|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** |  |

Јован Срданов

**Прогресивни преглед и анализа разлика између различитих техника сенчења у рачунарској графици**

ДИПЛОМСКИ РАД

- Основне академске студије -

Нови Сад, 2023.

**КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА**

|  |  |  |  |
| --- | --- | --- | --- |
| Редни број, **РБР**: | |  | |
| Идентификациони број, **ИБР**: | |  | |
| Тип документације, **ТД**: | | Монографска публикација | |
| Тип записа, **ТЗ**: | | Текстуални штампани документ/ЦД | |
| Врста рада, **ВР**: | | Завршни-bachelor рад | |
| Аутор, **АУ**: | | Јован Срданов | |
| Ментор, **МН**: | | Проф. др Драган Иветић | |
| Наслов рада, **НР**: | | Прогресивни преглед и анализа разлика између различитих техника сенчења у рачунарској графици | |
| Језик публикације, **ЈП**: | | Српски(ћирилица)/Српски (ћирилица) | |
| Језик извода, **ЈИ**: | | Српски/Енглески | |
| Земља публиковања, **ЗП**: | | Србија | |
| Уже географско подручје, **УГП**: | | Војводина | |
| Година, **ГО**: | | 2023 | |
| Издавач, **ИЗ**: | | Ауторски репринт | |
| Место и адреса, **МА**: | | Факултет Техничких Наука (ФТН), Д. Обрадовића 6, 21000 Нови Сад | |
| Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога) | | n/n/n/n/n/n/n где н замењује стварне вредности поглавља/страна/цитата/табела/слика/графика/прилога | |
| Научна област, **НО**: | | Електротехничко и рачунарско инжењерство | |
| Научна дисциплина, **НД**: | | Примењене рачунарске науке и информатика | |
| Предметна одредница/Кqучне речи, **ПО**: | | Рачунарска графика, темена у 3D простору, нормале темена, усредењивање нормала, сенчење, полигони, Фонг, Гуоро, OpenGL | |
| **УДК** | |  | |
| Чува се, **ЧУ**: | | Библиотека ФТН, Д. Обрадовића 6, 21000 Нови Сад | |
| Важна напомена, **ВН**: | |  | |
| Извод, **ИЗ**: | | Кратак опис рада који стаје у три до шест реченица. Треба навести проблем који се решавао, како се проблем решавао и који су резултати. | |
| Датум прихватања теме, **ДП**: | |  | |
| Датум одбране, **ДО**: | |  | |
| Чланови комисије, **КО**: | Председник: |  |
|  | Члан: |  | Потпис ментора |
|  | Члан, ментор: | Др Драган Иветић, ред. проф. |  |

**KEY WORDS DOCUMENTATION**

|  |  |  |  |
| --- | --- | --- | --- |
| Accession number, **ANO**: | |  | |
| Identification number, **INO**: | |  | |
| Document type, **DT**: | | Monographic publication | |
| Type of record, **TR**: | | Textual material, printed/CD | |
| Contents code, **CC**: | | Bachelor thesis | |
| Author, **AU**: | | Jovan Srdanov | |
| Mentor, **MN**: | | Dragan Ivetić, PhD, full professor | |
| Title, **TI**: | | Progressive overview and analysis of the differences between different shading techniques in computer graphics | |
| Language of text, **LT**: | | Serbian (cyrillic script)/Serbian (latin script) | |
| Language of abstract, **LA**: | | Serbian/English | |
| Country of publication, **CP**: | | Serbia | |
| Locality of publication, **LP**: | | Vojvodina | |
| Publication year, **PY**: | | 2010 | |
| Publisher, **PB**: | | Author reprint | |
| Publication place, **PP**: | | Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | n/n/n/n/n/n/n where respective n’s stand for the total number of chapters/pages/references/tables/pictures/graphs/appendices | |
| Scientific field, **SF**: | | Electrical and computer engineering | |
| Scientific discipline, **SD**: | | Applied computer science and informatics | |
| Subject/Key words, **S**/**KW**: | | Computer graphics, vertices in 3D space, vertex normals, normal averaging, shading, polygons, Phong, Gouraud, OpenGL | |
| **UC** | |  | |
| Holding data, **HD**: | | Library of the Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad | |
| Note, **N**: | |  | |
| Abstract, **AB**: | | Brief abstract of the paper. It should have between three and six sentences and should explain the problem being solved, the methods being used to solve it and the conclusions that can be drawn from its solution. | |
| Accepted by the Scientific Board on, **ASB**: | |  | |
| Defended on, **DE**: | |  | |
| Defended Board, **DB**: | President: |  |
|  | Member: |  | Menthor's sign |
|  | Member, Mentor: | Dragan Ivetić, PhD, full professor |  |

*(Податке уноси предметни наставник - ментор)*

| Врста студија: | Основне академске студије |
| --- | --- |
| Студијски програм: | РАЧУНАРСТВО И АУТОМАТИКА |
| Руководилац студијског програма: | Проф. др Милан Рапаић |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | Jovan Srdanov | Број индекса: | RA 145/2019 |
| Област: |  | | |
| 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:**

|  |
| --- |
|  |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
|  |

|  |  |
| --- | --- |
| 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](#_Toc145176069)

[2. Teoretske osnove 5](#_Toc145176070)

[2.1 Temena 5](#_Toc145176071)

[2.2 Model 5](#_Toc145176072)

[2.3 Normale temena 6](#_Toc145176073)

[2.4 Usrednjene normale temena 6](#_Toc145176074)

[2.5 Izvori svetlosti 7](#_Toc145176075)

[2.5.1 Direkcioni izvor svetlosti 7](#_Toc145176076)

[2.5.2 Tačkasti izvor svetlosti 7](#_Toc145176077)

[2.5.3 Reflektorski izvor svetlosti 7](#_Toc145176078)

[2.6 Svetlostne komponente 7](#_Toc145176079)

[2.6.1 Ambijentalna komponenta 7](#_Toc145176080)

[2.6.2 Difuzna komponenta 7](#_Toc145176081)

[2.6.3 Spekularna komponenta 7](#_Toc145176082)

[2.7 Iluminacioni model 7](#_Toc145176083)

[2.8 Senčenje 7](#_Toc145176084)

[2.8.1 Konstanto senčenje 7](#_Toc145176085)

[2.8.2 Guroovo senčenje 7](#_Toc145176086)

[2.8.3 Fongovo senčenje 7](#_Toc145176087)

[3. Implementacija 8](#_Toc145176088)

[3.1 Tehnologije 8](#_Toc145176089)

[3.1.2 C++ 8](#_Toc145176090)

[3.1.2 *OpenGL*, *GLEW*, *GLFW3* 8](#_Toc145176091)

[3.1.4 *VAO*, *VBO*, *EBO* 9](#_Toc145176092)

[3.1.5 *Dear ImGUI* 9](#_Toc145176093)

[3.2 Učitavanje i obrada modela i *mesh-*а 9](#_Toc145176094)

[3.3 Šejderi 15](#_Toc145176095)

[3.8 Rednerovanje modela u zavisnosti od odabranog moda 16](#_Toc145176096)

[4. Primer korišćenja 18](#_Toc145176097)

[4.1 Korisničke komande 18](#_Toc145176098)

[4.2 Modovi 18](#_Toc145176099)

[4.2.1 Mod 01 - Temena 19](#_Toc145176100)

[4.2.2 Mod 02 - Trouglovi 20](#_Toc145176101)

[4.2.3 Mod 03 - Popunjeni trouglovi bez vidljive mreže 20](#_Toc145176102)

[4.2.4 Mod 04 - Popunjeni trouglovi sa vidljivom mrežom 21](#_Toc145176103)

[4.2.5 Mod 05 - Sve normale 21](#_Toc145176104)

[4.2.6 Mod 06 - Usrednjene normale 22](#_Toc145176105)

[4.2.7 Mod 07- Senčenja 22](#_Toc145176106)

[4.2.8 Mod 08 - Teksture 24](#_Toc145176107)

[4.3 Baterijska lampa iz prvog lica 24](#_Toc145176108)

[4.4 Materijal modela na osnovu parametara svetlosnih komponenti 25](#_Toc145176109)

[4.5 Poređenje usrednjenih i svih normala 26](#_Toc145176110)

[4.6 Poređenje različitih tehnika senčenja 26](#_Toc145176111)

[5. Ograničenja i unapređenja 29](#_Toc145176112)

[5.1 Raznovrsnost modela i tekstura 29](#_Toc145176113)

[5.2 Specifičnost parametara 29](#_Toc145176114)

[5.3 Algoritam za izračunavanje usrednjenih normala 29](#_Toc145176115)

[5.4 Opcija za čuvanje stanja aplikacije 29](#_Toc145176116)

[5.5 Prelazak na internet 29](#_Toc145176117)

[6. Zaključak 30](#_Toc145176118)

[7. Literatura 31](#_Toc145176119)

[8. Dodatak A 32](#_Toc145176120)

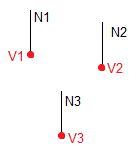
[9. Podaci o kandidatu 36](#_Toc145176121)

# 1. Uvod

# 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 mjau.



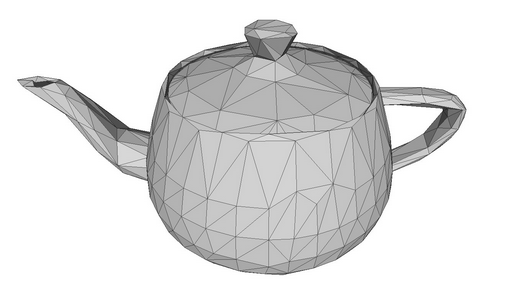
Slika 1 Temena sa normalama

Temena se koriste za opisivanje pozicije, boje, tekstura 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

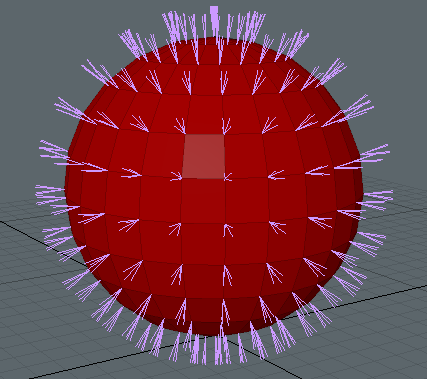
Model, u kontekstu računarstva i trodimenzionalne grafike, predstavlja digitalnu reprezentaciju trodimenzionalnog objekta ili scene. Ovaj modeli se mogu sastojati od mreža (*mesh*) koje se sastoje od temena povezanih trouglovima. Trouglovi su trostrane geometrijske figure koje povezuju temena, čineći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



Slika 2 Model čajnika

## 2.3 Normale temena

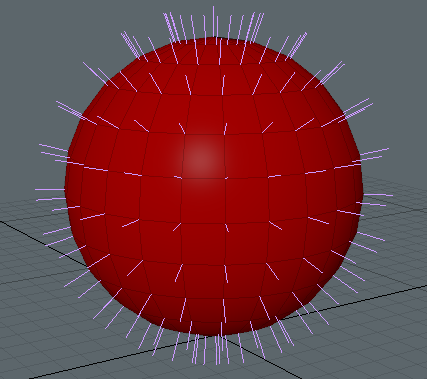
Normale temena su vektori koji se često koriste u računarskoj grafici kako bi se odredila orijentacija površine na svakom temenu unutar trodimenzionalnog modela. Ovi vektori pružaju informacije o tome kako svako teme reaguje na svetlost i kako će biti osenčeno na renderovanom objektu.Na slici mjau prikazan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



Slika 3 Model lopte sa neusrednjenim normalama temena

## 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 konstanog 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 mjau [mjau].



Slika 4 Model lopte sa usrednjenim normalama temena

## 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 maju su prikazani primeri tačkastog, reflektorskog i direkcionog izvora svetlа [mjau].



Slika 5 Primeri različitih izvora svetlosti

### 2.5.1 Tačkasti izvor svetlosti

Tačkasti izvor svetlosti simulira svetlost koja dolazi iz jedne tačke u prostoru i širi se u svim pravcima. Tackasti 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 tackastog 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 svetloske snopove. Reflektorski izvori svetlosti često se koriste za naglašavanje određenih delova scene, postizanje dramatičnih senki i stvaranje visokokontrastnih efekata.

### 2.5.3 Direkcioni izvor svetlosti

Direkcioni izvor svetlosti karakteriše snažan i usmeren snop svetlosti koji se širi u određenom smeru. Za razliku od tackastog izvora svetlosti koji emituje svetlost iz jedne tačke ili reflektorskog izvora sa koncentrisanim snopom svetlosti, direkcioni 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 Svetlostne 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 i prikazane su na slici mjau.



Slika 6 Svetlosne komponente

### 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 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 oštrih kontrasta.

### 2.6.3 Spekularna svetlost

Spekularna svetlost simulira refleksiju svetlosti sa 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, 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.

*–* intezitet reflektovane svetlosti, *–* vektor gledanja kamere ili posmatrača

*–* refleksioni koeficijent materijala za ambijentalno svetlo

*–* refleksioni koeficijent materijala za difuzno svetlo

*–* refleksioni koeficijent materijala za spekularno svetlo

*–* intezitet ambijentalne komponente svetla

*–* intezitet difuzne komponente svetla

*–* intezitet spekularne komponente svetla

*–* koeficijenti konstatnog, linearnog i kvadratnog slabljenja svetla na osnovu udaljenosti od svetlosnog izvora

*–* upadni ugao,  *–* koeficijent sjajnosti

*–* ugao između reflekcionog vektora i vektora gledana kamere ili posmatrača

## 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 konstanto senčenje, Guroovo senčenje i Phongovo senčenje, svaka sa svojim prednostima i nedostacima. Prikaz lopte osenčene različitim tehnikama senčenja se nalazi na slici mjau [mjau].



Slika 7 Uticaj različitih tehnika senčenja na model lopte

### 2.8.1 Konstanto 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.

### 2.8.2 Guroovo senčenje

### 2.8.3 Fongovo senčenje

# 3. Implementacija

U ovom poglavlju se detaljnije opisuju korišćene tehnologije kao i sama implementacija rešenja zadatka u tim tehnologijama.

## 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 omogućene 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 višeplatformski 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.[referenca]

*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 API. 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, a 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.

*Dear ImGui* je dizajniran da omogući brze iteracije i 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-*а

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

Za učitavanje se koristi biblioteka *Assimp* [mjau]*.* 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 3D modela za dalje upotrebe. 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 konstantog 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 konstatno 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 učitanog *mesh*-a. Rezultati izračunavanja se skladište u posebne datoteke i učitavaju pri svakom narednom pokretanju aplikacije. Na listingu 8 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**

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 7 Algoritam za izračunavanje usrednjenih normala**

Metoda smoothSetup () popunjava odgovarajuće podatke neophodne za prikaz modela prilikom Guroovog 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 8.

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 8 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 9 su implementacije metoda za 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 9 Metode za različite slučajeve renderovanja**

## 3.3 Šejderi

Šejderi su programi koji se koriste u procesu renderovanja trodimenzione 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, teksture 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 trodimenzionog prostora u dvodimenzione 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 tekstura 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 10.

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 10 Polja i metode klase Shader**

U glavnoj funckiji programa, Guroov i Fongov šejder su kreirani na način prikazan na listingu 11.

Shader gouraud\_shader\_material("shaders/gouraud.vert", "shaders/gouraud.frag");

Shader phong\_shader\_material("shaders/phong.vert", "shaders/phong\_material.frag");

**Listing 11 Kreiranje Guroovog i Fongovog šejdera**

Sadržaj datoteke gouraud.vert, gouraud.frag, basic.vert phong\_material.frag i se nalazi u dodatku A.

## 3.8 Rednerovanje modela u zavisnosti od odabranog moda

U glavnoj funkciji dolazi do renderovanja učitanog modela u željenom modu. Pored do sad 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 12.

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 12 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 13.

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 12 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 13 su prikazane implementacije 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 13 Prikaz dostupnih modova senčenja**

## 4. Primer korišćenja

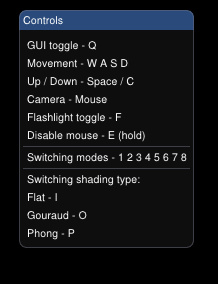
Kada korisnik pokrene aplikaciju, dočekuje ga početni ekran koji sjedinjuje dvodimenzionalni grafički interfejs i trodimenzionalni model prikazan u modu 01. Učitani model koji se prikazuje korisniku je model motora [1].

## 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 – Konstanto, O – Guroovo, P - Fongovo).

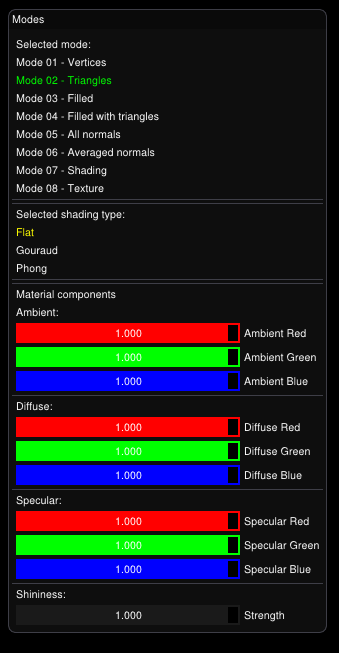
Izgled prozora za korisničke opcije prikazan je na slici MJAU.



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 MJAU



Slika 9 Prozor sa modovima i klizačima

### 4.2.1 Mod 01 - Temena

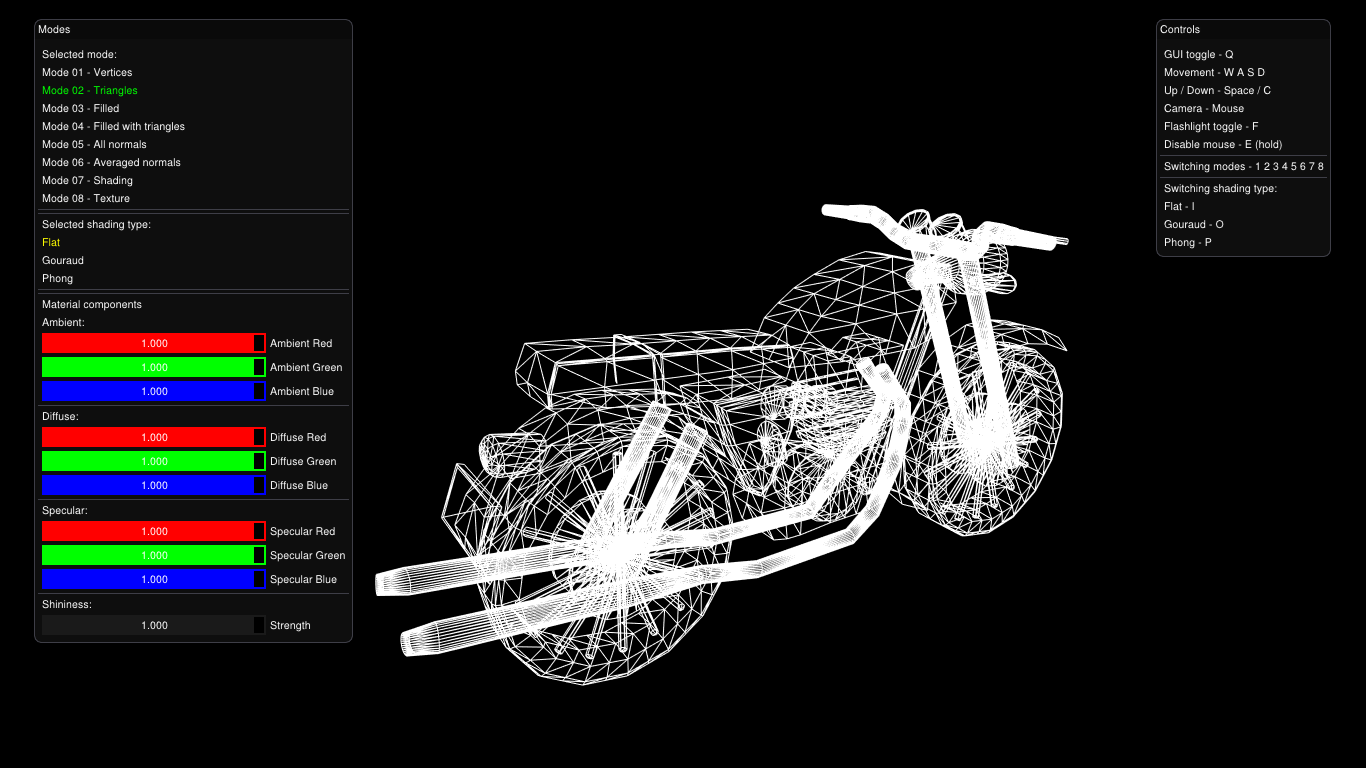
Prilikom odabira moda 01 korisniku se prikazuje model motora sačinjen samo od temena motora obojenih belom bojom. Na slici MJAU se nalazi prikaz ovog moda.



Slika 10 Mod 01 - Temena

### 4.2.2 Mod 02 - Trouglovi

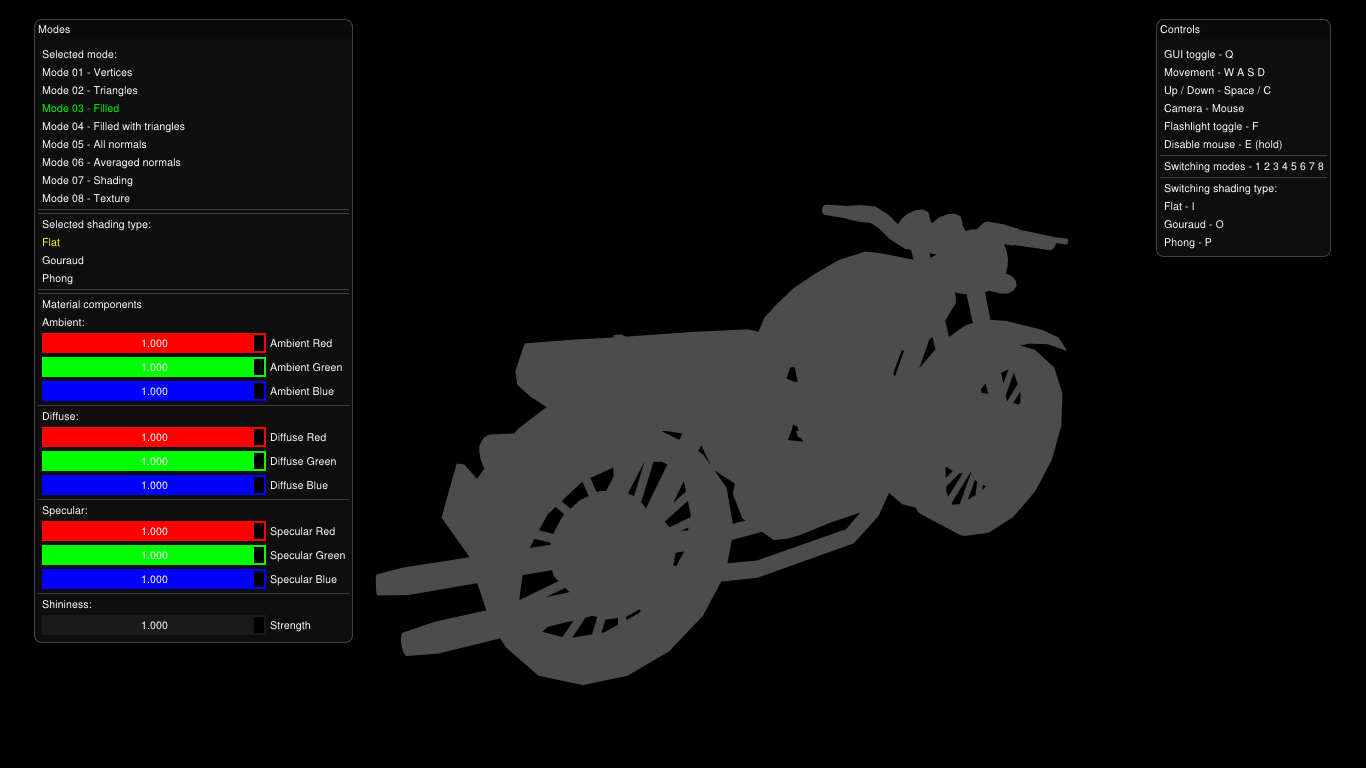
Prilikom odabira moda 02 korisniku se prikazuje model motora sačinjen od povezanih temena motora koji formiraju trouglove. Trouglovi nisu popunjeni. Ivice trouglova su obojene belom bojom. Na slici MJAU se nalazi prikaz ovog moda.



Slika 11 Mod 02 - Trouglovi

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

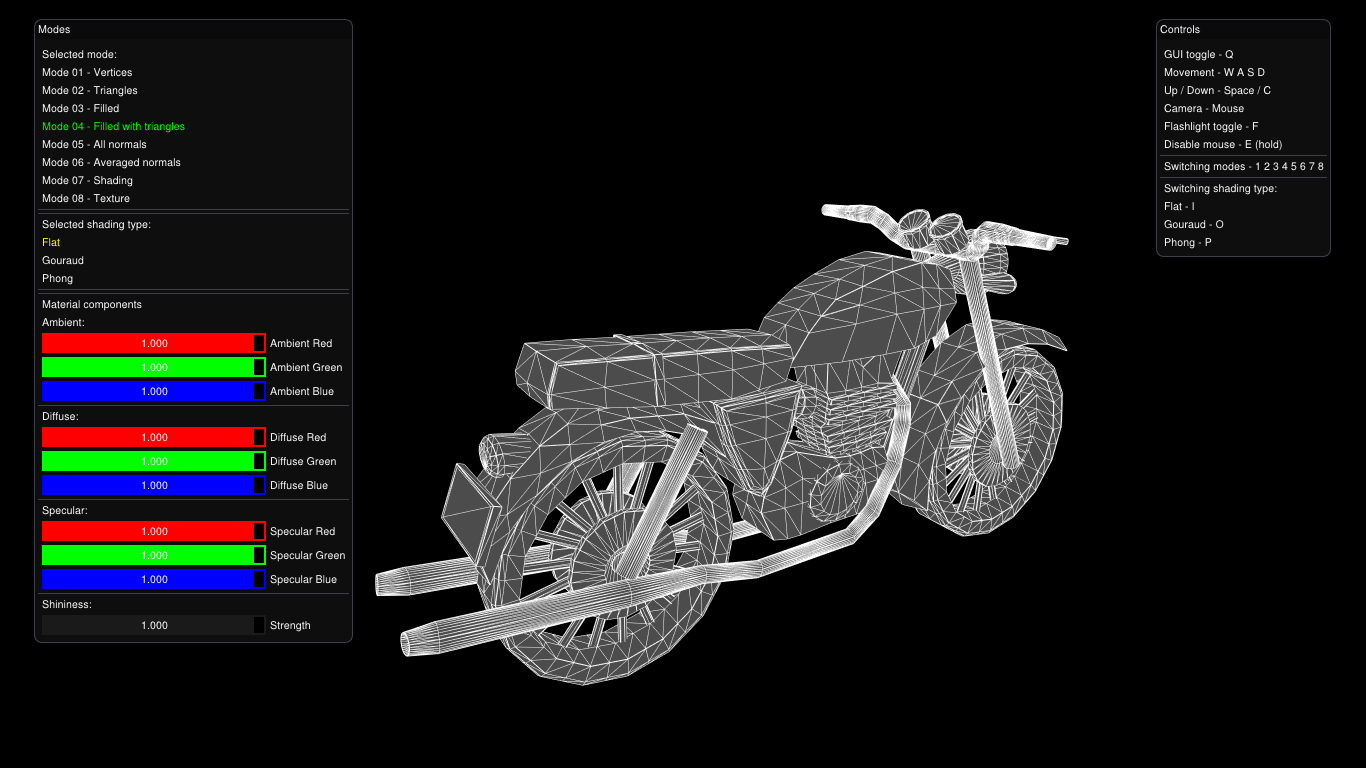
Prilikom odabira moda 03 korisniku se prikazuje model motora sačinjen od povezanih temena motora koji formiraju trouglove. Trouglovi su popunjeni sivom bojom. Na slici MJAU se nalazi prikaz ovog moda.



Slika 12 Mod 03 - Popunjeni trouglovi

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

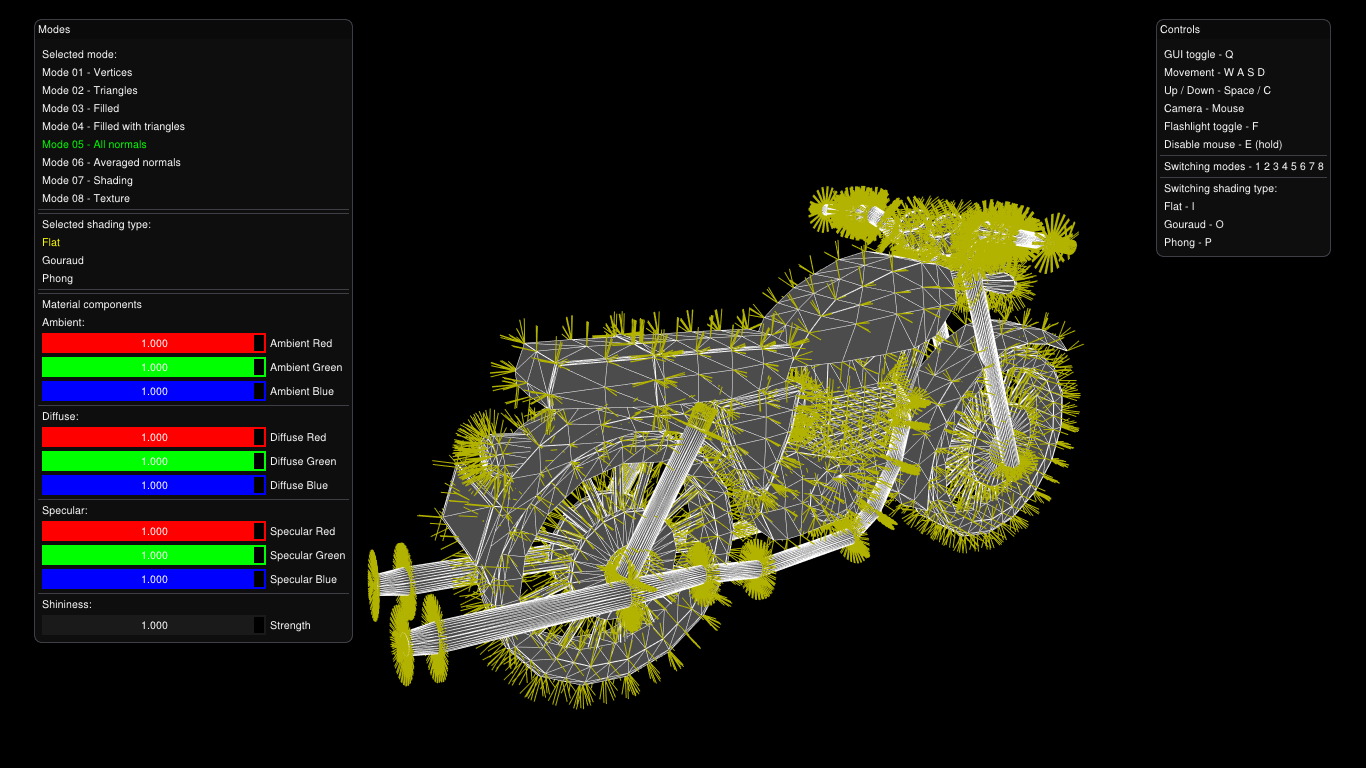
Prilikom odabira moda 04 korisniku se prikazuje model motora sačinjen od povezanih temena motora koji formiraju trouglove. Trouglovi su popunjeni sivom bojom. Ivice trouglova su obojene belom bojom. Na slici MJAU se nalazi prikaz ovog moda.



Slika 13 Mod 04 - Popunjeni trouglovi sa vidljivom mrežom

### 4.2.5 Mod 05 - Sve normale

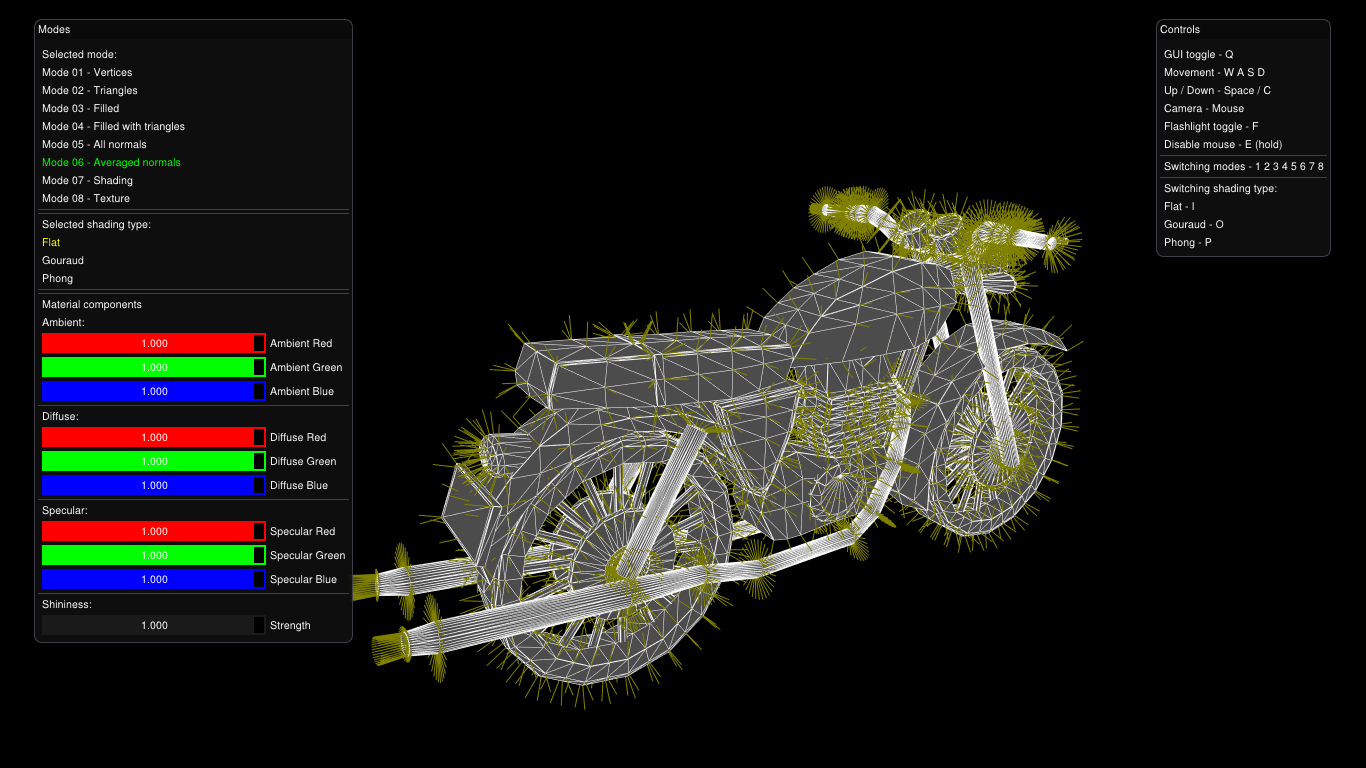
Prilikom odabira moda 05 korisniku se prikazuje isti prikaz kao i u modu 04, ali sa dodatim svim normalama koje izlaze iz temena. Normale su obojene žutom bojom. Na slici MJAU se nalazi prikaz ovog moda.



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 MJAU se nalazi prikaz ovog moda.



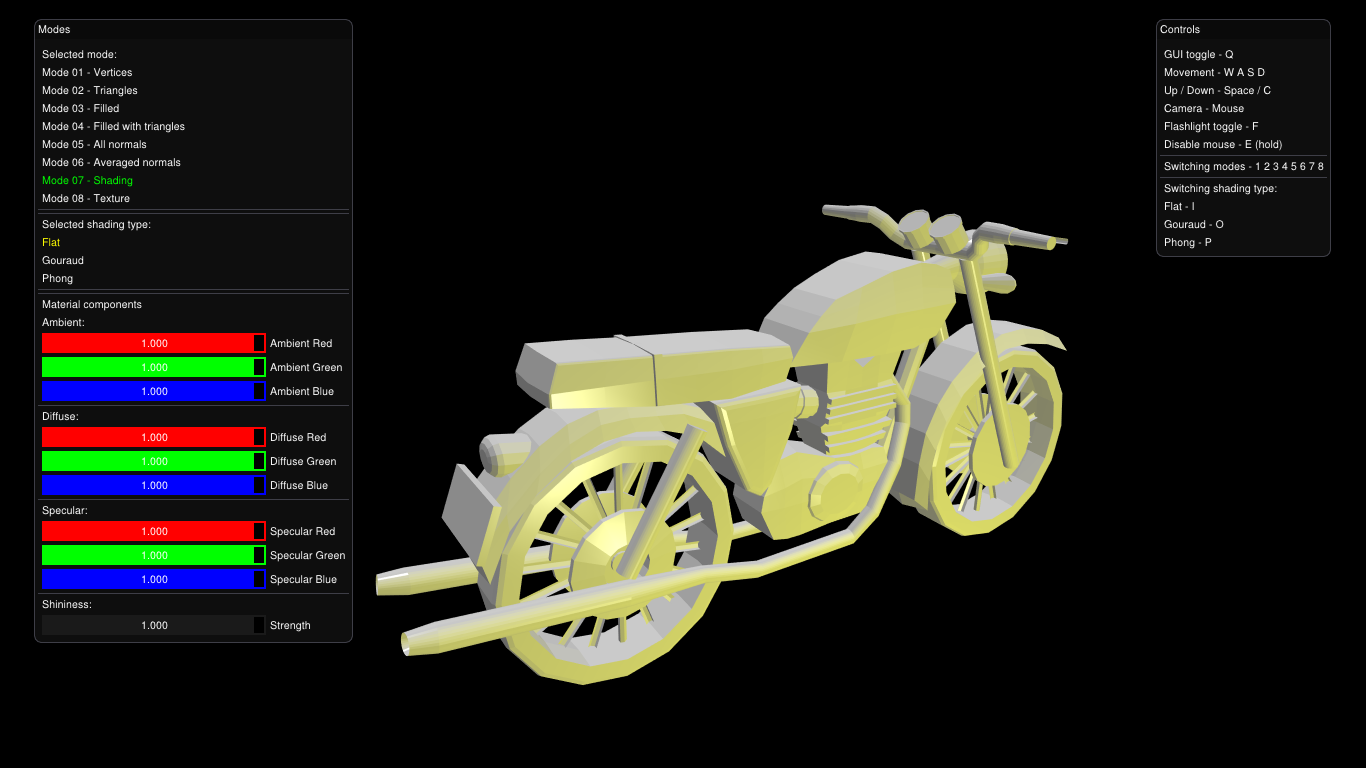
Slika 15 Mod 06 - Usrednjene normale

### 4.2.7 Mod 07- Senčenja

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

#### 4.2.7.1 Konstanto senčenje

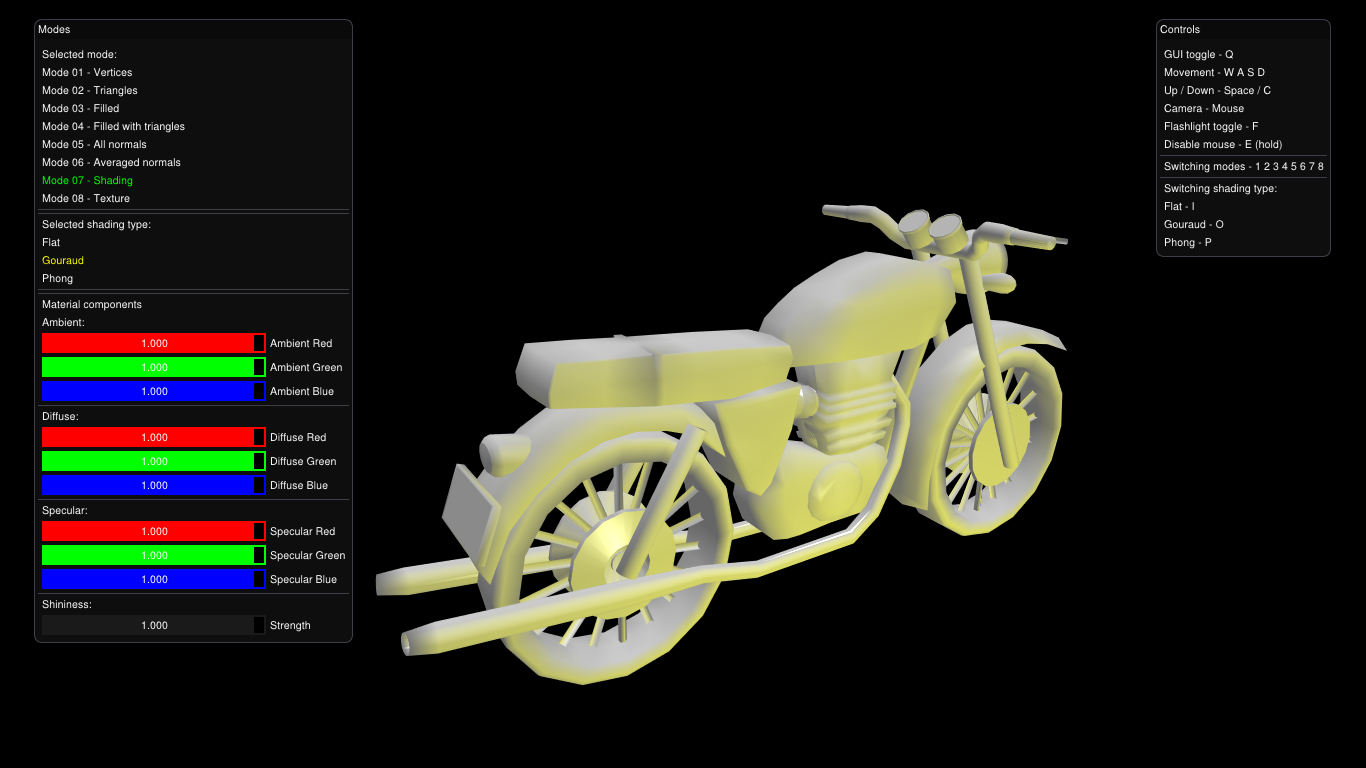
Prilikom odabira konstatnog senčenja korisniku se prikazuje model prikazan na slici mjau.



Slika 16 Mod 07 - Konstanto senčenje

#### 4.2.7.2 Guroovo senčenje

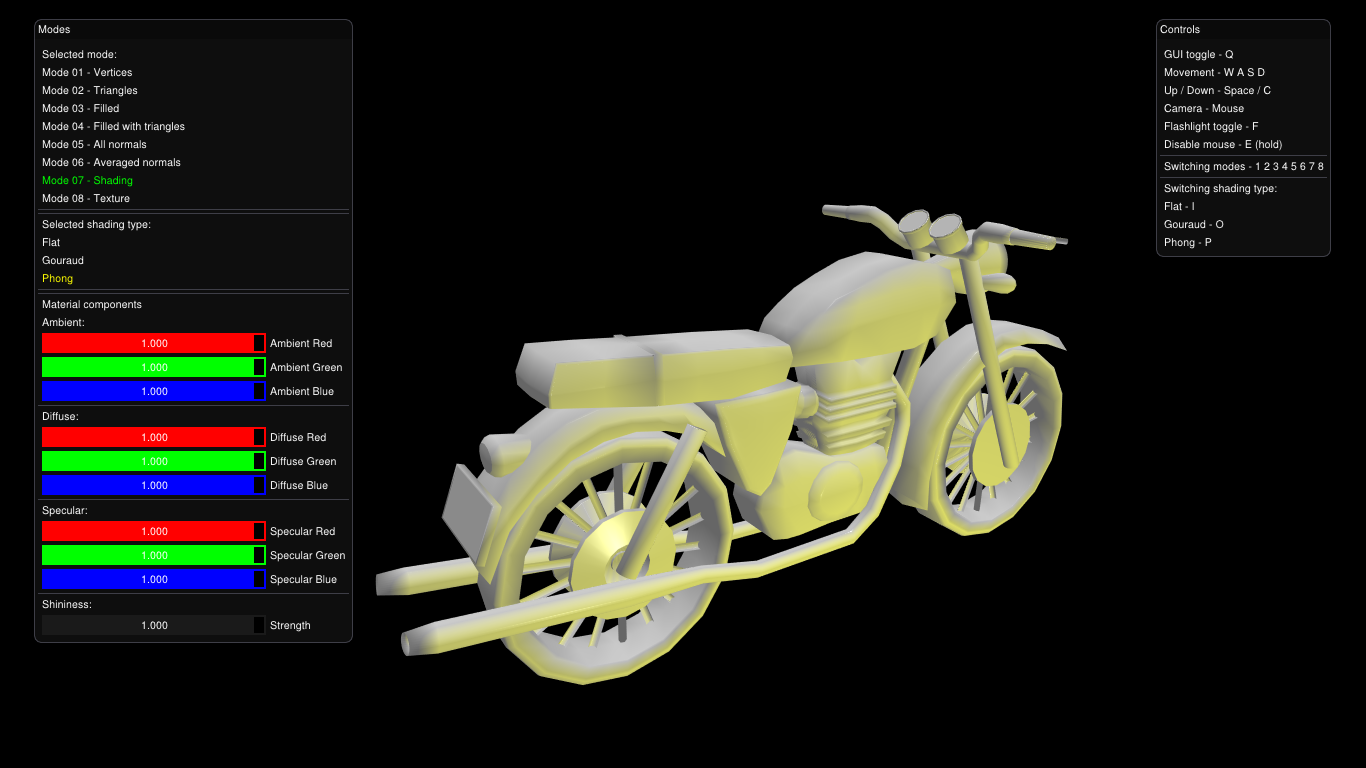
Prilikom odabira Guroovog senčenja korisniku se prikazuje model prikazan na slici mjau.



Slika 17 Mod 07 - Guroovo senčenje

#### 4.2.7.3 Fongovo senčenje

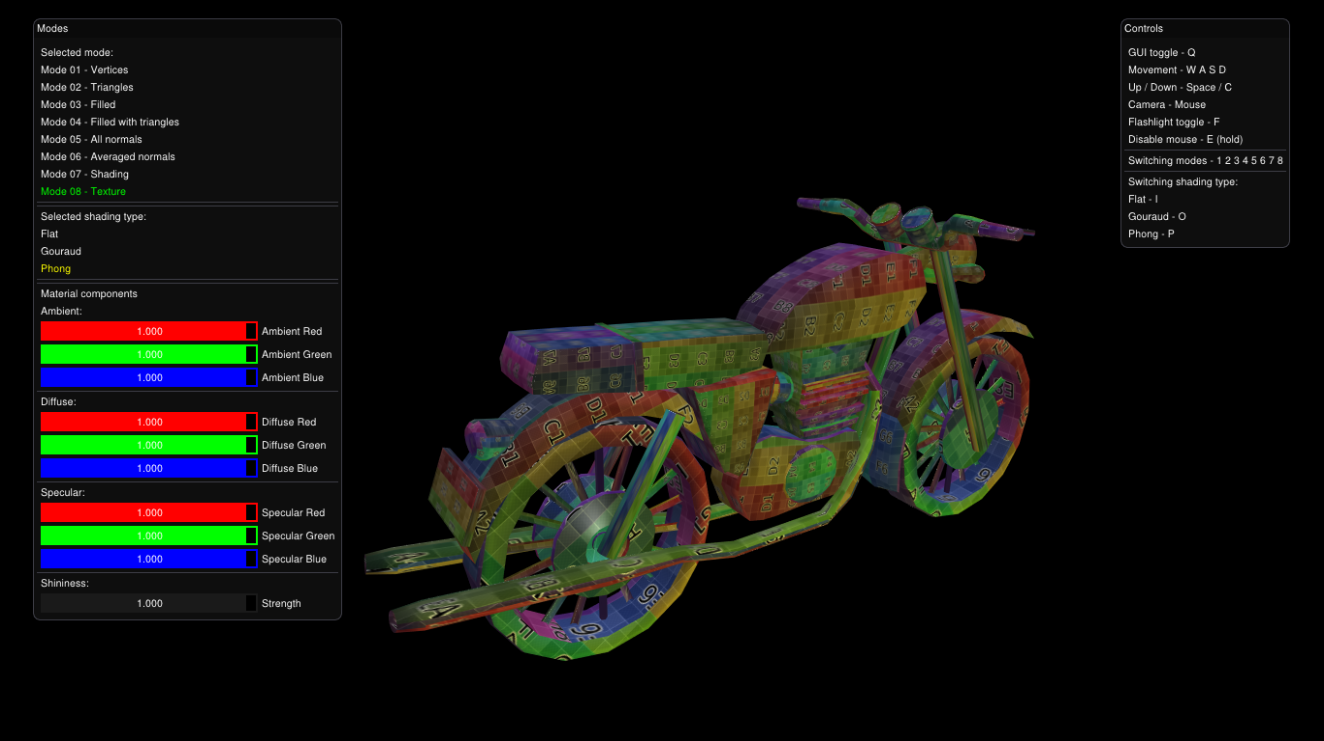
Prilikom odabira Fongovog senčenja korisniku se prikazuje model prikazan na slici mjau.



Slika 18 Mod 07 - Fongovo senčenje

### 4.2.8 Mod 08 - Teksture

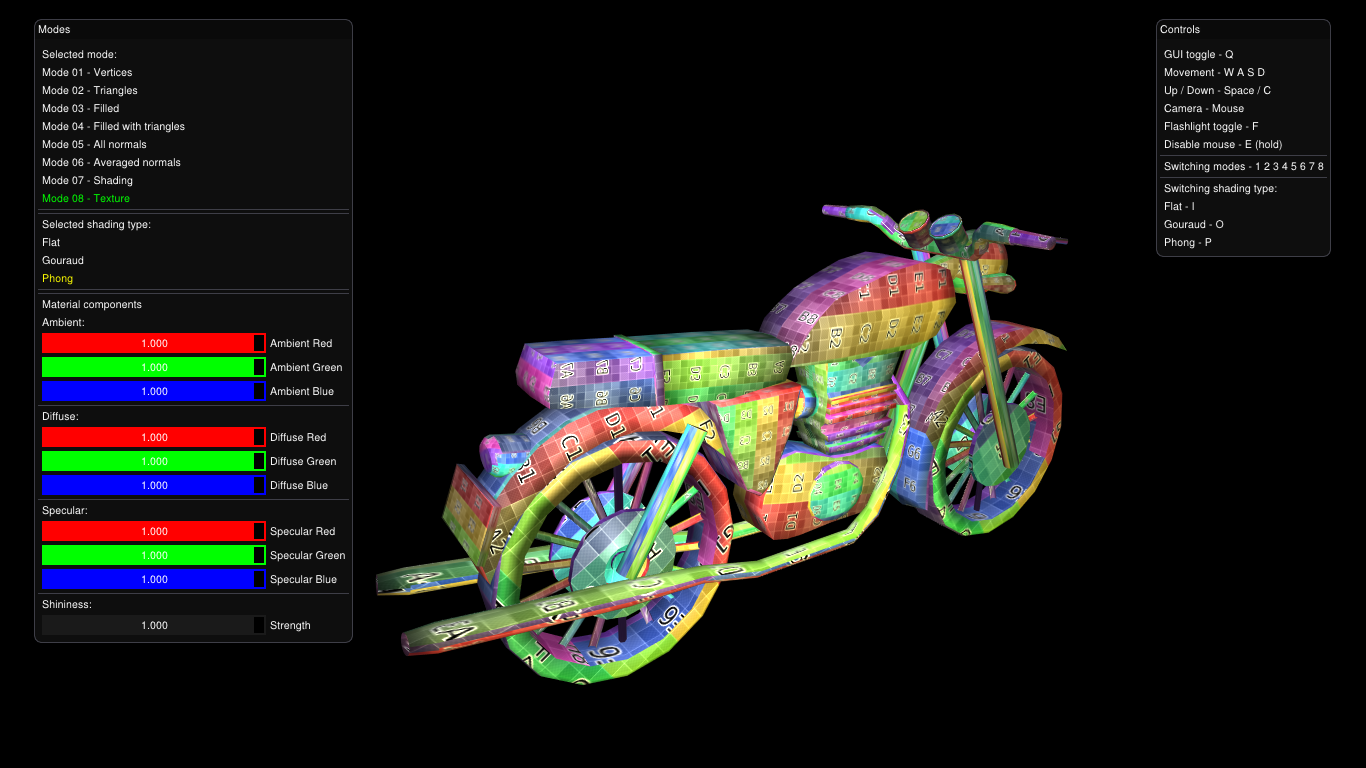
Prilikom odabira moda 08 korisniku se prikazuje model motora sa testnom difuznom i spekularnom teksturom. Primenjeno je Fongovo senčenje. Na slici MJAU se nalazi prikaz ovog moda.



Slika 19 Mod 08 - Teksture

## 4.3 Baterijska lampa iz prvog lica

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



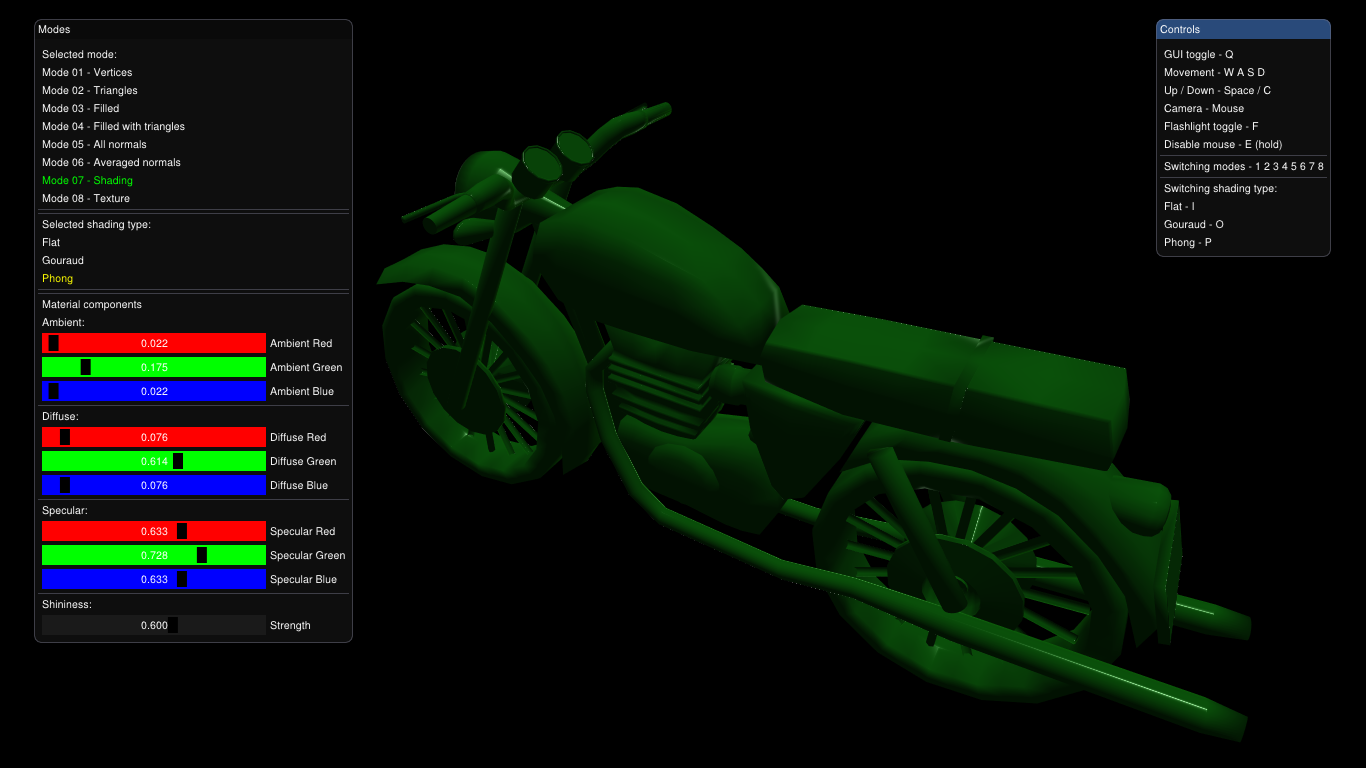
Slika 20 Osvetljene baterijskom lampom

## 4.4 Materijal modela na osnovu parametara svetlosnih komponenti

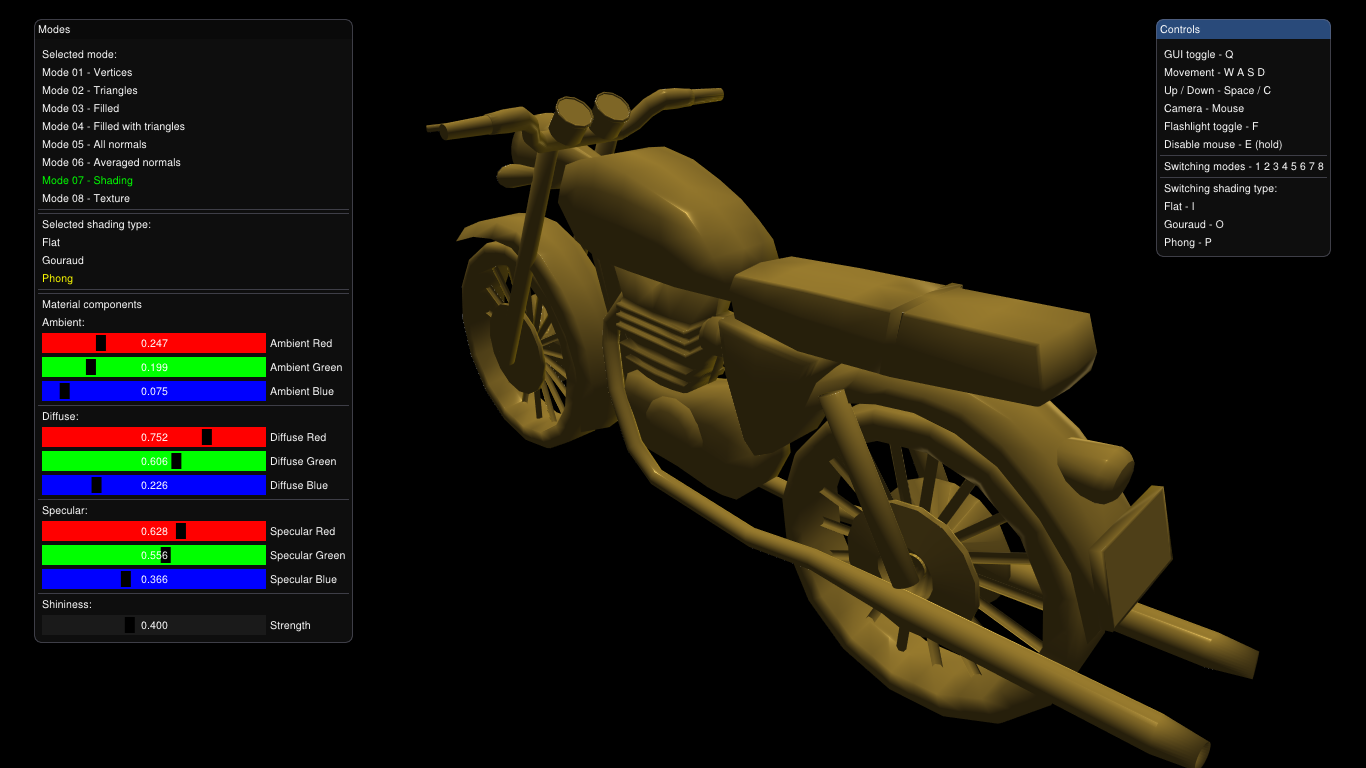
Promenom parametara klizača, menja se i izgled samog modela. Korišćenjem već određenih vrednosti za neke materijale, izgled modela se menja i deluje kao da je sačinjen od određenog materijala [[12](#materials)]. Vrednosti parametara koje razmatramo nalaze se u Tabeli 1. Prikaz smaragdnog motora nalazi se na slici 12, prikaz zlatnog motora nalazi se na slici 12 i prikaz motora sačinjenog od rubina nalazi se na slici 12.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 |

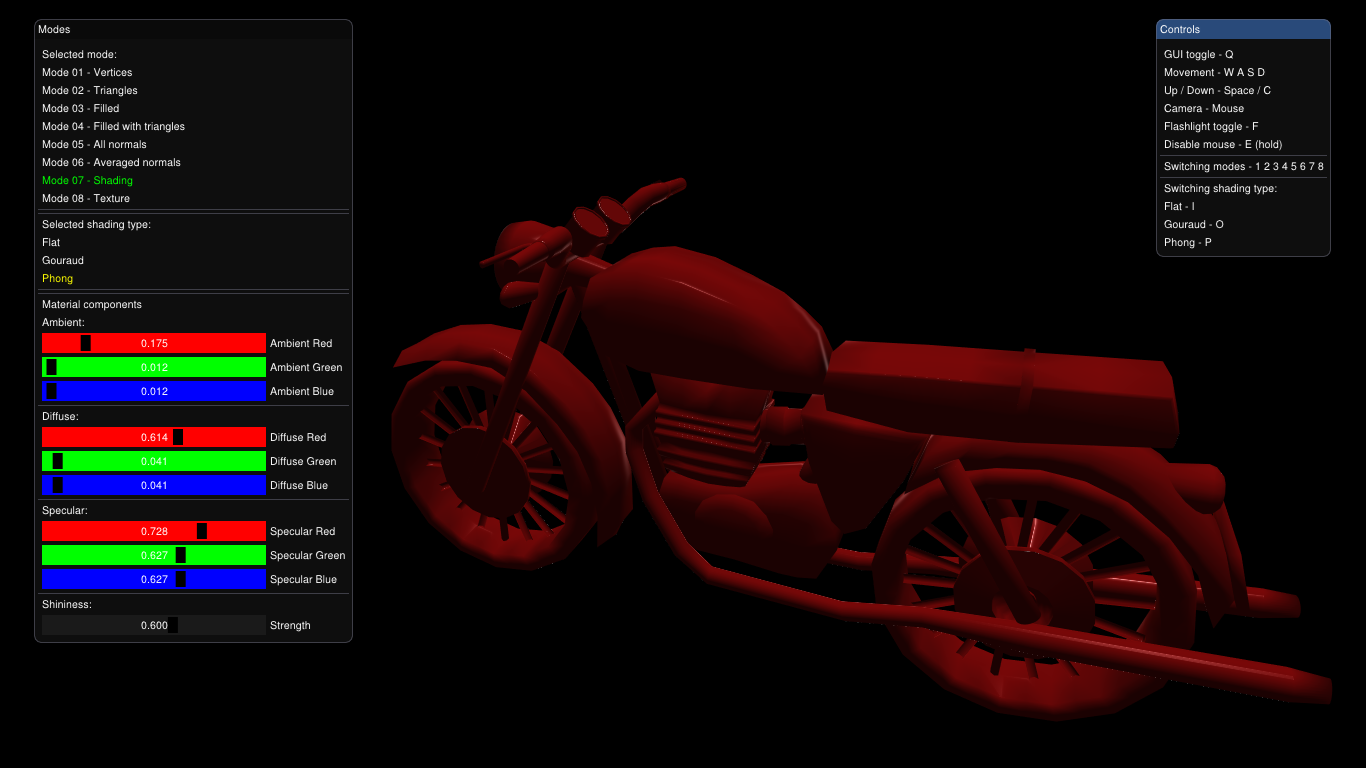
Tabela 1 Vrednosti svetlosnih komponenti za različite materijale



Slika 21 Smaragdni motor



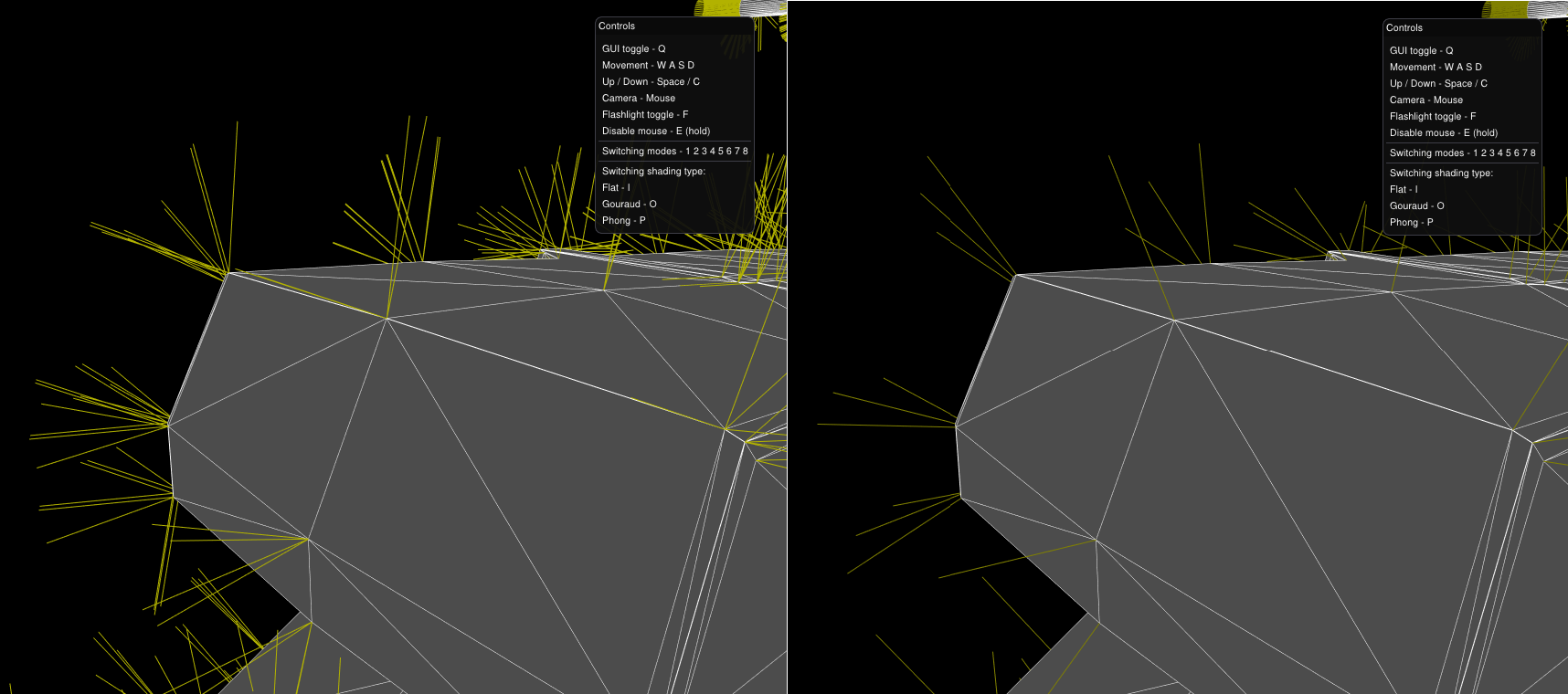
Slika 22 Zlatni motor



Slika 23 Motor sačinjen od rubina

## 4.5 Poređenje usrednjenih i svih normala

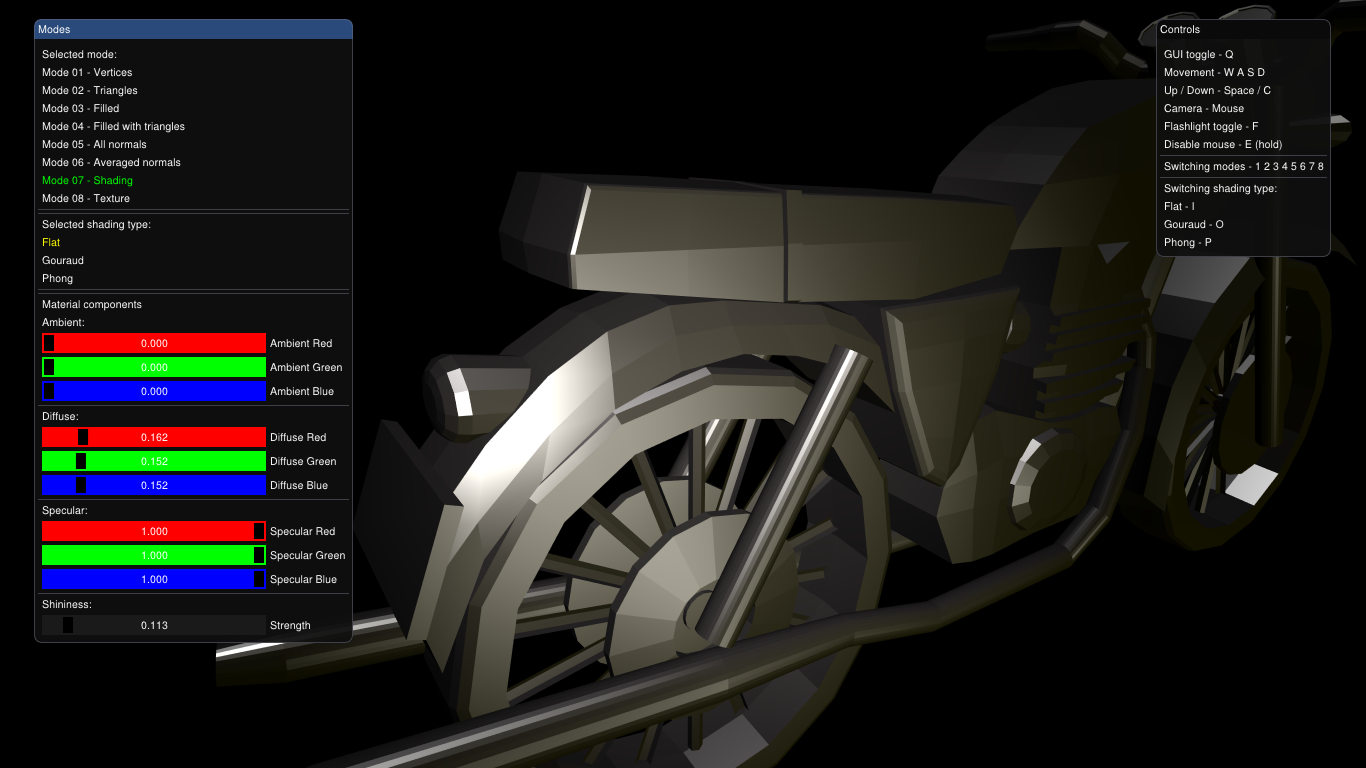
Na slici mjau je sa leve strane prikaz svih normala iz temena, a sa desne strane slike se nalazi prikaz usrednjenih normala. Broj normala u oba slučaja je isti samo se kod usrednjenih normala one nalaze na istom mestu pa daju privid toga da ih je manje.



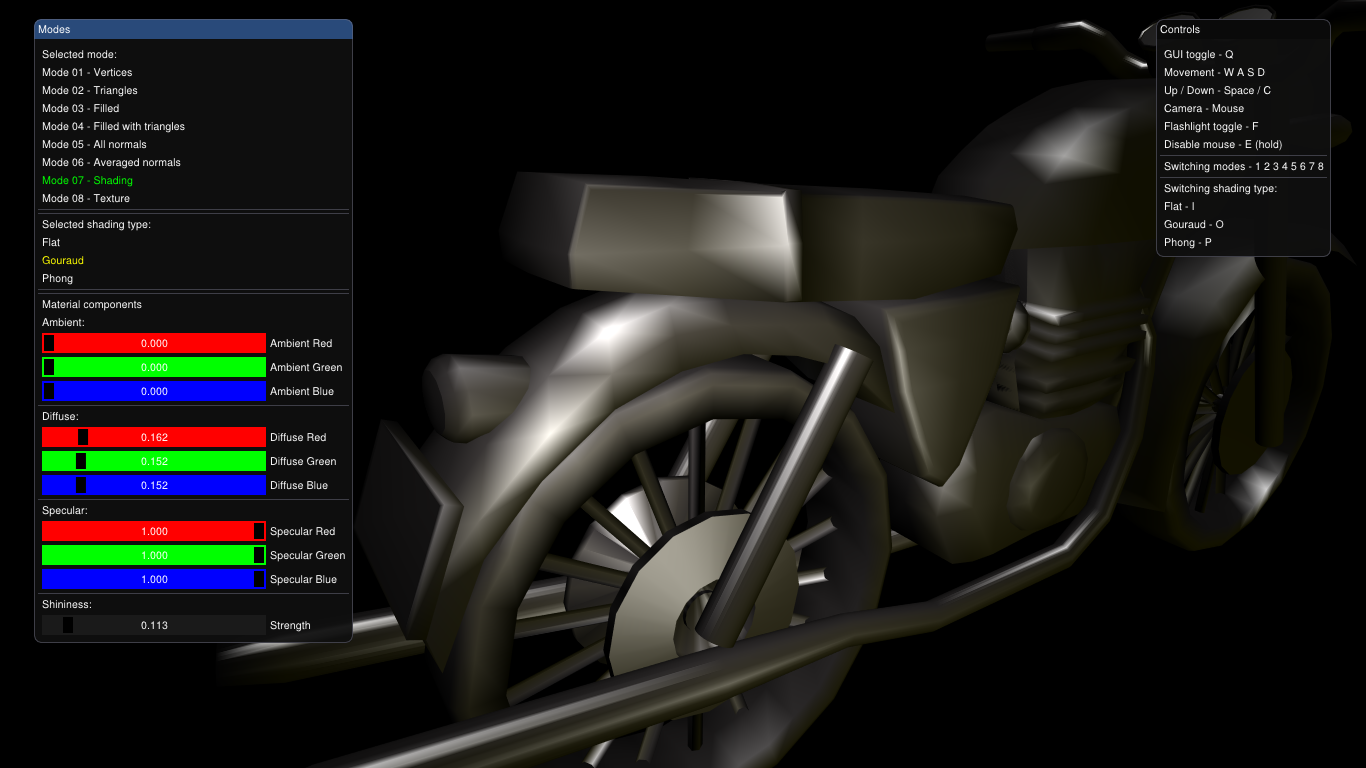
Slika 24 Poredjenje normala

## **4.6 Poređenje različitih tehnika senčenja**

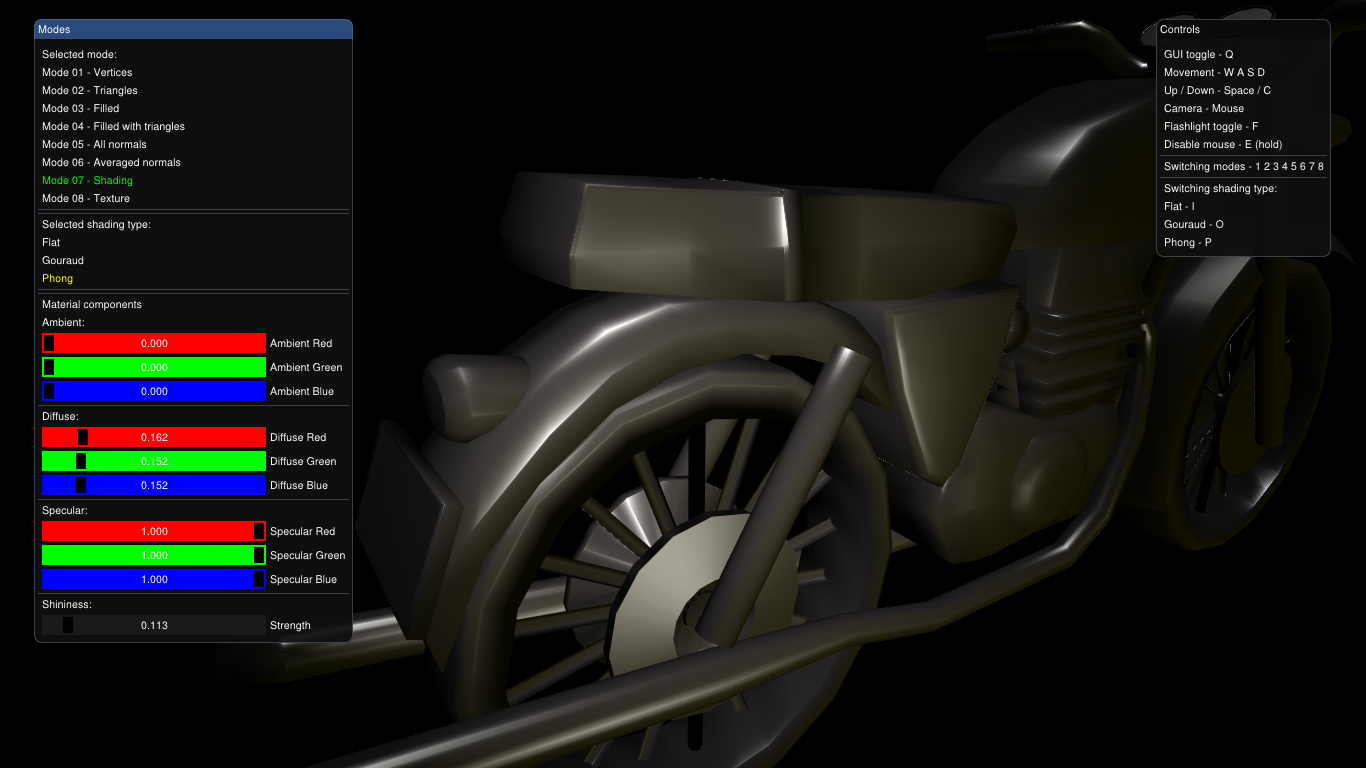
Najveća razlika imeđu tehnika senčenja se može uočiti prilikom smanjivanja parametara za sjajnost, korišćenjem baterijske lampe i približavanjem modelu. Prelaskom na svaki iduću tehniku senčenja model izgleda realnije. Na slici mjau prikazano je konstanto senčenje, na slici maju prikazano je Guroovo senčenje, na slici mjau prikazano je Fongovo senčenje.



Slika 25 Spekularna refleksija prilikom konstatnog senčenja



Slika 26 Spekularna refleksija prilikom Guroovog senčenja



Slika 27 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 učitan 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 učitanom 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

# 7. Literatura

1. 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
2. Mark J. Kilgard ,”OpenGL/VRML Materials”, [http://devernay.free.fr/cours/opengl/materials.html](http://devernay.free.fr/cours/opengl/materials.html%20) , stranica za parametre svetlosnih komponenti različitih materijaja
3. <https://sr.wikipedia.org/sr-el/OpenGL>
4. <https://github.com/ocornut/imgui>
5. <https://github.com/assimp/assimp>
6. <https://www.mql5.com/en/articles/7708>
7. <https://learn.foundry.com/modo/content/help/pages/uving/set_vertex_normals.html>
8. <https://www.cs.otago.ac.nz/cosc342/2017-notes/342-2017lect15.pdf>
9. <https://commons.wikimedia.org/wiki/File:Shading_models.png>

# 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. Završio 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.