastera Space korisnik ide gore u prostoru, dok pritiskom tastera C ide dole u prostoru.
- Pomeranjem miša korisnik menja poziciju u koju gleda.
- Pritiskom na taster F korisnik uključuje i isključuje efekat svetlosti baterijske lampe.
- Držanjem tastera E korisnik onemogućava promenu pozicije u koju gleda mišem.
- Pritiskom odgovarajućih brojeva(1-8) menja se prikaz na ekranu na odgovarajući mod.
- Pritiskom tastera I, O, P menja se tehnika senčenja (I – Konstantno, O – Guroovo, P - Fongovo).

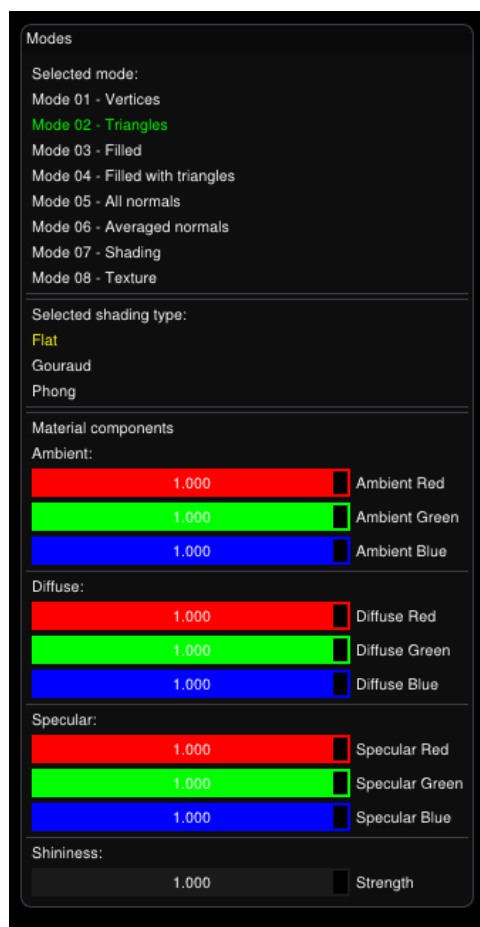
Izgled prozora za korisničke opcije prikazan je na slici 8.



Slika 8 Korisničke kontrole

### 4.2 Modovi

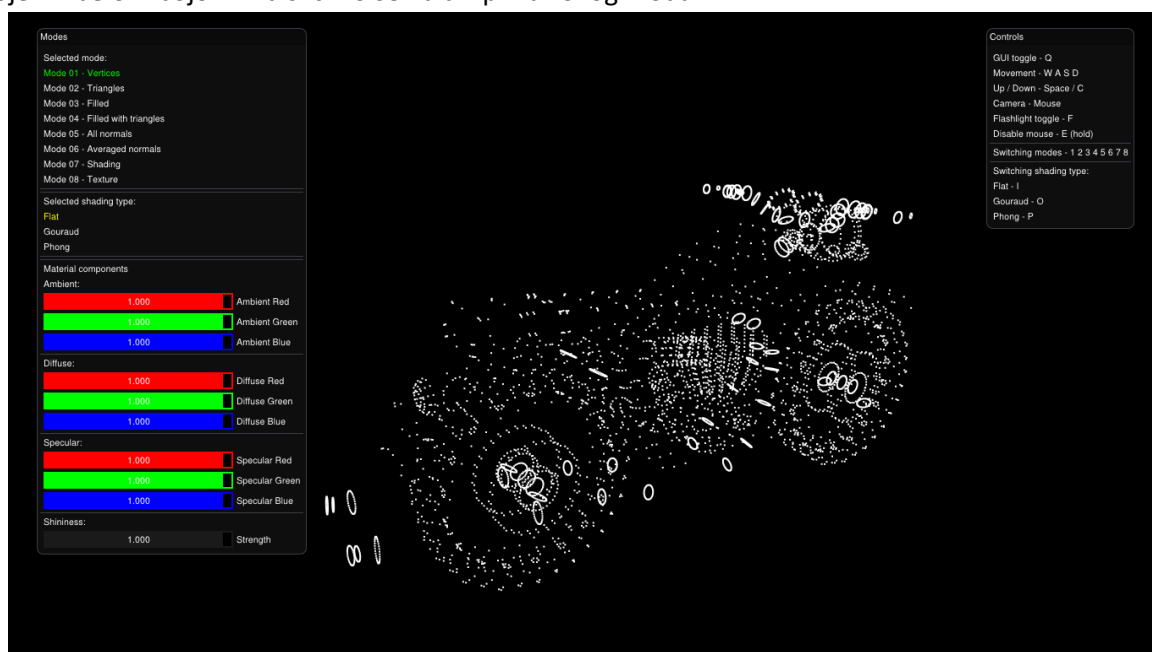
Prikaz u aplikaciji podeljen je u modove. Korisnik je informisan o tome u kom modu se trenutno nalazi pomoću prozora koji se nalazi na levoj strani ekrana. Mod u kome se korisnik nalazi je obojen zelenom bojom. Tehnika senčenja se na isti način ističe korisniku, ali umesto zelene boje, aktivna tehnika senčenja je označena žutom bojom. Ispod oznaka za modove i tehnike senčenja, nalaze se klizači za ambijentalnu, difuznu i spekularnu komponentu materijala kao i klizač za nivo sjajnosti materijala. Korisnik pomera klizače po potrebi koristeći levi klik miša. Izgled ovog prozora prikazan je na slici 9.



Slika 9 Prozor sa modovima i klizačima

#### 4.2.1 Mod 01 - Temena

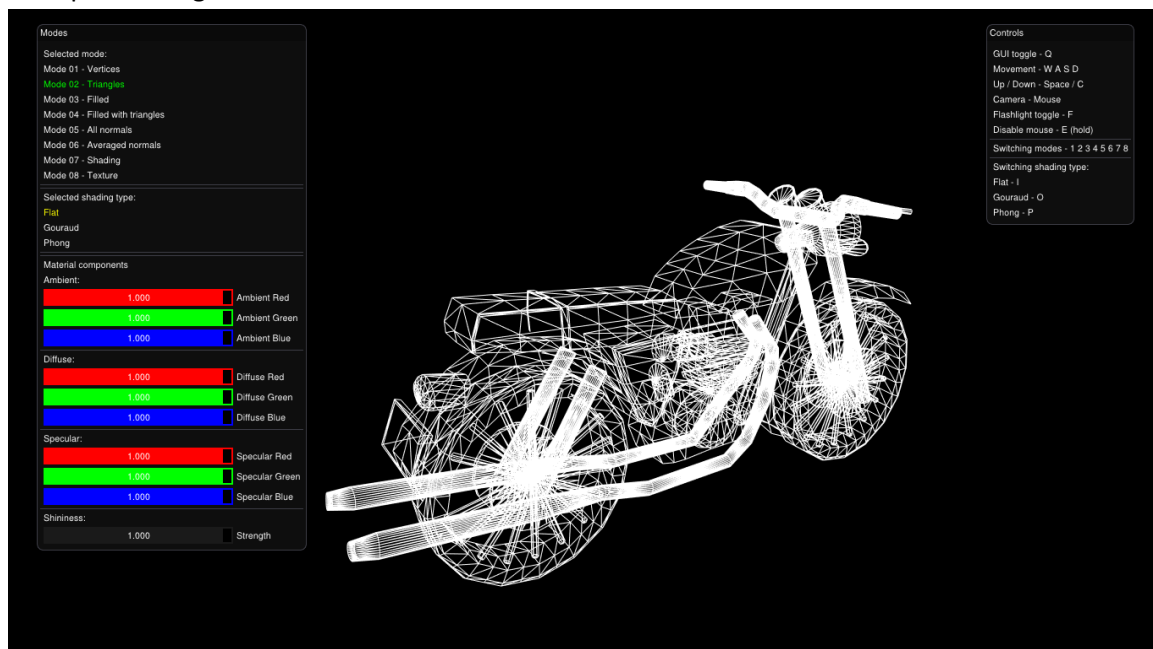
Prilikom odabira moda 01 korisniku se prikazuje model motora sačinjen samo od temena motora obojenih belom bojom. Na slici 10 se nalazi prikaz ovog moda.



Slika 10 Mod 01 - Temena

### 4.2.2 Mod 02 - Trouglovi

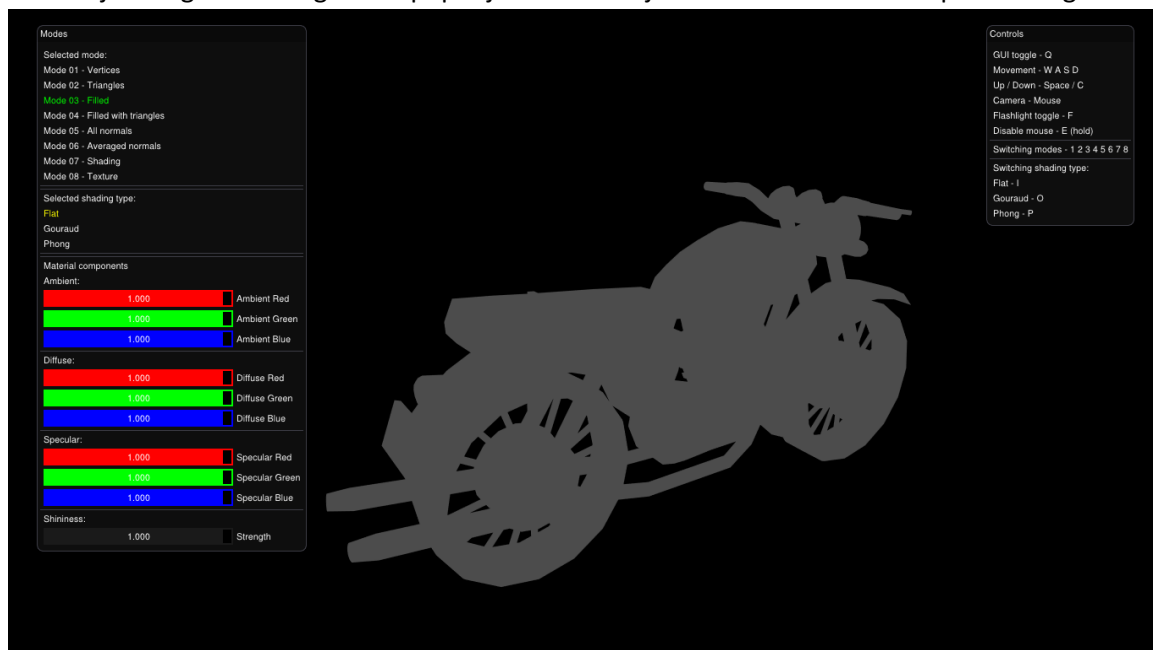
Prilikom odabira moda 02 korisniku se prikazuje model motora sačinjen od povezanih temena motora koji formiraju trouglove. Trouglovi nisu popunjeni. Ivice trouglova su obojene belom bojom. Na slici 11 se nalazi prikaz ovog moda.



Slika 11 Mod 02 - Trouglovi

### 4.2.3 Mod 03 - Popunjeni trouglovi bez vidljive mreže

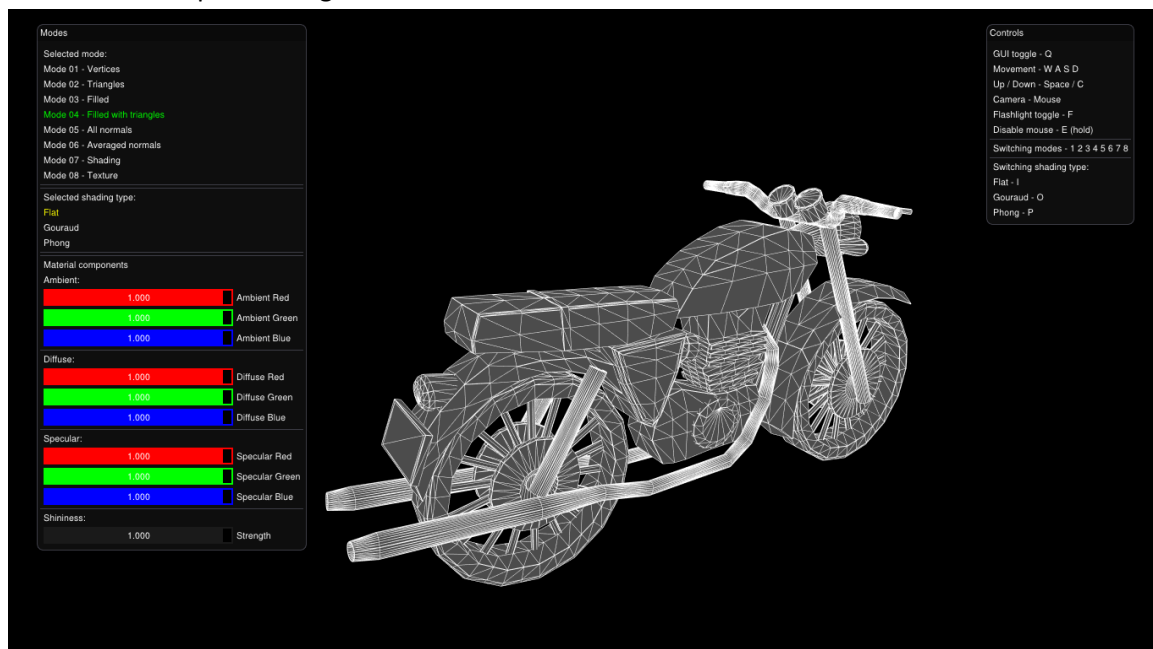
Prilikom odabira moda 03 korisniku se prikazuje model motora sačinjen od povezanih temena motora koji formiraju trouglove. Trouglovi su popunjeni sivom bojom. Na slici 12 se nalazi prikaz ovog moda.



Slika 12 Mod 03 - Popunjeni trouglovi

#### 4.2.4 Mod 04 - Popunjeni trouglovi sa vidljivom mrežom

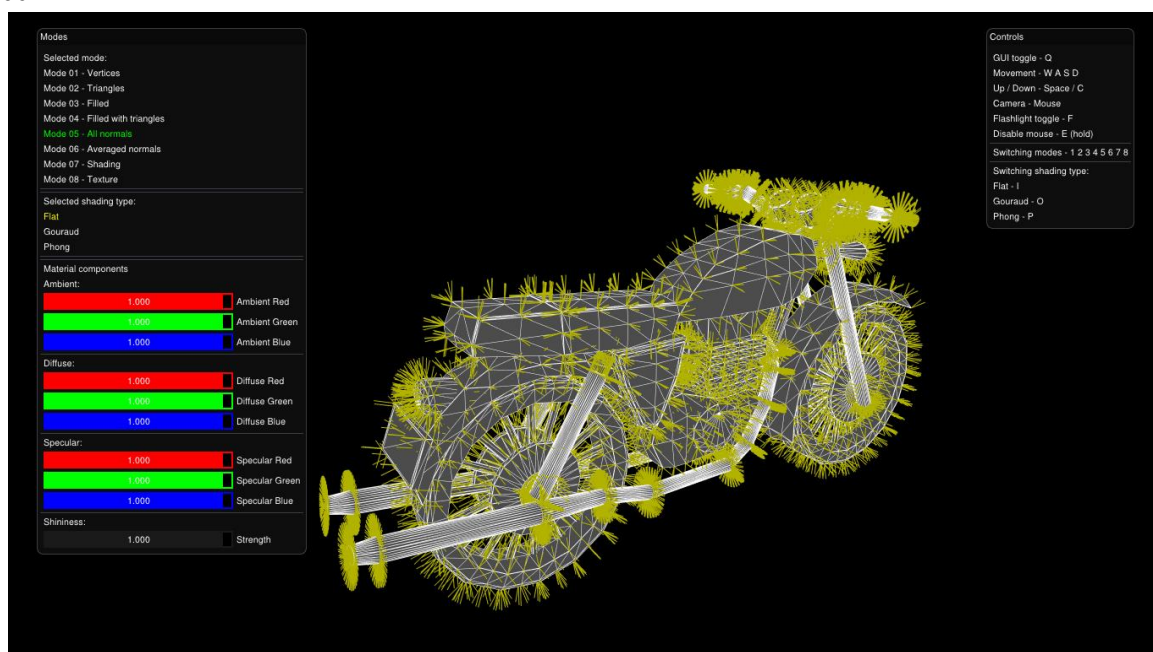
Prilikom odabira moda 04 korisniku se prikazuje model motora sačinjen od povezanih temena motora koji formiraju trouglove. Trouglovi su popunjeni sivom bojom. Ivice trouglova su obojene belom bojom. Na slici 13 se nalazi prikaz ovog moda.



Slika 13 Mod 04 - Popunjeni trouglovi sa vidljivom mrežom

#### 4.2.5 Mod 05 - Sve normale

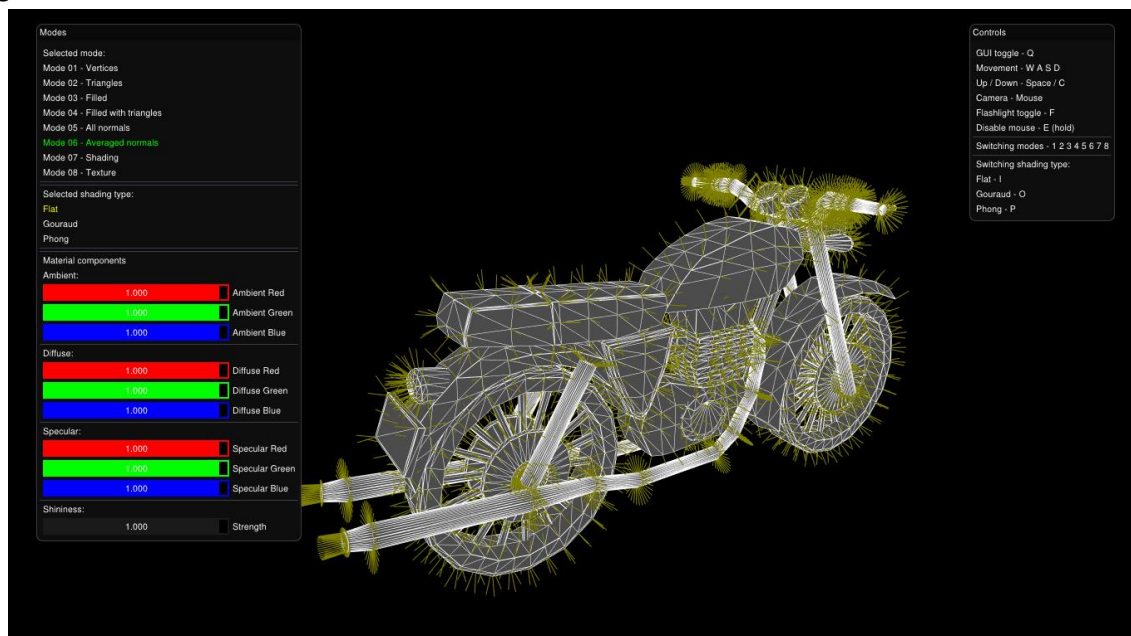
Prilikom odabira moda 05 korisniku se prikazuje isti prikaz kao i u modu 04, ali sa dodatim svim normalama koje izlaze iz temena. Normale su obojene žutom bojom. Na slici 14 se nalazi prikaz ovog moda.



Slika 14 Mod 05 - Sve normale

### 4.2.6 Mod 06 - Usrednjene normale

Prilikom odabira moda 05 korisniku se prikazuje isti prikaz kao i u modu 04 ali sa dodatim usrednjenim normalama koje izlaze iz temena. Normale su obojene tamno žutom bojom. Na slici 15 se nalazi prikaz ovog moda.



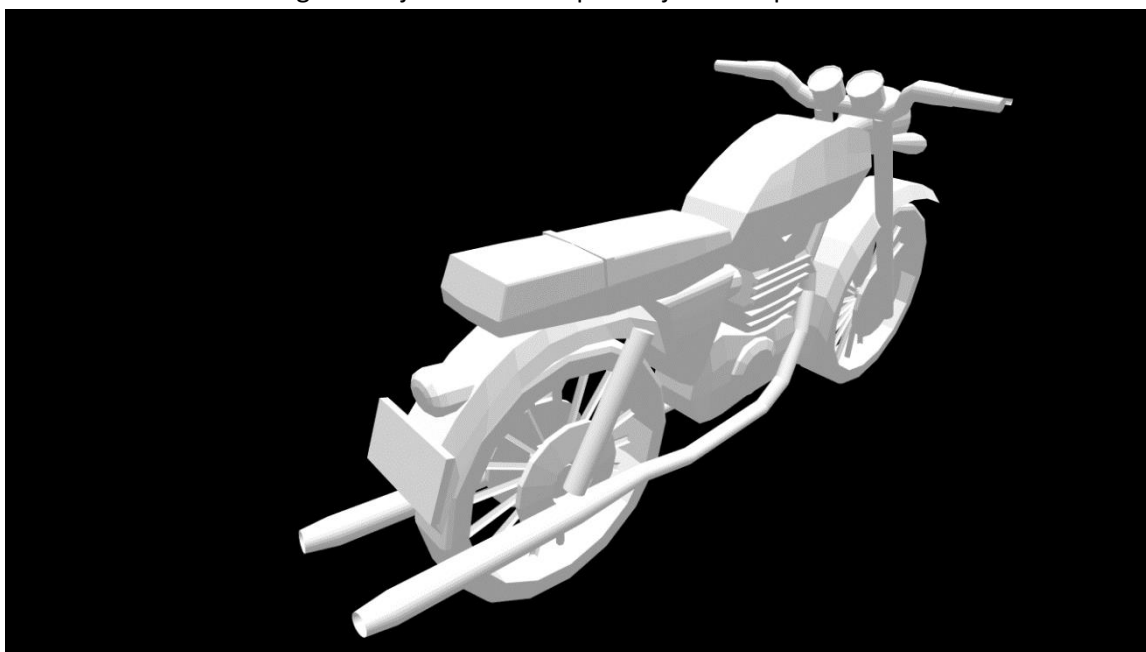
Slika 15 Mod 06 - Usrednjene normale

### 4.2.7 Mod 07- Senčenje

Korisnik u ovom modu može da bira kojom tehnikom će motor biti senčen.

#### 4.2.7.1 Konstantno senčenje

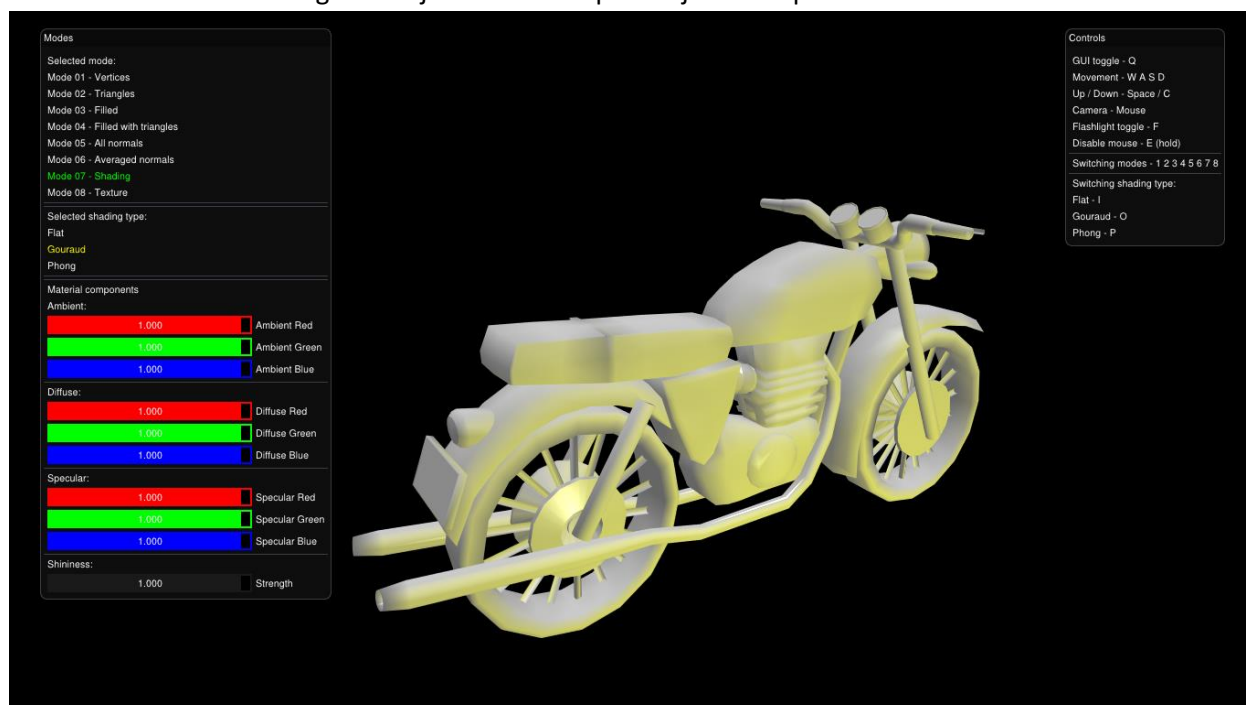
Prilikom odabira konstantnog senčenja korisniku se prikazuje model prikazan na slici 16.



Slika 16 Mod 07 - Konstantno senčenje

#### 4.2.7.2 Guroovo senčenje

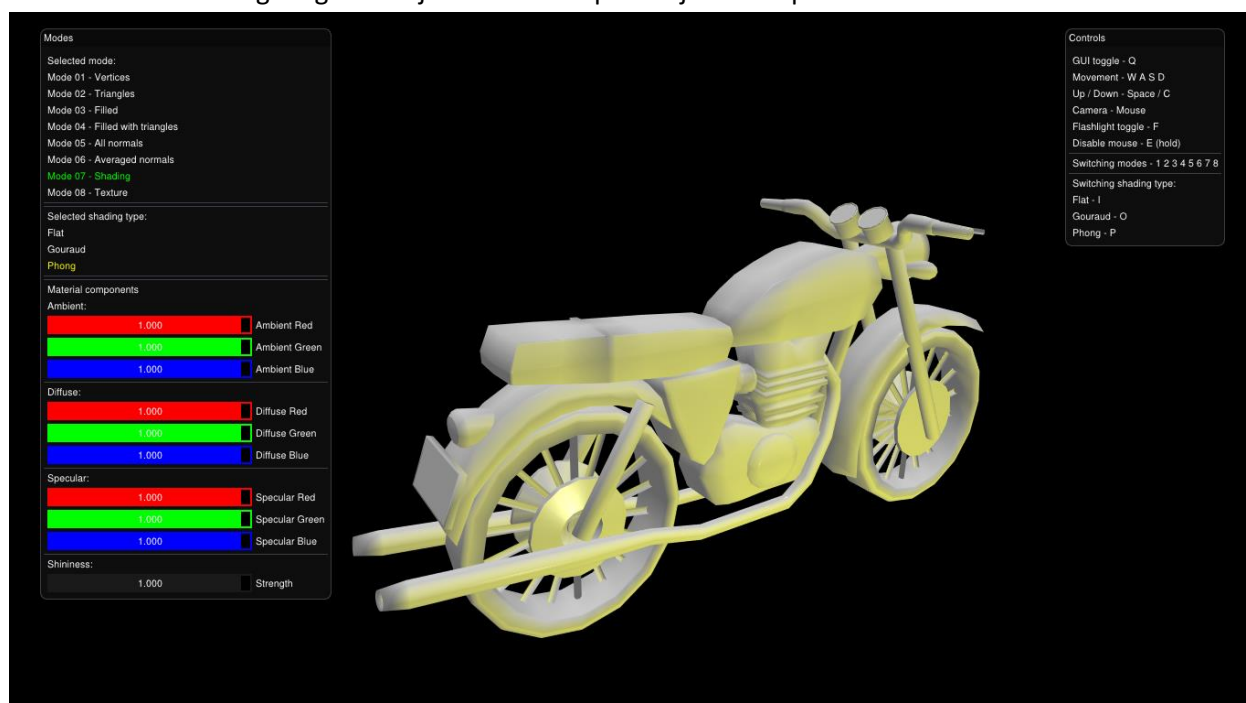
Prilikom odabira Guroovog senčenja korisniku se prikazuje model prikazan na slici 17.



Slika 17 Mod 07 - Guroovo senčenje

#### 4.2.7.3 Fongovo senčenje

Prilikom odabira Fongovog senčenja korisniku se prikazuje model prikazan na slici 18.

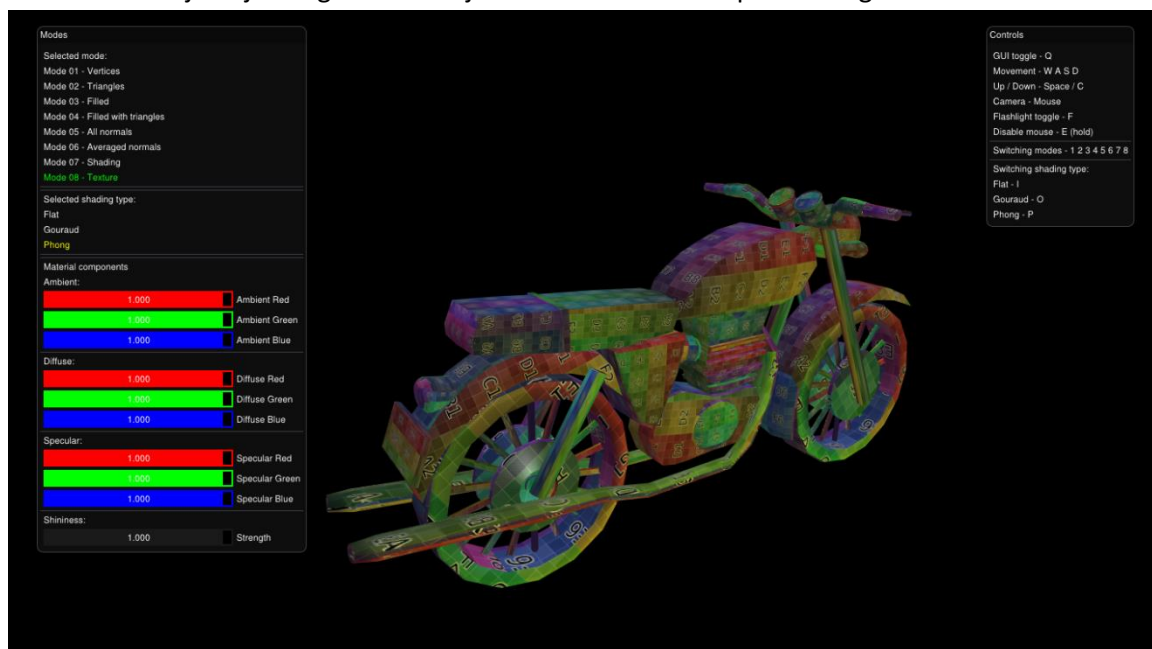


Slika 18 Mod 07 - Fongovo senčenje



## 4.2.8 Mod 08 - Teksture

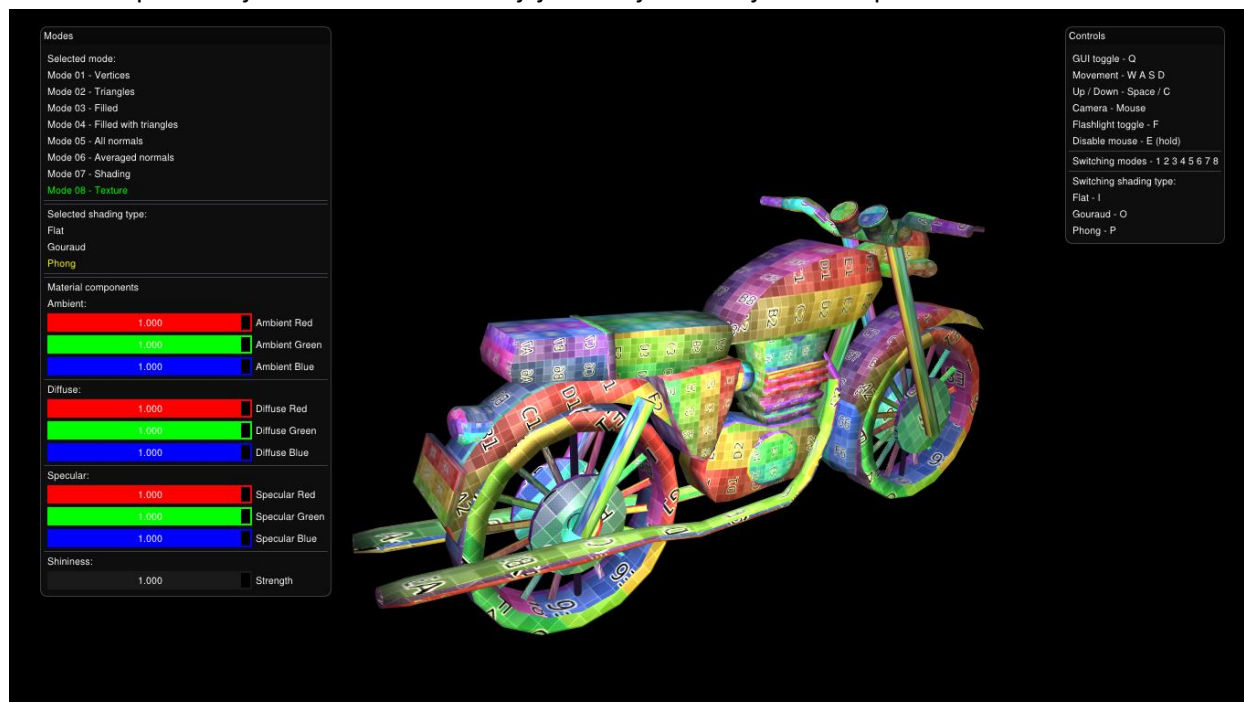
Prilikom odabira moda 08 korisniku se prikazuje model motora sa testnom difuznom i spekularnom teksturom. Primenjeno je Fongovo senčenje. Na slici 19 se nalazi prikaz ovog moda.



Slika 19 Mod 08 - Teksture

## 4.3 Baterijska lampa iz prvog lica

Na slici 20 prikazan je model u modu 08 koji je osvetljen baterijskom lampom.



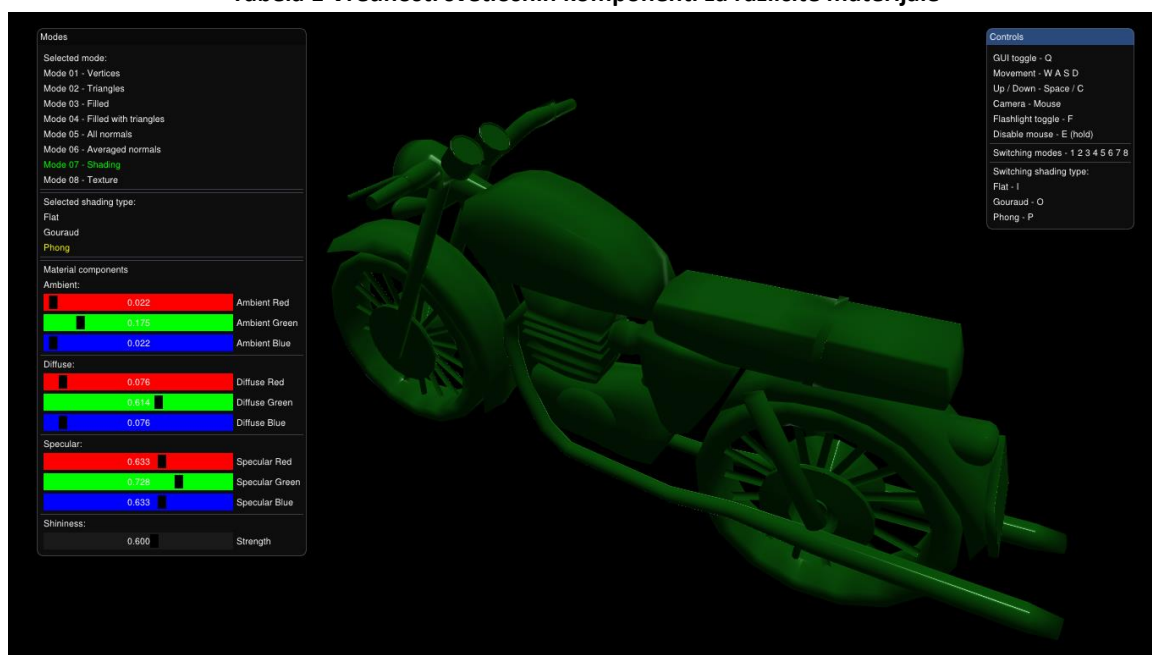
Slika 20 Osvetljene baterijskom lampom

## 4.4 Materijal modela na osnovu parametara svetlosnih komponenti

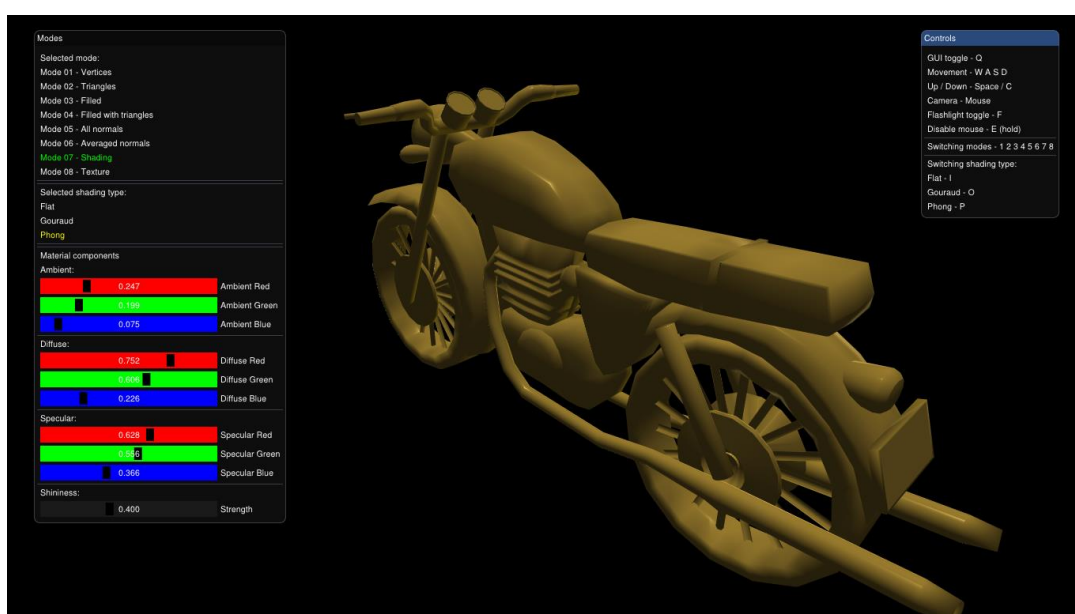
Promenom parametara klizača, menja se i izgled samog modela. Korišćenjem već određenih vrednosti za neke materijale, izgled modela se menja i deluje kao da je sačinjen od određenog materijala [9]. Vrednosti parametara koje razmatramo nalaze se u tabeli 1. Prikaz smaragdnog motora nalazi se na slici 21, prikaz zlatnog motora nalazi se na slici 22 i prikaz motora sačinjenog od rubina nalazi se na slici 23.

	Ambijentalna			Difuzna			Spekularna			Sjajnost
	Crvena	Zelena	Plava	Crvena	Zelena	Plava	Crvena	Zelena	Plava	
Smaragd	0.0215	0.1745	0.0215	0.07568	0.61424	0.07568	0.633	0.727811	0.633	0.6
Zlato	0.24725	0.1995	0.0745	0.75164	0.60648	0.22648	0.628281	0.555802	0.366065	0.4
Rubin	0.1745	0.01175	0.01175	0.61424	0.04136	0.04136	0.727811	0.626959	0.626959	0.6

Tabela 1 Vrednosti svetlosnih komponenti za različite materijale

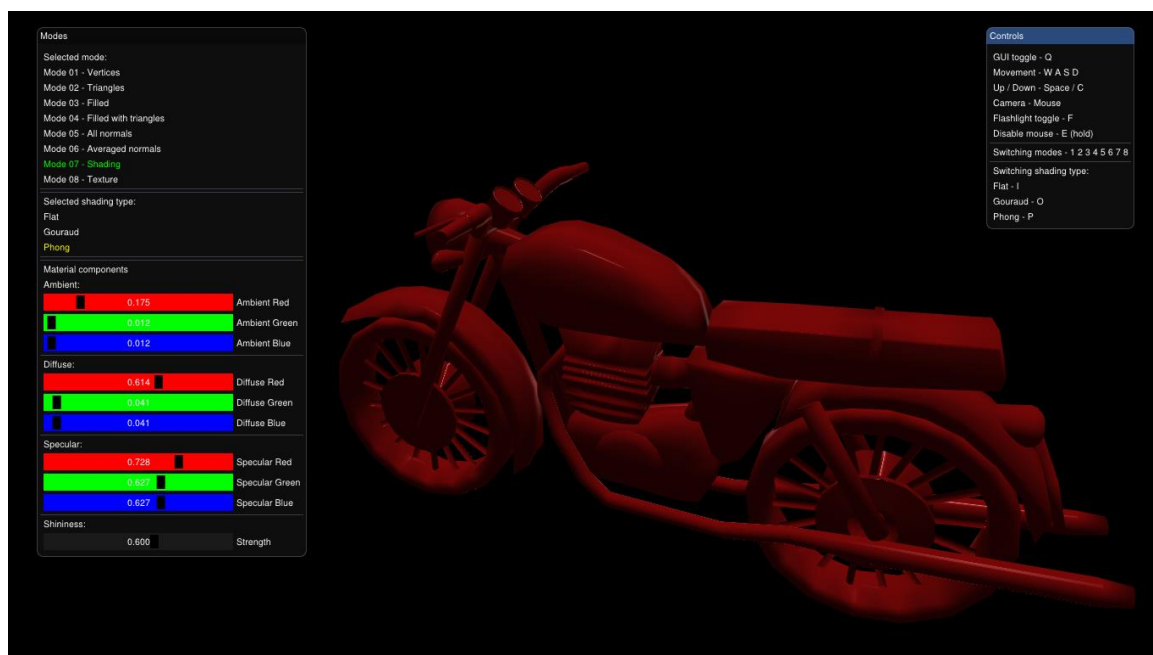


Slika 21 Smaragdni motor



Slika 22 Zlatni motor

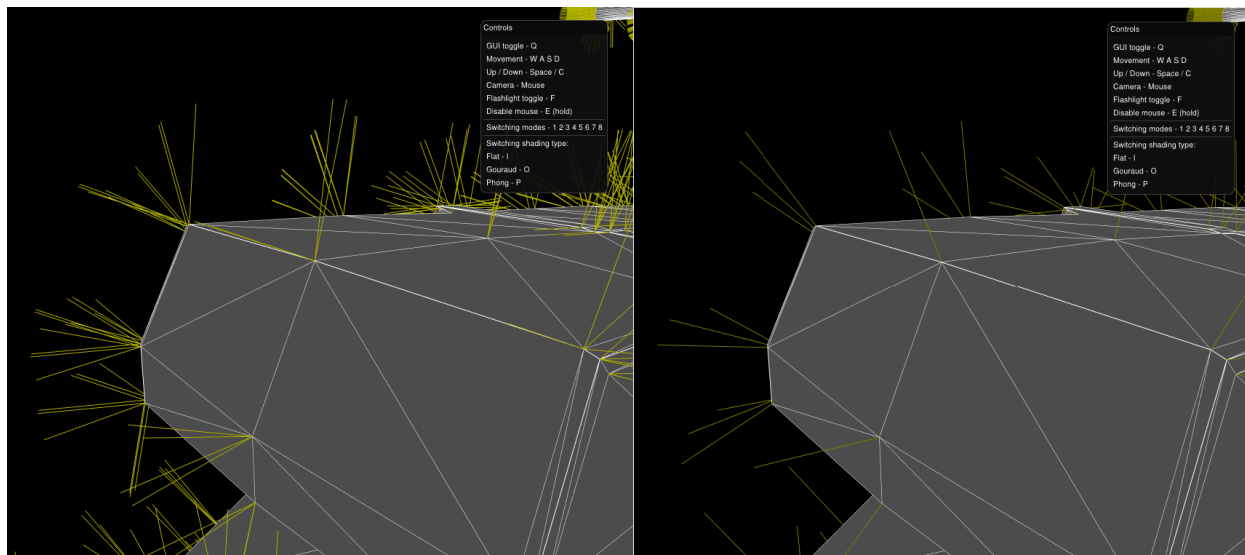




Slika 23 Motor sačinjen od rubina

## 4.5 Poređenje usrednjenih i svih normala

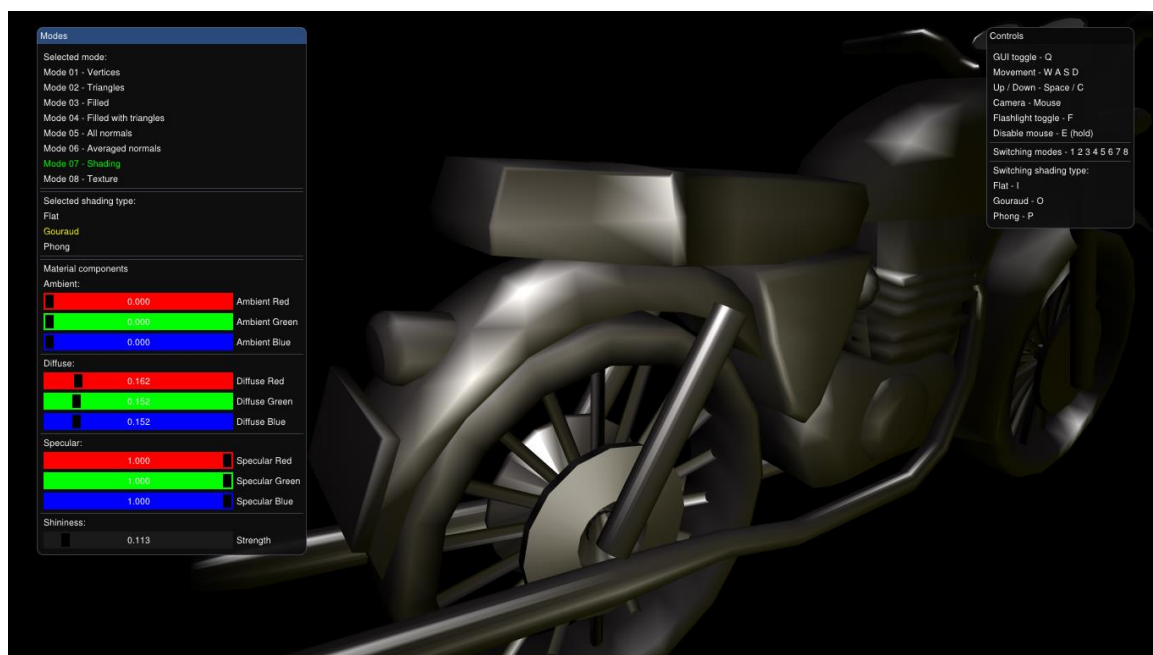
Na slici 24 je sa leve strane prikaz svih normala iz temena, a sa desne strane slike se nalazi prikaz usrednjenih normala. Broj normala u oba slučaja je isti samo se kod usrednjenih normala one nalaze na istom mestu pa daju privid toga da ih je manje.



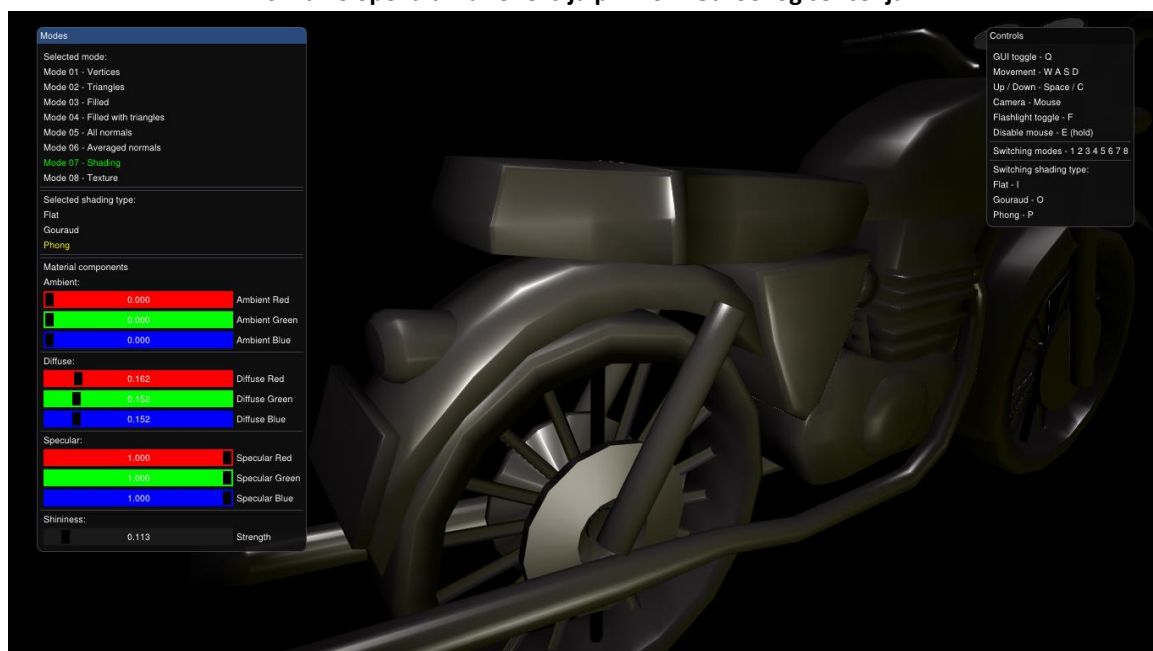
Slika 24 Poređenje normala

## 4.6 Poređenje različitih tehnika senčenja

Najveća razlika između tehnika senčenja se može uočiti prilikom smanjivanja parametara za sjajnost, korišćenjem baterijske lampe i približavanjem modelu. Na slici 25 prikazano je Guroovo senčenje, a na slici 26 prikazano je Fongovo senčenje.



Slika 25 Spekularna refleksija prilikom Gurovog senčenja



Slika 26 Spekularna refleksija prilikom Fongovog senčenja

## 5. Ograničenja i unapređenja

### 5.1 Raznovrsnost modela i tekstura

Aplikacija ne pruža nikakav jednostavan unos novih modela, kao ni tekstura. Sav prikaz se demonstrira na modelu motora koji je učitani u kodu. Omogućavanjem unosa drugih modela korisnik bi mogao da analizira razlike između tehnika senčenja na novim modelima, ili čak na istim modelima koji imaju različite stepene detalja. Unos novih difuznih i spekularnih tekstura bi takođe doprineo analiziranju modela.

### 5.2 Specifičnost parametara

Parametri poput jačine svetlosnih izvora, njihovih pozicija i boja se ne mogu menjati. Dodavanjem opcija za njihovo menjanje bi korisniku znatno unapredilo analiziranje i utvrđivanje razlika između tehnika senčenja. Pored ovih parametara, parametri rotacije, translacije i skaliranja modela su nepostojeći. Njihovim dodavanjem bi proces analiziranja bio još precizniji.

### 5.3 Algoritam za izračunavanje usrednjenih normala

Izračunavanje usrednjenih normala predstavlja najzahtevniji aspekt ovog zadatka. Brzina izvršavanja ovog algoritma direktno zavisi od broja temena prisutnih u učitanoj modelu. U slučaju učitavanja izuzetno detaljnog modela, postoji mogućnost da će izvršenje algoritma trajati veoma dugo, što je nepoželjno. Unapređenje ovog dela zadatka značajno bi poboljšalo efikasnost analize kompleksnijih modela.

### 5.4 Opcija za čuvanje stanja aplikacije

Opcija da korisnik može na aplikativnom nivou da sačuva stanje parametara poput pozicije u prostoru i vrednosti svetlosnih komponenti u eksternu datoteku i da ih učita po svojoj želji bi olakšalo analizu više različitih slučajeva u kojima se model može nalaziti. Na ovaj način se omogućava i razmenjivanje različitih stanja između korisnika.

### 5.5 Prelazak na internet

Razvijanje aplikacije tako da je podržana od strane web tehnologija bi u ogromnoj meri doprinelo pristupačnosti različitim korisnicima.

## 6. Zaključak

U ovom radu predstavljen je proces implementacije aplikacije za vizuelnu demonstraciju koncepta računarske grafike potkrepljen teoretskim osnovama, kao i njena upotreba.

Objašnjeni su pojmovi temena, normala i modela kao osnovnih gradivnih elemenata virtualnog prostora. Nakon toga fokus je bio na detaljima vezanim za svetlost poput različitih izvora svetlosti i komponenti refleksija svetlosnih zraka. Navedeni koncepti su zatim bili korišćeni za objašnjenje pojmova senčenja i izneti su detalji, prednosti i mane konstantnog, Guroovog i Fongovog senčenja.

Implementacija aplikacije je opisana u idućoj celini rada. Navedene su tehnologije koje su korišćene kao i razlozi zašto su one izabrane za implementaciju softverskog rešenja. Nakon toga su navedeni detalji implementacije ključnih klasa i algoritama potrebnih za realizaciju rešenja. Glavni fokus bio je na učitavanju i obradi modela, rukovanju sa šejderima i sjedinjavanju svega implementiranog u konkretnu celinu, gde bi na osnovu korisničkih unosa različiti delovi koda bili izvršavani.

Upotreba aplikacije je pojašnjena u narednoj celini rada. Nabrojane su korisničke komande kao i uputsvo za njihovo korišćenje. Prikazan je grafički korisnički interfejs koji pomaže pri upotrebi aplikacije. Nakon toga su uz slike prikazani svi koncepti računarske grafike koji su prethodno bili objašnjeni u celini vezanoj za teoretske osnove. Posebno je skrenuta pažnja na to kako različite vrednosti svetlosnih komponenti mogu da ostvare osećaj da je model napravljen od određenog materijala. Na kraju celine vezane za upotrebu aplikacije prikazana je i analiza razlike između Guroovog i Fongovog senčenja pri određenim uslovima.

U poslednjoj celini navedeni su potencijalni nedostaci ovog rešenja kao i predlozi za unapređenja. Postoje dva smera u kojima unapređenja ovog rešenja mogu da idu. Prvi smer je smer korisničke interakcije gde bi fokus bio na ostvarivanju još veće slobode pri promeni parametara vezanih za model i izvore svetlosti. Drugi smer jeste dodavanje još komplikovanijih koncepata računarske grafike poput senki i providnih objekata.

Ovaj rad daje jednostavan uvid u kompleksne koncepte računarske grafike koji služe za efikasno predstavljanje trodimenzionalnih objekata i scena na dvodimenzionalnoj površini ekrana. Kroz pojednostavljen pregled procesa ovaj rad analizira sve korake neophodne za formiranje odgovarajućeg nivoa realizma u renderovanim prikazima.

## 7. Literatura

- [1] <https://www.mql5.com/en/articles/7708>, stranica odakle je preuzeta slika čajnika
- [2] [https://learn.foundry.com/modo/content/help/pages/uv-ing/set\\_vertex\\_normals.html](https://learn.foundry.com/modo/content/help/pages/uv-ing/set_vertex_normals.html), stranica odakle je preuzeta slika crvene lopte sa svim i usrednjenim normalama
- [3] <https://www.cs.otago.ac.nz/cosc342/2017-notes/342-2017lect15.pdf>, stranica odakle je preuzeta slika svetlosnih izvora i svetlosnih komponenti
- [4] [https://commons.wikimedia.org/wiki/File:Shading\\_models.png](https://commons.wikimedia.org/wiki/File:Shading_models.png), stranica odakle je preuzeta slika različitih tehnika senčenja primenjenih na lopti
- [5] <https://sr.wikipedia.org/sr-el/OpenGL>, glavna stranica *OpenGL*-a
- [6] <https://github.com/ocornut/imgui>, repozitorijum biblioteke *Dear ImGui*
- [7] <https://github.com/assimp/assimp>, repozitorijum biblioteke *Assimp*
- [8] panosdalk, "Simple Motorcycle Free 3D Model - .blend .obj .mtl - Free3D ", <https://free3d.com/3d-model/simple-motorcycle-836902.html>, stranica odakle je preuzet model motora
- [9] Mark J. Kilgard, "OpenGL/VRML Materials", <http://devernay.free.fr/cours/opengl/materials.html>, stranica za parametre svetlosnih komponenti različitih materijaja

## 8. Dodatak A

```
#version 330 core

layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aUV;

uniform mat4 uProjection;
uniform mat4 uView;
uniform mat4 uModel;

out vec2 UV;
out vec3 vWorldSpaceFragment;
out vec3 vWorldSpaceNormal;
out vec3 FragColor;

struct PositionalLight {
    vec3 Position;
    vec3 Ka;
    vec3 Kd;
    vec3 Ks;
    float Kc;
    float Kl;
    float Kq;
};

struct DirectionalLight {
    vec3 Position;
    vec3 Direction;
    vec3 Ka;
    vec3 Kd;
    vec3 Ks;
    float InnerCutoff;
    float OuterCutoff;
    float Kc;
    float Kl;
    float Kq;
};

struct Material {
    vec3 Ka;
    vec3 Kd;
    vec3 Ks;
    float Shininess;
};

uniform PositionalLight uSunLight;
uniform DirectionalLight uFlashLight;
uniform DirectionalLight uDirLight;
uniform Material uMaterial;
uniform vec3 uViewPos;

void main() {
    vec3 WorldSpaceVertex = vec3(uModel * vec4(aPos, 1.0f));
    vec3 WorldSpaceNormal = normalize(mat3(transpose(inverse(uModel))) * aNormal);
    vec3 ViewDirection = normalize(uViewPos - WorldSpaceVertex);

    vec3 DirLightVector = normalize(-uDirLight.Direction);
    float DirDiffuse = max(dot(WorldSpaceNormal, DirLightVector), 0.0f);
    vec3 DirReflectDirection = reflect(-DirLightVector, WorldSpaceNormal);
    float DirSpecular = pow(max(dot(ViewDirection, DirReflectDirection), 0.0f),
uMaterial.Shininess);
```

```

vec3 DirAmbientColor = uDirLight.Ka * uMaterial.Ka;
vec3 DirDiffuseColor = uDirLight.Kd * DirDiffuse * uMaterial.Kd;
vec3 DirSpecularColor = uDirLight.Ks * DirSpecular * uMaterial.Ks;
vec3 DirColor = DirAmbientColor + DirDiffuseColor + DirSpecularColor;
vec3 PtLightVector = normalize(uSunLight.Position - WorldSpaceVertex);
float PtDiffuse = max(dot(WorldSpaceNormal, PtLightVector), 0.0f);
vec3 PtReflectDirection = reflect(-PtLightVector, WorldSpaceNormal);
float PtSpecular = pow(max(dot(ViewDirection, PtReflectDirection), 0.0f),
uMaterial.Shininess);
vec3 PtAmbientColor = uSunLight.Ka * uMaterial.Ka;
vec3 PtDiffuseColor = PtDiffuse * uSunLight.Kd * uMaterial.Kd;
vec3 PtSpecularColor = PtSpecular * uSunLight.Ks * uMaterial.Ks;
float PtLightDistance = length(uSunLight.Position - WorldSpaceVertex);
float PtAttenuation = 1.0f / (uSunLight.Kc + uSunLight.Kl * PtLightDistance +
uSunLight.Kq * (PtLightDistance * PtLightDistance));
vec3 PtColorSun = PtAttenuation * (PtAmbientColor + PtDiffuseColor +
PtSpecularColor);
vec3 SpotlightVector = normalize(uFlashLight.Position - WorldSpaceVertex);
float SpotDiffuse = max(dot(WorldSpaceNormal, SpotlightVector), 0.0f);
vec3 SpotReflectDirection = reflect(-SpotlightVector, WorldSpaceNormal);
float SpotSpecular = pow(max(dot(ViewDirection, SpotReflectDirection), 0.0f),
uMaterial.Shininess);
vec3 SpotAmbientColor = uFlashLight.Ka * uMaterial.Ka;
vec3 SpotDiffuseColor = SpotDiffuse * uFlashLight.Kd * uMaterial.Kd;
vec3 SpotSpecularColor = SpotSpecular * uFlashLight.Ks * uMaterial.Ks;
float SpotlightDistance = length(uFlashLight.Position - WorldSpaceVertex);
float SpotAttenuation = 1.0f / (uFlashLight.Kc + uFlashLight.Kl *
SpotlightDistance + uFlashLight.Kq * (SpotlightDistance * SpotlightDistance));
float Theta = dot(SpotlightVector, normalize(-uFlashLight.Direction));
float Epsilon = uFlashLight.InnerCutOff - uFlashLight.OuterCutOff;
float SpotIntensity = clamp((Theta - uFlashLight.OuterCutOff) / Epsilon, 0.0f,
1.0f);
vec3 SpotColor = SpotIntensity * SpotAttenuation * (SpotAmbientColor +
SpotDiffuseColor + SpotSpecularColor);
FragColor = DirColor + PtColorSun + SpotColor;
gl_Position = uProjection * uView * uModel * vec4(aPos, 1.0f);
}

```

#### Dodatak A 1 Sadržaj datoteke gouraud.vert

```

#version 330 core
in vec2 UV;
in vec3 vWorldSpaceFragment;
in vec3 vWorldSpaceNormal;
in vec3 FragColor;
out vec4 FinalColor;
void main() {
    FinalColor = vec4(FragColor, 1.0f);
}

```

#### Dodatak A 2 Sadržaj datoteke gouraud.frag

```

#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aUV;
uniform mat4 uProjection;
uniform mat4 uView;
uniform mat4 uModel;
out vec2 UV;
out vec3 vWorldSpaceFragment;
out vec3 vWorldSpaceNormal;
void main() {
    vWorldSpaceFragment = vec3(uModel * vec4(aPos, 1.0f));
    vWorldSpaceNormal = normalize(mat3(transpose(inverse(uModel))) * aNormal);
    UV = aUV;
    gl_Position = uProjection * uView * uModel * vec4(aPos, 1.0f);
}

```

#### Dodatak A 3 Sadržaj datoteke phong.vert

```

#version 330 core

struct PositionalLight {
    vec3 Position;
    vec3 Ka;
    vec3 Kd;
    vec3 Ks;
    float Kc;
    float Kl;
    float Kq;
};

struct DirectionalLight {
    vec3 Position;
    vec3 Direction;
    vec3 Ka;
    vec3 Kd;
    vec3 Ks;
    float InnerCutoff;
    float OuterCutoff;
    float Kc;
    float Kl;
    float Kq;
};

struct Material {
    vec3 Ka;
    vec3 Kd;
    vec3 Ks;
    float Shininess;
};

uniform PositionalLight uSunLight;
uniform DirectionalLight uFlashLight;

uniform DirectionalLight uDirLight;
uniform Material uMaterial;
uniform vec3 uViewPos;

in vec2 UV;
in vec3 vWorldSpaceFragment;
in vec3 vWorldSpaceNormal;

out vec4 FragColor;

void main() {
    vec3 ViewDirection = normalize(uViewPos - vWorldSpaceFragment);

    vec3 DirLightVector = normalize(-uDirLight.Direction);
    float DirDiffuse = max(dot(vWorldSpaceNormal, DirLightVector), 0.0f);
    vec3 DirReflectDirection = reflect(-DirLightVector, vWorldSpaceNormal);
    float DirSpecular = pow(max(dot(ViewDirection, DirReflectDirection), 0.0f),
uMaterial.Shininess);
    vec3 DirAmbientColor = uDirLight.Ka * uMaterial.Ka;
    vec3 DirDiffuseColor = uDirLight.Kd * DirDiffuse * uMaterial.Kd;
    vec3 DirSpecularColor = uDirLight.Ks * DirSpecular * uMaterial.Ks;
    vec3 DirColor = DirAmbientColor + DirDiffuseColor + DirSpecularColor;

    vec3 PtLightVector = normalize(uSunLight.Position - vWorldSpaceFragment);
    float PtDiffuse = max(dot(vWorldSpaceNormal, PtLightVector), 0.0f);
    vec3 PtReflectDirection = reflect(-PtLightVector, vWorldSpaceNormal);
    float PtSpecular = pow(max(dot(ViewDirection, PtReflectDirection), 0.0f),
uMaterial.Shininess);

```



```

    vec3 PtAmbientColor = uSunLight.Ka * uMaterial.Ka;
    vec3 PtDiffuseColor = PtDiffuse * uSunLight.Kd * uMaterial.Kd;
    vec3 PtSpecularColor = PtSpecular * uSunLight.Ks * uMaterial.Ks;
    float PtLightDistance = length(uSunLight.Position - vWorldSpaceFragment);
    float PtAttenuation = 1.0f / (uSunLight.Kc + uSunLight.Kl * PtLightDistance +
uSunLight.Kq * (PtLightDistance * PtLightDistance));
    vec3 PtColorSun = PtAttenuation * (PtAmbientColor + PtDiffuseColor +
PtSpecularColor);

    vec3 SpotlightVector = normalize(uFlashLight.Position - vWorldSpaceFragment);
    float SpotDiffuse = max(dot(vWorldSpaceNormal, SpotlightVector), 0.0f);
    vec3 SpotReflectDirection = reflect(-SpotlightVector, vWorldSpaceNormal);
    float SpotSpecular = pow(max(dot(ViewDirection, SpotReflectDirection), 0.0f),
uMaterial.Shininess);
    vec3 SpotAmbientColor = uFlashLight.Ka * uMaterial.Ka;
    vec3 SpotDiffuseColor = SpotDiffuse * uFlashLight.Kd * uMaterial.Kd;
    vec3 SpotSpecularColor = SpotSpecular * uFlashLight.Ks * uMaterial.Ks;
    float SpotlightDistance = length(uFlashLight.Position - vWorldSpaceFragment);
    float SpotAttenuation = 1.0f / (uFlashLight.Kc + uFlashLight.Kl *
SpotlightDistance + uFlashLight.Kq * (SpotlightDistance * SpotlightDistance));
    float Theta = dot(SpotlightVector, normalize(-uFlashLight.Direction));
    float Epsilon = uFlashLight.InnerCutOff - uFlashLight.OuterCutOff;
    float SpotIntensity = clamp((Theta - uFlashLight.OuterCutOff) / Epsilon, 0.0f,
1.0f);
    vec3 SpotColor = SpotIntensity * SpotAttenuation * (SpotAmbientColor +
SpotDiffuseColor + SpotSpecularColor);

    vec3 FinalColor = DirColor + PtColorSun + SpotColor;
    FragColor = vec4(FinalColor, 1.0f);
}

```

Dodatak A 4 Sadržaj datoteke phong\_material.frag

## 9. Podaci o kandidatu

Kandidat Jovan Srdanov je rođen 2000. godine u Zrenjaninu. Završio je osnovnu školu “Petar Petrović Njegoš” u Zrenjaninu, 2015. godine. Nakon osnovne škole, pohađao je elektrotehničku i građevinsku školu “Nikola Tesla” u Zrenjaninu, 2019. godine. Fakultet Tehničkih Nauka u Novom Sadu je upisao 2019. godine. Ispunio je sve obaveze i položio je sve ispite predviđene studijskim programom sa prosečnom ocenom od 8.72.