VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

INTERAKTIVNÍ WEBOVÉ APLIKACE PRO PODPORU VÝUKY 3D POČÍTAČOVÉ GRAFIKY

INTERACTIVE WEB APPLICATIONS SUPPORTING EDUCATION OF 3D GRAPHICS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Jaroslav Priščák

AUTHOR

VEDOUCÍ PRÁCE — prof. Mgr. Pavel Rajmic, Ph.D.

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Jaroslav Priščák ***ID:*** 195420

Ročník: 2 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Interaktivní webové aplikace pro podporu výuky 3D počítačové grafiky

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte teorii k níže konkretizovaným aplikacím, graficky a uživatelsky je navrhněte, implementujte a otestujte. Kombinací HTML a JavaScriptu vytvořte tři webové aplikace. Pro 3D modelování je vhodné využít knihoven jako jsou WebGL, three nebo P5.

Aplikace se tématicky týkají generování trojrozměrné počítačové grafiky, konkrétně budou zaměřeny na:

1) Ilustraci tečné roviny a normály k libovolnému bodu na třírozměrného povrchu,

2) Ilustraci vlivu parametrů na vzhled odrazu světla v Phongově osvětlovacím modelu,

3) Ilustraci rozdílu mezi stínováním kostantním, Gouraudovým a Phongovým.

Zaměřte se především na názornou podobu, interaktivitu a funkčnost pro potřebu výuky. Každou aplikaci vložte do HTML stránky, která bude obsahovat i stručný souhrn teorie.

DOPORUČENÁ LITERATURA:

[1] Beneš, B.; Sochor, J.; Felkel, P.; Žára, J.: Moderní počítačová grafika. Computer Press, Brno, 2005.

[2] Piegl, L.,; Tiller, W.: The NURBS Book. Druhé vydání. Springer, 1997

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 19.5.2023

Vedoucí práce: prof. Mgr. Pavel Rajmic, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Používanie interaktívnych webových aplikácií má potenciál výrazne zlepšiť výučbu 3D počítačovej grafiky. Využitím pohodlia a dostupnosti webu môžu tieto aplikácie poskytnúť študentom pútavejšiu a interaktívnejšiu výučbu. Táto práca sa zaoberá vývojom takýchto aplikácií s cieľom pomôcť pochopiť základy nutné pre výpočty, ako sú normálové vektory a dotykové roviny. Ďalej práca popisuje Phongov osvetľovací model, parametre prostredia a materiálu, ktoré vplývajú na výsledný svetelný odraz. Následne popisuje jednotlivé modely tieňovania ako konštantné, Gouraudovo a Phongovo, kde popisuje jednotlivé výhody a nevýhody, rozdiely náročnosti výpočtov pri určovaní farby fragmentu a jednotlivých pixelov. Zároveň vysvetľuje kód, na ktorom sú tieto aplety vytvorené.

KĽÚČOVÉ SLOVÁ

3D počítačová grafika, Phongov model, Gouraudov model, konštantné tieňovanie, tieňovanie, osvetlenie, Three.js, aplet, webová aplikácia

ABSTRACT

The use of interactive web applications has the potential to significantly improve the teaching of 3D computer graphics. By taking advantage of the convenience and accessibility of the web, these applications can provide students with more engaging and interactive learning. This thesis explores the development of such applications to help understand the fundamentals required for calculations such as normal vectors and tangent planes. Further, the thesis describes Phong's illumination model, the environmental and material parameters that affect the resulting light reflection. It then describes the different shading models such as flat, Gouraud and Phong shading models, describing the various advantages and disadvantages, the differences in computational complexity in determining the color of a fragment and individual pixels. It also explains the code on which these applets are built.

KEYWORDS

3D computer graphics, Phong model, Gouraud model, flat shading, shading, lighting, Three.js, applet, web application

Vysadené pomocou balíčku `thesis` verzie 4.07; `http://latex.feec.vutbr.cz`

PRIŠČÁK, Jaroslav. *Interaktivní webové aplikace pro podporu výuky 3D počítačové grafiky*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 77 s. Diplomová práca. Vedúci práce: prof. Mgr. Pavel Rajmic, Ph.D.

Vyhlásenie autora o pôvodnosti diela

Meno a priezvisko autora:	Bc. Jaroslav Prišč̆ák
VUT ID autora:	195420
Typ práce:	Diplomová práca
Akademický rok:	2022/2023
Téma závěrečnej práce:	Interaktivní webové aplikace pro podporu výuky 3D počítačové grafiky

Vyhlasujem, že svoju záverečnú prácu som vypracoval samostatne pod vedením vedúcej/cého záverečnej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej záverečnej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podpisuje iba v tlačenej verzii.

POĎAKOVANIE

Chcel by som poďakovať svojmu vedúcemu diplomovej práce prof. Mgr. Pavlovi Rajmicovi Ph.D., za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Obsah

Zoznam obrázkov

Úvod

Oblasť 3D počítačovej grafiky má dlhú a zaujímavú históriu, ktorá siaha až do počiatkov výpočtovej techniky. Prvé možnosti 3D grafiky boli obmedzené a používali sa predovšetkým na vedeckú a technickú vizualizáciu. S pokrokom v oblasti počítačového hardvéru a softvéru sa však 3D grafika stala čoraz sofistikovanejšou a v dnešnej dobe sa používa na širokú škálu účelov.

V súčasnosti sa 3D grafika vytvára pomocou rôznych špecializovaných softvérových nástrojov vrátane programov na 3D modelovanie, animáciu a vykresľovanie. Tieto nástroje umožňujú umelcom a dizajnérom vytvárať detailné a komplexné 3D modely, animovať ich v čase a renderovať ich do vysokokvalitných obrázkov a videí. Proces tvorby 3D grafiky zahŕňa kombináciu umeleckých zručností, technických znalostí a schopností riešiť problémy.

Oblasť 3D počítačovej grafiky sa naďalej vyvíja a napreduje rýchlym tempom, pričom sa neustále vyvíjajú nové technológie a techniky. Počítačová 3D grafika je neoddeliteľnou súčasťou modernej zábavy, od pokročilých vizuálnych efektov moderných filmov až po fotorealistické spracovanie videohier.

Známa je aj pod názvom CGI (z angl. *Computer-Generated Imagery*), filmoví tvorcovia môžu vďaka nej teraz oživiť svoje fantastické svety, vytvoriť realistických ľudí, príšery a zlepšiť vizuálny zážitok diváka vďaka počítačom generovaným obrazom. CGI sa stalo nevyhnutnou súčasťou filmového priemyslu vďaka technologickému pokroku a softvérovým nástrojom, ktoré umožňujú filmárom a producentom vytvárať progresívnu grafiku, prepracované špeciálne efekty a pohlcujúce virtuálne prostredie.

Diplomová práca je zameraná na 3D počítačovú grafiku, možnostiam pre jej implementáciu a vytvoreniu samotných interaktívnych apletov pre podporu výučby. Má za úlohu vytvorenie apletov pre zlepšenie výučby 3D grafiky, s motiváciou uľahčenia, názornej ilustrácie a pochopenia témy z oblasti priestorovej grafiky poskytnutím interaktívnych modelov umožňujúcich nastavenie jednotlivých parametrov a sledovanie ich vplyvu na vykreslenie.

V teoretickej časti bude táto diplomová práca zameraná na teoretický rozbor základov 2D grafiky a postupne nadviaže na teóriu týkajúcu sa 3D grafiky, kde budú popísané plochy, napríklad NURBS (z angl. *Non-Uniform Rational B-Spline*), ktorá bude následne aj využitá v apletoch ako vlnitý povrch. Ďalej budú popísané 3D operácie s objektom a ich matematický popis. Následne možnosti práce s 3D grafikou a ich jednotlivé implementácie prostredníctvom grafických knižníc.

V praktickej časti budú využité a aplikované znalosti z teoretickej časti na implementáciu a vývoj riešenia samotných edukatívnych apletov. Bude popísaná základná štruktúra projektov, popis prípravy prostredia pre prípad úprav, alebo nadviazania

na túto diplomovú prácu, ďalej budú popísané jednotlivé dôležité programové časti a následne aj popis použitia apletov a ich možnosti. Prvý aplet bude slúžiť na ukážku normálového vektoru a dotykovej plochy k akémukoľvek bodu na 3D povrchu. Druhý aplet bude znázorňovať Phongov osvetľovací model a vplyvy rôznych parametrov ako vlastnosti materiálu a nastavením okolitého prostredia na výsledný jav po vykreslení. Tretí aplet bude znázorňovať rozdiely jednotlivých tieňovaní a vlastností materiálu na výsledné vykreslenie. To bude ilustrovať výhody a nevýhody jednotlivých tieňovaní.

1 Základy 3D grafiky

1.1 2D krivky

Krivky a plochy sú v počítačovej grafike a súvisiacich aplikáciách využívané na mnohých miestach. Napríklad sa s nimi stretávame pri modelovaní v dvojrozmernom a trojrozmernom priestore, pri definícii fontov, pri určovaní dráhy pohybujúcich sa objektov v počítačovej animácii a mnoho ďalších [1].

Výrazný pokrok v tejto oblasti a zjednotenie už skôr používaných rôznorodých prístupov prinieslo používanie racionálnych B-spline kriviek a plôch s neuniformnou parametrizáciou, NURBS. Tieto metódy umožňujú generovať klasické geometrické prvky za pomoci rovnakých metód, ako ktoré umožňujú vytvoriť krivky a plochy so zložitými priebehmi a tvarmi [1].

Existujú tri hlavné spôsoby matematického určenia kriviek [2]:

- **Implicitné reprezentácie** kriviek definujú množinu bodov na krivke pomocou postupu, ktorý môže testovať, či je bod na krivke. Zvyčajne je implicitná reprezentácia krivky definovaná implicitnou funkciou v tvare:
  $f(x, y) = 0$, takže krivka je množina bodov, pre ktoré táto rovnica platí. Takáto implicitná funkcia f je skalárna funkcia, čo znamená, že vracia jedno reálne číslo.
- **Parametrické reprezentácie** kriviek poskytujú mapovanie z voľného parametra na množinu bodov na krivke. To znamená, že tento voľný parameter poskytuje index k bodom na krivke. Parametrická forma krivky je funkcia, ktorá priraďuje pozície hodnotám voľného parametra. Intuitívne si je možné predstaviť krivku, ako niečo, čo je možné nakresliť perom na papier, voľný parameter je čas, ktorý sa pohybuje v intervale od času, kedy sme začali kresliť krivku, do času, kedy sme ju dokončili. Parametrická funkcia tejto krivky vyjadruje, kde sa pero nachádza v ktoromkoľvek časovom okamihu:
  $(x, y) = f(t)$.
  Dôležité je, že parametrická funkcia je vektorová funkcia. Tento príklad je 2D krivka, takže výstupom funkcie je 2-vektor, pri 3D by to bol 3-vektor.
- Generatívne alebo procedurálne reprezentácie kriviek poskytujú postupy, ktoré môžu generovať body na krivke, ktoré nepatria do prvých dvoch kategórií. Medzi príklady generatívnych opisov kriviek patria schémy delenia a fraktály.

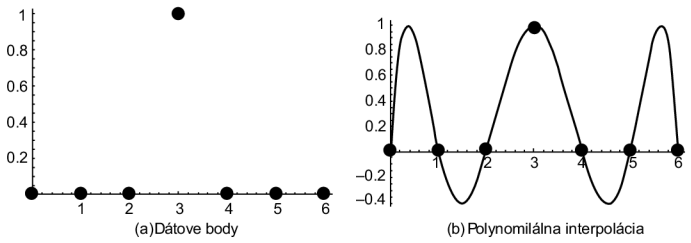
Krivka je množina bodov, vyššie spomenuté reprezentácie poskytujú spôsoby, ako tieto množiny špecifikovať. Každá krivka má mnoho možných reprezentácií. Kvôli tomu matematici zvyčajne starostlivo rozlišujú medzi krivkou a jej reprezentáciou. Z tohto pohľadu sa v počítačovej grafike zvyčajne odvoláva len na reprezentáciu, nie na samotnú krivku. Takže keď sa povie „implicitná krivka“, má sa na mysli to

buď krivka, ktorá je reprezentovaná nejakou implicitnou funkciou, alebo implicitná funkcia, ktorá je jednou z reprezentácií nejakej krivky. Takéto rozlišovanie zvyčajne nie je dôležité, pokiaľ nie je nepotrebné uvažovať o rôznych reprezentáciách tej istej krivky.

1.1.1 Interpolácia a aproximácia

Vedci a inžinieri používajú freeform krivky a povrchy na interpoláciu údajov a aproximáciu tvaru. Interpolácia a aproximácia však nie sú vždy kompatibilné operácie. Ak by boli údaje na obrázku 1.1a, tak na interpoláciu týchto údajov použijeme polynóm nízkeho stupňa, ako na obrázku 1.1b, tak polynóm osciluje okolo osi x, hoci v údajoch takéto oscilácie nie sú. Teda tvar interpolačného polynómu neodráža tvar údajov. Okrem toho aj poskytnutie ďalších a ďalších dátových bodov na požadovanej krivke nemusí odstrániť tieto nežiaduce oscilácie [3].

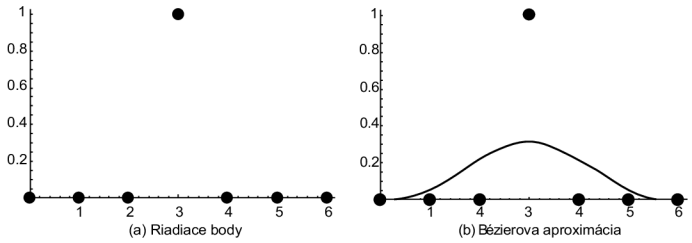
Cieľom aproximácie je zachytiť tvar požadovanej krivky alebo povrchu z niekoľkých dátových bodov bez nutnosti interpolácie bodov. Na rozdiel od interpolácie nie sú pri aproximácii samotné riadiace body sväté. Riadiace body sú skôr kontrolné body. Tieto body kontrolujú tvar krivky, alebo povrchu a možno ich upraviť tak, aby poskytovali lepšiu reprezentáciu požadovaného tvaru. Krivka na obrázku 1.1b aproximuje tvar údajov na obrázku 1.1a, aj keď krivka neprechádza všetkými dátovými bodmi. Krivka na obrázku 1.2b sa nazýva Bézierova krivka. Bézierove krivky majú mnoho praktických aplikácií, od návrhu nových písiem až po vytváranie mechanických súčiastok a zostáv pre veľké priemyselné dizajny a výroby [3].



Obr. 1.1: Polynomiálna interpolácia, prevzaté z [3].

Rozdiel je možné vidieť v interpolačnej polynomickej krivke sú oscilácie, hoci v pôvodných dátových bodoch oscilácie nie sú.

Bézierova krivka aproximuje tvar opísaný riadiacimi bodmi, ale Bézierova krivka neinterpoluje všetky riadiace body. Výšku krivky možno upraviť zmenou výšky stredného riadiaceho bodu. Porovnanie je možné s obrázkom 1.2.



Obr. 1.2: Bézierova aproximácia, prevzaté z [3].

1.1.2 Aproximácia kriviek

Môže sa javiť, že najjednoduchším spôsobom riadenia krivky je definovanie sady bodov, ktoré sa majú interpolovať. V praxi však majú interpolačné schémy často nežiaduce vlastnosti, pretože majú menšiu spojitosť a neponúkajú žiadnu kontrolu nad tým, čo sa deje medzi bodmi. Často sa uprednostňujú schémy kriviek, ktoré body iba aproximujú. Pri aproximačnej schéme riadiace body ovplyvňujú tvar krivky, ale presne ho neurčujú. Hoci sa vzdávame možnosti priamo určiť body, ktorými má krivka prechádzať, získavame lepšie správanie krivky a lokálnu kontrolu. Ak je nutné interpolovať sadu bodov, polohy riadiacich bodov možno vypočítať tak, aby krivka prechádzala týmito interpolačnými bodmi.

Dve najdôležitejšie typy aproximačných kriviek v počítačovej grafike sú Bézierove krivky a B-spline [2].

1.1.3 Bézierove krivky

Patria medzi populárne aproximačné krivky používané pre modelovanie v dvoch rozmeroch, ale aj pre definíciu trojrozmerných objektov. Bézierové krivky sa často používajú aj pri definícii písma (font) [1].

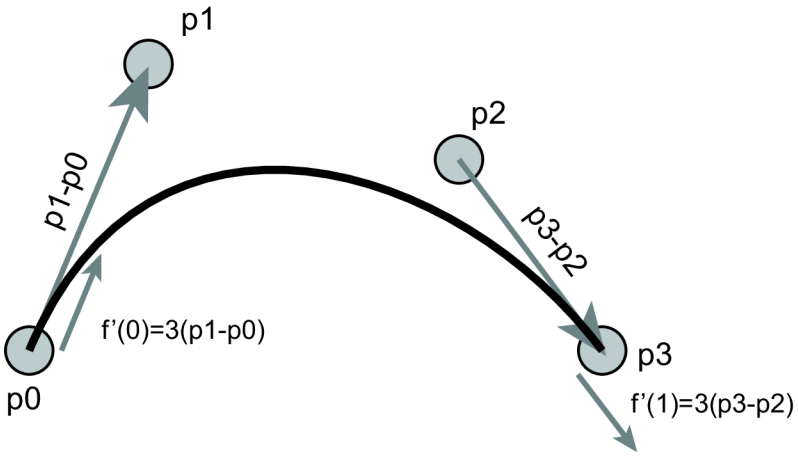
Bézierova krivka je polynomická krivka, ktorá aproximuje svoje riadiace body. Krivky môžu byť polynómy ľubovoľného stupňa. Krivka stupňa d je riadená $d + 1$ riadiacimi bodmi. Krivka interpoluje svoj prvý a posledný riadiaci bod, kde na tvar krivky majú priamy vplyv ostatné body.

Často sa zložité tvary vytvárajú spojením viacerých Bézierových kriviek nízkeho stupňa a v počítačovej grafike sa na tento účel bežne používajú kubické ($d = 3$) Bézierove krivky. Mnohé populárne ilustračné programy, ako napríklad Adobe Illustrator, a schémy reprezentácie písma, napríklad schémy používané v programe Postscript, používajú kubické Bézierove krivky. Bézierove krivky sú v počítačovej grafike veľmi populárne, pretože sa dajú ľahko ovládať, majú množstvo užitočných vlastností a existujú veľmi efektívne algoritmy na prácu s nimi [2].

Bézierove krivky sú konštruované tak, že [2]:

- Krivka interpoluje prvý a posledný riadiaci bod s bodmi označovanými ako $u = 0$ a 1, v uvedenom poradí, kde $u = 0$ reprezentuje počiatočný bod krivky a $u = 1$ koncový.
- Prvá derivácia krivky na jej začiatku (konci) je určená vektorom medzi prvým a druhým (predposledným a posledným) riadiacim bodom. Derivácie sú dané vektormi medzi týmito bodmi škálovanými podľa stupňa krivky.
- Vyššie derivácie na začiatku (konci) krivky závisia od bodov na začiatku (konci) krivky. n-tá derivácia závisí od prvých (posledných) $n + 1$ bodov.

Napríklad Bézierova krivka tretieho stupňa podľa obrázku 1.3. Krivka má štyri $d + 1$ riadiace body. Začína v prvom riadiacom bode $p0$ a končí v poslednom bode $p1$. Prvá derivácia na začiatku je úmerná vektoru medzi prvým a druhým riadiacim bodom $(p1 - p0)$. Konkrétne, $f'(0) = 3(p1 - p0)$. Podobne prvá derivácia na konci krivky je daná vzťahom $f(1) = 3(p3 - p2)$. Druhá derivácia na začiatku krivky môže byť určená z riadiacích bodov $p0, p1, p2$.



Obr. 1.3: Bézierova krivka, prevzaté z [2].

1.2 3D grafika a plochy

1.2.1 Bézierove plochy

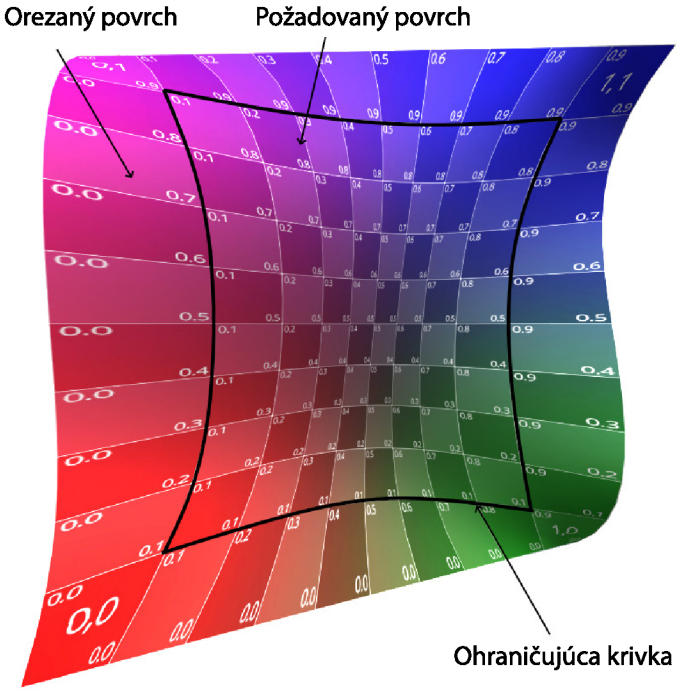
Niektoré modelovacie systémy používajú na reprezentáciu povrchov telies Bézierove plochy. Tieto plochy sú ľahko diferencovateľné, jednoduché a intuitívne na modelovanie a relatívne ľahké na výpočet priesečníka s lúčom. V súčasných systémoch sa na uchovanie plôch používajú zložitejšie reprezentácie založené na technológii NURBS. Tradičný užívateľský pohľad a možnosť pracovať s Bézierovými modelovacími nástrojmi sú však naďalej bežnou súčasťou týchto systémov. Je to možné

predovšetkým preto, že Bézierove plochy sú špeciálnym prípadom plôch NURBS. [1]. Bézierova plocha $n \times m$-tého stupňa je určená $(n+1) \times (m+1)$ riadiacimi bodmi P_i, j a vzťahom [1]:

$$Q(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{i,j} B_i^n(u) B_j^m(v)$$

1.2.2 NURBS

NURBS (z angl. *Non-Uniform Rational B-Splines*) sa stali de facto priemyselným štandardom pre reprezentáciu, návrh a výmenu údajov geometrických informácií spracovávaných počítačmi viď 1.4 [5]. Mnohé národné a medzinárodné normy, napr.



Obr. 1.4: Príklad NURBS plochy na obrázku, prevzaté z [6].

IGES, STEP a PHIGS, uznávajú NURBS ako výkonné nástroje na geometrické navrhovanie. Obrovský úspech NURBS je do veľkej miery spôsobený tým, že [5]:

- NURBS poskytujú jednotný matematický základ na reprezentáciu analytických tvarov, ako sú kužeľové rezy a štvorcové plochy, tak aj na voľné útvary, ako sú karosérie automobilov a trupy lodí.
- Navrhovanie pomocou NURBS je intuitívne; takmer každý nástroj a algoritmus má ľahko pochopiteľnú geometrickú interpretáciu.
- Algoritmy NURBS sú rýchle a numericky stabilné.
- Krivky a povrchy NURBS sú invariantné pri bežných geometrických transformáciách, ako sú translácia, rotácia, rovnobežné a perspektívne projekcie.

- NURBS sú zovšeobecnením neracionálnych B-spline a racionálnych a neracionálnych Bézierových kriviek a plôch.

NURBS plocha je matematicky popísaná ako [1]:

$$Q(\vec{u},\vec{v}) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j}P_{i,j}N_{i,p}(u)N_{j,q}(v)}{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j}N_{i,p}(u)N_{j,q}(v)}$$

,kde W_i, j sú váhy bodov (homogénne súradnice) P_i, j riadiace siete P, n a m je počet riadiacich bodov, p a q sú stupne polynómov a nakoniec $N_{i,p}(u)$, $N_{j,q}(v)$ sú normalizované B-spline bázové funkcie určené rekurentným vzťahom. NURBS plocha je ďalej určená dvoma uzlovými vektormi, vektorom $\vec{u}$ dĺžky $n+p+1$ a $\vec{v}$ dĺžky $m+q+1$, kde n, resp. m je počet riadiacich bodov v smere $\vec{u}$, resp. $\vec{v}$ a p, resp. q je stupeň plochy v smere $\vec{u}$, resp. $\vec{v}$. Uzlové vektory ovplyvňujú priebehy jednotlivých bázických funkcií a interval ich vplyvu [1].

1.2.3 Normálový vektor

Normálový vektor (k rovine) je vektor, ktorého smer je na danú rovinu kolmý. Normálový vektor je kolmý na akýkoľvek vektor, ktorého smer leží v rovine. Popisuje ho rovnica 1.1 [4].

$$\vec{n_u'}(u_0,v_0) = \frac{\vec{q_u'}(u_0,v_0)\times\vec{q_v'}(u_0,v_0)}{|\vec{q_u'}(u_0,v_0)\times\vec{q_v'}(u_0,v_0)|} \tag{1.1}$$

Používa sa predovšetkým na výpočty tieňovania, najmä v modeloch osvetlenia, ako je napríklad Phongov model osvetlenia 4.3.

1.2.4 Dotyková rovina

Dotyková rovina v ľubovoľnom bode P na povrchu, reprezentuje lokálnu orientáciu plochy v danom bode P. Dotyková rovina však neexistuje napríklad v bode na hrane kocky, alebo vo vrchole (dvojitého) kužeľa [4]. V 3D grafike je potrebné na realistickú simulácia správania sa svetla, pochopiť tvar, resp zakrivenie objektu a jeho orientáciu relatívnu k zdroju svetla. Bez koncepcie dotykovej roviny by bolo oveľa ťažšie presne simulovať interakciu svetla so zložitými povrchmi [10].

1.2.5 Siete vrcholov

Väčšina grafického hardvéru pracuje so špecifickými sadami geometrických primitív, pretože využívajú ich nižšiu náročnosť na spracovanie. Jednoduchšie primitívy sa dajú spracovať veľmi rýchlo. Výhradou je, že typy primitív musia byť všeobecné,

aby mohli modelovať širokú škálu geometrie od veľmi jednoduchej až po veľmi komplexné. Na typickom grafickom hardvéri sú typy primitív obmedzené na jednu alebo viacero z nasledujúcich možností [2]:

- **Body**: jednotlivé vrcholy (vertexy) používané na reprezentáciu bodov alebo časticových systémov.
- **Čiary**: dvojice vrcholov používané na znázornenie čiar, siluet alebo zvýraznenia okrajov.
- **Polygóny**: napríklad trojuholníky, trojuholníkové pásy, indexované trojuholníky, indexované trojuholníkové pásy, štvoruholníky, všeobecné konvexné mnohouholníky apod., ktoré sa používajú na opis trojuholníkových sietí, geometrických povrchov a iných pevných objektov, ako napr. gule, kužele, kocky alebo valce.

Pri odosielaní údajov do grafického hardvéru, ich posiela CPU (z angl. *Central Processing Unit*) cez jednu zo zberníc ako napríklad PCI (z angl. *Peripheral Component Interconnect*), AGP (z angl. *Accelerated Graphics Port*) alebo PCIe (z angl. *Peripheral Component Interconnect Express*) a nakoniec sa uložia do pamäte grafického hardvéru. V prípade veľmi veľkých trojuholníkových sietí reprezentujúcich komplexnú geometriu, môže mať prenášanie všetkých týchto údajov cez zbernicu za následok veľký zásah do výkonu, obzvlášť v prípadoch kedy sa jedná o vykresľovanie v okamžitom móde. Za účelom zníženia celkového prenosového pásma potrebného k prenosu geometrie cez grafickú zbernicu sú využívané rôzne spôsoby organizácie týchto geometrických sietí, napríklad [2]:

- **Trojuholníky**: sú zadané troma vrcholmi. Takto vytvorená trojuholníková sieť vyžaduje, aby bol každý trojuholník v sieti definovaný samostatne s mnohými potenciálne duplikovanými vrcholmi. Pre trojuholníkovú sieť obsahujúcu m trojuholníkov sa do grafického hardvéru odošle $3m$ vrcholov.
- **Pásy trojuholníkov**: trojuholníky sú usporiadané do pásov. Prvé tri vrcholy určujú prvý trojuholník v páse a každý ďalší vrchol pridáva trojuholník. Vytvorená trojuholníková sieť s m trojuholníkmi usporiadanými ako jeden trojuholníkový pás, pošle grafickému hardvéru tri vrcholy pre prvý trojuholník a potom jeden vrchol pre každý ďalší trojuholník v páse, spolu teda $m + 2$ vrcholy.
- **Indexované trojuholníky**: Namiesto toho, aby bol každý trojuholník uložený ako samostatná sada vrcholov, je uložené pole všetkých jedinečných vrcholov a potom je každý trojuholník definovaný odkazom na indexy troch vrcholov z tohto poľa. Takto sa spoločné vrcholy ukladajú len raz, čím sa výrazne znižuje množstvo údajov, ktoré je nutné uložiť.
- **Indexované trojuholníkové pásy**: podobne ako pri indexovaných trojuholníkoch sú vrcholy uložené v poli vrcholov. Trojuholníky sú však usporiadané

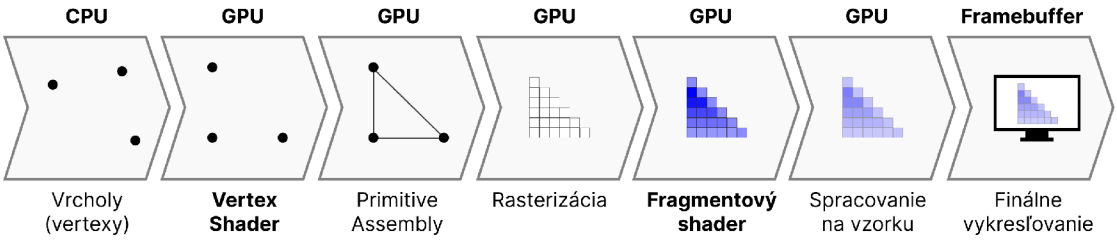
do pásov, pričom indexové pole definuje rozloženie pásov. Toto je najkompaktnejšia z organizačných štruktúr na definovanie trojuholníkových sietí, pretože kombinuje výhody pásov trojuholníkov s kompaktnosťou polí vrcholov.

1.2.6 STL

STL (z angl. *Stereolitografia*) je natívnym formátom súborov stereolitografického CAD (z angl. *Computer-Aided Design*) softvéru vytvoreného spoločnosťou 3D Systems. Stereolitografická technológia spoločnosti 3D Systems vytvorila éru digitálnych technológií na rýchlu výrobu modelov, prototypov a vzorov. Súbor STL aproximuje geometriu v súbore CAD, používa jednoduchú trojuholníkovú sieť na aproximáciu ohraničujúceho povrchu objektu a snaží sa presne aproximovať údaje s čo najmenším počtom prvkov. Súbor STL sa exportuje buď v binárnom formáte, alebo vo formáte ASCII (z angl. *American Standard Code for Information Interchange*) [7].

1.3 Grafický reťazec

Grafický reťazec, v angličtine známy ako rendering pipeline, opisuje kroky, ktoré sú potrebné, aby sa 3D objekt vykreslil na 2D obrazovku ako ukazuje obrázok 1.5. Pre účely tejto práce je najdôležitejší vertex shader a fragment shader, čo je popis operácií ktoré má grafická karta vykonať, zvyšné kroky sa vykonajú automaticky. Popis krokov grafického reťazca [10]:



Obr. 1.5: Kroky grafického reťazca, prevzaté z [20].

1. **Vrcholy (vertexy)**: sú základnými stavebnými prvkami 3D modelovania. Každý vrchol má niekoľko atribútov, napríklad polohu, farbu, normálu a súradnice textúry. V OpenGL sú tieto vrcholy zvyčajne uložené v objektoch VBO (z angl. *Vertex Buffer Objects*).
2. **Vertex Shader**: je programovateľná fáza shadera v renderovacej pipeline OpenGL. Spracúva každý vrchol samostatne. Tu sa aplikujú transformácie, ako napríklad MVP (z angl. *Model View Projection*), na prevod vrcholov z 3D do 2D priestoru obrazovky.

3. **Primitive Assembly**: po vertex shaderi zostaví OpenGL vrcholy do geometrických primitív (bodov, čiar alebo trojuholníkov) na základe vydaného príkazu na vykresľovanie.
4. **Rasterizácia**: je proces, ktorý vytvára virtuálny obraz scény. Má dva prvky: stratégiu prechádzania pixelov na virtuálnej obrazovke a metódu výpočtu hodnoty farby v každom pixely (t. j. vykonanie vzorkovania funkcie tieňovania) [9]. Primitíva sa potom rasterizujú do fragmentov. Rasterizácia je proces prevodu primitív na súbor fragmentov. Fragmenty sú v podstate pixely, ktoré primitíva pokrýva na obrazovke. V OpenGL môžete proces rasterizácie ovládať pomocou rôznych príkazov, napríklad `glPolygonMode()`.
5. **Fragmentový shader**: ďalší programovateľný stupeň shadera v reťazci OpenGL, spracováva fragmenty z predchádzajúceho kroku tým, že vypočítava konečnú farbu každého fragmentu. Fragmentový shader je možné naprogramovať tak, aby zahŕňal efekty, ako textúrovanie, osvetlenie a tiene.
6. **Spracovanie na vzorku**: táto fáza zahŕňa niekoľko operácií vrátane testovania hĺbky, testovania šablón, prelínania a ďalších. Testovanie hĺbky (napríklad pomocou Z-buffera) sa používa na určenie, či sa fragment nachádza za alebo pred inými fragmentmi. Testovanie šablón sa môže použiť na obmedzenie miesta, kde sa majú vykresliť pixely. Prelínanie kombinuje farbu nového fragmentu s farbou pixela, ktorý sa už nachádza vo framebufferi, čo umožňuje vytvárať efekty, ako priehľadnosť.
7. **Finálne vykresľovanie**: fragmenty, ktoré prešli všetkými testami, sa potom zapíšu do framebuffera, čo je v podstate obraz, ktorý je zobrazený na obrazovke.

2 Geometrické transformácie a premietanie

2.1 Transformácie

Geometrické transformácie sú jednou z najčastejšie používaných operácií v počítačovej grafike. Transformácie možno rozdeliť na lineárne a nelineárne. Lineárne transformácie zahŕňajú rotáciu, posun, škálovanie, skosenie a operácie vyplývajúce z ich zloženia. S nelineárnymi transformáciami sa v počítačovej grafike stretávame pri zložitejších zmenách tvaru grafických objektov, napr. pri deformáciách priestorových modelov alebo pri deformácii obrazu. Špeciálnou transformáciou je projekcia, ktorá prevádza objekty z viacrozmerného priestoru do priestoru menších rozmerov. S projekciami sa najčastejšie je možné stretnúť pri konverzii trojrozmernej scény do roviny obrazu [1].

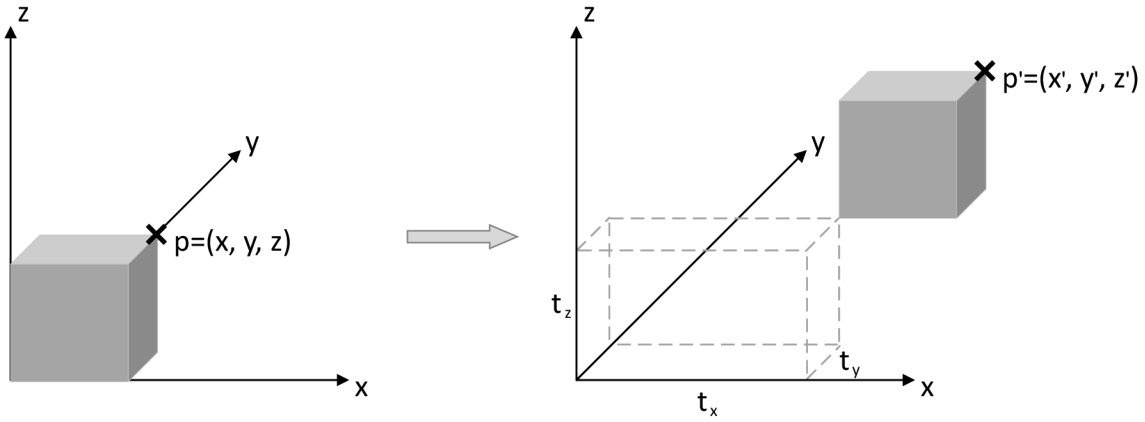
2.2 Transformácie v 3-priestore

Transformácie v 3-priestore sú v mnohých ohľadoch analogické transformáciám v 2-priestore. Projekčné transformácie sú podobné transformáciám v 2-priestore. Namiesto toho, aby boli neurčité na priamke, sú neurčité na celej rovine. V opačnom prípade sú úplne analogické. Transformácie mierky (škálovanie) môžu byť opäť rovnomerné alebo nerovnomerné. Tie, ktoré sú nerovnomerné, sa vyznačujú tromi ortogonálnymi invariantnými smermi a tromi faktormi mierky namiesto dvoch, inak sa výrazne nelíšia [10].

2.2.1 Translácia

Pri translácii sa všetky body, alebo vrcholy objektu posúvajú pozdĺž jednej osi (x, y, alebo z) [8]. Na transláciu homogénneho 3D bodu $p = [x, y, z, 1]^T$ do iného homogénneho 3D bodu $p' = [x', y', z', 1]^T$, použijeme 2.1, kde T je 4x4 matica reprezentujúca operáciu translácie pre homogénne body v 3D priestore, a kde je $[t_x, t_y, t_z]^T$ translačný vektor ako ukazuje obrázok 2.1.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.1}$$



Obr. 2.1: Translácia v 3D, prevzaté z [8].

2.2.2 Rotácia

Rotácie v 3D priestore je možné vykonať okolo ktorejkoľvek z troch osí. Štandardným spôsobom otáčania je použitie pravotočivého systému. Takýto systém, alebo pravidlo sa používa aj v matematike a fyzike, čiže ak ukáže palec pravej ruky v kladnom smere osi y a ukazovák v kladnom smere osi x, tak prostredník bude ukazovať v kladnom smere osi z. Takže rotácia by sa vykonala proti smeru hodinových ručičiek pri pohľade z kladného konca osi smerom k počiatku. Pokiaľ sa vykonáva viacero rotácií je dôležité poradie, v akom sa aplikujú, známe ako poradie otáčania, alebo poradie Eulerovho uhla [11]. Zároveň sa tieto operácie vykonávajú vzhľadom na počiatok súradnicového systému. Pokiaľ sa objekt otáča okolo iného bodu než je počiatok súradnicového systému, musí sa celá scéna posunúť tak, aby sa bod otáčania nachádzal v počiatku, vykoná sa otáčanie a potom scénu posunie naspäť. Rotácia okolo osi z o uhol θ [8]:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Analogicky rotácia okolo osi x o uhol θ [8]:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Tým bod $p = [x, y, z]^T$ bude mať nové koordináty označené ako $p' = [x', y', z']^T$.

2.2.3 Škálovanie

Objekt možno škálovať tak, aby sa proporcionálne zväčšil alebo zmenšil o faktor k. Ak je $s_x = s_y = s_z$, tak sa objekt zväčší v rovnakom pomere vo všetkých smeroch a takého škálovanie sa označuje ako rovnomerné, v opačnom prípade, sa škálovanie označuje ako nerovnomerné.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Kde matica 4×4, predstavuje operáciu škálovania pre homogénne body v 3D priestore a s_x, s_y, s_z sú faktory mierky v smeroch x, y, z.

2.3 Zobrazovanie priestorových dát

Zobrazovanie priestorových modelov a scén reprezentovaných v pamäti počítača je dôležitým prvkom počítačovej grafiky. Prevod trojrozmerných informácií do dvojrozmernej obrazovej podoby sa v anglickej literatúre nazýva rendering (vykresľovanie). Úlohu vykresľovania scény obsahujúcej telesá a zdroje svetla možno zjednodušene charakterizovať ako postupné riešenie nasledujúcich čiastkových úloh [1]:

- Globálne osvetlenie scény závisí najmä od zdrojov svetla a od optických vlastností telies a prostredia.
- Pohľadu na scénu z pozície pozorovateľa alebo nastavenia kamery a riešenia problému projekcie spolu s čiastočným alebo úplným riešením problému viditeľnosti.
- Vytvorenie rastrového obrazu vrátane riešenia viditeľnosti, lokálnych modelov osvetlenia a textúr.

2.4 Premietanie

2.4.1 Kamera

Základom úlohy premietania je formulácia geometrickej situácie, t. j. určenie miesta, kde stojí pozorovateľ, definovanie polohy a orientácie premietania a určenie smeru a cieľa pozorovania. Kamery snímajú obraz na projekciu, ktorá je kolmá na hlavnú optickú os kamery. Z hľadiska projektívnej geometrie je najvšeobecnejšia možná projekcia daná projektívnou transformáciou, ktorú v homogénnych súradniciach možno reprezentovať pomocou matíc [9]:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

Pomocou tejto transformácie máme 11 stupňov flexibility na definovanie virtuálnej kamery a je ju možné použiť na niekoľko typov kamier, napríklad perspektívnu kameru (ktorá používa stredovú projekciu), dierkovú, afinnú alebo ortografickú kameru [9].

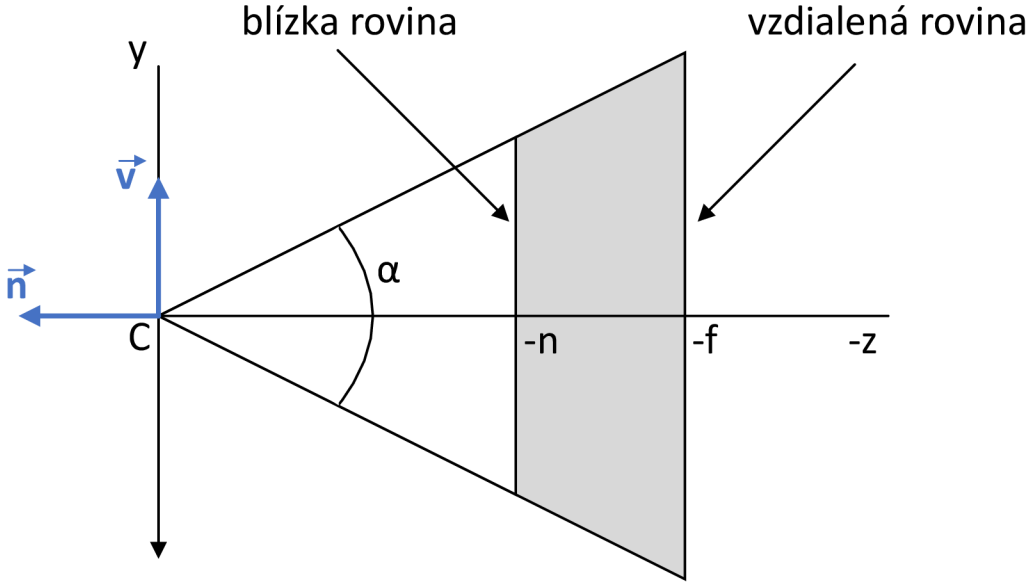
2.4.2 WebGL kamera

V grafickom systéme WebGL sa špecifikácia kamery získava zadaním stredu projekcie, pozorovacieho bodu (look-at point) a vektora sklonu V (tzv. up vector). Vo WebGL je však referenčný rámec kamery u, v, n kladný, to znamená, že vektor optickej osi vo WebGL ukazuje v smere opačnom, teda s rastúcou hĺbkou scény sa súradnica z zmenšuje [9]. WebGL používa pravotočivý súradnicový systém, v ktorom kladná os x smeruje doprava, kladná os y smeruje nahor a kladná os z smeruje von z obrazovky.

Three.js poskytuje dva hlavné typy kamier:

- Perspektívna kamera.
- Ortografická kamera.

Perspektívna kamera, ktorá sa bude v práci používať je aj tou najčastejšie používanou v 3D scénach ako ukazuje obrázok 2.2.



Obr. 2.2: Parametre kamery WebGL, prevzaté z [9].

Perspektívna kamera napodobňuje spôsob, akým ľudské oko vníma svet, pričom objekty sa zdajú byť tým menšie, čím sú vzdialenejšie od kamery. Definícia za pomoci týchto parametrov:

```
PerspectiveCamera(
fov : číslo,
aspect : číslo,
near : číslo,
far : číslo )
```

`PerspectiveCamera` v Three.js je definovaná pomocou zorného poľa α (v stupňoch), parametrom „fov“, pomeru strán, odpovedá širke v pixeloch na zaobrazovanej ploche voci výške, tým sa definuje parameter „aspect“. Blízka a vzdialená rovina udáva veľkosť orezania v blízkej rovine parametrom „near“ a na vzdialenej rovine parametrom „far“.

3 3D grafika pomocou JavaScript

V tejto kapitole budú zmienené a popísané webové technológie a niektoré z populárnych OpenGL/WebGL knižníc, ktoré je možné využiť pri spracovaní webových aplikácií týkajúcich sa trojrozmernej počítačovej grafiky.

3.1 Vybrané technológie

Táto časť popisuje technológie pre tvorbu webu a prácu z 3D grafikou pomocou knižníc. Následne použitím ich kombinácie, budú aplety nimi vytvorené.

3.1.1 HTML

HTML (z angl. *Hypertext Markup Language*) je jazyk určený pre zápis elementov webových stránok, ktorý prináša nezávislosť od zariadenia, čo znamená, že webové stránky možno vytvárať pre všetky typy platforiem, od počítačov až po smartfóny, bez nutnosti inštalovať doplnky do prehliadača alebo vyvíjať viacero verzií webových stránok pre mobilné zariadenia. Statické webové stránky v HTML majú príponu súboru .htm alebo častejšie .html [12].

3.1.2 CSS

CSS (z angl. *Cascading Style Sheets*), kaskádové štýly sa používajú na definovanie a prispôsobenie štýlov a rozloženia webových stránok. To znamená, že môžete vytvárať súbory štýlov na zmenu dizajnu, rozloženia a prispôsobenia rôznym veľkostiam obrazovky na rôznych zariadeniach od počítačov po smartfóny [12]. V dobe písania tejto diplomovej práce je najnovšou a aktuálnou verziou CSS3.

3.1.3 JavaScript

Prevažná väčšina webových stránok používa programovací jazyk JavaScript a všetky moderné webové prehliadače v počítačoch, tabletoch a telefónoch obsahujú interpretery JavaScriptu, takže JavaScript je najrozšírenejším programovacím jazykom v histórii. V poslednom desaťročí umožnil Node.js programovanie v jazyku JavaScript mimo webových prehliadačov a dramatický úspech Node znamená, že JavaScript je teraz najpoužívanejším programovacím jazykom aj medzi vývojármi softvéru. JavaScript je vysokoúrovňový, dynamický, interpretovaný programovací jazyk, ktorý je vhodný pre objektovo orientované a funkcionálne štýly programovania. Premenné jazyka JavaScript nie sú typované. Jeho syntax je voľne založená na jazyku Java,

ale inak tieto jazyky nie sú príbuzné. JavaScript odvodzuje svoje first-class funkcie z jazyka Scheme a svoju prototypovú dedičnosť z menej známeho jazyka Self [13].

3.1.4 TypeScript

TypeScript bol vytvorený interne v spoločnosti Microsoft začiatkom roka 2010 a potom uvoľnený a otvorený v roku 2012. Vedúcim jeho vývoja je Anders Hejlsberg, ktorý je známy aj tým, že viedol vývoj populárnych jazykov C# a Turbo Pascal. TypeScript sa často opisuje ako „nadmnožina jazyka JavaScript“ alebo „JavaScript s typmi“. TypeScript je charakteristický pre [14]:

- **Programovací jazyk**
  - Jazyk, ktorý obsahuje všetku existujúcu syntax jazyka JavaScript a novú syntax špecifickú pre TypeScript na definovanie a použitie typov.
- **Kontrola typov**
  - Zabezpečuje predovšetkým to, aby sa do funkcií odovzdávali údaje správneho typu ukladali do premenných. Je možné definovať aj návratové type, teda aby funkcie vracali premenné istého typu a podobne.
- **Kompilátor**
  - Pri TypeScripte sa nazýva aj „tsc“. Hlavnou úlohou je kontrola typov, či definované premenné pri behu programu obsahujú definovaný typ, teda včas zahlási prípadné problémy ako uložene typu text do typu, ktorý očakáva číslo, označí kód, ktorý sa ani nepoužíva, prípadne slučky, ktoré sú nekonečné apod. Zároveň sa stará o preloženie kódu TypeScript do JavaScriptového kódu.
- **Jazyková služba**
  - Program, ktorý používa kontrolu typov na to, aby predal editorom, ako je napríklad VisualStudio Code, ako poskytnúť vývojárom užitočné nástroje a analytiku nad kódom. Pomáha pri písaní kódu, tým že dokončuje mená, premenné, a ukazuje očakávané typy argumentov.

Tie možnosti pomáhajú včasne predísť čo najväčšiemu počtu chýb pri vývoji a uľahčil sa tým samotný vývoj a ladenie programov.

3.1.5 Node.js

Node.js používa koncept modulu ako základného prostriedku na štruktúrovanie kódu programu. Je to stavebný prvok na vytváranie aplikácií a opakovane použiteľných knižníc. V Node.js je jedným z najviac evanjelizovaných princípov navrhovanie malých modulov (a balíkov), a to nielen z hľadiska veľkosti hrubého kódu, ale predovšetkým, z hľadiska rozsahu. Pomocou svojich správcov modulov, z ktorých najpopulárnejšie sú NPM (z angl. *Node Package Manager*) a YARN (z angl. *Yet Another*

Resource Negotiator), tak pomáha Node.js riešiť problém závislostí tým, že zabezpečuje, aby dva (alebo viac) balíkov závislých od rôznych verzií toho istého balíka používali svoje vlastné inštalácie takéhoto balíka, čím sa zabráni konfliktom. Tento aspekt umožňuje balíkom závisieť na vysokom počte malých, dobre sústredených závislostí bez rizika vytvorenia konfliktov [15].

3.1.6 WebGL

WebGL je multiplatformový otvorený webový štandard bez licenčných poplatkov pre nízkoúrovňové 3D grafické API (z angl. *Application Programming Interface*) rozhranie založené na OpenGL ES, ktoré je sprístupnené v jazyku ECMAScript (JavaScriptový štandard) prostredníctvom prvku HTML5 Canvas. Vývojári, ktorí poznajú OpenGL ES 2.0, rozpoznajú WebGL ako API založené na shaderoch využívajúcich GLSL, s konštruktmi, ktoré sú sémanticky podobné základnému API OpenGL ES. Zostáva veľmi blízko špecifikácii OpenGL ES s určitými ústupkami, ktoré vývojári očakávajú od jazykov so správou pamäte, ako je JavaScript. WebGL 1.0 poskytuje súbor funkcií OpenGL ES 2.0, WebGL 2.0 poskytuje API OpenGL ES 3.0.

WebGL prináša do moderných webových prehliadačov integráciu 3D bez nutnosti externých doplnkov. Hlavní distribútori webových prehliadačov Apple (Safari), Google (Chrome), Microsoft (Edge) a Mozilla (Firefox) sú členmi pracovnej skupiny WebGL [16].

3.1.7 Three.js

Jednoznačne najpopulárnejšou knižnicou na vývoj grafických aplikácií WebGL je Three.js, bola využitá aj pre vývoj mnohých vlajkových demonštračných aplikácií WebGL. Poskytuje jednoduchú a intuitívnu sadu objektov, ktoré sa bežne používajú v 3D grafike. Je rýchla, výkonná a využíva mnohé osvedčené techniky grafického enginu, obsahuje niekoľko zabudovaných typov objektov a praktických nástrojov. Three.js obsahuje aj vykresľovací systém, ktorý umožňuje vykresľovanie 3D obsahu (s určitými obmedzeniami) do 2D Canvas API, SVG a CSS3 s 3D transformáciami.[17]. Three.js je kontinuálne udržiavanou knižnicou, v čase písania tejto diplomovej práce voľne dostupnou prostredníctvom repozitára na GitHube, pod licenciou MIT [18]. Aplety využívajú spoločné knižnice z Three.js:

```
import * as THREE from "three";
```

Táto implementuje univerzálnu knižnicu pre 3D s podporou rôznych prehliadačov a vykresľovaním za pomoci WebGL.

```
import {OrbitControls}
```

```
from "three/examples/jsm/controls/OrbitControls";
```

Implementuje ovládanie pohľadu kamery otáčať sa okolo predmetu.

```
import {STLLoader}
from "three/examples/jsm/loaders/STLLoader";
```

Umožňuje načítať .stl súbor, ktorý popisuje geometriu povrchu trojrozmerného objektu.

```
import {NURBSSurface}
from "three/examples/jsm/curves/NURBSSurface.js";
```

Toto má definície `NurbsCurve` a `NurbsSurface`. Stačí odovzdať vstupy a získať tieto triedy na generovanie bodov na krivke a povrchu.

```
import {ParametricGeometry} from
 "three/examples/jsm/geometries/ParametricGeometry.js";
```

Umožní vytvoriť invertovanú plochu NURBS, tým potom dokážeme modelovať obojstranné povrchy.

```
import * as BufferGeometryUtils
from "three/examples/jsm/utils/BufferGeometryUtils";
```

Umožňuje prepojiť a skombinovať povrchy. Zvyšné k úlohe špecifické knižnice, alebo šablóny materiálov budú popísané jednotlivo k danej úlohe.

3.1.8 Knižnica p5.js

Knižnica p5.js je ďalšou z JavaScriptových knižníc so zameraním na sprístupnenie kódovania pre umelcov, dizajnérov, pedagógov. Syntax jazyka p5.js je takmer identická s JavaScriptom, avšak knižnica pridáva vlastné funkcie týkajúce sa grafiky, interakcie a poskytuje jednoduchší prístup k natívnym funkciám HTML5, ktoré sú modernými webovými prehliadačmi už podporované. Medzi základné koncepty a vlastnosti knižnice patria [19]:

- **Základné oprerácie**: funkcie `setup()` a `draw()` knihovny p5.js sú jej dve základné operácie. Na začiatku programu sa funkcia `setup()` spustí raz, aby sa nastavili počiatočné konfigurácie, napríklad veľkosť plátna. Vizuálne prvky na plátne sa v cykle priebežne aktualizujú pomocou funkcie `draw()`.
- **Rendering**: knižnica podporuje 2D aj 3D vykresľovanie. V 2D poskytuje p5.js funkcie na vykresľovanie tvarov, farieb, obrázkov a textu. Na vykresľovanie v 3D je využívaný WebGL.
- **Užívateľské vstupy**: ponúka širokú škálu funkcií pre integráciu interakcií od používateľa vrátane vstupu z myši, klávesnice, dotykovej obrazovky a dokonca aj gamepadov.

- **Matematické funkcie**: knižnica umožňuje napríklad jednoducho generovať náhodné čísla, mapovať rozsahy a vykonávať ďalšie matematické operácie.
- **Manipulácia DOM**: okrem canvasu ponúka p5.js aj nástroje na vytváranie a modifikáciu HTML elementov.

4 Osvetľovacie modely a tieňovanie

4.1 Svetlo

Pre počítačovú grafiku sú fundamentálne algoritmy, ktoré simulujú interakciu svetla s povrchmi objektov a s opticky aktívnym prostredím. Svetlo je prostriedkom vizuálneho vnímania sveta a pochopenie jeho interakcie s materiálmi a prechádzania priestorom je základom pre vytváranie a zobrazovanie virtuálnych scén [1]. Svetlo má dualistickú povahu a správa sa ako vlny aj častice. Vlnový opis je vhodný na vysvetlenie javov, ako je rozptyl svetla, difrakcia, interferencia atď. Časticovým opisom sa dá na druhej strane vysvetliť odraz svetla, interakciu s drsným povrchom materiálu atď. Optika, je veda zaoberajúca sa štúdiom svetla, sa delí na tieto podoblasti [1]:

- **Geometrická optika** modeluje svetlo ako nezávislé lúče, ktoré sa pohybujú priestorom a ich trajektórie možno opísať geometrickými pravidlami.
- **Vlnová optika** modeluje elektromagnetické vlny a umožňuje opísať väčšinu javov, v ktorých geometrická optika nedokáže opísať, najmä difrakciu a interferenciu.
- **Elektromagnetická optika** zahŕňa vlnovú optiku a navyše opisuje polarizáciu svetla a disperziu na hranách.
- **Fotónová optika** (nazývaná aj kvantová optika) je základom pre vysvetlenie interakcie svetla s materiálom.

V počítačovej grafike sa takmer výlučne používa geometrická optika a v niektorých prípadoch opis svetla založený na časticiach. Tieto modely umožňujú opísať a modelovať prevažnú väčšinu javov, ktoré sú nevyhnutné pre vizuálne vnímanie sveta. Počítačová grafika sa nezaoberá javmi, ako napr. polarizácia alebo difrakcia svetla, hoci tieto rozšírenia sú možné a pomerne jednoduché [1].

4.1.1 Zdroje svetla

Plochy v scéne, ktoré vyžarujú svetlo, sa nazývajú zdroje svetla. Obzvlášť výrazné zdroje svetla sa bežne nazývajú primárne zdroje svetla. Objekty, ktoré nepriamo vyžarujú svetlo odrazom, sa nazývajú sekundárne zdroje svetla.

Primárne zdroje svetla v počítačovej grafike môžeme rozdeliť svetelné zdroje do štyroch kategórií: smerové svetlá, bodové svetlá, reflektory a plošné svetlá.

- Smerové svetlá, ako napríklad slnko, sa nachádzajú na nekonečnom (t.j. veľmi vzdialenom) mieste. Smerové svetlo sa šíri do určitého smeru. Tento typ svetla je špecifikovaný smerom šírenia a vyžarovanou energiou (žiarivosťou).

- Bodové svetlo, napríklad sviečka alebo žiarovka, je lokalizovaný zdroj svetla v bode scény, ktorý vyžaruje do všetkých smerov izotropným spôsobom. Jeho energia sa oslabuje so vzdialenosťou. Tento typ svetla je špecifikovaný bodom jeho umiestnenia a vyžarovanou hodnotou žiarivej energie.
- Reflektor, napríklad stolná lampa, alebo divadelný reflektor, je bodové svetlo s kužeľom osvetlenia: vyžarovaná žiarivá energia je mimo tohto kužeľa nulová. Tento zdroj osvetľuje ohraničenú oblasť scény. Je špecifikovaný svojou polohou na scéne, vyžarovanou energiou a kužeľom šírenia svetla (ten sa dá parametrizovať tak, aby mal ohnisko s premenlivým uhlom plného svetla).
- Zdroj plošného svetla je nenulová, konečná plocha. Tieto zdroje svetla vytvárajú oblasti osvetlenia úplného tieňa (umbra) a čiastočných tieňov (penumbra) na scéne. Príkladom je žiarivka.

Sekundárne zdroje svetla, v skutočnej scéne funguje väčšina objektov ako sekundárne zdroje svetla. Modelovanie tohto druhu svetla je však veľmi náročné. V počítačovej grafike sa tomuto efektu približujeme zavedením ambientného svetla. Toto svetlo má konštantnú hodnotu vyžarovania a je nesmerové: jeho svetlo, keď sa odráža od objektov v scéne, má rovnakú hodnotu vyžarovania, nezávisle od polohy a orientácie objektov alebo polohy pozorovateľa [9].

4.2 Farba a intenzita

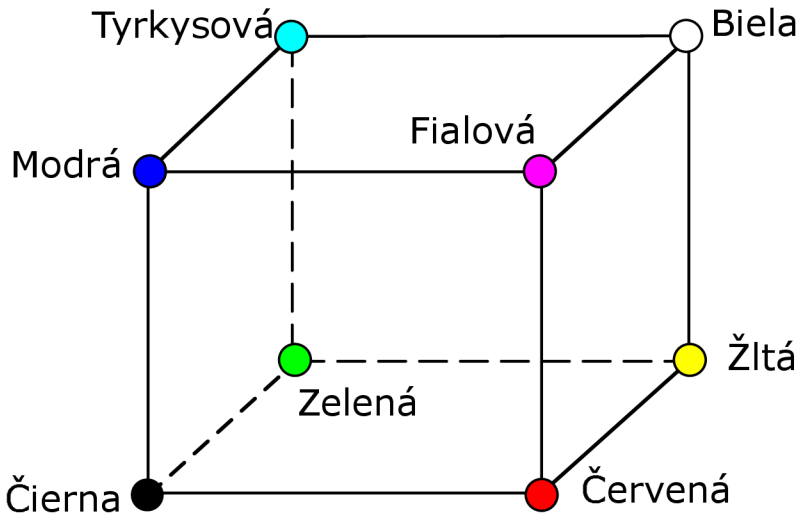
Pre zobrazenie obrázka pomocou počítačovej grafiky, je nutné vypočítať farbu a intenzitu svetla v každom jeho bode. Cieľom tejto podkapitoly je vysvetliť, ako numericky reprezentovať farbu a intenzitu, a potom vytvoriť modely osvetlenia, ktoré nám umožnia vypočítať farbu a intenzitu v každom bode, keď poznáme farbu, umiestnenie a intenzitu zdrojov svetla a optické vlastnosti objektov na scéne [3].

4.2.1 RGB priestor

RGB (z angl. *Red, Green, Blue*) predstavuje trojrozmerný farebný vektorový priestoru. Tri základné farby: červená, zelená a modrá, sú základom tohto vektorového priestoru. Každá farba môže byť reprezentovaná ako lineárna kombinácia červenej, zelenej a modrej.

Na reprezentáciu farebného priestoru sa často používa jednotková kocka, ako ukazuje obrázok 4.1. Čierna farba sa nachádza v jednom z rohov kocky, ktorý sa zvyčajne spája s počiatkom súradnicového systému, a tri základné farby sú umiestnené pozdĺž troch ortogonálnych osí. Intenzita sa mení pozdĺž hrán kocky, pričom plná intenzita každej primárnej farby zodpovedá jednotkovej vzdialenosti pozdĺž príslušnej hrany. Každá kombinácia farby a intenzity je potom reprezentovaná určitou

lineárnou kombináciou červenej farby, zelenej a modrej, kde koeficienty každej primárnej farby ležia v rozsahu od 0 po 1, bývajú uvádzané aj v celočíselnom rozsahu od 0 po 255, čo odpovedá kódovaniu každej zo zložiek RGB v jednom byte, teda pri 8-bitovej hĺbke farby. Pre každú farbu f existuje jedinečná množina súradníc (R, G, B) vnútri jednotkovej kocky, ktorá reprezentuje farbu f. Vyjadrenie pomocou 3 bytov je v súčasnosti najbežnejšie a umožňuje reprezentovať $256^3 = 16777216$ farebných odtieňov. Číselné hodnoty RGB predstavujú intenzitu: čím vyššia je hodnota R, G, alebo B, tým väčšia je ich intenzita: čím je hodnota väčší podiel červenej, zelenej alebo modrej farby na farbe f. Tento model je známy ako farebný model RGB a je jedným z najbežnejších farebných modelov v počítačovej grafike [1, 3].



Obr. 4.1: RGB kocka, prevzaté z [3].

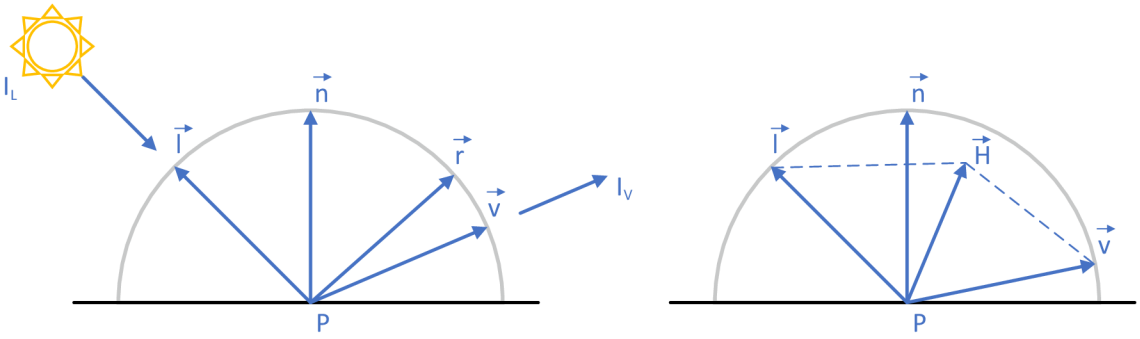
Vyhodnotenie modelu osvetlenia v každom bode, ktorý sa vykresľuje na obrazovke, je zdĺhavé, preto bolo vyvinutých niekoľko metód, ktoré umožňujú podrobný výpočet modelu osvetlenia len pre niekoľko bodov na povrchu telesa a odvodiť farebné odtiene ostatných bodov, ktoré sa majú zobraziť.

Tieto metódy zhrnieme pod spoločný názov tieňovanie (ang. shading). Pomocou tieňovania je možné rozlíšiť možné zakrivenie a zaoblenie povrchov, a tak dosiahnuť prirodzený vzhľad priestorových objektov, hoci mnohé výpočty týkajúce sa spracovania svetla boli zjednodušené alebo vynechané [1].

4.3 Phongov osvetľovací model

Phongov osvetľovací model je empirický osvetľovací model pre výpočet odrazeného svetla, ktorý navrhol v roku 1975 Bui Tuong Phong. Odraz na povrchu materiálu je určený smerom dopadajúceho svetla $\vec{l}$, smerom k pozorovateľovi $\vec{v}$ bodom na povrchu

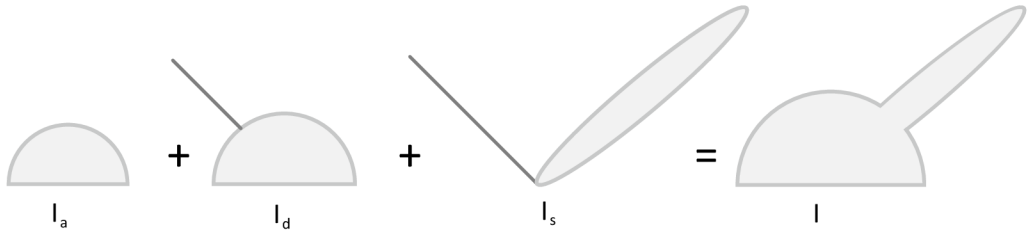
P, normálovým vektorom v mieste dopadu $\vec{n}$ a zrkadlovo odrazeným lúčom $\vec{r}$ tak, ako je uvedené na obrázku 4.2. Vektor $\vec{H}$ je stred uhlu medzi zdrojom svetla $\vec{l}$ a smerom ku kamere $\vec{v}$ a používa sa na výpočet zrkadlového odrazu. Čím bližšie je vektor $\vec{H}$ k normále povrchu $\vec{n}$, tým silnejší je zrkadlový odraz v danom bode povrchu [1].



Obr. 4.2: Geometria odrazu (vľavo) a zjednodušenie výpočtu pomocou pol-vektora (vpravo), prevzaté z [1].

Rovnica 4.5 sa nazýva Phongov model osvetlenia. Treba opäť zdôrazniť, že ide o empiricky určený výraz, ktorý priamo nesúvisí s fyzikálnou podstatou šírenia a odrazu svetla. Napriek tomu sa vďaka svojej jednoduchosti široko používa v počítačovej grafike a stal sa štandardom v aplikáciách bežiacich v reálnom čase. Prakticky všetky grafické procesory obsahujú hardvérovú implementáciu algoritmu modelu osvetlenia [1].

Ambientné, difúzne a zrkadlové zložky sú tri faktory, ktorý Phongov model osvetlenia používa na určenie výslednej farby pixela. Množstvo svetla odrazeného od všet-



Obr. 4.3: Odrazené svetlo vo Phongovom modeli osvetlenia. Ambientná, difúzna a zrkadlová zložka sa sčítajú do konečného odrazu, prevzaté z [1].

kých povrchov v obraze predstavuje zložka okolia, ktorá má pevnú hodnotu. Difúzna zložka, ktorá je úmerná uhlu medzi normálou povrchu a smerom zdroja svetla, je svetlo, ktoré je rovnomerne rozptýlené povrchom. Zrkadlová zložka, ktorá je úmerná uhlu medzi smerom odrazeného svetla a zorným uhlom diváka, označuje svetlo, ktoré sa od povrchu odráža v jednom smere. Sčítaním získame vzťah pre celkové vnímané

svetlo pozorovateľa na povrchu objektu 4.3.

$$I_a + I_d + I_s = I \tag{4.1}$$

Ambientná zložka je vyjadrená ako

$$I_a = I.K_a \tag{4.2}$$

je odrazom nešpecifikovaného okolitého svetla prichádzajúceho zo všetkých smerov a označuje sa I_a. Okolité rozptýlené svetlo je spôsobené viacnásobným odrazom od iných telies a rozptylom od molekúl vzduchu. Je preto väčšinou biele [1].

Difúzna zložka je vyjadrená ako

$$I_d = I.K_d \left(\vec{l} \times \vec{n}\right) \tag{4.3}$$

kde K_d je koeficient difúzneho odrazu, trojčlenný vektor. Označuje zastúpenie difúznej zložky v celkovom odrazenom svetle a do značnej miery predstavuje to, čo vnímame ako farbu telesa. Množstvo svetla I_d je tým väčšie, čím je smer dopadu bližšie k normále. Čo je Lambertov zákon difúzneho odrazu $I_d = I. \cos \alpha$. Tento vzťah má zmysel len pre $\vec{l} \times \vec{n} > 0$, pretože inak je povrch odvrátený od svetla a difúzna zložka svetla I_d je nulová [1].

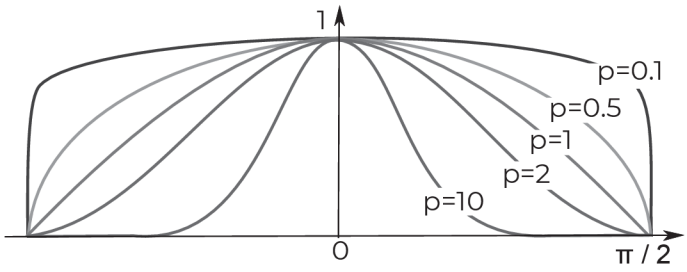
Zrkadlová zložka, nazývaná aj ako spekulárna je vyjadrená ako

$$I_s = I.K_s \left(\vec{r} \times \vec{v}\right)^p \tag{4.4}$$

kde I predstavuje farebné zloženie dopadajúceho lúča, $\vec{v}$ je jednotkový vektor pohľadu. Vektor $\vec{r}$ vyjadruje smer ideálneho zrkadlového odrazu a je symetrický k vektoru $\vec{l}$ pozdĺž normály - možno ho vypočítať zo vzťahu
$\vec{r} = 2(\vec{l} \times \vec{r})\vec{n} - \vec{l}$. Pripomeňme si, že skalárny súčin jednotkových vektorov sa rovná kosínusu uhla, ktorý zvierajú.

Koeficient zrkadlového odrazu $0 \leq K_s \leq 1$ určuje stupeň zastúpenia odrazenej zrkadlovej zložky svetla v celkovom odrazenom svetle. Tento koeficient je ternárnym farebným vektorom. Násobenie farebných vektorov sa na rozdiel od geometrických súčinov vykonáva po jednotlivých zložkách vektorov [1].

Exponent p riadi rýchlosť poklesu funkcie definujúcej osvetlenie. Keď $p \to \infty$, funkcia sa približuje k funkcii odrazivosti (označovanej aj ako δ), čo znamená, že budeme mať dokonalý zrkadlový odraz (osvetlenie sa zredukuje na bod). Na druhej strane, keď $p \to 0$, plocha osvetlenia sa zväčšuje, čo znamená, že povrch je drsnejší (zvyšuje sa rozptyl odrazu). Z fyzikálneho hľadiska je teda možné exponent p považovať za drsnosť povrchu. Nazýva sa preto aj exponentom drsnosti alebo Phongovým exponentom.



Obr. 4.4: Exponent zrkadlovej zložky, prevzaté z [9].

Výsledný vzťah pre Phongov tieňovací model vyzerá nasledovne.

$$I = I_a.K_a + I_d.K_d.\max(0, \vec{n} \times \vec{l}) + I_s.K_s.\max(0, \vec{r} \times \vec{v})^p \tag{4.5}$$

V rovnici je I výsledná farba pixela, ktorá sa skladá z troch typov zložiek odrazu svetla od materiálu. Z týchto prípadov sa potom skladá výsledný odraz. Odraz sa delí na I_s zrkadlový (spekulárny), I_d difúzny a I_a ambientný.

4.4 Tieňovanie

Tieňovanie je proces v počítačovej grafike, ktorý popisuje ako sa má aplikovať svetlo a farby na 3D objekt. Tým je možné dosiahnuť ilúziu vlastností materiálov ako aj hĺbky trojrozmerného priestoru vykreslenej na 2D obrazovku.

4.4.1 Shadery

Moderné shadery sú v skutočnosti grafické programy, ktoré sa neobmedzujú len na výpočet farieb bodov. Existujú geometrické shadery, ktoré môžu meniť zoznam trojuholníkov, ktoré sa majú spracovať v nasledujúcich fázach, a teselačné shadery, ktoré prijímajú vysokoúrovňové opisy povrchov a vytvárajú z nich zoznamy trojuholníkov. Príkladom je shader deliaceho povrchu (subdivision surface shader), ktorý môže ako vstup prijať vrcholy a štruktúru siete riadiacej siete deliaceho povrchu a ako výstup vytvoriť kolekciu malých trojuholníkov, ktoré tvoria dobrú aproximáciu hraničného povrchu. Existujú aj vertex shadery, ktoré slúžia len na transformáciu umiestnenia vrcholov a spravidla nemajú nič spoločné s prípadnou farbou [9].

Hoci typický grafický program môže mať geometrický shader, teselačný shader, vertexový shader a fragmentový shader, existuje aj možnosť vypnúť ktorúkoľvek časť reťazca a prikázať: „Počítaj len po tento úsek a potom skonči“. Program teda môže spustiť svoj geometrický a teselačný shader a potom vrátiť údaje CPU, ktorý ich môže nejakým spôsobom upraviť pred tým, ako ich vráti GPU, aby ich spracoval rasterizačnou a orezávacou jednotkou a potom fragment shaderom [9].

4.4.2 Konštantné tieňovanie

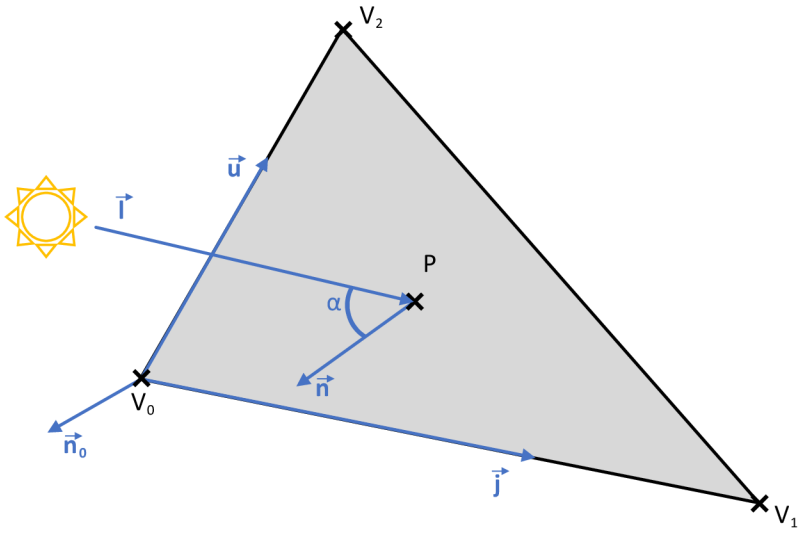
Táto metóda, známa aj ako flat shading, je veľmi rýchla, jednoduchá a používa sa na zobrazovanie rovinných povrchov. Predpokladá, že každý povrch má len jednu normálu. Ak neexistuje žiadna normála implicitne obsiahnutá v údajoch priestorového modelu, možno ju použiť pre konvexné rovinné povrchy určiť ako výsledok, ktorý je správne orientovaný tak, aby smeroval do vonkajšieho polpriestoru.

Z normály sa vypočíta jedna farba a pri rasterizácií povrchu sa priradí všetkým jeho pixelom. Konštantné tieňovanie sa používa tam, kde je potrebné dosiahnuť vysokú rýchlosť vykreslenia [1].

Nech je tvar trojuholníka definovaný jeho 3 vrcholmi V_0,V_1,V_2 a normálami ($\vec{n_0}$, $\vec{n_1}$, $\vec{n_2}$). Nech je zdroj svetla smerový s vektorom svetla $\vec{l}$. Svetlo má okolitú a smerovú časť s farbami [R, G, B]. Vypočítať je možné ktorúkoľvek normálu, keďže normálový vektor reprezentuje smer ktorým trojuholník leží, k výpočtu normály plochy sa použije vektorový súčin vektoru $\vec{u}$ s vektorom $\vec{v}$, čoho výsledkom je vektor kolmý na obidva pôvodné vektory $\vec{n}$ a rovnaký po celej ploche trojuholníku.
$\vec{u} = V_1 - V_0 = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$
$\vec{j} = V_2 - V_0 = (x_3 - x_1, y_3 - y_1, z_3 - z_1)$
$\vec{n} = \vec{u} \times \vec{j} = (u_y v_z - u_z v_y, u_z v_x - u_x v_z, u_x v_y - u_y v_x)$
Kosínus uhlu, ktorý je medzi zdrojom svetla a normálovým vektorom je potom:
$\cos\alpha = \vec{l} \cdot \vec{n}$, kde uhol sa vypočíta ako: $\alpha = \cos^{-1}(\vec{l} \cdot \vec{n})$, funkcia vráti uhol v radiánoch. Ak chceme uhol previesť v stupňoch, môžeme ho vynásobiť $\alpha = \alpha\frac{180}{\pi}$. Farba samotného polygónu sa určí napríklad použitím Phongového výpočtu zložiek svetla v danom bode podľa rovnice 4.5.
$C = I$
Farbu vo fragment shaderi následne priradí všetkým pixelom plochy po rasterizácii 4.5. Výsledná farba je teda počítaná pre jeden bod trojuholníka napríklad cez Phongov osvetľovací model popísaný rovnicou 4.5 a následne sa farba potom priradí vo fragment shaderi celému trojuholníku.

4.4.3 Gouraudovo tieňovanie

Metódu navrhol Henri Gouraud a je vhodná na tieňovanie telies, ktorých povrch je aproximovaný množinou rovinných plôch. Pre fungovanie algoritmu je dôležitá znalosť farieb všetkých vrcholov spracovávaného povrchu. Farbu vrcholov určíme vyhodnotením modelu osvetlenia. Vzhľadom na metódu výpočtu, ktorá interpoluje farby, nemá zmysel zohľadňovať zrkadlovú zložku svetla. Preto sa počítajú len ambientná a difúzna zložka svetla. Potom sa bilineárnou interpoláciou vypočítajú farebné odtiene percepčných bodov danej oblasti. Preto sa Gouraudovo tieňovanie nazýva aj farebná interpolácia. Farbu môžeme interpolovať vo forme trojitého vektora [R, G, B]



Obr. 4.5: Výpočet u konštantného tieňovania, prevzaté z [1].

na základe jednotlivých zložiek. Každú zložku v pôvodnom rozsahu od 0 po 1 možno previesť na celočíselný rozsah od 0 do 255. Na interpoláciu možno použiť celočíselnú aritmetiku. Celočíselná bilineárna interpolácia sa dá ľahko realizovať technickými prostriedkami. Preto je Gouraudovo tieňovanie v súčasnosti štandardnou metódou používanou v grafických akcelerátoroch [1]. Najprv sa pre každý vertex vypočíta jeho normálový vektor. Normála povrchu sa používa na určenie uhla medzi zdrojom svetla a povrchom, ktorý sa používa na výpočet farby vrcholu. Najprv sa vypočítajú normálové vektory vo vrcholoch V_0, V_1, V_2 čím dostaneme $(\vec{n_0}, \vec{n_1}, \vec{n_2})$. Následne sa určí intenzita a farba svetla na základe uhlu medzi dopadajúcim svetlom $\vec{l}$ a príslušnou normálou 4.6.

$C_0 = I_0(\vec{l} \cdot \vec{n_0})$

$C_1 = I_1(\vec{l} \cdot \vec{n_1})$

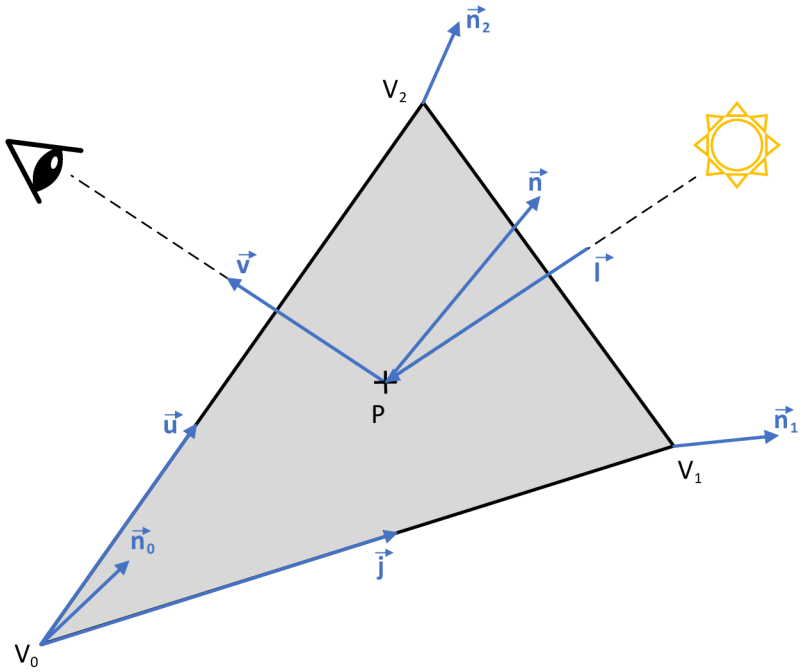
$C_2 = I_2(\vec{l} \cdot \vec{n_2})$

Interpolujú sa farby vrcholov na povrchu polygónu pomocou barycentrických súradníc. To možno vykonať lineárnou interpoláciou farieb vrcholov v každom bode povrchu polygónu, resp. pre každý pixel po rasterizácií. Teraz trojuholník s vrcholmi V_0, V_1 a V_2 a farbami C_0, C_1 a C_2. Ak chceme interpolovať farbu v bode P vnútri trojuholníka, najprv vypočítame barycentrické súradnice bodu P vzhľadom na vrcholy: Nech A_0, A_1 a A_2 sú plochy trojuholníkov s vrcholmi V_0, V_1 a V_2 a w_0, w_1 a w_2 sú barycentrické hodnoty vzhľadom k bodu P od vrcholov V_0, V_1 a V_2. Vypočíta sa celková plocha trojuholníka:

$A = \frac{1}{2}\|(V_1 - V_0) \times (V_2 - V_0)\|$

Jednotlivé časti plochy, ktoré delí bod P potom:

$A_0 = \frac{1}{2}\|(V_1 - P) \times (V_2 - P)\|$



Obr. 4.6: Výpočet u Gouraudového tieňovania, prevzaté z [1].

$A_1 = \frac{1}{2}\|(V_2 - P) \times (V_0 - P)\|$
$A_2 = \frac{1}{2}\|(V_0 - P) \times (V_1 - P)\|$
Z týchto hodnôt vieme vypočítať následne váhu jednotlivých pomerov plôch k ploche celého trojuholníka ako:
$w_0 = \frac{A_0}{A}$
$w_1 = \frac{A_1}{A}$
$w_2 = \frac{A_2}{A}$
Keďže A_0, A_1 a A_2 sú plochy čiastkových trojuholníkov vytvorených spojením P s každým vrcholom a A je plocha hlavného trojuholníka, barycentrické súradnice w_0, w_1 a w_2 predstavujú relatívne váhy vrcholov V_0, V_1 a V_2 pri interpolácii hodnôt v bode P. Použitím týchto súradníc na interpoláciu hodnôt, ako sú farby vrcholov alebo normály, v bode P môžeme získať hladký efekt tieňovania na celej ploche trojuholníka. Interpolujú sa farby vrcholov v bode P pomocou barycentrických súradníc ako:
$C = w_0.C_0 + w_1.C_1 + w_2.C_2$
Rovnica predstavuje vážený priemer farieb vrcholov C_0, C_1 a C_2 , pričom váhy sú dané barycentrickými súradnicami w_0, w_1 a w_2. Barycentrické súradnice určujú, ako veľmi prispieva každá farba vrcholu k interpolovanej farbe v bode P vnútri trojuholníka, pričom súčet váh je 1. C je výsledná interpolovaná farba v bode P.

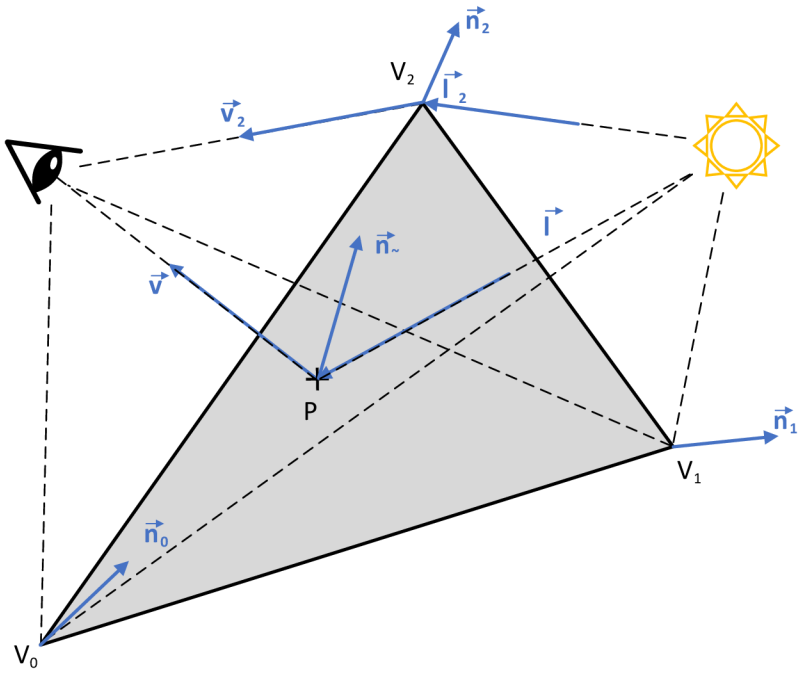
Počas rasterizácie sa farba každého pixela v polygóne určí interpolovanou farbou v príslušných barycentrických súradniciach. Keďže interpolácia sa vykonáva

pomocou lineárnej kombinácie farieb vrcholov a váhy tejto kombinácie sú dané barycentrickými súradnicami, môžeme interpolovanú farbu pre každý pixel efektívne vypočítať.

Gouraudovo tieňovanie vypočíta farby vrcholov polygónu na základe uhla medzi zdrojom svetla a normálou povrchu a potom interpoluje farby na povrchu polygónu pomocou barycentrických súradníc.

4.4.4 Phongovo tieňovanie

Účelom algoritmov tieňovania je vyhladiť vzhľad polygonálnych modelov znížením vplyvu ostrých hrán, aby oko nadobudlo dojem hladkého zakriveného povrchu. Phongovo tieňovanie simuluje zakrivené povrchy interpolovaním normálových vektorov namiesto intenzít (ako v Gouraudovom modeli tieňovania). Normálové vektory sa pozdĺž zakriveného povrchu menia, takže Phongovo tieňovanie poskytuje lepšiu aproximáciu intenzity pre zakrivené povrchy modelované rovinnými polygónmi 4.7 [3]. Po rasterizácii sa pre každý pixel ležiaci v trojuholníku sa interpolujú jeho 3 nor-



Obr. 4.7: Výpočet u Phongovho tieňovania, prevzaté z [1].

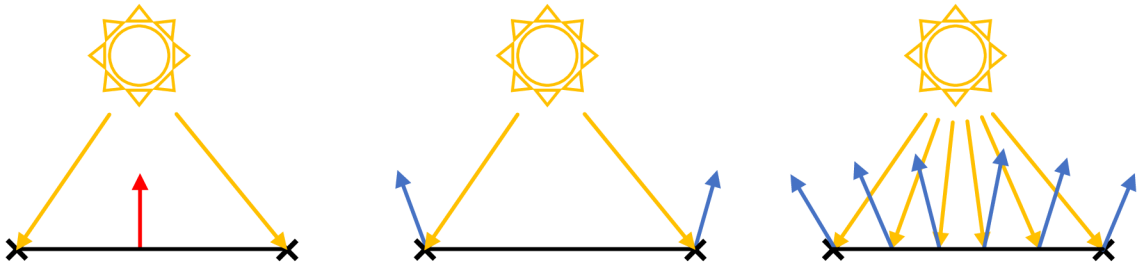
mály vrcholov na jeden celkový normálový vektor na základe váh barycentrických koordinátov. Vo Phongovom tieňovaní vedie interpolácia normál k ich postupnému nakloneniu k normále najbližšiemu vrcholu. Pre bod P vo vnútri trojuholníka, sa vypočítajú jeho barycentrické súradnice $[u, v, w]$ ako:
$u = \frac{((V_{1y}-V_{2y})(P_x-V_{2x})+(V_{2x}-V_{1x})(P_y-V_{2y}))}{((V_{1y}-V_{2y})(V_{0x}-V_{2x})+(V_{2x}-V_{1x})(V_{0y}-V_{2y}))}$

$v = \frac{((V_{2y}-V_{0y})(P_x-V_{2x})+(V_{0x}-V_{2x})(P_y-V_{2y}))}{((V_{1y}-V_{2y})(V_{0x}-V_{2x})+(V_{2x}-V_{1x})(V_{0y}-V_{2y}))}$
$w = 1 - u - v$
Na základe barycentrických súradníc interpolujeme normálové vektory vo vrcholoch trojuholníka na výsledný $\vec{n_\sim}$ ako:
$\vec{n_\sim} = u.\vec{n}_0 + v.\vec{n}_1 + w.\vec{n}_2$ týmto spôsobom sa vypočítajú aj pre ostatné pixely, kde tento proces umožňuje váhovanie naprieč trojuholníkom a teda, čím bližšie je bod resp. pixel jednému z vrcholov, tým viac je po váhovom priemere naklonená pôvodnej normále z daného vrcholu a to umožňuje potom plynulé tieňovanie. Následne sa vypočíta vektor $\vec{l}$, z bodu P smerom k zdroju svetla. Ďalej sa vypočíta vektor $\vec{v}$, z bodu P smerom ku kamere. Tým sa získajú všetky potrebné pre výpočet dielčich zložiek podľa vzťahu 4.1 a rozpísaný je v 4.5. Ambientná zložka je:
$I_a = I_a.K_a$ ako v 4.2.
Difúzna zložka (použitím vypočítaného váhovo priemerovaného vektoru $\vec{n_\sim}$:
$I_d = I_d.K_d.(\vec{n_\sim} \times \vec{l})$ podľa 4.3.
Zrkadlová zložka:
$I_s = I_s.K_s.(\vec{r} \times \vec{v})^p$ ako v 4.4.
reflexný vektor sa taktiež vypočíta použitím váhovo priemerovaného vektoru $\vec{n_\sim}$ smerom k svetlu $\vec{l}$. Medzi reflexným vektorom a vektorom smerujúcim do kamery sa vypočíta ich vektorový súčin a ten sa následne umocní na koeficient lesklosti p. Zložky budú sčítané podľa vzťahu z 4.5:
$I = I_a.K_a + I_d.K_d.\max(0, \vec{n} \times \vec{l}) + I_s.K_s.\max(0, \vec{r} \times \vec{v})^p$

Táto technika poskytuje hladký a realistický efekt tieňovania na povrchu polygónu. Gouraudovo tieňovanie teda vyniká výpočtovou rýchlosťou, ale nedosahuje rovnakú kvalitu obrazu ako Phongovo tieňovanie. Je úplne nevhodné používať ho v dynamických scénach, kde sú viditeľné zmeny jasu na povrchu pri snímaní panelov. Ak by sme napríklad otáčali guľu okolo svojej osi, videli by sme zreteľne jasný pruh vždy, keď sa okraj plášťa otočí kolmo na smer svetla $\vec{l}$ [1].

4.5 Porovnanie tieňovaní

Zjednodušenú predstavu o rozdieloch tieňovania predstavuje obrázok 4.8. Pre kon-



Obr. 4.8: Porovnanie Konštantného, Gouraudovho a Phongovho tieňovania, prevzaté z [1].

štantné tieňovanie platí, že počíta svetlo iba raz pre celý trojuholník a následne vypočítanú farbu priradí každému pixelu v trojuholníku.

- Výhody:
  1. Rýchlosť
  2. Výpočtová nenáročnosť
- Nevýhody:
  1. Vzhľad

Gouraudovo tieňovanie je náročnejšie na výpočet ako konštantné, avšak umožňuje hladšie vykreslenie. Keďže, vypočítanú farbu na vertexoch interpoluje váhovým priemerom v trojuholníku, z toho vyplýva aj jeho nevýhoda a to že, ak je odlesk v strede trojuholníka a nie na jeho vrcholoch, tak sa tento odlesk nemusí prejaviť.

- Výhody:
  1. Efektivita pri výpočte
  2. Lepší vzhľad ako pri konštantnom tieňovaní
- Nevýhody:
  1. Skreslenie odleskov
  2. Machov efekt pri prechodoch

Phongove tieňovanie interpoluje normály a váhovo spriemeruje pre daný pixel v trojuholníku vzhľadom na vzdialenosť k vrcholom, ale farba pixelu sa počíta samostatne pre každý z nich, čo sa prejaví realistickým výsledkom.

- Výhody:
  1. Realistický vzhľad
  2. Odlesky
- Nevýhody:
  1. Výpočtovo náročný

5 Vytvorené aplety

Pri tvorbe apletov bola použitá kombinácia HTML 3.1.1, CSS 3.1.2 a TypeScriptu 3.1.4 s použitím 3D knižnice Three.js 3.1.7. TypeScript prináša do projektov JavaScript voliteľné statické typovanie. Hlavnou výhodou statického typovania je, že typové chyby sa zisťujú a opravujú v čase zostavovania, takže kód bude po nasadení do produkcie s väčšou pravdepodobnosťou fungovať správne.

5.1 Príprava prostredia a kompilácia kódu

Nasledujúci príklad bude uvedený pre operačný systém Ubuntu 20.04.5, NPM balíček verzie 9.1.2 a Node.js verzie 16.19.1. V prvom kroku sa nainštaluje správcu balíčkov pre JavaScriptový programovací jazyk, ktorým je NPM 3.1.5, ten pri spustení kontroluje závislosti a získa balíčky potrebné pre spustenie apletov. Tie budú zabalené do bundle.js, takže nebude potrebné pre ich spustenie mať tieto závislosti nainštalované.

```
sudo apt update
sudo apt install curl -y
sudo apt install npm -y
sudo npm install npm -g
```

Následne sa stiahne a spustí skript na pridanie Node.js verzií 16.x do Ubuntu systému.

```
declare URL="https://deb.nodesource.com/setup_16.x"
curl -sL $URL | sudo -E bash -
```

Nakoniec sa nainštaluje samotné Node.js príkazom:

```
sudo apt install nodejs -y
```

Pre kompiláciu zo zdrojových kódov sa v koreňovej zložke s danou úlohou spustia príkazy:

```
cd $hlavná_zložka_apletu
npm i
npm run start
npm run dev
```

Kde výsledkom by malo byť automatické otvorenie `http://localhost:8080/` v prehliadači s daným riešením. Následne každá zmena v kóde v ceste `./src` sa prejaví automatickou rekompiláciou a znovunačítaním v prehliadači, takže sa zmena prejaví hneď. Pre prejavenie sa zmien v produkčnej zložke `./dist` je nutné zopakovať príkaz:

```
npm run start
```

Štruktúra v hlavnej zložke s apletom je pre každú úlohu rovnaká a to:

- ./public
  - Obsahuje statické súbory, v našom prípade 3D modely vo formáte stl.
- app.ts
  - Hlavný TypeScript súbor, bude následne prekompilovaný do JavaScriptu ako bundle.js v zložke `./dist`, vrámci kódu hľadá element v html `canvas_frame` do ktorého sa vykreslí ako DOM element (dynamicky element).
  - Je kód ktorý popisuje základné nastavenie, správanie a možnosti apletu a prijíma parametre z HTML stránky ako vstup.
- index.html
  - Obsahuje základnú štruktúru stránky a vpísané formátovacie kaskádové štýly. Zároveň obsahuje div element s identifikátorom `canvas_frame`. Po načítaní stránky sa spustí aplet, kde tento identifikátor v kóde stránky vyhľadá a dynamicky sa doň vykreslí.
- package.json
  - Inicializuje a popisuje Node.js projekt, parametre a potrebné závislosti.
- tsconfig.json
  - Konfiguračný súbor, ktorý určuje, ktoré súbory sa majú kompilovať, a možnosti kompilátora pre projekt.
- webpack.config.js
  - Popisuje ako vytvoriť jeden súbor JavaScript (bundle.js), ktorý obsahuje všetky závislosti potrebné v sebe pre spustenie. Práve preto je ho možné spustiť bez toho, aby bolo nutné mať nainštalované všetky závislosti pri zobrazení na stránke.
- ./dist
  - Zložka obsahuje finálne súbory po kompilácií, teda výsledný bundle.js a index.html.

Pre vytvorenie scény a zobrazenie osí, sú použité:

```
const scene: THREE.Scene = new THREE.Scene();
scene.add(new THREE.AxesHelper(5));
```

Pre pridanie ambientného bieleho svetla:

```
const ambient_light = new THREE.AmbientLight(0xffffff);
scene.add(ambient_light);
```

Nastavenie perspektívy kamery 2.4.2

```
const camera: THREE.PerspectiveCamera =
```

```
new THREE.PerspectiveCamera(
    75,
    desired_width / desired_height,
    0.1,
    1000
);
camera.position.z = 3;
```

Prvý parameter udáva vertikálne zorné pole kamery, potom druhý parameter podľa pomeru veľkosti scény na web stránke udá nastaví pomer strán, posledné dva parametre udávajú blízku a vzdialenú rovinu, ako podľa obrázku 2.2. Renderer generuje 2D obraz podľa scény a zobrazuje na HTML canvas, ďalej pre priradenie možností, ktoré umožňujú pohyb resp. rotáciu a približovanie a oddiaľovanie scény:

```
const renderer: THREE.WebGLRenderer =
    new THREE.WebGLRenderer();
const controls: OrbitControls =
    new OrbitControls(camera, renderer.domElement);
```

Nakoniec, sa pridá smyčka pre vykresľovanie dookola:

```
function render() {
  renderer.render(scene, camera);
}
function animate() {
  requestAnimationFrame(animate);
  controls.update();
  render();
}
animate();
```

Tým je vytvorená základná funkčnosť scény.

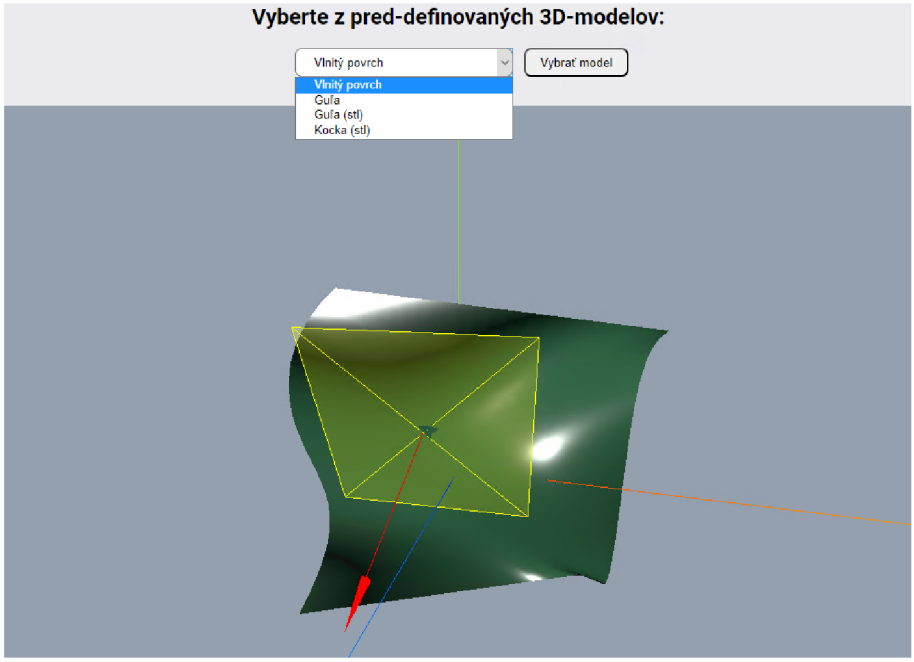
5.2 Ilustrácia dotykovej roviny a normály k ľubovolnému bodu na trojrozmernom povrchu

Prvý aplet znázorňuje dotykovú rovinu a normálu k ľubovolnému bodu na vybranom trojrozmernom povrchu. Východzia scéna obsahuje zvlnený povrch (NURBS) 1.2.2, pričom po kliknutí do scény na tento povrch trojrozmerného objektu sa k bodu vykreslí jeho normálový vektor a dotyková rovina. Scénu je možné ľubovolne rotovať

konštantným držaním ľavého tlačidla a pohybom myši. Taktiež je umožnené približovanie a oddiaľovanie pomocou kolieska na myši, keď je kurzor v scéne. V rozbaľovacej ponuke pod preddefinovanými 3D modelmi je možné vybrať z prednastavených objektov, po kliknutí na tlačidlo „Vybrať model“ sa zo scény odstráni aktuálne zobrazený model a do scény sa pridá sa nový užívateľom vybraný 3D model. Na výber sú 4 objekty:

- Vlnitý povrch.
- Guľa.
- Guľa (stl).
- Kocka (stl).

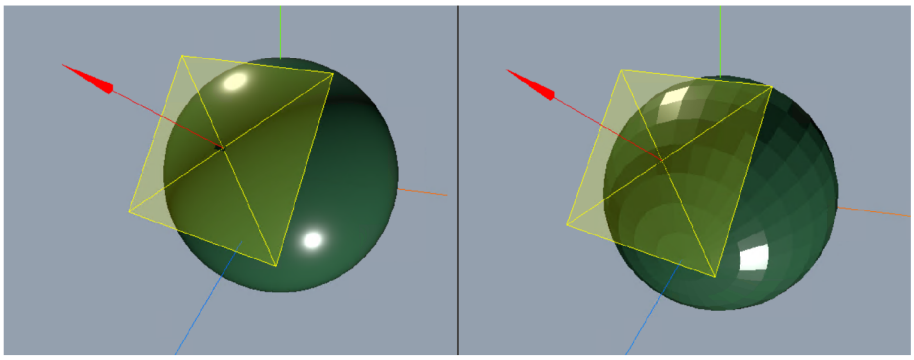
Tieto základné možnosti a východzí stav stránky ilustruje obrázok 5.1. Aplet je dostupný na prehliadanie cez odkaz: `https://priscak.sk/prva-uloha`.



Obr. 5.1: Základné možnosti a východzí stav prvého apletu.

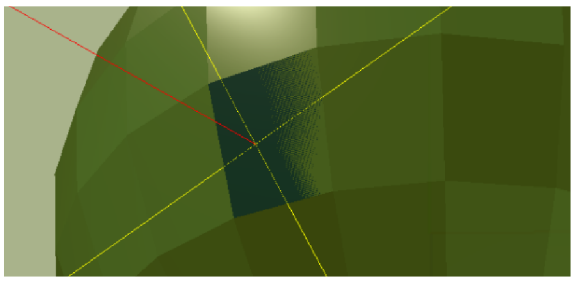
Vlnitý povrch je tvorený obojstrannou NURBS plochou. Následne možnosti guľa a guľa (stl) sú rozdielne v tom, že guľa je vytvorená cez Three.js modul SphereGeometry, teda je definovaná vektorovo (ideálne nekonečne plynulo). Naproti tomu guľa (stl) je popísaná ako sieť trojuholníkov (teselácií) 1.2.6, ktoré nemajú medzi sebou medzery, nastavením jemnosti resp. veľkosti jednotlivých častí vieme určite ako presne bude výsledný objekt aproximovať vektorovo definovaný objekt. Veľkosti objektov guľa a guľa (stl) sú veľmi podobné, takže vytvárajú ukážkový príklad medzi vektorovo definovaným objektom a jeho aproximáciou. Taktiež, zobrazuje že v danom trojuholníku pri aproximácii v danom trojuholníku sa mení len poloha normá-

lového vektoru, ale uhol ostáva zachovaný, pričom dotyková plocha ostáva rovnaká, ukážka na obrázku 5.2. Plochu ktorú vidíme akoby „preskakovať“ medzi materiá-



Obr. 5.2: Dotyková rovina a normála k bodu na guli.

lom gule (stl) a dotykovou rovinou je spôsobená javom, resp. artefaktom zvaným Z-buffer. Z-buffer sa niekedy označuje aj ako hĺbkový buffer, si ukladá informáciu o hĺbke pixlov v scéne a na základe toho sa potom vykresľujú len fragmenty viditeľné v scéne z pohľadu kamery. To pomáha potom určiť, ktoré objekty, resp. ich časti má zmysel vykresľovať a počítať pri renderovaní. Avšak môže nastať situácia ako ukazuje priblíženie artefaktu z obrázku 5.2 na obrázku 5.3. Keďže dotyková plocha



Obr. 5.3: Príklad Z-buffer artefaktu.

1.2.4 sa vykresľuje v identickej hĺbke z pohľadu kamery ako samotný materiál, pri renderovaní sa engine nevie rozhodnúť, ktorý fragment vykresliť, čo sa prejavuje preblikávaním v mieste s rovnakou hĺbkou. Tento artefakt nie je uniformný vzhľadom na vzdialenosť kamery. Z-buffer je presnejší, čím bližšie je kamera objektu, a čím sa vzdiaľuje, tak začína byť menej precízny a tak po oddialení sa jeden z povrchov vykreslí nad druhým a preblikávanie zanikne. Ošetriť tieto prípady sa dá zvýšením hĺbky vyhodnocovania, alebo implementáciou LOD (z angl. *Level of Detail*) [21].

5.2.1 Programové spracovanie

Spracovanie užívateľovho kliknutia vyhodnotí metóda:

```
function process_users_click(position)
```

Tá vracia pozíciu kliknutia, v prípade, že bod, kde užívateľ klikol má intersekciu s objektom vo scéne. Pozícia je ďalej odovzdaná ako vstup do modulu:

```
import {CustomPlaneHelper} from "./custom_plane_helper";
```

Pred samotným vykreslením sa veľkosť roviny prispôsobuje veľkosti objektu. Určí sa jeho ohraničujúci rámec, čo v 3D grafike predstavuje najmenší obdĺžnikový rámec, ktorý môže obsahovať celý objekt. Často sa používa na efektívnu detekciu kolízií, pretože jeho výpočet je menej náročný na zdroje ako výpočet skutočnej geometrie objektu.

```
const object_max_size = Math.max(object_bbox_size.x,
            object_bbox_size.y, object_bbox_size.z);
```

Táto veľkosť objektu sa potom pošle spolu so súradnicami intersekcie a hexadecimálnou farbou 0xFFFF00 RGB (255,255,0), čo reprezentuje žltú farbu na vykreslenie dotykovej plochy. Následne je prenásobený hodnotou 0,6 pre lepšie prispôsobenie veľkosti k objektu `plane_helper_size_multiplier`.

```
const plane_helper = new CustomPlaneHelper(
    position.point,
    plane,
    object_max_size * plane_helper_size_multiplier,
    0xffff00
);
```

Objektu je ešte priradená hodnota priehľadnosti 0,2 kvôli lepšej názornosti. Pri vykreslení normálového vektoru sa využije východzí `ArrowHelper`:

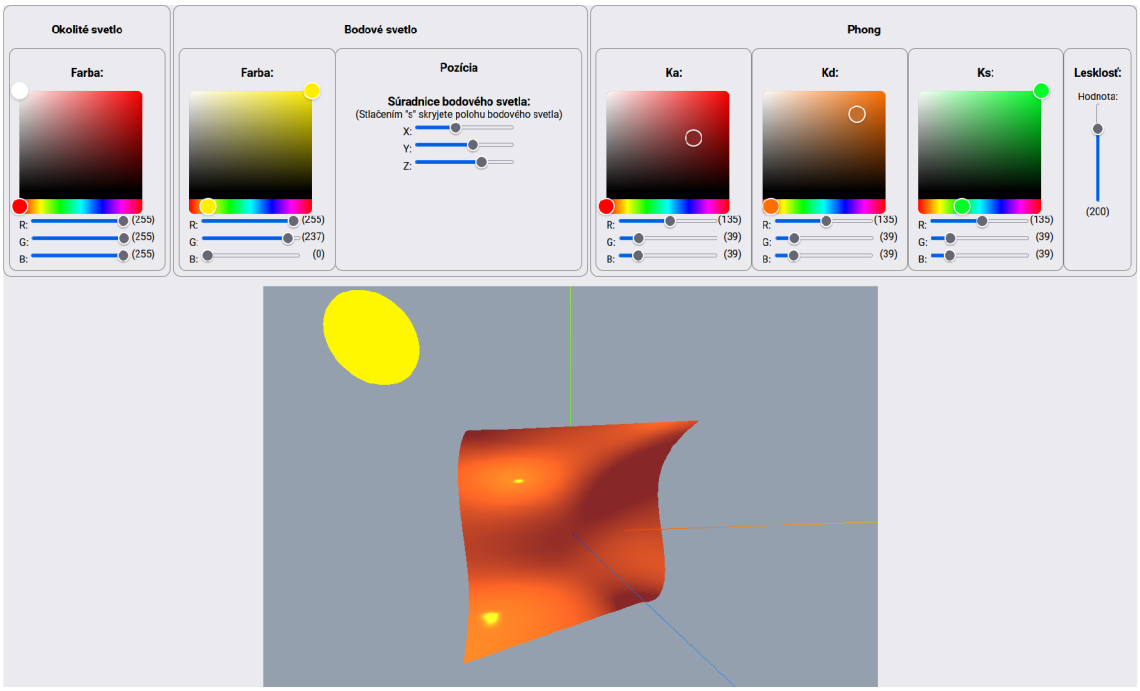
```
const normal_helper = new THREE.ArrowHelper(
    position.face.normal,
    position.point,
    object_max_size * normal_helper_size_multiplier,
    0xff0000
);
```

Predajú sa mu rovnaké hodnoty pozície intersekcie, prispôsobí veľkosť na základe objektu a nastaví smer a farba 0xFF0000 RGB (255,0,0), teda červená.

5.3 Ilustrácia vplyvu parametrov na vzhľad odrazu svetla v Phongovom osvetľovacom modeli

Aplet ilustruje Phongov osvetľovací model 4.3 a vplyv parametrov na výsledný vzhľad odrazu svetla ako ukazuje obrázok 5.4.

Aplet je dostupný na prehliadanie cez odkaz: `https://priscak.sk/druha-uloha`.



Obr. 5.4: Vplyv parametrov na odraz svetla vo Phongovom osvetľovacom modeli.

5.3.1 Parametre prostredia a materiálu

Nastavením parametrov resp. pomerom je možné modelovať rôzne svetelné podmienky a zároveň materiál.

Vrámci nastavenia prostredia sa dajú upraviť parametre pre zdroje svetla 4.1.1:

- Ambientné svetlo, farba a intenzita.
- Bodové svetlo, farba, intenzita a jeho pozícia.

Nastavenia vlastnosti samotného Phongovho materiálu určujú:

- Parameter „K_a“.

Určuje akú ambientnú (vlastnú) farbu má daný zobrazený predmet, tým sa určuje aké zložky svetla odráža a aké pohlcuje.

- Parameter „K_d“.

Určuje difúznu farbu pri odraze náhodným smerom (neriadi sa pravidlom, že uhol dopadu sa rovná uhlu odrazu).

- Parameter „K_s“.

Určuje zrkadlovú (spekulárnu) zložku pri odraze (riadi sa pravidlom, že uhol dopadu sa rovná uhlu odrazu)

- Parameter Lesklosti.

Lesklosť materiálu, tá sa vo výpočte zrkadlového odlesku umocňuje na túto hodnotu, vo vzorci 4.5 je to exponent p.

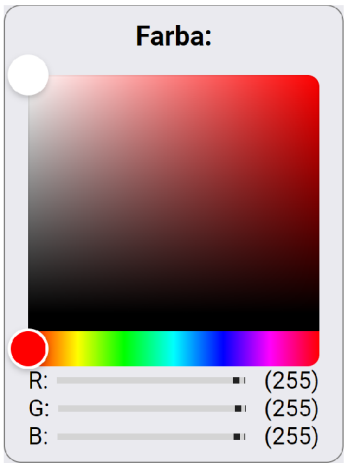
Pomerom týchto parametrov je možné modelovať rôzne svetelné podmienky prostredia, ďalej modelovať vlastnosti materiálov a polohovaním kamery meniť pozorovacie uhly a vplyv jednotlivých parametrov a zložiek svetla na výsledný efekt.

5.3.2 Programové spracovanie

Pridané komponenty z hľadiska projektu sú:

```
import "vanilla-colorful";
```

Slúži na vytvorenie výberu farieb z palety ako ukazuje obrázok 5.5 a vracia jej hexadecimálnu hodnotu. Nastavovaním v úplne ľavom okraji je stále biele svetlo RGB



Obr. 5.5: Paleta pre výber farby.

(255,0,0). Pohybom smerom dole sa mení intenzita svetelného zdroja, ale nie farba. Nastavovaním v úplne pravom okraji smerom dole sa mení intenzita danej farby, teda napríklad keď máme plne červenú RGB (255,0,0), tak v strede bude RGB (127,0,0) a na spodku RGB (0,0,0) čím zdroj vypneme. Farba sa nastavuje pomocou farebného prúžku pod paletou, čím sa zmení potom aj paleta vyššie. Prípadne nastavením jednotlivo RGB zložky pomocou posuvníkov. Apletu je potom vrátená hexadecimálna hodnota, napríklad 0xFFFFFF, čo sa potom transformuje do RGB zložiek nasledovne:

```
const r = parseInt(color.substr(1, 2), 16);
```

```
const g = parseInt(color.substr(3, 2), 16);
const b = parseInt(color.substr(5, 2), 16);
```

Každá zložka je definovaná dvoma znakmi v hexadecimálnom zápise. Zložka červená v prvých dvoch znakoch, zložka zelená v treťom a štvrtom znaky a modrá v posledných dvoch znakoch. Funkcia `parseInt`, na prvom parametri zoberie tieto znaky a podľa druhého parametru 16, vie že sa jedná o hexadecimálny základ 00 až FF a konvertuje ich potom do desiatkového od 0 až 255. Výstupom bude, že vstup 0xFFFFFF bude preložení do `r`, `g` a `b` rovné hodnotám 255.

Šablóna vytvorí Phongov materiál na základe definovanej predlohy a parametre sa mapujú na zmeny v HTML stránke a prenášajú do samotného výpočtov.

```
import {create_custom_phong_material}
    from "./custom_phong_material";
```

Zjednodušene grafický reťazec sa stále vykonáva dookola a prijíma parametre okolia a materiálu následne predá vertex sharedu, potom grafická karta vykoná rasterizáciu (mapovanie objektu, resp. jeho vrcholov na rozmer pixlov na obrazovke), kde nakoniec sa predá fragment shaderu na výpočet farby jednotlivých pixlov.

Phongov materiál načíta hodnoty z web stránky a uloží do uniforms dátovej štruktúry, ktoré slúžia ako parametre pre shadre. Tieto hodnoty sa od momentu zavolania shadru vrámci jednej iterácie nemenia.

```
uniforms: {
    ambientLightColor: { value: ambient_light },
    spotLightColor: { value: spot_light },
    Ka: { value: Ka },
    Kd: { value: Kd },
    Ks: { value: Ks },
    spotLightPosition: { value: spot_light_position },
    shininess: { value: shininess },
}
```

Všetky hodnoty okrem shiness, sú definované ako vektory troch čísel, v prípade farieb to slúžia na uloženie RGB zložiek, v prípadne `spotLightPosition` na uloženie koordinátov x, y, z a shiness je definovaný ako jedno číslo. Vo vertex shaderi sa získajú jednotlivé vertexy, spoja sa do trojuholníkov a prebehne rasterizácia, teda priradenie jednotlivých vertexov na pixly na obrazovke, konkrétny výpočet pre Phong:

```
void main(){
  vec4 vertPos4 = modelViewMatrix * vec4(position, 1.0);
  vertPos = vec3(vertPos4) / vertPos4.w;
  normalInterp = vec3(normalMatrix * normal);
```

```
  transformedLightPosition = vec3(modelViewMatrix *
  vec4(spotLightPosition, 1.0));
  gl_Position = projectionMatrix * vertPos4;
}
```

vertPos4 získame ako násobenie `position`, ktorá popisuje lokálne súradnice objektu v scéne, východzie sú x, y, z [0, 0, 0] a `modelViewMatrix`, čo je výsledok násobenia medzi `viewMatrix` a `modelMatrix`, kde `viewMatrix` definuje umiestnenie (polohu) a orientáciu kamery, zatiaľ čo `modelMatrix` definuje priestor objektu, ktorý sa budete vykresľovať. `normalInterp` je normálový vektor z priestoru modelu do priestoru pohľadu. `transformedLightPosition` je výpočet pozície zdroja bodového svetla ku kamere a objektu. `projectionMatrix` je projekčná matica, teda pohľad z kamery rešpektujúc jej nastavené zorné pole, blízku a ďalekú scénu. Z výstupu `gl_Position` dostávame pozíciu jednotlivých vertexov a ich priradenie pixlom na obrazovke monitora, z pohľadu ako to uvidí kamera.

Výstupy z vertex shaderu sú odovzdané fragment shaderu, ten popisuje na základe čoho vypočíta grafická karta farbu každého pixelu na Phongovom materiáli a teda, aby priradil farbu `gl_FragColor` každému pixelu.

```
void main() {
    vec3 N = normalize(normalInterp);
    vec3 L = normalize(transformedLightPosition-vertPos);
    float lambertian = max(dot(N, L), 0.0);
    float specular = 0.0;
    if(lambertian > 0.0) {
        vec3 R = reflect(-L, N);
  vec3 V = normalize(-vertPos);
  float specAngle = max(dot(R, V), 0.0);
  specular = pow(specAngle, shininess);
  }
    gl_FragColor = vec4(
        Ka * ambientLightColor +
        Kd * lambertian * spotLightColor+
        Ks * specular * spotLightColor, 1.0
    );
}
```

Shader najprv vypočíta normalizovanú normálu povrchu $\vec{n}$. Ďalej normalizovaný vektor smeru svetla $\vec{l}$ medzi transformovanou polohou bodového svetla a polohou vrcholu.

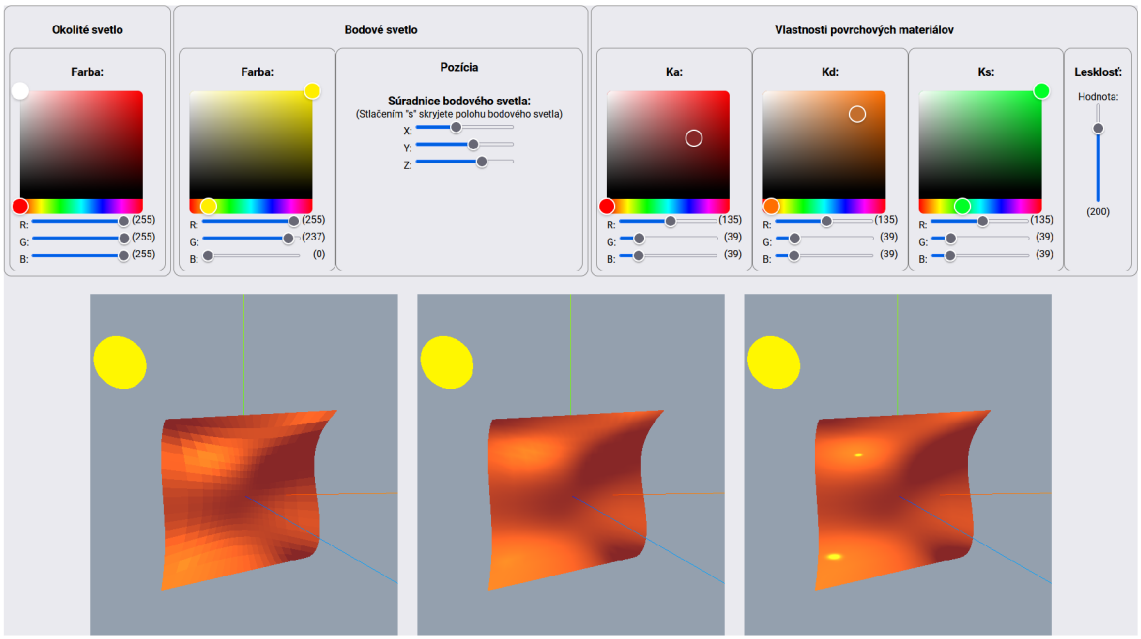
Normalizácia zabezpečí, aby tieto vektory mali dĺžku 1, čo je dôležité pre výpočty z dôvodu ich konzistentnosti, keďže výsledok vektorového súčinu potom vráti hodnoty od -1 po 1, čo sa následne interpretuje kosínus uhlu medzi nimi. Hodnota -1 znamená, že vektory smerujú opačnými smermi, hodnota 0 znamená, že uhly sú na seba ortogonálne (zvierajú pravý uhol) a 1 znamená, že vektory smerujú rovnakým smerom. Túto operáciu je možné spraviť aj medzi dvoma nenormalizovanými vektormi, avšak výpočet je potom náročnejší a pri zaokrúhľovaní môže vzniknúť chyba, teda posledným dôvodom je aj rýchlosť výpočtu [22].

Potom použije lambertov kosínusový zákon na výpočet lambertovského člena, ktorý predstavuje difúznu odrazivosť materiálu. Ak je lambertovský člen väčší ako nula, shader vypočíta vektor odrazeného svetla $\vec{r}$ a vektor ku kamere $\vec{v}$. Následne `specAngle` ako bodový súčin odrazeného svetla R a vektora ku kamere $\vec{v}$. Ak je uhol medzi oboma vektormi väčší ako 90 stupňov, výsledok bodového súčinu bude v skutočnosti záporný a dostaneme zápornú zrkadlovú zložku. Z tohto dôvodu používame funkciu `max`, ktorá vracia najvyšší z oboch svojich parametrov, aby sme sa uistili, že zrkadlovú zložka (a teda ani farby) nikdy nebudú záporné. Potom sa vypočíta hodnota zrkadlového odrazu ako mocnina zrkadlového uhlu na hodnotu lesklosti. Nakoniec shader vypočíta konečnú farbu pixelu súčtom okolitej, difúznej a zrkadlovej zložky farby. Pri pohľade na výpočet, ktorý `gl_FragColor` robí, je vidieť zhoda s Phongovou rovnicou pre výpočet 4.5. Tým sa určí výsledná farba fragmentu a priradí premennej `gl_FragColor`, čo je zabudovaná premenná, ktorá predstavuje konečnú farbu pixelu.

5.4 Ilustrácia rozdielu medzi tieňovaním konštantným, Gouraudovým a Phongovým

Aplet ilustruje tri paralelne bežiace scény, kde každá zdieľa model, polohu a orientáciu kamery ako aj prijímané parametre z web stránky. Názorne sú tým ilustrované rozdiely v reálnom čase a za rovnakých podmienok na obrázku 5.6. Prvé z ľava ilustrované konštantné tieňovanie, druhé je tieňovanie Gouraudovo a tretím je Phongovo tieňovanie. Aplet je dostupný na prehliadanie cez odkaz: `https://priscak.sk/tretia-uloha`.

Prvé z ľava je konštantné tieňovanie viď 4.4.2. To najjednoduchšie z modelov tieňovania. Každý vykresľovaný polygón má jeden normálový vektor. Tieňovanie celého polygónu je konštantné na celom povrchu polygónu a bude mu priradená jedna farba vrámci neho. Pri malom počte polygónov to so zakrivením povrchu dáva sekaný vzhľad. Čo ešte zhoršuje vnímanie a amplifikuje prechody vrámci konštantného tieňovanie je Machov efekt [23], ten popisuje že jas je okom vnímaný inak v strede



Obr. 5.6: Rozdiely odrazu svetla v rôznych tieňovacích modeloch.

polygónu než na okraji polygónu s rôznym jasom. Na hranici svetlej a tmavej farby sa svetlá farba javí sa svetlá farba javí svetlejšie a tmavá farba tmavšie.

5.4.1 Programové spracovanie

Z hľadiska programového je to vytvorené pomocou troch `canvas_framov`, ktoré má v sebe šablóna HTML a potom sa doň vykreslia tri paralelne bežiace aplety.

```
const containers: HTMLElement[] = [
  document.getElementById("canvas_frame"),
  document.getElementById("canvas_frame2"),
  document.getElementById("canvas_frame3")
];
containers[0].appendChild(renderers[0].domElement);
containers[1].appendChild(renderers[1].domElement);
containers[2].appendChild(renderers[2].domElement);
```

Parametre predávané sú zhodné a popísane pri Phongovom aplete 5.3.1. Rozdiel je iba v definícií parametru `color` ako `flat out vec4 color;`, to znamená, že sa priradí len jedna hodnota celému trojuholníku. Z hľadiska kódu väčšina výpočtu prebieha vo vertex shaderi:

```
vec3 transformedLightPosition = vec3(modelViewMatrix *
                                vec4(spotLightPosition, 1.0));
vec4 vertPos4 = modelViewMatrix * vec4(position, 1.0);
```

```
vertPos = vec3(vertPos4) / vertPos4.w;
normalInterp = vec3(normalMatrix * normal);
gl_Position = projectionMatrix * vertPos4;

vec3 N = normalize(normalInterp);
vec3 L = normalize(transformedLightPosition - vertPos);

float lambertian = max(dot(N, L), 0.0);
float specular = 0.0;

if(lambertian > 0.0) {
vec3 R = reflect(-L, N);
vec3 V = normalize(-vertPos);

float specAngle = max(dot(R, V), 0.0);
specular = pow(specAngle, shininess);
}

color = vec4(
Ka * ambientLightColor +
Kd * lambertian * spotLightColor +
Ks * specular * spotLightColor, 1.0
);
```

Prvý úsek kódu vypočíta tranformáciu polohy svetla a polohy vrcholu zo zdroja smerom ku kamere. Potom sa vypočíta normála na jednom z vrcholov. `NormalMatrix` je transpozíciou inverznej matice modelu a pohľadu, ktorá správne transformuje normálu z priestoru modelu do priestoru kamery. Normalizujú sa vektory: normalizuje normálové vektory $\vec{n}$ a vektory smeru svetla $\vec{l}$.

Následne lambertovská odrazivosť, čo predstavuje difúznu časť odrazu svetla od materiálu do všetkých smerov vektovovým súčinom normálových vektorov $\vec{n}$ a vektorov smeru svetla $\vec{l}$. Funkcia `max()` zabezpečí, že ak je bodový súčin menší ako 0, nastaví sa `lambertian` na hodnotu 0, pretože také svetlo je na druhej strane povrchu.

Ak tento člen je nenulový, kód potom vypočíta zrkadlovú časť osvetlenia, ktorá predstavuje lesklé odlesky. Najprv vypočíta smer odrazu svetla $\vec{r}$, potom smer pohľadu do kamery $\vec{v}$ a nakoniec uhol medzi nimi `specAngle`.

Zrkadlová odrazivosť sa potom vypočíta ako tento uhol zvýšený na mocninu „lesklosti“ povrchu shiness označovaný ako exponent p v rovnici 4.5.

Nakoniec sa vypočíta farba pixelu ako súčet ambientnej, difúznej a zrkadlovej zložky 4.1, podľa vzorca 4.1 resp. 4.5. Okolité svetlo predstavuje svetlo, ktoré je rozptýlené natoľko, že prichádza zo všetkých smerov rovnako, difúzne svetlo predstavuje priame svetlo rozptýlené vo všetkých smeroch a zrkadlové svetlo predstavuje priame svetlo odrážajúce sa v určitom smere. Každá časť je pritom ovplyvnená materiálovými vlastnosťami popísaných v parametroch 5.3.1. Následne výsledná farba `color`, je odovzdaná fragment shaderu na priradenie farby, farba je definovaná ako `flat`, takže grafická karta, ako aj samotný fragment shader vie, že má priradiť rovnakú farbu každému pixelu v trojuholníku, neprebieha už žiadna ďalšia interpolácia.

```
flat in vec4 color;
void main() {
    gl_FragColor = color;
}
```

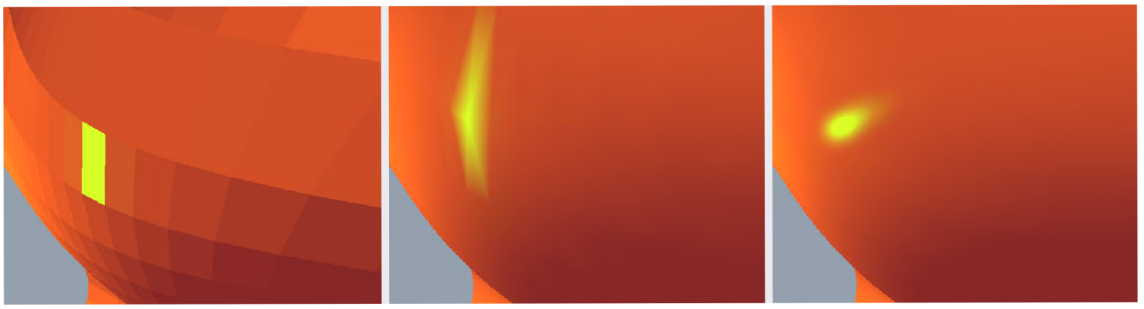
Gouraud 4.4.3 priemeruje farby vertexov v trojuholníku. Farba sa vypočíta na základe normály vrcholu pomocou vertex shadera a potom sa spriemeruje medzi fragmentmi jedného trojuholníka na základe vzdialenosti od každého vrcholu trojuholníka, čím bližšie jednému vertexu, tým väčšia dominancia farby daného vertexu.

Výpočet vo vertex shaderi pre Gouraudov tieňovací model prebieha úplne rovnako ako pri konštantnom tieňovaní. Sú mu odovzdané rovnaké parametre, až na definíciu `color` ako `varying vec4 color;`, tá je vypočítaná vo vertexoch a potom váhovo spriemerovaná automaticky grafickou kartou a vo fragment shaderi pridelená po pixloch na základe vzdialenosti k vertexom:

```
varying vec4 color;
void main() {
    gl_FragColor = color;
}
```

Phong priemeruje 4.4.3 normály vertexov v trojuholníku. Farba sa vypočíta pomocou fragment shadera viď 62 na základe spriemerovania 3 vrcholových normál trojuholníka, podľa blízkosti pixelu k jednotlivým vrcholom, ktoré prešli z vertex shadera, ako bolo popísane v predchádzajúcej úlohe a následne podľa 4.5 vo fragment shaderi vypočítaná pre každý pixel osobitne. Grafická karta nemá definované jaký model tieňovania má použiť, to sa definuje tým ako sa napíše kód v shaderoch.

Najviditeľnejšie sú tieto rozdiely pri bližšom pohľade pri odraze svetla, ako ukazuje obrázok 5.7. Tento pohľad zároveň detailne poukazuje na výhody a nevýhody jednotlivých metód popísaných v 4.5.



Obr. 5.7: Detailnejšia ilustrácia rozdielov medzi tieňovacími modelmi.

Záver

Diplomová práca sa v teórii venovala 3D grafike, popísala operácie s objektami a možnosti implementácií v rôznych dostupných knižniciach. Z popísaných možností bola vybraná kombinácia Node.js, TypeScriptu a 3D knižnice Three.js. Táto knižnica kombinuje výhody intuitívneho programovania a nevyžaduje inštalácie softvérových závislostí pri vykresľovaní apletov do prehliadača.

Grafická knižnica Three.js bola vybraná, kvôli jej širokej podpore zo strany webových prehliadačov a výhody, že nepotrebuje žiadne inštalované závislosti od systému.

Aplety vytvorené pre túto prácu majú za úlohu slúžiť ako užitočné výukové zdroje, pretože umožňujú používateľom interaktívne skúmať rôzne modely tieňovania, ich parametre a sledovať, ako ovplyvňujú zobrazené objekty. Tieto aplety nielenže zlepšujú vedomosti, ale poskytujú používateľom priestor na skúmanie s rôznymi nastaveniami a materiálmi, čo pomáha používateľom lepšie pochopiť základné myšlienky.

Prvý aplet názorne dokáže vysvetliť a ilustrovať normálový povrch a normálový vektor vykreslený na 3D materiál a pomáha užívateľom pri vizualizácii a pochopení funkcie normál povrchu v počítačovej grafike. Ponúka interaktívny spôsob skúmania normál povrchu a pochopenia ich úloh pri výpočtoch osvetlenia a tieňovania.

Druhý aplet ukazuje techniku osvetlenia Phongového modelu aplikovanú na 3D predmet, je určený na zobrazenie a interakciu. Phongov model osvetlenia, zohľadňuje ambientné svetlo, difúzne a zrkadlové zložky svetla. Umožňuje nastavovať tieto parametre a tak interaktívne skúmať vplyv jednotlivých parametrov na výsledné vykresľovanie.

Tretí aplet vizualizuje a interaktívne ilustruje rôzne prístupy k tieňovaniu v počítačovej grafike. Porovnáva zároveň konštantné tieňovanie, Gouraudovo a Phongovo tieňovanie, kde umožňuje používateľom meniť faktory, ako je okolité svetlo, bodové svetlo, odrazovú farbu difúznej a zrkadlovej zložky a polohu bodového zdroja svetla. Takýto aplet poskytuje intuitívne poznanie toho, ako rôzne modely tieňovania vedú k rôznym vizuálnym výsledkom na 3D objektoch za rôznych svetelných podmienok. Je rýchlo viditeľné, že Phongovo tieňovanie zohľadňuje interakciu svetla s materiálovými vlastnosťami objektu, pričom je možné dosiahnuť realistické a hladké povrchy. Zobrazené je zároveň Gouraudovské tieňovanie, ktoré robí per-vertexovú interpoláciu farieb, výpočetne je menej náročné ako Phongovo tieňovanie, ale poskytuje výsledky estetickejšie ako v prípade konštantného tieňovania. Analýzou výhod a nevýhod jednotlivých modelov tieňovania sme objasnili faktory, ktoré vplývajú na samotné vykreslenie objektov a ktoré treba zohľadniť pri výbere vhodného prístupu k tieňovaniu pre rôzne aplikácie.

Literatúra

[1] ŽÁRA Jiří, Bedřich BENEŠ, Jiří SOCHOR a Petr FELKEL. *Moderní počítačová grafika*. 2., přepracované a rozšířené vydání. Brno: Computer Press, 2010. ISBN 80-251-0454-0.

[2] SHIRLEY, Peter, Michael ASHIKHMIN a Steve MARSCHNER. *Fundamentals of Computer Graphics*. 2. vyd. CRC Press, 2005. ISBN 9781439864319.

[3] GOLDMAN, Ronald. *An Integrated Introduction to Computer Graphics and Geometric Modeling*. Boca Raton: CRC Press, 2009. ISBN 978-1-4398-0334-9.

[4] CLAPHAM, Christopher a James NICHOLSON. *The concise Oxford dictionary of mathematics*. 5th ed. Oxford: Oxford University Press, 2014. Oxford paperbacks. ISBN 978-0-19-967959-1.

[5] PIEGL, Les a Wayne TILLER. *The NURBS book*. 2nd ed. Berlin: Springer-Verlag, 1997. ISBN 35-406-1545-8.

[6] Three.js.org. In: https://threejs.org/ [online], 10.05 2023 [cit. 2023-05-10]. Dostupné z: <`https://threejs.org/examples/webgl_geometry_nurbs.html`>

[7] GRIMM, Todd. *User's guide to rapid prototyping*. Dearborn, Mich.: Society Of Manufacturing Engineeers, 2004. ISBN 9780872636972.

[8] *3D Math Overview and 3D Graphics Foundations* [online]. Austin, TX: Multimedia Applications Division Freescale Semiconductor, Inc., 2010 [cit. 03.05.2023]. Dostupné z: <`https://www.nxp.com/docs/en/application-note/AN4132.pdf`>.

[9] GOMES, Jonas, Luiz VELHO a Mario COSTA SOUSA. *Computer graphics: theory and practice*. Boca Raton: CRC Press, 2012. ISBN 978-156-8815-800.

[10] HUGHES, John F. Computer graphics: principles and practice. 3rd ed. Upper Saddle River: Addison-Wesley, 2013. ISBN 978-0-321-39952-6.

[11] DUNN, Fletcher, and Ian PARBERRY. *3D Math Primer for Graphics and Game Development* 2010. (Jones & Bartlett). ISBN 9781568817231.

[12] WILSON, Kevin. *The Absolute Beginner's Guide to HTML and CSS the Absolute Beginner's Guide to HTML and CSS: A Step-by-Step Guide with Examples and Lab Exercises*. First edition. Berlin, Germany: APress, 2023. ISBN 9781484292495.

[13] FLANAGAN, David. *JavaScript: the definitive guide : master the world's most-used programming language*. Seventh edition. Beijing: O'Reilly, 2020. ISBN 978-149-1952-023.

[14] GOLDBERG, Josh. *Learning TypeScript : enhance your web development skills using type-safe JavaScript*. Sebastopol, CA: O'Reilly Media, Inc., 2022. ISBN 9781098110307.

[15] CASCIARIO, Mario a Luciano MAMMINO. *Node.js design patterns: get the best out of Node.js by mastering its most powerful components and patterns to create modular and scalable applications witch ease*. Second edition. Birmingham: Packt, [2016]. ISBN 978-1-78588-558-7.

[16] WebGL - OpenGL ES for the Web. *In: The Khronos Group*. [online] 19.7.2011 [cit. 17.03.2023]. Dostupné z: <`https://www.khronos.org/webgl/`>

[17] PARISI, Tony. *Programming 3D Applications with HTML5 and WebGL 3D Animation and Visualization for Web Pages*. Sebastopol, Ca O'reilly & Associates, 2014. ISBN 9781449362966.

[18] MRDOOB. Three.js. *In: GitHub [online]*. 17. 3. 2023 [cit. 17.03.2023]. Dostupné z: <`https://github.com/mrdoob/three.js/blob/master/LICENSE`>

[19] MCCARTHY, Lauren, Casey REAS a Ben FRY. *Getting Started with p5.js*. Maker Media, Inc., 2015. ISBN 9781457186738.

[20] ANDREUSSI, Francesco. *Realtime-Rendering with OpenGL The Graphics Pipeline* [online]. Bauhaus-Universität Weimar, 2019 [cit. 16.05.2023]. Dostupné z: <`https://www.uni-weimar.de/fileadmin/user/fak/medien/professuren/Computer_Graphics/CG_WS_19_20/Computer_Graphics/01_Introduction.pdf`>

[21] AKENINE-MÖLLER, Tomas, Eric HAINES, Naty HOFFMAN, Angelo PESCE, Michael IWANICKI a Sébastien HILLAIRE. *Real-time rendering*. Fourth edition. Boca Raton: CRC Press/Taylor & Francis Group, 2018. ISBN 978-1-138-62700-0.

[22] LANKHAM, Isaiah, Bruno NACHTERGAELE a Anne SCHILLING. *Linear algebra as an introduction to abstract mathematics*. New Jersey: World Scientific, [2016]. ISBN 978-981-4730-35-8.

[23] PANIKKATH, Ragesh a Deepa PANIKKATH. *Mach Band Sign: An Optical Illusion*. Baylor University Medical Center Proceedings [online]. 2017, 27(4), 364-365 [cit. 2023-05-15]. ISSN 0899-8280. Dostupné z: doi:10.1080/08998280.2014.11929161

Zoznam symbolov a skratiek

CGI	Computer Generated Imagery
NURBS	Non-Uniform Rational B-Spline
PCI	Peripheral Component Interconnec
CPU	Central Processing Unit
GPU	Graphics Processing Unit
PCI	Peripheral Component Interconnect
AGP	Accelerated Graphics Port
PCIe	Peripheral Component Interconnect express
STL	Stereolitografia
CAD	Computer-Aided Design
ASCII	American Standard Code for Information Interchange
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
NPM	Node Package Manager
YARN	Yet Another Resource Negotiator
API	Application Programming Interface
VBO	Vertex Buffer Objects
MVP	Model View Projection
RGB	Red, Green, Blue
LOD	Level of Detail

Obsah elektronickej prílohy

- `/` ..... Koreňový adresár priloženého archívu
  - `prva-uloha` ..... Prvý aplet
    - `./dist` ..... HTML stránka a aplet pripravený na spustenie
    - `./public` ..... 3D STL modely
    - `./src` ..... Zdrojový kód stránky, apletu a materiálu
      - `app.ts`
      - `custom_plane_helper.ts`
      - `index.html`
    - `package.json` ..... Konfigurácia Node.js projektu
    - `tsconfig.json` ..... Konfiguračný súbor pre kompilátor
    - `webpack.config.js` ..... Popis pre zabalenie apletu do bundle.js
  - `druha-uloha` ..... Druhý aplet
    - `./dist` ..... HTML stránka a aplet pripravený na spustenie
    - `./public` ..... 3D STL modely
    - `./src` ..... Zdrojový kód stránky, apletu a materiálu
      - `app.ts`
      - `custom_phong_material.ts`
      - `index.html`
    - `package.json` ..... Konfigurácia Node.js projektu
    - `tsconfig.json` ..... Konfiguračný súbor pre kompilátor
    - `webpack.config.js` ..... Popis pre zabalenie apletu do bundle.js
  - `tretia-uloha` ..... Tretí aplet
    - `./dist` ..... HTML stránka a aplet pripravený na spustenie
    - `./public` ..... 3D STL modely
    - `./src` ..... Zdrojový kód stránky, apletu a materiálov
      - `app.ts`
      - `custom_flat_material.ts`
      - `custom_gouraud_material.ts`
      - `custom_phong_material.ts`
      - `index.html`
    - `package.json` ..... Konfigurácia Node.js projektu
    - `tsconfig.json` ..... Konfiguračný súbor pre kompilátor
    - `webpack.config.js` ..... Popis pre zabalenie apletu do bundle.js