



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**  
BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**  
FACULTY OF INFORMATION TECHNOLOGY

**ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ**  
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

**VIZUALIZACE ROZSÁHLÝCH GRAFICKÝCH DAT**  
VISUALIZATION OF LARGE GRAPHIC DATA

**BAKALÁŘSKÁ PRÁCE**  
BACHELOR'S THESIS

**AUTOR PRÁCE**  
AUTHOR

**PETER ĎURICA**

**VEDOUCÍ PRÁCE**  
SUPERVISOR

**Ing. MICHAL VLNAS**

**BRNO 2023**

## Zadání bakalářské práce



146302

Ústav: Ústav počítačové grafiky a multimédií (UPGM)  
Student: **Ďurica Peter**  
Program: Informační technologie  
Specializace: Informační technologie  
Název: **Vizualizace rozsáhlých grafických dat**  
Kategorie: Počítačová grafika  
Akademický rok: 2022/23

### Zadání:

1. Nastudujte možnosti vizualizace rozsáhlých dat, zejména z oblasti v počítačové grafiky (světelné cesty, mračna bodů, heat mapy apod.)
2. Navrhněte vhodné řešení pro vizualizaci vybraných typů dat formou vlastní aplikace nebo zásuvného modulu.
3. Implementuje navržené řešení.
4. Otestuje implementované řešení a jeho využitelnost v praxi.
5. Vytvořte demonstrační video.

### Literatura:

- Kirk, Andy. Data Visualisation. 2nd ed., SAGE Publications, 2019.
- Dougherty, Jack and Ilyankou, Ilya. Hands-On Data Visualization: Interactive Storytelling From Spreadsheets to Code. 1st ed., 2021.

Při obhajobě semestrální části projektu je požadováno:

Body 1, 2 a experimenty vedoucí k bodu 3.

Podrobné závazné pokyny pro vypracování práce viz <https://www.fit.vut.cz/study/theses/>

Vedoucí práce: **Vlnas Michal, Ing.**

Vedoucí ústavu: Černocký Jan, prof. Dr. Ing.

Datum zadání: 1.11.2022

Termín pro odevzdání: 10.5.2023

Datum schválení: 31.10.2022

## **Abstrakt**

Táto práca rieši vizualizáciu konkrétnych a špecializovaných rozsiahlych grafických dát používaných vo vedeckej oblasti typu mračna bodov, tepelných máp na objektoch alebo vizualizácie harmonických funkcií na pologuli. V práci sa využíva jazyk Python na vykreslenie scén a jazyk GLSL na implementáciu tepelných máp a harmonických funkcií. Výsledný program umožňuje vykreslovanie scén z programu Mitsuba renderer s tisícami bodov v reálnom čase. Tepelné mapy umožňujú dva rozdielne spôsoby vykreslovania a to na každý pixel v obraze, alebo na vrcholy objektov. Harmonické funkcie umožňujú rôzne úrovne kvality pre kontrolu výkonu. Prínosom práce je zjednodušenú analýza scény, kvôli vizualizácii rozsiahlych dát konkrétneho typu v reálnom čase.

## **Abstract**

This thesis solves visualization of large, specific and specialized graphic data used in science field like point clouds, heat-map on objects in scene or harmonic functions on hemisphere. The work uses the Python language for rendering scenes and the GLSL language for the implementation of heat maps and harmonic functions. The final program can render scenes from Mitsuba renderer with thousands of points in real-time. Heatmaps have 2 types of rendering and that are per pixel and per vertex on objects. Harmonic functions have different quality options for performance control. The main benefit is easier scene analysis thanks to visualization of large data of specific type in real time.

## **Kľúčové slová**

vizualizácia, grafické dáta, tepelné mapy, mračná bodov, Mitsuba renderer, Wavefront .obj, Python, OpenGL, GLSL, hemisférické harmonické funkcie

## **Keywords**

visualization, graphical data, heatmaps, point clouds, Mitsuba renderer, Wavefront .obj, Python, OpenGL, GLSL, hemispheric harmonic functions

## **Citácia**

ĎURICA, Peter. *Vizualizace rozsáhlých grafických dat*. Brno, 2023. Bakalářská práce. Vysočné učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Vlnas

# Vizualizace rozsáhlých grafických dat

## Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Michala Vlnasa. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....  
Peter Ďurica  
9. mája 2023

## Podčakovanie

Chcel by som sa podčakovať vedúcemu práce Ing. Michalovi Vlnasovi za pomoc a rady pri tvorbe práce a za ústretovosť v posledných týždňoch pred odovzdaním. Ďalej by som sa chcel podčakovať rodine a priateľom za podporu počas celého roka.

# Obsah

<b>1</b>	<b>Úvod</b>	<b>2</b>
<b>2</b>	<b>Zobrazovanie špecifických dátových typov na GPU v reálnom čase</b>	<b>3</b>
2.1	OpenGL . . . . .	3
2.2	Postup vykreslovania objektov na GPU a Shadery . . . . .	4
2.3	Kamerová projekcia . . . . .	5
2.4	Mračno bodov . . . . .	7
2.5	Tepelná mapa a akceleračné štruktúry . . . . .	8
2.6	Hemisférické harmonické funkcie . . . . .	9
2.7	XML, Mitsuba renderer a Wavefront .obj . . . . .	12
<b>3</b>	<b>Návrh</b>	<b>14</b>
3.1	Dáta na vykreslenie . . . . .	14
3.2	Operácie na vykreslenie . . . . .	16
3.3	Užívateľské rozhranie . . . . .	18
<b>4</b>	<b>Implementácia</b>	<b>20</b>
4.1	Načítanie dát . . . . .	20
4.2	Nastavenie kamery a pohyb v priestore . . . . .	23
4.3	Reprezentácia dát na GPU . . . . .	26
4.4	Render Loop . . . . .	27
4.5	Inštančné renderovanie bodov a HSH funkcií . . . . .	28
4.6	Renderovanie objektov a tepelnej mapy . . . . .	31
4.7	Ovládanie / GUI . . . . .	34
4.8	Testovanie . . . . .	37
<b>5</b>	<b>Záver</b>	<b>41</b>
	<b>Literatúra</b>	<b>42</b>
	<b>A Obsah priloženého média</b>	<b>44</b>

# Kapitola 1

## Úvod

Táto práca slúži k zobrazovaniu jedinečných typov grafických dát v trojrozmernom priestore. Informácie o týchto dátach sú ukladané neštandardným spôsobom a preto nie sú jednoduché na zobrazenie klasickými metódami. Práca sa zaoberá spôsobmi na analýzu týchto vysoko konkrétnych typov údajov a na ich správne zobrazenie. Pri zobrazovaní používa matematické operácie medzi typmi dát na ukázanie závislosti medzi objektami v priestore.

Vizualizácia grafických dát je dôležitá na správnu analýzu trojrozmerných grafických scén. Síce sa nejedná o realistické zobrazenie scény ale program určený na vizualizáciu dát môže zjednodušiť analýzu konkrétnej scény zobrazením dát vo formáte lepšie určenom na identifikáciu informácií o scéne.

V tejto dobe neexistuje voľne dostupná aplikácia, alebo jednoduchý spôsob ktorý by umožňoval čítanie grafických dát uložených vo veľmi konkrétnych a neštandardných formátoch a ich následnú vizualizáciu.

Preto bolo cieľom tejto práce nájsť spôsob ako zobraziť tieto dátá, hlavne mračná bodov, uložené vysoko konkrétnym spôsobom nad scénou zo vstupného súboru pre Mitsuba renderer. Následným cieľom bolo zobraziť teplotnú mapu na objektoch v scéne podľa prítomnosti a počtu bodov z mračna v blízkosti objektu, prípadne priamo na bodoch vizualizácia harmonických funkcií pomocou pologule.

V následnej kapitole je vysvetlený spôsob kreslenia objektov na grafickej karte a spôsoby kamerovej projekcie pre zobrazenie scény v trojdimenzionálnom priestore spoločne s typmi grafických dát a operáciami nad nimi. V tretej kapitole sa rieši návrh čítania vstupných dát, matematických operácií nad týmito dátami a návrh grafického rozhrania pre interakciu s aplikáciou. Samotná konkrétna implementácia týchto návrhov a následné testovanie sa rieši v štvrtej kapitole.

## Kapitola 2

# Zobrazovanie špecifických dátových typov na GPU v reálnom čase

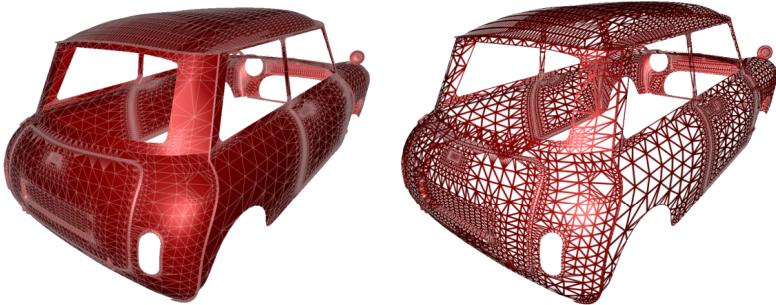
V tejto kapitole sa prejde cez detailný popis vykreslovania objektu použitím grafickej karty až k informáciám o konkrétnych dátových typoch a technikách zobrazovania. Tieto informácie nepredstavujú základnú vedomosť priemerného človeka a sú dôležité na plné pochopenie implementačných detailov aplikácie.

### 2.1 OpenGL

OpenGL je prostredie na vývoj aplikácie [16] (v angličtine skratka API), ktoré poskytuje softvérovú knižnicu na prístup k možnostiam grafického hardvéru. Verzia OpenGL 4.3, ktorá je použitá v aplikácii o ktorú sa táto práca zaoberá, obsahuje 500 rozdielnych príkazov, ktoré sa používajú na špecifikáciu objektov, obrázkov a operácií potrebných na vytvorenie interaktívnych počítačových aplikácií založených na počítačovej grafike.

Pri programovaní v jazyku Python sa používa sa knižnica PyOpenGL [12], ktorá spája už existujúcu knižnicu s týmto programovacím jazykom. Rozdiel medzi oboma prostrediami nie je takmer žiadny, keďže takmer každý príkaz v jednom API má podobný v tom druhom. Nevýhodou PyOpenGL môže byť pomalsia rýchlosť kvôli ohybným dátovým typom v jazyku Python. Inak nemá používanie tejto knižnice takmer žiadny rozdielny efekt ako klasické OpenGL.

Prvá verzia OpenGL [16] vyšla v júli roku 1994 firmou Silicon Graphics Computer Systems. OpenGL bolo navrhnuté ako zjednodušené rozhranie nezávislé na hardvéri a preto ľahko implementovateľné na rôznych systémoch. Avšak práve kvôli tomu neumožňuje žiadnu interakciu s oknom v ktorom sú zobrazované grafické prvky a ani neumožňuje užívateľské rozhranie alebo vlastnosti na spracovanie užívateľského vstupu. Trojdimenzionálne objekty sa v OpenGL musia skladať zo sád geometrických primitívov ako body, čiary alebo trojuholníky. Po vykreslení všetkých primitívov objektu sa nám zdá objekt ako celistvý aj keď sa v skutočnosti skladá z množstva menších častí. Obrázok 2.1 zobrazuje objekt a trojuholníky z ktorých sa skladá.



Obr. 2.1: Grafický objekt so zvýraznenými primitívmi [3].

## 2.2 Postup vykreslovania objektov na GPU a Shadery

Rasterizácia [10] je proces mapovania geometrie definovanej vo vektorovom formáte na dvojdimenzionálnu mriežku (obrazovku). Samotný postup tohto procesu je vcelku jednoduchý a je ho možné rozdeliť do sekvencie nasledovných krokov:

1. Koordinačné vektorové vstupné geometrické vrcholy sa vynásobia transformačnou maticou na určenie pozície vrcholu na obrazovke.
2. Trojuholníky zložené z týchto vrcholov, ktoré sa nachádzajú na obrazovke, sa zapĺnia fragmentami (pixlami) obrazovky.
3. Výsledné fragmenty sa zoradia podľa pozícii na obrazovke a hĺbky. Ak sa dva fragmenty nachádzajú na rovnakom mieste, tak sa zahodí fragment s väčšou hĺbkou od pohľadu kamery.
4. Každému zvyšnému fragmentu je pridelená korešpondenčná farba.

Softvérová aplikácia [10] získava prístup ku grafickému hardvéru a teda k tomuto procesu pomocou špecializovaných knižníc ako napríklad OpenGL. Pomocou takejto knižnice vieme identifikovať a inicializovať konkrétny grafický hardvér, vytvoriť dátové úložiská s geometrickými informáciami a pripojiť tieto úložiská ako vstupné hodnoty pre rasterizačný proces.

Grafický hardvér spoločne s aplikačnými knižnicami vytvárajú vykreslovací reťazec [10] (pipeline). Proces, ktorý je vykonávaný v tomto reťazci sa volá renderovanie. Pozostáva z rôznych fáz, z ktorých každá vykonáva nejakú konkrétnu úlohu. Tieto fázy korešpondujú s jednotlivými krokmi v rasterizácii. Väčšina z týchto krokov sú programovateľné a preto si ich vie užívateľ upraviť podľa seba:

- Ovládač vstupu – Číta dátu z užívateľských úložísk a skladá ich do primitívov,
- Vertex shader – Programovateľný krok na operácie nad vrcholmi,
- Surface shader – Programovateľný krok, v ktorom sa pripravujú dátá pre tesalátor,
- Tesalátor – Počíta detailné geometrické povrchy,
- Domain shader – Programovateľný krok, v ktorom sa počíta pozícia vrcholov na základe výsledkov teselátora,
- Geometry shader – Programovateľný krok, ktorý vie generovať dodatočné vrcholy,

- Rasterizátor – Prevádzza vektorové informácie do rasterizovaného obrazu,
- Fragment shader – Programovateľný krok pre operácie nad fragmentami obrazu,
- Spájanie výstupu – Vygenerovanie finálnych farieb pre pixely na obrazovke.

Nie všetky kroky v tomto procese sú povinné pri vytváraní grafickej aplikácie. Zvyšná časť sekcie sa venuje shaderom ktoré boli využité vo finálnej aplikácii.

## Vertex shader

Vertex shader [1] je prvý krok pri spracovaní trojuholníkov skladajúcich sa z vrcholov objektu. Tieto vrcholy sa skladajú z pozície v objekte, prípadne jeho normálovej hodnoty, farby alebo pozícii na textúre. Avšak dátá o konkrétnych trojuholníkoch, nemá shader k dispozícii. Zaujímajú ho len prichádzajúce vrcholy.

Umožňuje upravovať, vytvárať alebo aj ignorovať hodnoty spojené s vrcholom ako jeho pozíciu, farbu alebo normálovú hodnotu. Vždy však musí preniesť pozíciu vrcholu v objekte do jednotného dvojrozmerného priestoru a túto hodnotu poslať ďalej do reťazca na rasterizáciu.

Na každom vrchole sa v grafike pracuje samostatne a paralelne, preto vypočítané hodnoty na jednom vrchole neovplyvňujú ostatné. Každá vypočítaná hodnota sa však interpoluje vrámci celého trojuholníka a nemusíme teda napríklad počítať farbu každého fragmentu v trojuholníku.

## Fragment shader

Vypočítané interpolované hodnoty z vertex shaderu sa stávajú vstupné hodnoty pre fragment shader [1]. Kedže sa nachádza fragment shader za rasterizátorom, každý fragment pozná svoju pozíciu na obrazovke. Úlohou fragment shadera býva zafarbiť daný pixel konkrétnou farbou.

Limitácia fragment shaderu je, že dokáže ovplyvňovať len fragment na jednej pozícii. Nevie čítať ani upravovať vstupy svojich susedných fragmentov.

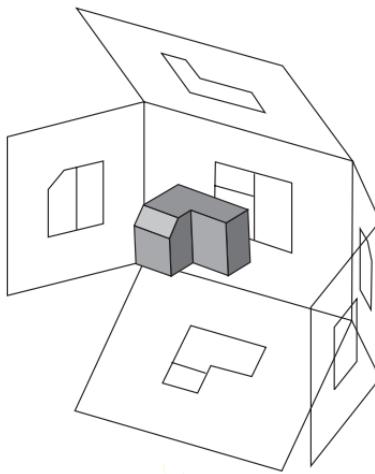
## 2.3 Kamerová projekcia

Kamerová projekcia [14] slúži k projekcii objektov v scéne do dvojrozmernej roviny, podobne ako fotoaparát alebo kamera v reálnom svete. Medzi najviac využívané typy projekcií v počítačovej grafike sa radí ortogonálna a perspektívna projekcia.

### Ortogonalna projekcia

Ak sa použije ortogonálna projekcia [1], tak si všetky objekty uchovávajú svoju veľkosť nezávisle od ich vzdialenosť od kamery. Matica 2.1 ukazuje jednoduchú ortogonálnu maticu, ktorá necháva x- a y-hodnoty nezmenené zatiaľ čo nastavuje hodnotu z na nulu. Premieta všetky body objektu na rovinu  $z = 0$ .

$$O = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.1)$$



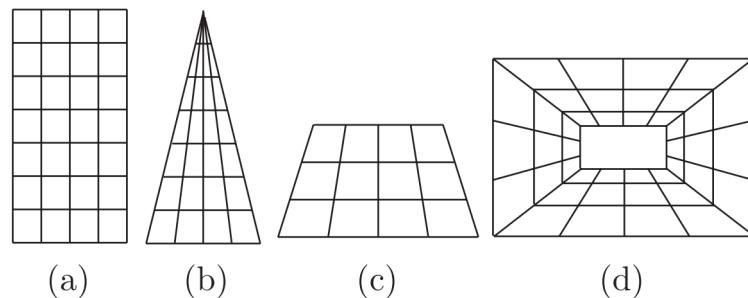
Obr. 2.2: Objekt vykreslovaný ortogonálnou projekciou [14]

Obrázok 2.2 ukazuje objekt vykreslený ortogonálnou projekciou z každej roviny obkolesenej kocky.

Projekcia týchto objektov [1] je rozdielna od projekcie v reálnom svete, keďže v skutočnosti sa nám objekty ktoré sú vzdialenejšie od kamery zdajú byť menšie. Pre projekciu podobnú skutočnému svetu je potrebné použiť perspektívnu projekciu.

### Perspektívna projekcia

Komplexnejšia transformácia ako ortogonálna projekcia je perspektívna projekcia [1], ktorá sa pravidelne používa pri väčšine aplikácií počítačovej grafiky. Paralelné čiary v tejto projekcii nemusia zostať paralelné a môžu konvergovať do jedného bodu. V obrázku 2.3 je vidno v časti (a) plochú rovinu a v časti (b) tú istú rovinu ale konvergujú k jednému bodu, čo vytvára ilúziu hĺbky. Následne je v časti (c) udržaný efekt hĺbky aj bez konvergentného bodu a v časti (d) je tento efekt použitý na vytvorenie kvázi miestnosti.



Obr. 2.3: Ukážka perspektívnej projekcie [14]

Perspektívna matica [1] na zobrazenie scény je zobrazená v matici 2.2. Skratky hodnôt znamenajú:

- l – Ľavá hodnota roviny obrazovky,
- r – Pravá hodnota roviny obrazovky,
- b – Spodná hodnota roviny obrazovky,
- t – Vrchná hodnota roviny obrazovky,
- n – Hodnota hĺbky najbližšie sa zobrazujúcich objektov,
- f – Hodnota hĺbky najďalej sa zobrazujúcich objektov.

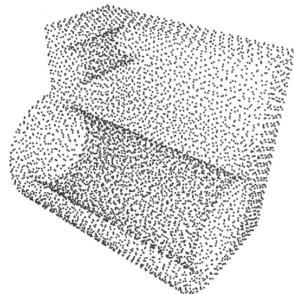
$$P = \begin{pmatrix} \frac{2n}{r-l} & 0 & -\frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (2.2)$$

Po vynásobení maticou 2.2 [1] vznikne bod  $q = (q_x, q_y, q_z, q_w)$ , takže na zistenie reálnej hodnoty bodu je treba ešte spočítať nový bod  $p = (q_x/q_w, q_y/q_w, q_z/q_w, 1)$ . Tieto hodnoty už sú správne.

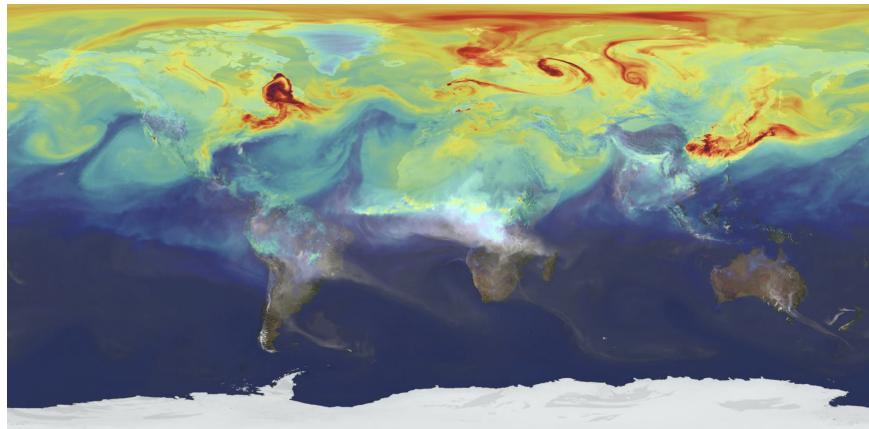
## 2.4 Mračno bodov

Mračno bodov je sada bodov [9], kde každý bod reprezentuje nejaké miesto v trojdimenziónom priestore. Môže sa skladať z miliónov jedinečných políčok, kde každé obsahuje minimálne polohu bodu. Na samotné vykreslenie bodov to stačí, ale body často obsahujú informácie naviac ako napríklad farba bodu alebo normálový vektor. Ak je týchto bodov v mračne dostatok vieme ich využiť na vykreslovanie objektov v scéne. Takto vykreslený objekt je viditeľný na obrázku 2.4. Je to veľmi účinná metóda ako uchovať objekt z reálneho sveta v digitálnej podobe.

Na zaznamenanie sa môže použiť laserový skener, fotogrametria alebo sa dá vygenerovať softvérom nad grafickou scénou. Laserový skener býva kvalitnejšia metóda ako fotogrametria pretože laser dokáže presne identifikovať pozíciu bodu v priestore a vie to spraviť s extrémnou presnosťou.



Obr. 2.4: Objekt zobrazený ako mračno bodov [11]



Obr. 2.5: Hodnoty teploty namerané vo svete z dňa 5. apríla 2019 [18]

## 2.5 Tepelná mapa a akceleračné štruktúry

Tepelná mapa je dvojrozmerná grafická reprezentácia dát [2], kde sú hodnoty premenných zobrazované ako farby. Príklad tepelnej mapy sa nachádza v obrázku 2.5. Tepelné mapy sú dôležité kvôli dvom dôvodom. Prvým dôvodom je to, že intuitívne farby použité vo farebnej škále inšpirované teplotou, znižujú čas potrebný na pochopenie mapy. Je známe, že žltá je teplejšia ako zelená, oranžová je teplejšia ako žltá a červená je horúca. Následne je jednoduché pochopiť že „teplota“ farby je priamo úmerná hodnote v bode. Druhý dôvod je, že tepelné mapy zobrazujú hodnotu priamo nad meraným priestorom.

Kvôli tomuto je tepelná mapa veľmi účinná pri vizualizácii meraných hodnôt pri porovnaní s klasickým zobrazením numerických hodnôt. Tepelné mapy nám umožňujú si rýchlo všimnúť prípadné vzory alebo sklonky v dátach.

Pri práci s obrovským obsahom dát, hlavne pri vykreslovaní v reálnom čase, je potrebné zaručiť rýchly prístup k dátam. Na tieto účely existujú akceleračné štruktúry.

### k-D stromy

k-D strom je binárny vyhľadávací strom[5], kde dáta v každom uzle majú k-dimenzióvnú pozíciu v priestore. Nelistový uzol v k-D strome rozdeľuje priestor do dvoch častí, nazývané polo-priestory. Body naľavo od tohto priestoru sú reprezentované ľavým podstromom a body napravo zase pravým podstromom.

Pre generalizáciu sa dá očíslovať rovina pre každú jednu dimenziu stromu. Roviny sú teda očíslované  $0, 1, 2, \dots, (k-1)$ . Bod v hĺbke  $d$  je orientovaný podľa roviny  $A$  ktorá je vypočítaná  $A = d \bmod k$ . Ak je koreňový uzol orientovaný podľa roviny  $A$ , tak ľavý podstrom obsahuje všetky body, ktorých hodnota polohy na rovine  $A$  je menšia ako v koreňovom uzli. Podobne v pravom podstrome sú všetky body, ktorých hodnota je väčšia.

Pre ďalšiu akceleráciu sa dajú jednotlivé body ukladať do tried podľa klasifikačných algoritmov. Následne pri kalkuláciách stačí rátať len s bodmi v konkrétnych triedach.

### kNN algoritmus

kNN (k-Nearest Neighbor) je klasifikačný algoritmus [17] ktorý pracuje na princípe žiadnych predpokladov. Tento princíp znamená že algoritmus nerobí žiadne predpoklady ohľadom parametrov. Bod je klasifikovaný podľa najpočetnejšej triedy jeho susedov. Ak je  $k = 1$

tak bod len zdedí triedu najbližšieho suseda. Ak je  $k > 1$  tak sa zdedí trieda s najväčším zastúpením medzi susedmi. Trieda  $y$  sa dá predpovedať pomocou rovnice 2.3:

$$y = \arg \max_{(c)} \left[ \sum_{i=0}^K I_c(n_i) \right], \quad (2.3)$$

kde:

$$I_c(n) = 1, \text{ ak } n \in \text{v triede c.}$$

### kNN s váhou

kNN s váhou je variant kNN algoritmu. Najjednoduchší spôsob je spočítať dominantnú triedu v KNN. Problém môže nastať ak vzdialenosť najbližších susedov sú príliš rozdielne a napríklad susedia ktorý sú vzdialenejší prehlasujú bližšieho suseda. Kvôli tomuto problému sa vymyslel vážený KNN algoritmus.

Ak  $k > 1$  Každý sused má priradenú váhu, ktorá je využitá pri priradovaní triedy. Táto váha  $W_n$  môže značiť napríklad inverznú vzdialenosť suseda od bodu. Bod sa priradí do takej triedy, ktorej suma po zrátaní všetkých susedov bola najväčšia. Táto trieda  $y$  sa dá predpovedať pomocou funkcie 2.4.

$$y = \arg \max_{(c)} \left[ \sum_{i=0}^K I_c(n_i) W_{n_i} \right] \quad (2.4)$$

### kNN s váhou a predpokladom triedy

kNN s váhou a predpokladom triedy [17] používa pravdepodobnosť že bod bude patriť do niektornej z tried. Táto pravdepodobnosť sa používa pri klasifikovaní bodu. Ak je  $k > 1$  pravdepodobnosť sa počíta na každom susedskom bode.

Ak  $c_1$  je majoritná trieda a  $c_2$  je minoritná, tak sa pravdepodobnosť týchto tried dá vypočítať pomocou rovníc 2.5 a 2.6:

$$L_1 = \sum_{i=1}^n p(x_i | y_i = c_1), \quad (2.5)$$

$$L_2 = \sum_{i=1}^n p(x_i | y_i = c_2), \quad (2.6)$$

a trieda  $y$  sa dá predpovedať pomocou rovnice 2.7:

$$y = \arg \max_{(c)} \left[ p(x_i | y_i) \sum_{i=0}^K I_c(n_i) \right]. \quad (2.7)$$

## 2.6 Hemisférické harmonické funkcie

Sférické harmonické funkcie (SH) reprezentujú funkcie definované na guli [4]. V grafických aplikáciách sa používajú hlavne na reprezentáciu funkcií na odraz svetla od povrchu objektu (BRDF) alebo na environmentálne mapy. Pri použití oboch sa hodnota svetelného integrálu dá zapísat ako skalárny súčin vektorov. Naviac sa SH dajú využiť aj pre uloženie predom vypočítaných tieňov a dát vnútorných odrazov. Avšak tieto funkcie ako tieňovanie, vnútorné

odrazy alebo BRDF sú definované na pologuli, zatiaľ čo základné SH funkcie sú definované na celej guli. Preto je potrebných viacero koeficientov SH pre presnejšiu reprezentáciu.

Pre vyriešenie tohto problému sa môžu použiť hemisférické harmonické funkcie (HSH). Tieto hemisférické funkcie sú adaptáciou na verziu SH, ktorá sa spolieha na združené Legendreove polynómy. V HSH sa tieto polynómy posúvajú pre vytvorenie nového základu.

## Ortogonalne polynómy

Množina polynómov  $\{p_l(x)\}$  sú ortogonalne [4] v intervale  $[a, b]$  ak  $p_l(x)$  je polynom stupňa  $l$ , pre  $n, m \geq 0$ :

$$\int_a^b w(x)p_n(x)p_m(x)dx = \delta_{nm}c_n, \quad (2.8)$$

kde  $\delta_{nm}$  je 0 alebo 1 ak  $n \neq m$  alebo  $n = m$  a  $w(x)$  je pozitívna funkcia váhy. Ak  $c_n = 1$  tak sú polynómy ortonormálne.

## Posunuté polynómy

Posúvanie je lineárna transformácia [4]  $x$  do  $k_1x + k_2$ , pričom  $k_1 \neq 0$ . Ak sú polynómy  $\{p_l(x)\}$  ortogonalne na intervale  $[a, b]$  s funkciou váhy  $w(x)$ , tak polynómy  $\{p_l(k_1x + k_2)\}$  sú ortogonalne na intervale  $[\frac{a-k_2}{k_1}, \frac{b-k_2}{k_1}]$  s funkciou váhy  $w(k_1x + k_2)$ . Ak  $\{p_l(x)\}$  sú ortonormálne tak aj  $\{(\pm k_1)^l \sqrt{|k_1|} p_l(k_1x + k_2)\}$  sú ortonormálne. Polynómy  $\{p_l(k_1x + k_2)\}$  sú posunuté verzie  $\{p_l(x)\}$ .

## Asociované Legendreove polynómy

Asociované Legendreove polynómy (ALP) [4],  $\{P_l^m(x)\}$  pričom  $m \in \{0, \dots, l\}$ , sú množiny ortogonalných polynómov na intervale  $[-1, +1]$  s ohľadom na  $l$  s funkciou váhy  $w(x) = 1$ :

$$\int_{-1}^1 P_l^m(x)P_{l'}^{m'}(x)dx = \frac{2(m+l)!}{(2l+l)(l-m)!} \delta_{ll'}. \quad (2.9)$$

Nahradením argumentu  $x$  s  $\cos \theta$ , sa získa množina funkcií  $\{P_l^m(\cos \theta)\}$  definovaných cez uhlový interval  $[0, \pi]$ . Hodnoty SH funkcií, ortogonalne cez  $[0, \pi] \times [0, 2\pi]$ , sú zostavené z  $P_l^m(\cos \theta)$  ako:

$$Y_l^m(\theta, \phi) = \begin{cases} \sqrt{2}K_l^m \cos(m\phi)P_l^m(\cos \theta) & \text{ak } m > 0 \\ \sqrt{2}K_l^m \sin(-m\phi)P_l^{|m|}(\cos \theta) & \text{ak } m < 0 \\ K_l^0 P_l^0(\cos \theta) & \text{ak } m = 0, \end{cases} \quad (2.10)$$

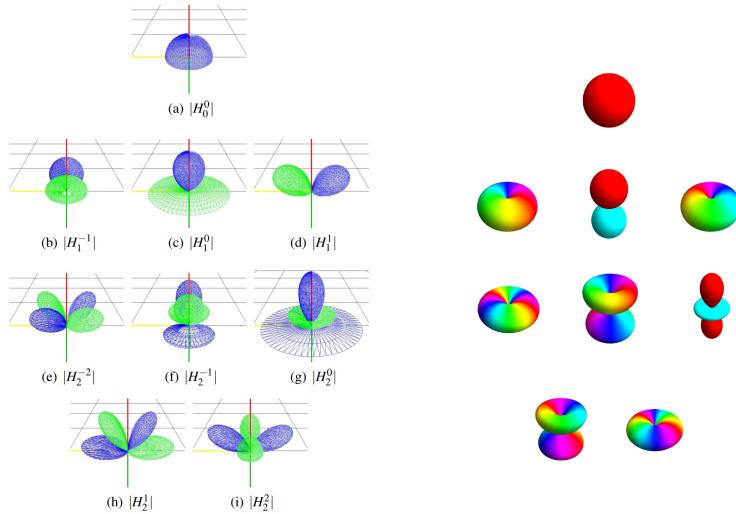
kde  $K_l^m$  je normalizačná hodnota:

$$K_l^m = \sqrt{\frac{(2l+1)(l-|m|)!}{4\pi((l+|m|)!)}}. \quad (2.11)$$

## Posunuté ALP

Použitím lineárnej transformácie  $x$  do  $2x - 1$  sa získajú posunuté ALP [4] v intervale  $x \in [0, 1]$ :

$$\tilde{P}_l^m(x) = P_l^m(2x - 1). \quad (2.12)$$



Obr. 2.6: Grafy funkcie  $H_l^m$  a  $Y_l^m$  pre  $l$  od 0 do 2[4][15]

Dôvod tohto posunu je, že po nahradení argumentu  $x$  s  $\cos \theta$ , sa získá funkcia  $\{\tilde{P}_l^m(\cos \theta)\}$  ktorá je definovaná na intervale  $\theta \in [0, \frac{\pi}{2}]$ , čo je rozsah polárneho uhla hemisféry:

$$\tilde{P}_l^m(\cos \theta) = P_l^m(2 \cos \theta - 1) \text{ pričom } \theta \in [0, \frac{\pi}{2}]. \quad (2.13)$$

Posunuté ALP zostávajú ortogonálne, ale zmení sa normalizácia podľa definície v rovnici 2.13. Funkcia 2.14 ukazuje ortogonálny vzťah s ohľadom na  $l$ , pričom hodnota funkcie váhy je 1:

$$\begin{aligned} \int_0^1 \tilde{P}_l^m(x) \tilde{P}_{l'}^m(x) dx &= \int_0^1 P_l^m(2x-1) P_{l'}^m(2x-1) dx \\ &= \frac{(m+l)!}{(2l+1)(l-m)!} \delta_{ll'}. \end{aligned} \quad (2.14)$$

### Hemisférické funkcie z posunutých ALP

Rovnakým spôsobom ako sú SH funkcie zostavené z ALP, sa zostaví základ aj pre HSH [4]  $\{H_l^m(\theta, \pi)\}$  z posunutých ALP:

$$H_l^m(\theta, \phi) = \begin{cases} \sqrt{2} \tilde{K}_l^m \cos(m\phi) \tilde{P}_l^m(\cos \theta) & \text{ak } m > 0 \\ \sqrt{2} \tilde{K}_l^m \sin(-m\phi) \tilde{P}_l^{|m|}(\cos \theta) & \text{ak } m < 0 \\ \tilde{K}_l^0 \tilde{P}_l^0(\cos \theta) & \text{ak } m = 0, \end{cases} \quad (2.15)$$

s následnou normalizačnou hodnotou:

$$\tilde{K}_l^m = \sqrt{\frac{(2l+1)(l-|m|)!}{2\pi((l+|m|)!)}}. \quad (2.16)$$

HSH sú ortogonálne v rozsahu  $[0, \frac{\pi}{2}] \times [0, 2\pi]$  s ohľadom na  $l$  a aj  $m$ . Troj-dimenziorné grafy prvých pár HSH funkcií sú v obrázku 2.6.

## 2.7 XML, Mitsuba renderer a Wavefront .obj

Informácie o grafickej scéne môžu byť uložené v rôznych formátoch. Jedným zo spôsobov, ktorý využíva aj Mitsuba renderer, je uchovávať si informácie o scéne v súboroch typu XML a informácie o objektoch v scéne v súboroch typu .obj.

### XML

XML (Extensive Markup Language) [13] je značkovací jazyk umožňujúci anotovanie textových dát pomocou tagov. Základ tohto formátu vychádza zo správne štrukturovaných (well-formed) dokumentov a zameraný je na čitateľnosť a univerzálnu používateľnosť.

Well-formed dokumenty obsahujú značkové elementy, ktoré majú začínajúci a koncový tag, ako v príklade nižšie:

```
<tag> textové dátá </tag>.
```

Elementy sa môžu aj vkladať do ostatných elementov a elementové tagy môžu obsahovať aj asociované atribúty. Tieto atribúty poskytujú informácie o tagu, bez toho aby zasahovali do vložených dát:

```
<tag1 atribút1 = "hodnota" atribút2 = "42">
    <tag2> textové dátá </tag2>
</tag1>.
```

Dokument môže obsahovať pokyny spracovania pre modul XML, ktorý bude čítať dokument a rovnako aj komentáre. Riadia sa však rozdielnou syntaktickou formou ako elementové tagy a môžu sa objaviť na ľubovoľnom mieste v dokumente:

```
<?xmlstylesheet type="application/xslt + xml" href="#style1"?>
<!-- Toto je koment --&gt;.</pre>
```

Dokument musí obsahovať presne jeden element najvyššej úrovne. Poradie elementov je dôležité, keďže značia tok informácií v dokumente.

### Mitsuba renderer

Mitsuba renderer [7] je renderovací systém orientovaný na výskum pre obojsmernú simuláciu presunu svetla vyvinutý v inštitúte EPFL vo Švajčiarsku. Pozostáva z knižníc a pluginov, ktoré implementujú funkciaľitu pre rôzne materiály a zdroje svetla pre skompletizovanie renderovacích algoritmov. Je postavený na Dr.Jit, špecializovanom just-in-time prekladači zostavenom špeciálne pre Mitsuba renderer. Inštaluje sa pomocou PyPI a funguje ako knižnica pre programovací jazyk Python.

Mitsuba používa na reprezentáciu scén [6] zjednodušený formát založený na XML. Jednotlivé časti scény sú zapísané ako elementy s atribútmi obsahujúcimi informácie. Jednoduchá scéna s jedným objektom bez nasvietenia by vyzerala takto:

```
<scene version="3.0.0">
    <shape type="obj">
        <string name="filename" value="mesh.obj"/>
    </shape>
</scene>.
```

Ostatné časti scény [6] sa ukladajú podobným spôsobom, ale s použitím rozdielnych tagov. Priamo v XML sa dá nastaviť nasvietenie v scéne, upraviť pozíciu a nastavenia kamery alebo upraviť materiály a transformačné matice objektov. Následne sa dá v Python kóde ešte upraviť scénu pred samotným renderom. Príklad scény vykreslenej Mitsuba rendererom je na obrázku 2.7.

## Wavefront .obj

Wavefront .obj [8] je jedným z najviac používaných formátov pre textové uloženie trojdimenzionálnych objektov. Geometrické informácie o objekte ukladá priamo do textového súboru. Niekoľko súborov sa k .obj súboru prikladá .mtl súbor, ktorý označuje materiál z ktorého sa objekt skladá.

Formát používa tagy pre špecifikovanie geometrických elementov objektu. Každý tag sa píše v samostatnom riadku. Poradie písania elementov je dôležité, pretože pri spájaní plôch sa používa poradie vertexu. Jednotlivé tagy a príklady použitia sú ukázané v tabuľke 2.1.

Komentár:	# Komentár
Pozícia vertexu:	v $x \ y \ z$
Hodnota normálu:	vn $x_n \ y_n \ z_n$
Pozícia na textúre:	vt $x_t \ y_t$
Zostavovanie plochy:	f $v_1/t_1/n_1 \ v_2/t_2/n_2 \ v_3/t_3/n_3$
Zostavovanie plochy len s vertexom:	f $v_1 \ v_2 \ v_3$
Zostavovanie plochy s vertexom a normálom:	f $v_1//n_1 \ v_2//n_2 \ v_3//n_3$
Zostavovanie plochy s vertexom a textúrov:	f $v_1/t_1 \ v_2/t_2 \ v_3/t_3$

Tabuľka 2.1: Syntax písania tagov v .obj formáte [8].



Obr. 2.7: Scéna vykreslená v Mitsuba renderer [6].

# Kapitola 3

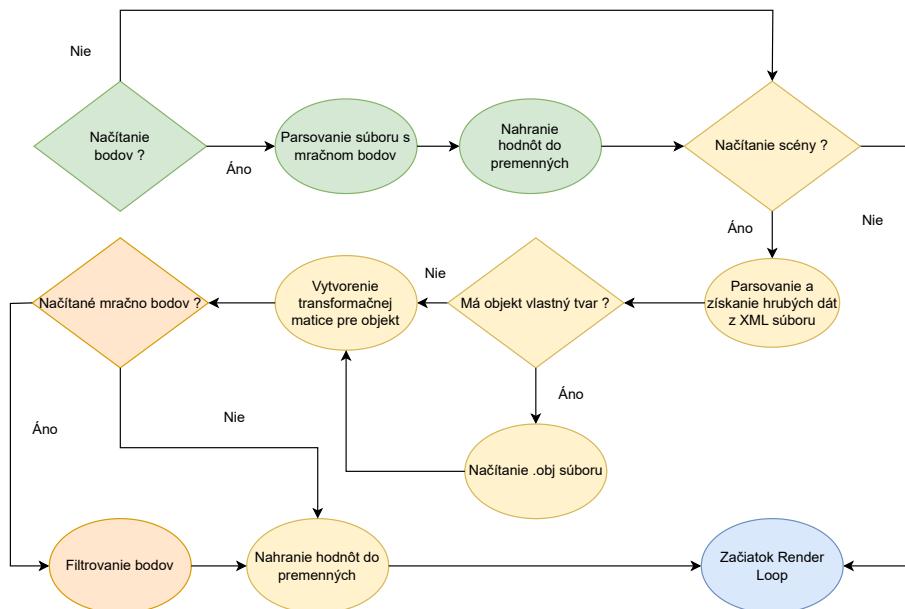
## Návrh

Táto kapitola obsahuje návrhy riešení jednotlivých problémov, ktoré bolo treba vyriešiť pri tvorbe aplikácie. Hlavne sa sústredí na parsovanie vstupných dát, návrhom na vykreslenie tepelných máp a hemisférických funkcií a návrhom užívateľského rozhrania.

V tejto kapitole sa nachádzajú pojmy, ktoré nemusia byť verejne známe a preto je odporúčané si prečítať kapitolu 2 pre čitateľov, ktorí nie sú oboznámení s problematikou.

### 3.1 Dáta na vykreslenie

Pred samotným vykreslením je potrebné si správne prečítať a uložiť vstupné súbory. Okrem súboru s mračom bodov, ktorý nemá definovaný presný formát, sú všetky typy súborov bližšie opísané v sekcií 2.7. Diagram návrhu spracovania vstupných dát je na obrázku 3.1. Jednotlivé úseky diagramu sú bližšie vysvetlené v tejto sekcií.



Obr. 3.1: Návrh načítania vstupných dát

## Parsovanie súboru s mračnom bodov

Súbor s mračom bodov 2.4 má jedinečnú štruktúru a nie je to klasický formát ukladania mračien. Skladá sa z hodnôt zapísaných v riadkoch. Každý riadok reprezentuje jeden bod v mračne. Formát riadku vyzerá takto:

$$\vec{p}, \vec{s}, \vec{t}, \vec{n}, \text{HSH}.$$

Hodnoty sú zapísané ako čísla s desatinou čiarkou. Prvé tri hodnoty zodpovedajú x-, y- a z-koordináciám bodu v priestore. Nasledujú tangenta, bitangenta a normála bodu. Každá zodpovedá trom hodnotám.

Koeficienty HSH funkcií 2.6 sa nachádzajú za nimi a môžu mať ľubovoľný počet hodnôt. Avšak je potrebné aby každý bod v mračne mal rovnaký počet koeficientov a aby sa počet koeficientov rovnal  $x^2$ , pričom  $x$  je celé číslo väčšie ako 0 a označuje rámček koeficientov HSH.

Po úspešnom parsovaní súboru je potrebné uložiť získané hodnoty bodov do pamäte grafickej karty, alebo do premenných pre filtrovanie bodov, ktoré sa vykonáva neskôr v programe.

## Parsovanie XML súboru

Vstupný XML súbor je rovnaký ako vstupný súbor v Mitsuba renderer. Viac o tomto rendereri je spomenuté v sekcií 2.7. Avšak pre jednoduché zobrazenie scény bez nasvietenia nie je potrebné čítať všetky informácie zo súboru. Zaujímavé sú jedine elementy s tagom **shape**.

V týchto elementoch sa nachádzajú informácie o objektoch v scéne. Každý **shape** má informáciu o svojom tvaru, prípadne o transformáciách objektu. Všetky tieto hodnoty je potrebné uložiť pre následné korektné zobrazenie objektu.

Základný postup pri získavaní hrubých dát z XML je teda najskôr odfiltrovať len informácie, ktoré sú zaujímavé pre program. Potom uloženie týchto hodnôt do premenných pre ich následné použitie v programe.

## Načítanie .obj súboru a tvorba transformačnej matice

Element **shape** má tri podporované možnosti tvaru predmetu. Sú to **sphere**, **rectangle** a **obj**. Pri **sphere** a **rectangle** sa načítajú predom určené objekty. V prípade **obj** je tvar objektu určený **.obj** súborom. Viac o **.obj** súboroch je spomenuté v sekcií 2.7.

Prvý krok pri načítaní **.obj** súboru je získať k nemu cestu. Tá je vždy zapísaná v pod-elementoch typu **obj**, ktoré sú uložené v premenných. K tejto ceste je však dôležité pridať aj cestu k samotnému XML súboru od umiestnenia aplikácie. Pri čítaní **.obj** súboru je potrebné si uložiť a zoradiť všetky vrcholy a normály. Význam tagov a informácií uložených v **.obj** súbore je bližšie popísaný v sekcií 2.7. Pri skladaní primitívov v tagoch **f** sa používa práve postupnosť vrcholov a normálov na zostavenie trojuholníkov z ktorých sa objekt skladá. Toto poradie sa nahradzuje konkrétnymi hodnotami vrcholov a normálov, ktoré sa ukladajú do bufferov (ukladací priestor, vyrovnaná pamäť). V bufferoch sú zoradené body a normály tak, ako budú vykreslované v primitívoch. Napríklad trojuholník tvoria vždy tri body v bufferi za sebou. Takto zoradené informácie zo súboru **.obj** sa nahrávajú do grafickej pamäte.

Transformačná matica pre objekt slúži k umiestneniu vrcholov objektu do priestoru scény. Vykreslovaniu dát na grafickej karte sa venuje sekcia 2.2. Informácie o transformáciach sú uložené v elemente **transform**, ktorý je pod-elementom **shape**. Môžu sa tam rovno

nachádzat už zostavené matice, alebo len postupy škálovania, rotácií a posunov objektu. Je potrebné zostaviť transformačné matice pre každý jeden typ transformácie a tie vynásobiť medzi sebou. Kedže násobenie matíc nie je komutatívne, tak je dôležité udržať poradie násobenia. Teda najprv vynásobiť škálovanie s rotáciou a potom výslednú maticu z predchádzajúceho výpočtu s maticou posunu. Finálne transformačné matice sa uložia do premenných a posielajú sa do pamäte grafiky pri renderovaní objektu s ktorým súvisia.

## Filtrovanie bodov

Filtrovanie bodov slúži na akceleráciu procesu vykreslenia tepelnej mapy. Namiesto toho aby sa pri určovaní zafarbenia v bode objektu používala vzdialenosť každého bodu v mračne, tak sa body prefiltrujú aby sa počítalo len s bodmi, ktoré sa nachádzajú v blízkosti objektu. Pre každý objekt sa zistí kváder v ktorý ho ohraničuje. Predtým je však potrebné vrcholy objektu upraviť transformačnou maticou, aby sa nachádzali na správnom mieste z hľadiska scény.

Pozícia každého bodu z mračna bodov sa porovná s kvádom v ktorom sa objekt nachádza. Ak sa bod nachádza vnútri kvádra, zaznačí sa. Následne pri renderovaní tepelných máp sa použijú pre každý objekt len body ktoré sú v jeho blízkosti a tým sa zabráni zbytočnej kalkulácii s nerelevantnými bodmi.

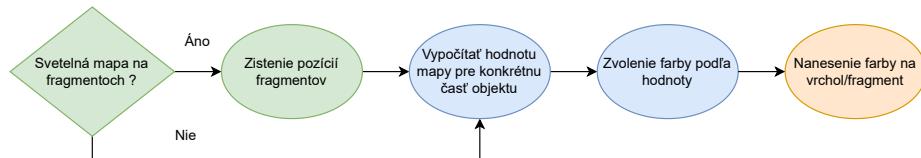
## 3.2 Operácie na vykreslenie

Táto sekcia sa zaoberá návrhmi operácií pri vykreslovaní tepelných máp a hemisférických funkcií. Tieto typy zobrazovania sú bližšie opísané v sekciách 2.5 a 2.6.

### Tepelné mapy

Diagram návrhu vykreslovania tepelných máp je na obrázku 3.2. Vyvinutá aplikácia vie vykreslovať tepelné mapy na vrcholoch objektov a aj na fragmentoch. Pri oboch módoch sa výrazne využíva interpolovanie hodnôt vrcholov v fragment shaderi. Pri vykreslovaní tepelnej mapy na vrchole sa vypočítaná farba interpoluje medzi fragmentami. Pri vykreslovaní na fragmentoch sa získava pozícia fragmentu v scéne interpoláciou pozície vrcholov trojuholníkov. Vykreslovanie na fragmentoch je presnejšie ako na vrcholoch, ale zároveň aj náročnejšie na grafickú kartu.

Pri samotnom počítaní farby je však jedno aký mód sa používa. Pre každý bod z mračna bodov, ktorý je v blízkosti objektu, sa vypočíta vzdialenosť od pozície vrcholu/fragmentu. Ak je vzdialenosť dostatočne nízka, začína mať tento bod efekt na zafarbenie. Veľkosť efektu je nepriamo úmerná od vzdialnosti. Efekt každého bodu sa sčítava dokopy. Ak je teda v blízkosti vrcholu/fragmentu veľa bodov, zvyšuje sa aj hodnota mapy pre konkrétnu časť objektu. Farba sa určí vypočítanou hodnotou, teda ak je hodnota nízka, farba v bode



Obr. 3.2: Návrh vykreslenia tepelnej mapy

má odtiene modrej. Ak sa zvyšuje tak začína nadobúdať ostatné farby ako zelenú, žltú alebo červenú. Po zvolení farby sa táto farba nanesie do výstupnej farby vrcholu/fragmentu.

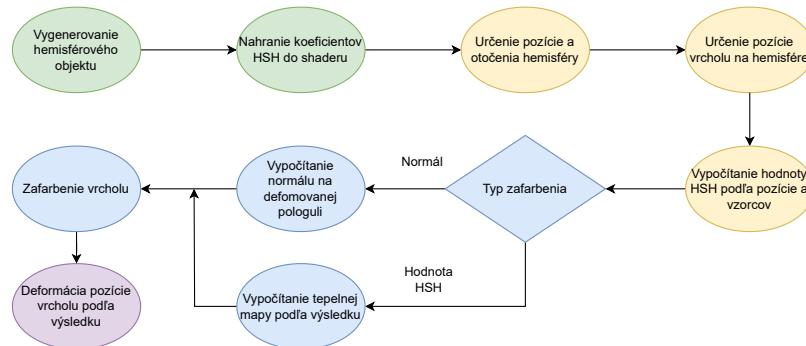
## Hemisférické funkcie

Diagram návrhu vykreslovania hemisférických funkcií je na obrázku 3.3. Konkrétnie vzorce na výpočet hodnoty hemisférických funkcií sú v sekcií 2.6.

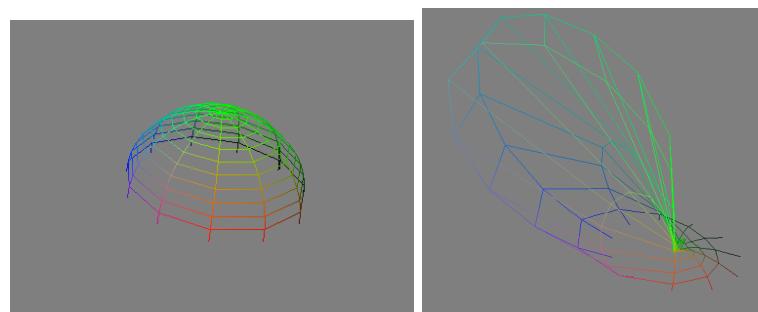
Na začiatku sa vygeneruje hemisférový objekt. Generácia tohto objektu je užitočná pretože sa tým vie kontrolovať kvalita zobrazenia hemisférických funkcií, alebo v prípade veľkého zaťaženia grafickej karty, znížiť počet vrcholov pre počítanie. Hemisféra má vždy určený počet riadkov a stĺpcov, a skladá sa z buniek v tvare štvorcu ktoré vzniknú medzi riadkami a stĺpcami. Ešte pred počítaním je však dôležité nahrať koeficienty HSH do pamäte grafickej karty, aby mohli byť použité vo výpočtoch.

Aby sa hemisféra zobrazovala na správnom mieste a aby bola otočená správnym smerom, je potrebné vypočítať transformačnú maticu na ktorú sa použijú hodnoty pozicie, normálu, tangenty a bitangenty bodu. Ďalej je potrebné vypočítať pozíciu samotného vrcholu na hemisfére. Využíva sa k tomu vedomosť o počte riadkov a stĺpcov na hemisfére a to, že vrcholy boli zoradené od spodu na vrch pri generovaní. Keď je známa pozícia vrcholu na hemisfére vrámcí buniek, je možné zistiť hodnoty y- a z-uhlov, ktoré určujú presnú polohu vrcholu. Tieto hodnoty uhlov spoločne s Legendrevými polynomami a koeficientami HSH sa využijú na výpočet hodnoty funkcie na vrchole.

Podľa zvoleného typu farbenia týchto funkcií je potrebné vypočítať normálky deformovanej pologule alebo využiť výsledok funkcie na vytvorenie jednoduchej tepelnej mapy. Normálka sa počíta podľa transformačnej matice a pozícii vrcholu na hemisfére. Tepelná



Obr. 3.3: Návrh vykreslenia hemisférických funkcií



Obr. 3.4: Ukážka deformácie pologule

mapa sa počíta ako gradient medzi zelenou a červenou farbou. Pozícia na gradiente sa určí hodnotou HSH na vrchole. Vypočítaná farba sa priradí k vrcholu. Pozícia vrcholu sa nакoniec posunie o hodnotu výsledku HSH smerom určeným pozíciou vrcholu na hemisfére a tým sa deformeuje pologuľa vid. obrázok 3.4.

### 3.3 Užívateľské rozhranie

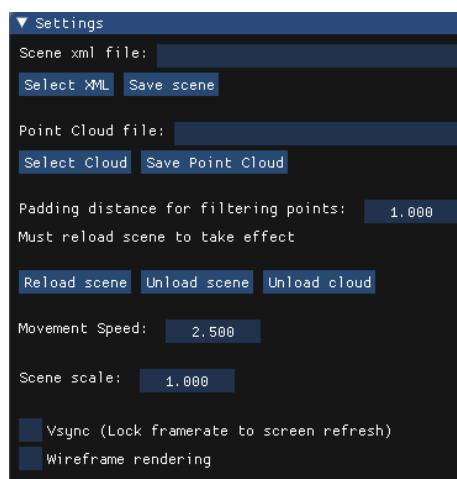
Užívateľské rozhranie slúži na interakciu užívateľa s aplikáciou. V prípade tejto aplikácie ho reprezentuje okno s nastaveniami. Tieto nastavenia sa delia do troch častí, kde každá časť sa sústredí na inú oblasť aplikácie. Táto sekcia se venuje priblíženiu funkcií aplikačných nastavení a vysvetleniu navigacie medzi nimi.

#### Načítanie vstupných súborov a základné nastavenia zobrazovania scény

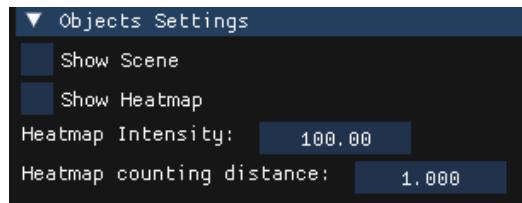
V prvej časti sa nachádzajú nastavenia, ktoré sa zobrazia pri zapnutí aplikácie. Sú zoobrazené na obrázku 3.5. Políčka **Scene xml file** a **Point Cloud file** obsahujú cesty k vstupný súborom. Tieto cesty sa dajú zapísat ručne a uložiť tlačítkami **Select scene** a **Select Point Cloud**. Alternatívne sa môžu použiť tlačítka **Select XML** a **Select Cloud** na výber súborov pomocou vyskakovacieho okna so súborovým systémom. Pri použití týchto tlačítok sa automaticky vyfiltrujú len typy súborov, ktoré sú podporované aplikáciou.

Nasleduje políčko na úpravu hodnoty, o ktorú sa zväčší kváder, ktorý pri filtrovaní bodov obkolesuje objekty. Pri renderovaní tepelnej mapy je niekedy užitočné počítať aj s bodmi, ktoré neležia priamo na objekte. Ak však scéna zabera väčší priestor je potrebné zväčsiť kváder o potrebnú hodnotu. Rovnako ak sa scéna odohráva na menšom priestore alebo je renderovanie tepelnej mapy náročné na grafickú kartu, je možné týmto znížiť záťaž alebo spresniť vykreslovanie. Ďalšie tlačítka slúžia na načítanie hodnôt zo vstupných súborov, prípadne na ich uvoľnenie ak už nie sú potrebné.

**Movement Speed** upravuje rýchlosť pohybu kamery v scéne. **Scene scale** zas škálovanie všetkých objektov v scéne. Políčko **Vsync** je účinné pri jednoduchých scénach, kedy nie je potrebné grafickú kartu využívať naplno a tak ušetriť grafické zdroje. Pri zaškrtnutí **Wireframe** sa budú vykreslovať len čiary medzi vrcholmi trojuholníkov.



Obr. 3.5: Základné nastavenia aplikácie



Obr. 3.6: Nastavenia objektov v scéne

### Nastavenia nad objektami v scéne

V druhej časti sa nachádzajú nastavenia nad objektami a sú zobrazené na obrázku 3.6. Táto časť sa zobrazuje len ak je načítaná scéna s objektmi a položky, ktoré upravujú tepelné mapy sa zobrazia len, keď je k objektom načítané aj mračno bodov.

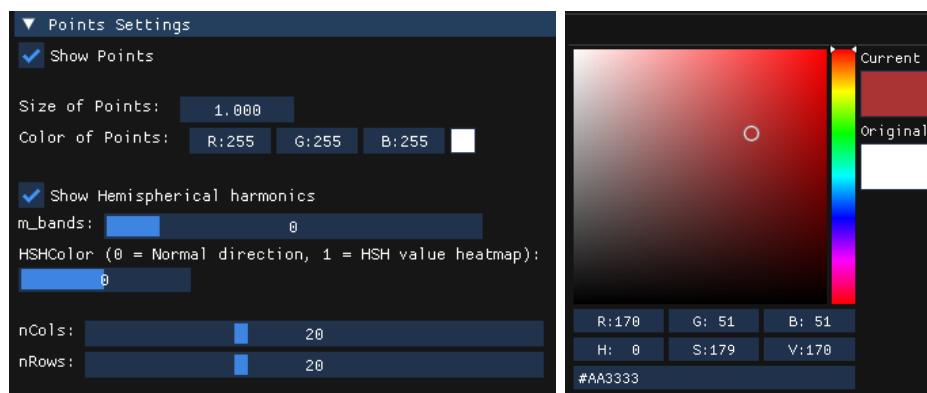
Poľička **Show Scene** a **Show Heatmap** nastavujú spôsob zobrazovania objektov. Ak sa nezobrazuje tepelná mapa tak sú objekty zafarbené náhodným odtieňom modrej. V rámci tepelnej mapy je možné upraviť jej intenzitu, teda upraviť váhu blízkeho bodu z mračna na zafarbenie objektu. Posuvníkom **Heatmap counting distance** je možné upraviť vzdialenosť, podľa ktorej sa určuje, či je bod z mračna dostatočne blízko ku bodu na objekte.

### Nastavenia mračien bodov a hemisférických funkcií

V tretej časti sa nachádzajú nastavenia nad mračnami bodov a sú zobrazené na obrázku 3.7. Zobrazujú sa len ak je načítaný súbor s mračnom bodov.

Body z mračna sa zobrazia po zaškrtnutí políčka **Show Points**. Vložením hodnoty do **Size of Points** a **Color of Points** je možné určiť veľkosť a farbu zobrazených bodov. Pri farbe sa dajú jednotlivé hodnoty RGB zapísat ručne, alebo využiť vyskakovacie okno na výber farby.

Pri hemisférických funkciách sa dá pomocou posuvníka **m\_bands** zvoliť počet koeficientov používaných pri výpočte funkcií. Čím väčšie číslo, tým sú výpočty náročnejšie. V **HSHColor** sa určuje spôsob farbenia funkcií a teda či budú zafarbené na základe ich smeru alebo podľa hodnoty HSH funkcie. Ako posledné sa dá nastaviť počet stĺpcov a riadkov v hemisférovom objekte, na ktorom sa vykonávajú HSH funkcie. Čím je menší počet stĺpcov a riadkov, tak sa zmenšuje aj počet vrcholov na ktorých sa počítajú HSH funkcie. Teda čím menej vrcholov, tým rýchlejšie prebieha renderovanie, za cenu zníženej kvality zobrazenia HSH.



Obr. 3.7: Nastavenia nad mračnami bodov a zobrazenie výberu farby

# Kapitola 4

## Implementácia

Táto kapitola do hĺbky vysvetľuje jednotlivé procesy potrebné pre fungovanie programu a algoritmy navrhnuté v predchádzajúcej kapitole. Obsah kapitoly sa teda často odkazuje na sekcie v kapitolách 2 a 3, v ktorých sú prípadné detaľy vysvetlené podrobnejšie.

### 4.1 Načítanie dát

Sekcia načítanie dát sa venuje bližšiemu vysvetleniu a priblíženiu algoritmov, ktoré boli použité pri parsovaní vstupných súborov. Návrhy týchto algoritmov sú zobrazené v sekcií 3.1.

#### Parsovanie súboru s mračnom bodov

Pri parsovaní súboru s mračom sa najprv otvorí súbor pomocou jeho cesty a načíta sa prvý riadok zo súboru do premennej `line`. Parsovanie prebieha po jednotlivých riadkoch. Používa sa pritom metóda cyklovania `while`, ktorá kontroluje hodnotu v premennej `line`. Po prečítaní všetkých riadkov posledné volanie `readline` vracia prázdny string a cyklus končí.

Hodnoty v riadku sú oddelené medzerou alebo čiarkou. Pre zabezpečenie oboch možností sa najprv nahradia v stringu všetky čiarky za medzery a potom sa string rozdelí podla medzier na menšie stringy, kde každý obsahuje práve jednu hodnotu v riadku. Hodnoty sa ukladajú do premenných v poradí, ako sú opísané v návrhu v sekcií 3.1. Pri koeficientoch HSH je parsovanie zložitejšie, keďže ich počet môže byť rozdielny podľa konkrétneho mračna bodov. Preto sa koeficienty ukladajú do generovaného zoznamu, kam sa vloží každá zvyšná hodnota v riadku.

Po uložení všetkých hodnôt v riadku do vektorov sa tieto vektory pripoja na poslednú pozíciu polí so všetkými hodnotami ostatných bodov v mračne. Teda pozícia vektora v tomto poli je rovnaká, ako pozícia riadku v súbore. S týmito polami sa následne pracuje ďalej v programe. Ako posledné sa vypočíta maximálna úroveň zanorenia HSH funkcií, ktorá sa rovná druhej odmocnine počtu koeficientov HSH v riadku a zatvorí sa súbor.

#### Parsovanie XML

Parsovanie XML je navrhnuté v sekcií 3.1. XML súbor má formát scény v Mitsuba renderer, ktorý je opísaný v sekcií 2.7. Pri parsovaní sa používa Python knižnica `ElementTree`. Táto knižnica rozdelí XML dokument do dátových štruktúr, kde každá štruktúra znázor-

```

<shape type="obj" >
  <string name="filename" value="models/Mesh011.obj" />
  <transform name="toWorld" >
    |   <matrix value="1.8 0 0 2.3 0 1 0 0 0 0 1 0 0 0 0 1" />
  </transform>
  <boolean name="faceNormals" value="true" />
  <ref id="Floor" />
</shape>

<shape type="rectangle" >
  <transform name="toWorld" >
    <scale y="0.5" x="0.5" />
    <rotate y="1" angle="30" />
    <matrix value="0.811764 1.25471e-014 5.4" />
    <translate z="1"/>
    <translate x="-1.2" />
    <translate y="0.6" />
  </transform>
  <bsdf type="conductor" >
    |   <string name="material" value="none" />
  </bsdf>
</shape>

```

Obr. 4.1: Príklady `shape` elementu v XML súboore

ňuje element stromu. Tieto štruktúry sú medzi sebou prepojené rovnako ako v elementy v dokumente. Ako prvá vec pri parsovaní sa uloží koreňový element XML stromu do premennej.

Parsovanie prebieha prechádzaním pod-elementov koreňa stromu. Dôležité sú elementy s tagom `shape`, ktorého príklady sú ukázané v obrázku 4.1. Ak sa pri kontrole tagu pod-elementu zhoduje tag s názvom `shape`, tak sa vytvorí prázdny slovník pre ukladanie informácií vnútri tohto elementu.

Atribúty elementu v strome sú už rovno uložené v štruktúre ako slovník, kde názov atribútu je kľúč k hodnote atribútu. Podobne sa atribúty uložia aj do slovníka `shape`. Pod-elementy, ktoré sú dôležité, sú `string` a `transform`. Prvý obsahuje cestu k .obj súboru a druhý zase transformácie objektu.

Atribúty sa pridávajú do slovníka pod-elementu rovnako ako do `shape` slovníka. V týchto atribútoch sa napríklad nachádza cesta k .obj súboru. Transformácie sú pod-elementy elementu `transform` a sú zoradené v poradí akom majú byť vykonané. Preto sa ukladajú do klasického zoznamu, kde ich pozícia značí poradie operácie nad objektom.

Pod-elementy sa ku slovníkom svojich rodičovských elementov pripoja s kľúčom, ktorý je ich tag. Finálny slovník `shape` sa pripojí na koniec zoznamu `Objects` s ktorým sa pracuje ďalej v programe.

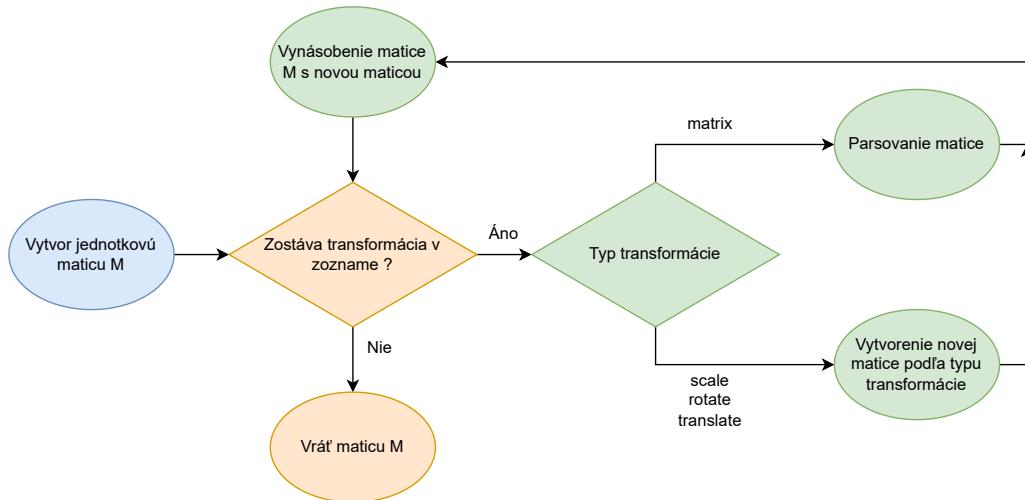
## Čítanie slovníkov s objektami v scéne

Objekt v scéne môže byť typu `obj`, `sphere` alebo `rectangle`. Pre posledné dve možnosti je cesta k .obj súboru konštantná. Pre typ `obj` je cesta jedinečná pre každý objekt a nachádza sa v slovníku pod klúčmi `string` a `value`. V tomto prípade je dôležité pred samotnú cestu k objektu pridať aj cestu k XML súboru.

Ak je v slovníku transformácia, musí sa vypočítať transformačná matica, inak je táto matica jednotková. Do funkcie na vytvorenie transformačnej matice sa posielala len zoznam s transformáciami.

## Tvorba transformačnej matice

Transformačná matica začína ako jednotková matica, ktorá sa postupne násobí s každou transformáciou. Diagram tvorby transformačnej matice je na obrázku 4.2. Typy transformácií sú `scale`, `rotate`, `translate` alebo už predom pripravená matica. Predpokladá sa z formátu XML súboru, že transformácie sú zoradené v správnom poradí a nie je treba toto poradie kontrolovať. Na tvorenie matíc pre každú operáciu a ich násobenie sa využíva knižnica `pyrr`, ktorá obsahuje funkcie na prácu s maticami pre OpenGL.



Obr. 4.2: Diagram tvorby transformačnej matice

Hodnoty, ako koordinácie  $x, y, z$  alebo uhol rotácie, sú uložené v slovníku transformácie pod ich menami. Rovnako aj typ transformácie, ktorý je uložený pod klíčom `type`. Hodnoty  $x, y, z$  môžu byť aj pod kľúčom `value`, kde sú písané v stringu vedľa seba oddelené čiarkami. Normálne tam bývajú všetky tri hodnoty, ale pri škálovaní môže byť len jedna hodnota. Ak je tam len jedna hodnota, znamená to že do každej dimenzie sa objekt zväčší alebo zmenší rovnako. Pri výbere hodnôt z `value` sa rozdelí tento string podľa čiarok na tri menšie stringy, kde každý obsahuje jednu koordinačnú hodnotu. Následne sa už len tieto hodnoty vytiahnu zo stringov do pred-pripravených premenných.

Konkrétne matice transformácie sa posielajú v stringu a parsujú sa podobne ako `value`. Môžu byť vo formáte  $3 \times 3$  alebo  $4 \times 4$ . V prípade kratšej matice sa musí pred použitím rozšíriť o štvrtý riadok a stĺpec. Hodnoty matice sú od seba oddelené medzerami a je ich potrebné transponovať pred použitím v OpenGL.

Po vynásobení jednotkovej matice všetkými operáciami sa táto matica vracia ako transformačná matica objektu.

## Parsovanie .obj súboru

Parsovanie .obj súboru prebieha podobne ako pri mračne bodov. Každý riadok sa číta samostatne a hodnoty sa ukladajú do bufferov, z ktorých sa na konci zostaví finálny buffer so zoradenými hodnotami. Formát .obj súboru je bližšie popísaný v sekcií 2.7. Pri načítaní riadku sa skontroluje typ dát. Relevantné typy dát sú vrcholy (`v`) a zostavené primitívy (`f`). Ak je typ dát `v` tak sa pomocou funkcie nahrajú koordinácie vrcholu do zoznamu `coords`. Ak je typ `f` tak sa zapíšu pozície vrcholov do zoznamu pozícii `indices`. Pseudokód parsovania .obj súboru je zobrazený v algoritme 1.

Ak sa primitív skladá z troch vrcholov, jednoducho stačí len poradie vrcholov zapísať za sebou do `indices`. Ak sa však skladá zo štyroch, treba ho rozdeliť na dva trojuholníky, ktorých vrcholy sa postupne zapíšu do zoznamu. Tieto trojuholníky za skladajú z vrcholov 1, 2, 3 a 1, 3, 4.

Pred uložením vrcholu do bufferu sa koordinácie zmenia na float. Výsledná hodnota sa ešte pred nahratím škáluje. Hodí sa to napríklad pri načítaní objektu pre body, ktorý je jednotková gula. Rovno pri načítaní sa vie naškálovať na veľkosť menšieho bodu.

---

**Algoritmus 1:** Parsovanie .obj súboru

---

```
while line = readline() do
    if line.empty() then
        | continue
    end
    line = line.split()
    if line[0] == 'v' then
        | coords.addVertex(line[1], line[2], line[3])
    end
    if line[0] == 'f' then
        if len(line[1:]) == 3 then
            | indices.addIndices(line[1], line[2], line[3])
        else if len(line[1:]) == 4 then
            | indices.addIndices(line[1], line[2], line[3])
            | indices.addIndices(line[1], line[3], line[4])
        end
    end
end
```

---

Pri zostavovaní finálneho zoradeného bufferu sa prechádza cez hodnoty v zozname `indices` podľa ktorých sa zistí pozícia vrcholu v zozname `coords` a vloží sa do finálneho bufferu. Zoradený buffer a dĺžka zoznamu `indices` sú vratné hodnoty, ktoré sa používajú ďalej v programe.

## Filtrovanie bodov

Návrh filtrovania bodov je v sekcii 3.1. Na začiatku filtrovania bodov sa upraví a skopíruje buffer objektu, aby sa s ním mohlo voľne pracovať. Potom sa pridá k pozíciam bodov štvrtý stĺpec pre umožnenie skalárneho súčinu so  $4 \times 4$  transformačnou maticou. Týmto súčinom sa vrcholy objektu transformujú na rovnaké miesto ako budú vykreslené v scéne.

Ak sú už vrcholy na správnych miestach, je možné začať s filtráciou. Kváder sa vytvára nájdením najväčších a najmenších hodnôt medzi vrcholmi objektu. Preto je potrebné otočiť osi v dvojrozmernom poli. S takto otočenými osami sú všetky hodnoty jedného typu koordinácie v jednom poli a tak sa dá ľahšie identifikovať najmenšia a najväčšia hodnota. K týmto hodnotám sa ešte pripočítava výplň, pre počítanie bodov aj v blízkom okolí objektu.

Ako posledné sa prejde cez všetky body v mračne a zhodnotí sa, či sa ich pozícia nachádza vnútri kvádra. Indexy bodov, ktoré sa nachádzajú dnu sa ukladajú do zoznamu, ktorý sa využíva ďalej v programe.

## 4.2 Nastavenie kamery a pohyb v priestore

Pri vykreslovaní scény v reálnom čase je dôležité mať možnosť pohybu kamery po scéne. Inak nie je potrebné scénu vykreslovať v reálnom čase. Pre pohyb a zobrazenie objektov v kamere sa používajú matice, ktoré presúvajú a deformujú objekty v scéne pre navodenie ilúzie pohybu v troj-dimenzionálnom priestore. Tieto matice sú matica projekcie a matica pohľadu. V shaderi sa nimi násobia vrcholy objektov po ich umiestnení v scéne.

## Matica perspektívnej projekcie

Bližšie informácie o projekčných maticiach sú v sekcií 2.3. Pri začatí programu sa vypočíta matica perspektívnej projekcie pre veľkosť základného okna. Používa sa pritom znova knižnica `pyrr`. Funkcia `create_perspective_projection` vytvára matice perspektívnej projekcie priamo použiteľné v OpenGL. Berie štyri parametre a to uhol pohľadu v stupňoch, pomer dĺžky a výšky obrazu a hodnoty blízkeho a vzdialejšieho fokálneho bodu. Dĺžka a výška obrazu sa určuje v počte pixelov okna v ktorom sa aplikácia zobrazuje a fokálne body určujú oblasť v ktorej sa objekty budú zobrazovať.

Táto matica sa mení iba pri zmene rozlíšenia vykreslovacieho okna. Cez `callback` funkciu, ktorá sa zavolá vždy pri zmene rozlíšenia, sa pomocou nových hodnôt vypočíta nová matica. Ak by sa nevypočítala, objekty by v scéne vyzerali deformované.

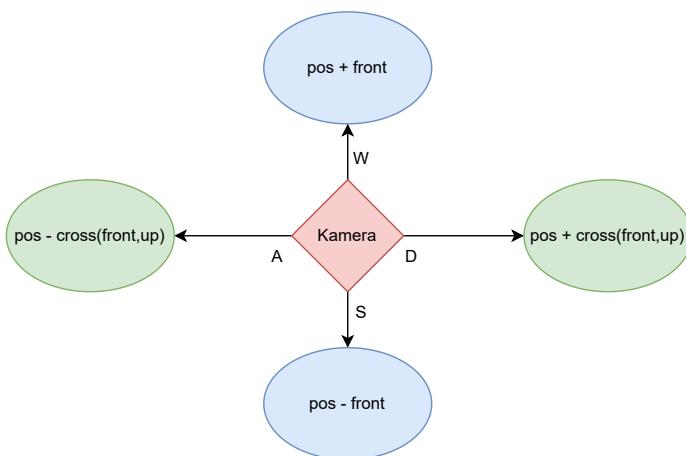
## Matica pohľadu

Matica pohľadu slúži na určenie polohy a smeru pohľadu kamery v scéne. Jej úloha je posúvať objekty a tým navodiť ilúziu pohybu kamery. Na jej vytvorenie slúži funkcia `create_look_at` z knižnice `pyrr`. Vstupné parametre funkcie sú tri vektory. Prvý vektor je pozícia kamery v scéne. Druhý vektor je cieľ v scéne na ktorý sa kamera pozerá. Ciel môže byť súčet pozície kamery a vektoru smeru otočenia kamery. Tretí vektor smeruje od kamery nahor.

Táto matica sa generuje pri každom novom snímku, pretože ovplyvňuje pohyb po scéne. Na začiatku render loopu sa vždy prepočítajú kamerové vektory, podľa ktorých sa vypočíta nová matica pohľadu. Táto matica sa následne nahrá do vertex shaderu ako `uniform` premenná.

## Pohyb po scéne

Pohyb po scéne sa realizuje stlačením tlačítok W, A, S, D na klávesnici. Diagram pohybu kamery je na obrázku 4.3. Na začiatku render loopu sa volá funkcia, ktorá spracúva vstup klávesnice a myši nad vykreslovacím oknom. Pre zachovanie konzistentnej rýchlosťi pohybu kamery pri rôznych časoch vykreslenia snímky sa násobí konštanta rýchlosťi s časom medzi poslednými dvoma vykreslenými snímkami. Pri stlačení tlačítka W sa k pozícii kamery



Obr. 4.3: Diagram pohybu kamery

pripočíta vektor smeru kamery vynásobený rýchlosťou pohybu. Pri stlačení S sa zase tento výsledok odpočíta.

Pri pohybe do strán sa vypočíta skalárny súčin vektorov smerujúcich pred a nad kameru. Vypočítaný vektor je kolmý k obom vektorom a smeruje doprava od smeru kamery. Pri stlačení D sa teda k pozícii kamery tento vektor pripočítava a zase pri stlačení A odpočítava.

## Pohľad kamery

Pri pohlade kamery sa upravujú vektory `camera_front` a `camera_up`. Pre uchovanie rozdielu polohy myšky medzi snímkami sa používa `callback` funkcia, ktorá tento rozdiel počíta. Posúvanie pohľadu funguje len pri stlačení pravého tlačítka myši. Preto je potrebné kontrolovať stav tohto tlačítka pomocou boolean premennej. Ak tlačítko na myši nieje stlačené, premenná má hodnotu `True` a nezaznamenáva sa rozdiel pohybu myši. Ak je stlačené tak sa správne počíta rozdiel medzi momentálnou pozíciou myši a pozíciou myši na minulom snímku. Rozdiel osi  $y$  sa počíta opačne ako osi  $x$  pretože rasterizácia obrazu prebieha zľava doprava, odhora dolu. Čiže veci nižšie na obrazovke majú vyššiu hodnotu  $y$  ako veci vyššie. Rozdiely pozície podla osi sa uložia do premenných `xoffset` a `yoffset`.

Nové hodnoty vektorov sa počítajú v metódach objektu `cam`. Pseudokód výpočtu je v algoritme 2. Najskôr sa upravia hodnoty `xoffset` a `yoffset` pomocou prednastavenej senzitivity a pripočítajú sa k `jaw` a `pitch`. `Jaw` predstavuje pohyb sprava dolava a `pitch` zhora dole. Pred výpočtom nových vektorov sa ešte skontroluje hodnota `pitch` či nepresahuje priamy uhol. Ak by presahovala tak by sa kamera otočila a nefungovala by správne.

Najprv sa vypočíta podla vzorca vektor smeru kamery. Vektor `camera_right` sa vypočíta skalárny súčinom novo-vytvoreného `camera_front` vektora a vektora smerujúceho nahor. Vektor `camera_up` sa vypočíta skalárny súčinom vektorov `camera_front` a `camera_right`. Pre násobenie vektorov sa používa knižnica `pyrr`.

Tieto vektory sa ukladajú do premenných používaných v programe vo funkcií na spracovanie vstupov, ak je stlačené pravé tlačítko myši.

---

### Algoritmus 2: Počítanie nových vektorov kamery

---

```
jaw += xoffset *sensitivity
pitch += yoffset *sensitivity
if pitch > 89.9 then
| pitch = 89.9
end
if pitch < -89.9 then
| pitch = -89.9
end

camFront.x = cos(rad(jaw)) * cos(rad(pitch))
camFront.y = sin(rad(pitch))
camFront.z = sin(rad(jaw)) * cos(rad(pitch))

camFront = norm(camFront)
camRight = norm(camFront × [0.0, 1.0, 0.0])
camUp = norm(camFront × camRight)
```

---

## 4.3 Reprezentácia dát na GPU

Po načítaní dát zo vstupných súborov je dôležité tieto dát uložiť do pamäte na grafickej karte, aby sa s nimi dalo pracovať v shaderoch. Táto sekcia sa venuje spôsobu uloženia dát na GPU a následnému prístupu k nim.

### Reprezentácia objektov

Pre každý objekt v scéne sa vytvorí samostatný Vertex Array Object (VAO) a Vertex Buffer Object (VBO). VAO slúži na identifikáciu VBO v ktorom sú uložené pozície vrcholov objektu na grafickej karte. Po prepojení VBO na VAO a nastavení typu bufferu sa musí pripraviť miesto na grafickej karte a špecifikovať adresu skadial sa bude do bufferu zapisovať pomocou funkcie `glBufferData`. Na určenie spôsobu uloženia dát v bufferi slúži funkcia `glVertexAttribPointer`. Parametrom tejto funkcie je aj lokácia vstupnej premennej v shaderi s ktorou sa bude pracovať na grafickej karte. Táto lokácia je prvým parametrom funkcie. Nasleduje počet hodnôt, ktoré patria k sebe a typ hodnoty. Nasledujúce parametre slúžia na voľbu normalizácie hodnôt a k prípadným skokom v bufferi. Kedže vrcholy sú v zoradenom bufferi uložené tesne vedľa seba nie je potrebné tieto parametre nastavovať.

### Reprezentácia mračna bodov

Pre mračno bodov sa vytvoria dve VAO. Jedno slúži na objekt bodu, ktorý sa bude vykreslovať na pozíciah bodov v mračne a druhý slúži na hemisférický objekt, ktorý sa bude deformovať pomocou HSH funkcií. Vrcholy oboch objektov sa nahrávajú rovnako ako vrcholy objektov v scéne. Avšak je potrebné nahrať do grafickej pamäte aj posuvné matice pre zmenu pozície bodu. Tieto matice sa počítajú podobne ako posuvné matice pri skladaní transformačných matíc pre objekty. Na posun sa používa pozícia bodu v scéne. Každá hodnota z matice sa následne nahrá samostatne za sebou do zoznamu.

Pri nahrávaní matice do klasického bufferu však nastáva problém a to, že parameter funkcie `glVertexAttribPointer` s množstvom hodnôt vedľa seba môže mať hodnotu maximálne štyri a matica ich má šestnásť. Samotná vstupná premenná v shaderi však môže byť typu  $4 \times 4$  matice, ale zaberá 4 lokácie. Pre každú z týchto lokácií je potrebné zavolať funkciu `glVertexAttribPointer` a pri týchto volaniach sa už naplno využije preskakovanie v rámci bufferu. Možnosť normalizácie stále ostáva nevyužitá, ale hodnoty v bufferi už nie sú tesne vedľa seba. Vzdialenosť od jednotlivých matíc v bajtoch je veľkosť jednej hodnoty v zozname krát šestnásť a začiatok čítania sa pre každý riadok matice bude posúvať o veľkosť predchádzajúcich riadkov rovnako v bajtoch. Kedže sa body v mračne renderujú inštančne, pomocou funkcie `glVertexAttribDivisor` sa nastaví, že matice sa budú striedať každú inštanciu.

Tieto matice sa nahrajú rovnako do oboch VAO bodov v mračne. K hemisférickému objektu je potrebné ešte nahrať normály, tangenty a bitangenty bodov v mračne. Avšak keďže sú to znova len vektory s troma hodnotami, nahrávajú sa podobne ako vrcholy objektu s rozdielom nastavenia `Divisor` znova na jednu inštanciu.

### Shader Storage Buffer Object

Do Shader Storage Buffer Object (SSBO) je možné uložiť veľké množstvo dát, ktoré sú v rámci shader kódu vždy prístupné. Je teda vhodným miestom na uloženie pozícií a vyfiltrovaných indexov bodov pre renderovanie tepelných máp, alebo na uloženie koeficientov

HSH funkcií. Ukladanie do SSBO je podobné ako do klasického bufferu. Do parametra funkcie `glBindBufferBase` je potrebné zapísť binding point SSBO, ktorý symbolizuje jeho lokáciu v shaderi.

### Prístup k hodnotám v shaderi

Nahrávané hodnoty z bufferov sú v shaderi uložené vo vstupných premenných. Pri deklaráciách týchto premenných sa nastavuje lokácia, ktorá sa využíva pri nahrávaní hodnôt z bufferov. Hodnoty vo vstupných premenných sa aktualizujú vždy podľa aktuálneho vrcholu, ktorý sa vykresluje.

SSBO vnútri shaderu vyzerá ako štruktúra v programovacom jazyku C a k hodnotám vnútri sa pristupuje rovnako ako ku hodnotám vnútri štruktúry. Rovnako ako pri vstupných premenných sa pri deklarácii píše aj binding point.

Transformačné matice sa ukladajú do shaderov ako `uniform` hodnoty a tiež sú dostupné vždy v rámci shader kódu.

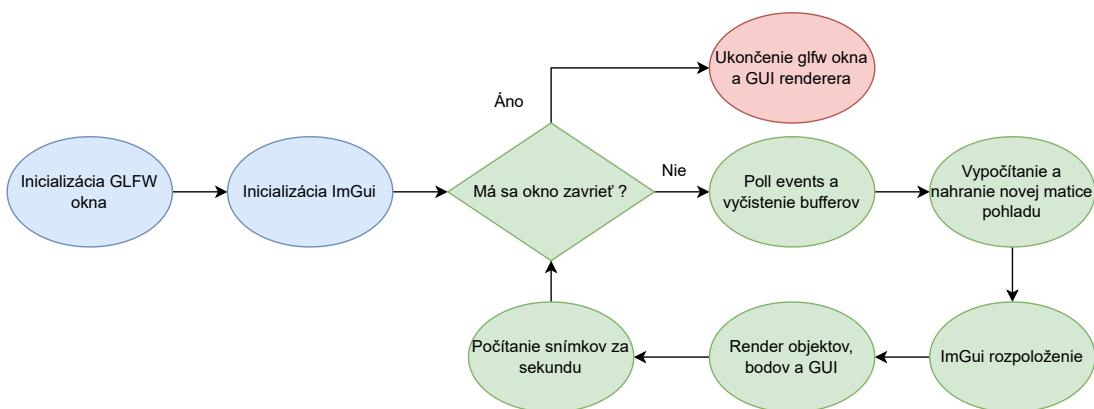
## 4.4 Render Loop

Pri renderovaní v reálnom čase sa musí pre každý snímok vykresliť scéna samostatne. Ak je týchto snímkov dostatočne veľa za sekundu, navádzia to ilúziu pohybu. Snímky sa vykresľujú v nekonečnom cykle do okna programu. Tento cyklus sa volá render loop a jeho diagram je v obrázku 4.4.

### Inicializácia pred render loopom

Pred samotným render loopom je potrebné inicializovať okno, do ktorého sa bude scéna renderovať. Okno sa inicializuje pomocou knižnice `glfw`, ktorá je priamo určená na použitie pri renderovaní pomocou OpenGL. Po vytvorení a určení pozicie na obrazovke sa pre okno určí kontext. Znamená to, že príkazy grafickej knižnice sa budú vykonávať nad týmto oknom.

Užívateľské rozhranie je riešené cez knižnicu `ImGui`, ktorú treba rovnako inicializovať a vytvoriť objekt `Renderer`, ktorý vykreslí okno užívateľského rozhrania do `glfw` okna.



Obr. 4.4: Diagram pochytu kamery

## Vnútri render loopu

Pre render loop je ideálne použiť cyklus typu `while`. Aj keď je to nekonečný cyklus, keď sa užívateľ rozhodne zatvoriť program, mal by prestať prebiehať aj render scény. V `glfw` knižnici preto existuje funkcia `window_should_close`, ktorá vracia `True` ak sa užívateľ rozhodol zavrieť okno. Na začiatku render loopu sa zhromaždia všetky udalosti pomocou funkcie `poll_events`. V tejto funkcií sa zaznamenajú všetky interakcie s klávesnicou a myšou a zavolajú sa `callback` funkcie. Potom sa vyčistia bufferi a upravia sa vektory kamery podľa postupu v sekcií [4.2](#).

Po vypočítaní a nahratí novej matice pohľadu do shaderu sa určuje podoba užívateľského rozhrania a renderujú sa objekty, body v scéne a užívateľské prostredie. Po renderení sa vypočíta trvanie vykreslenia snímky a podľa tohto času sa vypočíta počet snímkov za sekundu. Po skončení `while` cyklu treba korektne ukončiť `Renderer` užívateľského rozhrania a zavrieť okno.

## 4.5 Inštančné renderovanie bodov a HSH funkcií

Táto sekcia opisuje implementáciu renderovania bodov z mračna a HSH funkcií. Návrh implementácie HSH funkcií je v sekcií [3.2](#) a teória k týmto funkciám je v sekcií [2.6](#).

### Renderovanie bodov v mračne

Renderovanie bodov v mračne je v tomto bode, kedy sú už všetky hodnoty nahrané v grafickej pamäti, celkom jednoduché. V kóde programu sa nastaví prepínač v shaderi na renderovanie mračna bodov pomocou zmeny jeho hodnoty v `uniform` premennej. Následne sa naviaže VAO objektu bodu na renderovanie. Inštančné renderovanie sa spúšta funkciou `glDrawArraysInstanced`. Táto funkcia má 4 parametre, a to spôsob renderovania, začiatok index vrcholu, počet vrcholov a počet inštancií. Spôsob renderovania je uložený v premennej `RenderType`, pretože je volený užívateľom. Typy používané v aplikácii sú `GL_LINES` a `GL_TRIANGLES`, ktoré vykresľujú len čiary medzi vrcholmi alebo aj vnútra trojuholníkov.

Kód vnútri shaderov je tiež jednoduchý a pozostáva len z vypočítania pozície vrcholu na obrazovke a uloženia farby bodu. Vo fragment shaderi sa len táto farba vrcholu vloží do fragmentu.

### Generovanie hemisférického objektu

Pseudokód generovania hemisférického objektu je vložený v algoritme [3](#). Najprv sa musia pre hemisférický objekt vygenerovať vrcholy. Vrcholy sa generujú pomocou dvoch uhlov. Uhол  $\theta$  znázorňuje vektor smeru a jeho hodnota v radiánoch je v rozsahu  $[0, 2\pi]$ . Uhol  $\phi$  znázorňuje výšku a jeho hodnota v radiánoch je v rozsahu  $[0, \frac{\pi}{2}]$ . Podľa vzorca sa pomocou týchto uhlov vypočíta pozícia vrcholu, ktorá sa pridá do zoznamu s ostatnými vrcholmi. Riadkov sa počíta o jeden viac pretože je potrebné vygenerovať vrcholy aj pre úplne vrchný riadok.

Podobne ako pri načítaní `.obj` súboru, je potrebné vytvoriť buffer v ktorom sú zoradené vrcholy do trojuholníkov. Tu je však potrebné si poradie vrcholov vkladaných do bufferu vypočítať. Keď sa pri počítaní príde na koniec riadku, poradie vrcholu vedľa je rovné prvému vrcholu v tom riadku. Preto je potrebné tieto hodnoty upraviť aby sa nepresiahol objem pola pri ukladaní do bufferu.

---

**Algoritmus 3:** Generovanie hemisférického objektu

---

```
vertices = []
for (i = 0; i < nRows + 1; i++) do
    θ =  $\frac{i}{nRows} * \frac{\pi}{2}$ 
    for (j = 0; i < nCols; j++) do
        φ =  $\frac{j}{nCols} * 2\pi$ 
        x = cos(φ) * sin(θ)
        y = cos(θ)
        z = sin(φ) * sin(θ)
        vertices.addVertex(x, y, z)
    end
end

buffer = []
for (i = 0; i < nRows; i++) do
    for (j = 0; i < nCols; j++) do
        v0 = i * nCols + j
        v1 = v0 + 1
        v2 = (i + 1) * nCols + j
        v3 = v2 + 1
        if v1 mod nCols == 0 then
            v1 = v1 - nCols
            v3 = v3 - nCols
        end

        buffer.addTriangle(v0, v1, v2)
        buffer.addTriangle(v2, v1, v3)
    end
end
```

---

Po vytvorení bufferu sa tento buffer nahrá do VAO hemisférického objektu a môže sa s ním ďalej pracovať.

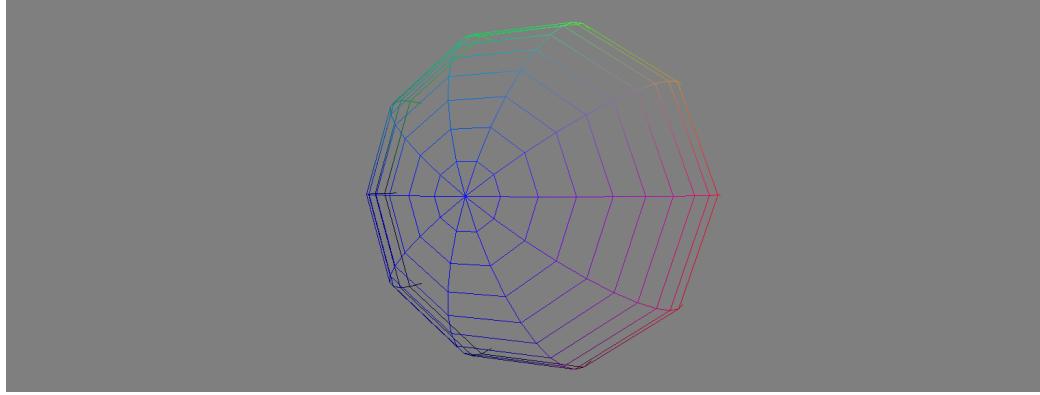
### Vytvorenie matice modelu pre bod

Matica modelu slúži na orientáciu pologule v smere normálu bodu. Vytvára sa vynásobením matice s pozíciou bodu s maticou obsahujúcou tangentu, bitangentu a normál bodu. Rovnica 4.1 zobrazuje tento výpočet.

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ pos.x & pos.y & pos.z & 1 \end{pmatrix} \cdot \begin{pmatrix} t.x & t.y & t.z & 0 \\ b.x & b.y & b.z & 0 \\ n.x & n.y & n.z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.1)$$

### Určenie pozície vrcholu na pologuli

Pred určením presnej polohy je potrebné identifikovať obdĺžnik na pologuli kde sa vrchol nachádza. Keďže je pologuľa generovaná, je známy počet vrcholov podľa počtu stĺpcov



Obr. 4.5: Pologuľa rozdelená na obdlžníky

a riadkov. S týmito informáciami sa dá presne určiť poloha obdlžníka. Obrázok 4.5 ukazuje pologuľu rozloženú na obdlžníky pre lepšiu predstavu.

Poloha vrcholu sa počíta rovnako ako pri generovaní hemisférického objektu a znázorňujú ju 2 uhly  $\theta$  a  $\phi$ . Podľa stĺpca a riadka vrcholu sa vypočíta normalizovaná hodnota týchto uhlov, ktorá sa prevedie do radiánov. Hodnota HSH funkcie sa počíta s použitím týchto uhlov a indexom vykresloanej inštancie `shID`.

### Výpočet hodnoty HSH na vrchole

Hodnota HSH sa počíta podľa vzorcov v sekcii 2.6 a pseudokód jej výpočtu je v algoritme 4. Pri výpočtoch sa používajú posunuté asociované Legendreove polynómy. V premennej `m_bands` je maximálna úroveň zanorenia do koeficientov HSH a je ovládaná užívateľom. Čím je táto hodnota vyššia, tým je náročnejší výpočet na grafiku. Koeficienty z SSBO sa získavajú vo funkcií `coef`, kde sa vyberajú podľa poradia inštancie a pozícií zanorenia koeficientu 1 a `m`.

### Deformácia a zafarbenie vrcholu

Poloha vrcholu sa znova vypočíta s použitím uhlov  $\theta$  a  $\phi$  pre zaistenie, že vypočítaná hodnota HSH sa nanesie na správnu pozíciu na hemisfére. Táto poloha sa vynásobí výsledkom z HSH funkcie a upraví sa škálovaním hemisférového objektu čím sa deformeuje do žiadanej pozície.

---

#### Algoritmus 4: Výpočet HSH funkcie

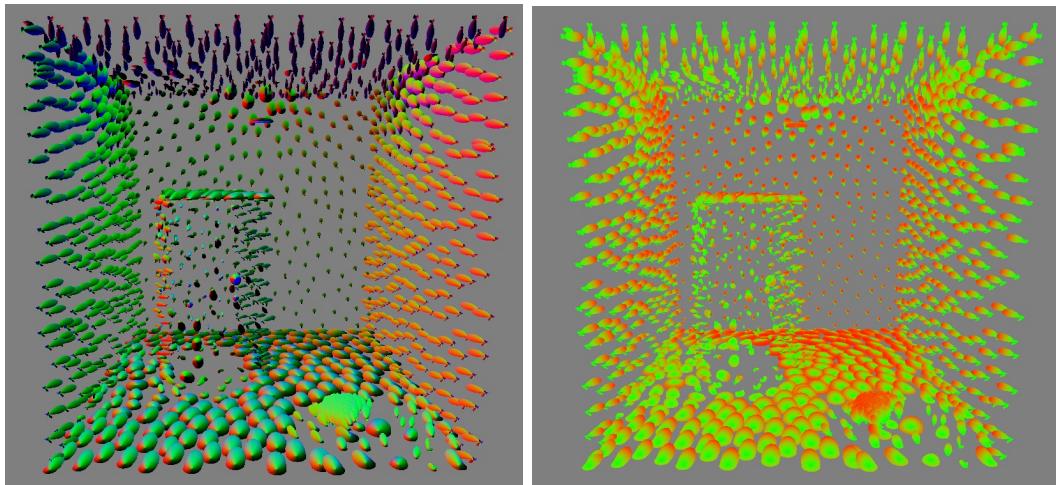
---

```

result = 0
for ( $l = 0; l < m\_bands; l++$ ) do
    for ( $m = 1; m \leq l; m++$ ) do
         $L = \tilde{P}_l^m(\cos \theta) * \tilde{K}_l^m(\cos(m\phi))$ 
         $result += coef(shID, l, -m) * \sqrt{2} * \sin(m\phi) * L$ 
         $result += coef(shID, l, +m) * \sqrt{2} * \cos(m\phi) * L$ 
    end
     $result += coef(shID, l, 0) * \tilde{P}_l^0(\cos \theta) * \tilde{K}_l^0(\cos(m\phi))$ 
end

```

---



Obr. 4.6: Ukážky farbenia HSH funkcií

Vrchol môže byť zafarbený dvoma spôsobmi. Prvý spôsob je farba podľa normálu vrcholu a druhý spôsob je tepelná mapa podľa hodnoty HSH funkcie. Normál vrcholu sa nenahráva do shaderu a preto je ho potrebné vypočítať pomocou sčítania inverznej a transponovanej matice modelu s pozíciou vrcholu. Rovnica 4.2 zobrazuje tento výpočet. Hodnotu farby je potrebné pred použitím normalizovať.

$$Color = (M^{-1})^T \cdot (verPos, 1) \quad (4.2)$$

Jednoduchá tepelná mapa podľa HSH hodnoty prechádza zo zelenej farby v bode, ktorý je deformovaný negatívne, ku červenej farbe v bode, ktorý bol deformovaný pozitívne. Aby sa zmenila farba zo zelenej na červenú, musí sa odčítať hodnota  $g$  a pripočítať hodnota  $r$  v  $rgb$  vektore. Ak sa budú tieto hodnoty upravovať výsledkom HSH funkcie vznikne tepelná mapa medzi zelenou a červenou na základe HSH hodnoty. Rovnica 4.3 zobrazuje tento výpočet. Farbu je potrebné pred použitím normalizovať.

$$Color = (|HSH|, 1.0 - |HSH|, 0.0) \quad (4.3)$$

Oba spôsoby zafarbenia sú zobrazené na obrázku 4.6.

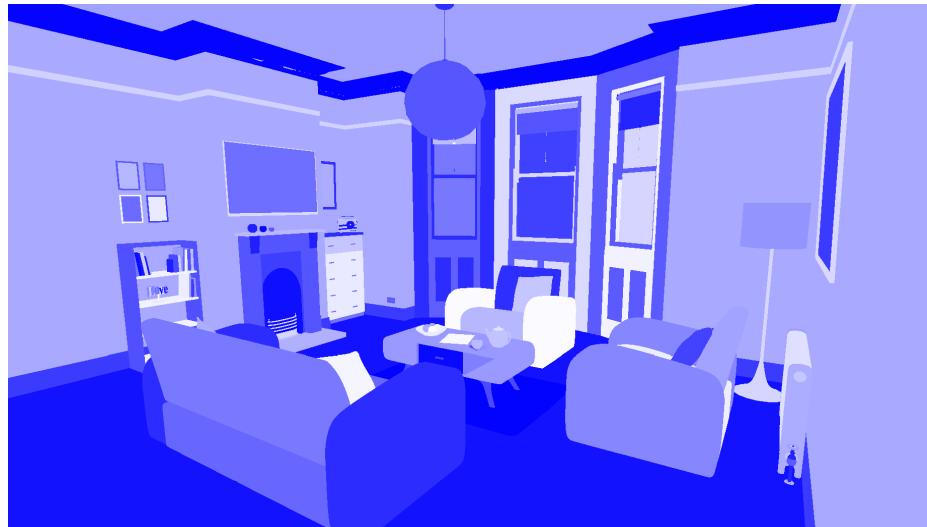
## 4.6 Renderovanie objektov a tepelnej mapy

Táto sekcia sa zaobráva spôsobom implementácie renderovania objektov v scéne a zafarbenia týchto objektov tepelnou mapou. Návrh renderovania tepelných máp je opísaný v sekcií 3.2 a teória o tepelných mapách sa nachádza v sekcií 2.5.

### Renderovanie objektov bez tepelnej mapy

Aby bolo možné rozlíšiť objekty v scéne od seba, musia byť rozdielne zafarbené. Preto sa pre každý objekt sa vytvorí náhodný odtieň farby od bielej po modrú. Tieto farby boli vybrané, pretože nie sú príliš nápadné a zároveň sú lahko rozlišiteľné.

Po parsovaní .obj súboru sa pre každý objekt vygeneruje pseudonáhodné číslo od 0 do 1. Toto číslo sa následne odčíta od  $r$  a  $g$  hodnôt jednotkového vektoru, ktorý sa potom nahrá



Obr. 4.7: Scéna vykreslená v aplikácii bez tepelnej mapy

do zoznamu vektorov na pozíciu totožnú s pozíciou objektu. Toto znamená, že ak je pseudonáhodné číslo nízke, farba objektu je bližšie k bielej. Čím je číslo väčšie, tým je farba objektu modrejšia.

Vnútri render loopu sa v cykle pre každý objekt naviaže VAO, kde sú uložené zoradené vrcholy objektu. Pred renderom je potrebné nahrať maticu modelu a farbu modelu do **uniform** premenných v shaderi. Vykreslenie objektu sa spúšťa funkciou `glDrawArrays`, ktorá má tri vstupné parametre. Prvým je spôsob renderovania, čiže či sa budú vyfarbovať len čiary medzi vrcholmi alebo celé trojuholníky. Ďalšie dva sú prvý index vrcholu a počet vrcholov v zoradenom bufferi.

Vnútri shader kódu sa pomocou matíc vypočíta pozícia vrcholu na obrazovke a nastaví sa farba vrcholu na farbu objektu. Scéna vykreslená bez tepelnej mapy je na obrázku 4.7.

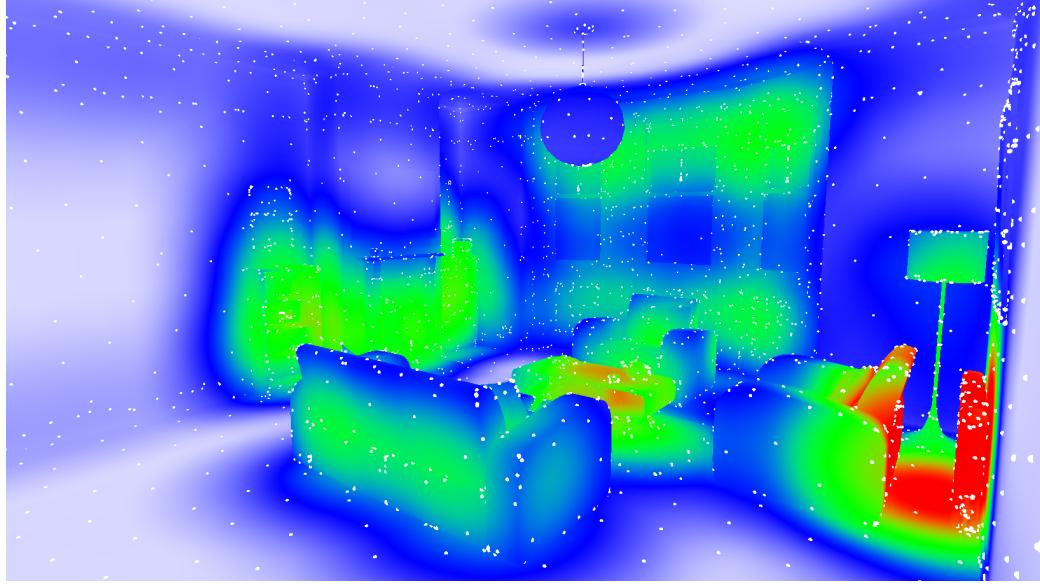
## Dáta ukladané do SSBO

Pri počítaní tepelných mapách sa používajú dve SSBO. V prvom sú pozície všetkých bodov v mračne zoradené za sebou a v druhom sú prefiltrované indexy bodov, ktoré sa nachádzajú vnútri objektov. Každý objekt v scéne má vlastnú zbierku týchto indexov a preto sú tieto zbierky zoradené za sebou v jednom bufferi podľa poradia vykreslovaných objektov. Pomocou týchto indexov sa pri počítaní tepelnej mapy vyberajú koordinácie bodov z prvého SSBO.

## Renderovanie tepelnej mapy

V render loope je postup podobný ako pri renderovaní bez tepelnej mapy. Je však potrebné nahrať do **uniform** premenných v shaderi dve hodnoty naviac. Prvá hodnota je počet bodov vnútri objektu v zozname `NobjectPoints`. Druhá hodnota je `offset` a znázorňuje počet bodov vnútri objektov, ktoré boli vykreslené skorej.

Vnútri shader kódu sa najprv vypočíta hodnota tepelnej mapy v konkrétnej pozícii na objekte. Pomocou filtrovaných indexov bodov v mračne sa prejde cez každý bod, ktorý sa nachádza v blízkosti objektu. Vzdialenosť bodu v mračne s pozíciou na objekte sa vydeli maximálnou povolenou vzdialenosťou aby sa normalizovala pre následné porovnanie.



Obr. 4.8: Scéna vykreslená v aplikácii s tepelnou mapou

Efekt bodu na zafarbenie sa počíta pomocou vzorca  $\frac{1-dis}{intensity}$ . V princípe ak je bod vzdialý má minimálny efekt a ak je blízko má efekt 1. Tento efekt sa ešte delí intenzitou tepelnej mapy určenou užívateľom.

Farba sa určuje gradientom medzi farbami podľa vypočítanej hodnoty tepelnej mapy. Ak je hodnota v rozsahu  $[0, 1]$  gradient je od bielej k modrej, aj je v rozsahu  $[1, 2]$  gradient je od modrej k zelenej a ak v rozsahu  $[2, \infty]$  gradient je medzi zelenou a červenou. Hodnoty nad tri sa zobrazujú tiež ako červená farba. Pseudokód výpočtu tepelnej mapy je v algoritme 5. Scéna vykreslená s tepelnou mapou je na obrázku 4.8.

---

**Algoritmus 5:** Výpočet tepelnej mapy

---

```

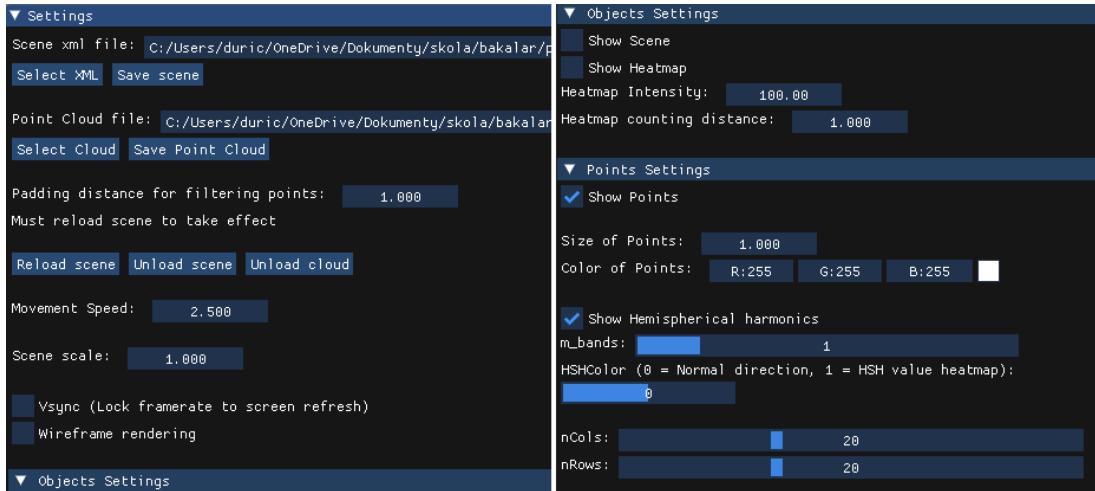
for ( $i = offset; i < offset + NPointsIn; i++$ ) do
     $point = pointPos[objectPoints[i]]$ 
     $dis = distance(verPos, point) / MaxDistance$ 

    if  $dis \leq 1.0$  then
        |  $color\_cor += \frac{1-dis}{intensity}$ 
    end

    end
    if  $color\_cor < 1.0$  then
        |  $color = vec3(1.0 - color\_cor, 1.0 - color\_cor, 1)$ 
    else if  $color\_cor < 2.0$  then
        |  $color\_cor = color\_cor - 1$ 
        |  $color = vec3(0.0, color\_cor, 1.0 - color\_cor)$ 
    else
        |  $color\_cor = color\_cor - 2$ 
        |  $color = vec3(color\_cor, 1.0 - color\_cor, 0.0)$ 
    end

```

---



Obr. 4.9: GUI aplikácie

## 4.7 Ovládanie / GUI

Táto sekcia sa venuje implementácii ovládania aplikácie a grafického užívateľského rozhrania. Návrh užívateľského rozhrania je v sekcií 3.3. Implementuje sa pomocou knižnice ImGui, ktorá umožňuje vkladať hodnoty upravené užívateľom do premenných programu.

Knižnica obsahuje funkcie, ktoré vykresľujú elementy ako posuvníky, input políčka, zaškrtačacie políčka (checkbox) alebo tlačítka. Vratné hodnoty z týchto funkcií sú konkrétnie hodnoty, ktoré boli zapísané užívateľom a boolean hodnota, ktorá značí zmenu zapísanú užívateľom. Celé GUI je na obrázku 4.9.

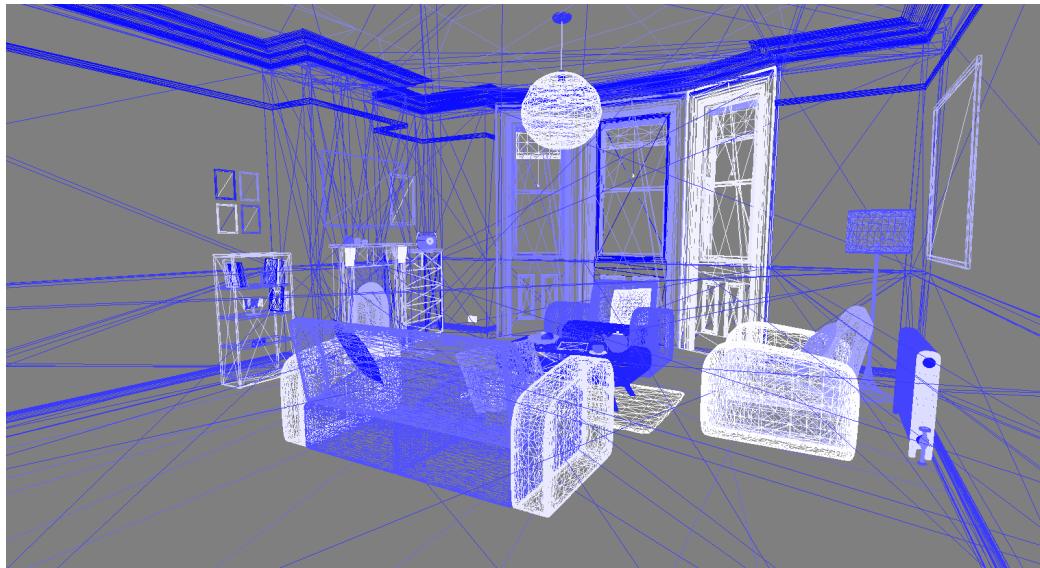
### Načítanie ciest vstupných súborov

Po zvolení cesty ku vstupnému súboru sa pred začatím parsovania skontroluje formát súboru. Pri XML sa kontroluje, či knižnica na čítanie XML dokáže prečítať súbor. Ak ho nedokáže prečítať, tak súbor nie je well-formed a teda sa nemôže použiť pri parsovaní. Pri súbori s mračnom bodov sa kontroluje prvý riadok súboru. Ak sa úspešne prečíta tak sa predpokladá že celý súbor je v poriadku.

Po skontrolovaní súborov sa nastavia boolean premenné `validPC` a `validXML` na hodnotu podľa toho či súbor prešiel kontrolou alebo nie. Načítavanie dát zo súborov sa vykoná len v prípade, že hodnoty v týchto premenných sú `True`. Aplikácia vie načítať aj len objekty alebo len mračno bodov. Niektoré funkcie však budú nedostupné. `Padding distance` sa vkladá priamo do premennej pre výplň vo funkcií na filtrovanie bodov v sekcií 4.1. Po načítaní vstupných súborov sa nastavia boolean premenné `SceneLoaded` a `CloudLoaded`. Bez toho aby tieto premenné boli v hodnote `True` sa nezobrazia objekty v scéne. Pri uvoľňovaní dát sa tiež uvoľnia len dátá, ktoré sú načítané pomocou týchto premenných.

### Základné nastavenia scény

Základné nastavenia scény sú nastavenia, ktoré sa vzťahujú na všetky objekty na scéne, alebo na scénu samotnú. Hodnota `Movement Speed` sa priamo posielá do funkcie na počítanie pozície a pohľadu kamery v sekcií 4.2. Hodnota škálovania sa posielá do shadera



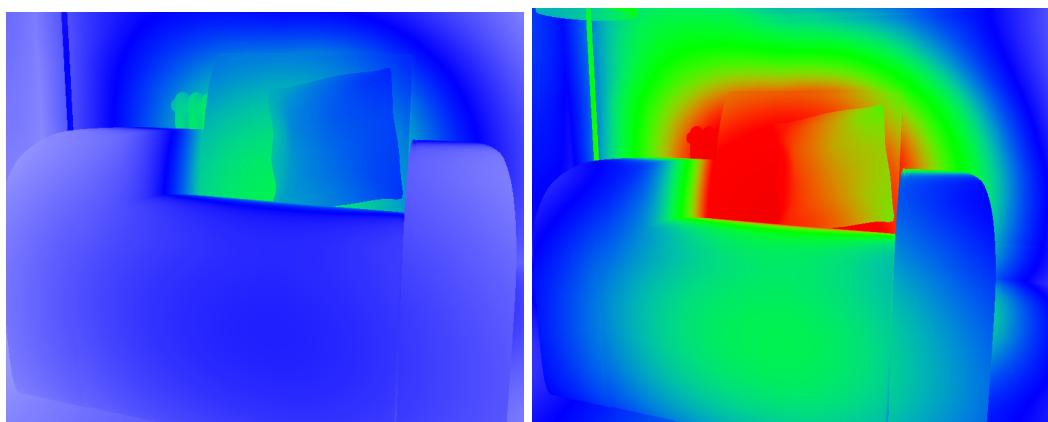
Obr. 4.10: Wireframe render scény

pomocou `uniform` premennej. Z tejto hodnoty sa zostaví matica škálovania, ktorá je pred maticou modelu pri počítaní pozície vrcholu na obrazovke.

Ak je checkbox `Vsync` zaškrtnutý spustí sa funkcia `glfw.swap_interval(1)`. Parameter funkcie značí kolko refreshov displeja má čakať `glfw` na zmenu snímku. Ak je to 0, snímka sa po renderovaní hned vykreslí. Ak je to 1, ako v tomto prípade, čaká sa na refresh monitora na vykreslenie snímky. Checkbox `Wireframe` mení hodnotu v premennej `RenderType` viditeľnej v sekciách 4.5 a 4.6. V tejto premennej sa nachádza typ vykreslovania primitívov v OpenGL a teda makrá `GL_LINES` a `GL_TRIANGLES`. Ak je použité makro `GL_LINES` tak sa vykreslujú len čiary medzi vrcholmi viditeľné v obrázku 4.10.

### Nastavenia objektov tepelných máp

Objekty sa zobrazujú len v prípade ak checkbox `Show Scene` je zaškrtnutý. Bez zaškrnutia `Show Heatmap` sa zobrazuje scéna zafarbená odtieňmi modrej. Ak sa zaškrte zobrazuje



Obr. 4.11: Rozdiel medzi intenzitou tepelnej mapy 100 a 50

sa nad objektami tepelná mapa. Intenzita a vzdialenosť počítania sa posiela do `uniform` premenných v shaderi a používajú sa v kóde v sekcií 4.6. Čím je hodnota intenzity menšia, tým je efekt bodu na zafarbenie väčší. Je to kvôli tomu že sa týmto číslom delí základná hodnota efektu bodu. Rozdiel medzi intenzitami tepelnej mapy je na obrázku 4.11.

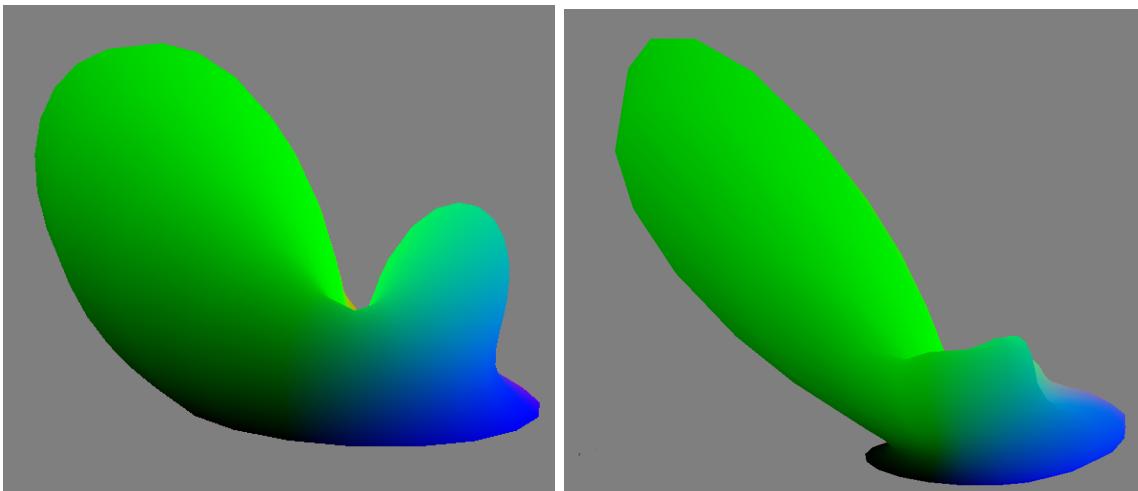
### Nastavenia mračna bodov

Zobrazovanie bodov funguje rovnako ako zobrazovanie bodov. Mračno sa zobrazí ak sa zaškrte Show points. Ak chce užívateľ zobraziť hemisférické funkcie, musia byť zaškrtnuté oba checkboxy. Veľkosť bodu sa posiela do `uniform` premennej v shaderi s ktorou sa násobí poloha vrcholu pri počítaní jeho pozícia na obrazovke alebo polomer HSH funkcie. Rovnako aj farba bodu sa posiela do shadera ako `uniform` a používa sa na zafarbenie vrcholu objektu bodu. Kód pri ktorom sa tieto `uniform` premenné používajú je v sekcií 4.5.

### Nastavenia hemisférických funkcií

Posuvník `m_bands` posiela hodnotu do `uniform` premennej, ktorá určuje počet cyklov vo `for` cykle pri počítaní hodnoty HSH. Čím je hodnota väčšia, tým sa pre každý vrchol hemisféry počíta s väčším zanorením koeficientov a zobrazenie funkcie je tým pádom presnejšie. Kód kde sa táto premenná používa je v sekcií 4.5. Rozdiel medzi úrovňami `m_bands` je na obrázku 4.12. Typ zafarbenia HSH funkcií je v posuvníku `HSHColor` kde na 0 zafarbuje podľa smeru normálu a na 1 podľa hodnoty HSH funkcie na vrchole. Tiež sa to posiela do shaderu ako `uniform` premenná. Obrázok, ktorý zobrazuje rozdiel medzi týmito zafarbeniami je obrázok 4.6 v sekcií 4.5.

Ked sa zmení počet stĺpcov a riadkov v posuvníkoch `nCols` a `nRows`, vygeneruje sa nový hemisférový objekt s týmto počtom stĺpcov a riadkov. Generovanie hemisférového objektu je opísané v sekcií 4.5.



Obr. 4.12: Rozdiel medzi `m_bands` 3 a 6 pri HSH

## 4.8 Testovanie

Táto sekcia sa venuje testovaniu implementácie funkcií aplikácie. Testovanie bolo vykonané na počítači s procesorom AMD Ryzen 5 5600H a s grafickou kartou NVIDIA RTX 3060 mobile. Informácie o testovaných scénach sú v tabuľke 4.1.

Scéna	Počet objektov	Počet bodov v mračne
<i>kitchen</i>	292	8123
<i>cornel box</i>	8	1792
<i>living room</i>	185	5246
<i>spaceship</i>	92	15934
<i>staircase</i>	27	1421
<i>veach-ajar</i>	22	4771
<i>veach-bidir</i>	14	1998

Tabuľka 4.1: Detaily testovaných scén

## Načítanie dát

Tabuľka 4.2 obsahuje časy načítania jednotlivých častí scén. Časy, ktoré sú v tabuľke, sú aritmetickým priemerom piatich meraní pre zaručenie presnosti a zamedzenie anomálií. Každý načítaný objekt v scéne má rozdielny počet vrcholov a teda aj rozdielny čas načítania. Toto môže byť jeden z faktorov, prečo sa niektoré scény s menším počtom objektov načítavajú dlhšie. Ďalší zaujímavý údaj je čas načítania mračna bodov v scéne **spaceship**. Kedže každý bod má rovnaký počet hodnôt v riadku, predpokladá sa že rozdiel v časoch bude lineárne stúpať podľa počtu bodov v mračne. V tomto prípade je to však až príliš veľký rozdiel. Je možné, že vzniká spomalenie pri kopírovaní polí interpretom jazyka Python počas vkladania nových hodnôt. Časy filtrovania bodov celkom zodpovedajú časom načítania objektov a počtom bodov v mračne.

Scéna	Body	Objekty	Filtrovanie	Spolu
<i>kitchen</i>	0,652	3,977	2,436	<b>7,065</b>
<i>cornel box</i>	0,079	0,096	0,029	<b>0,204</b>
<i>living room</i>	0,292	1,774	0,916	<b>2,982</b>
<i>spaceship</i>	3,236	1,559	1,339	<b>6,134</b>
<i>staircase</i>	0,070	0,102	0,040	<b>0,212</b>
<i>veach-ajar</i>	0,350	1,043	0,182	<b>1,575</b>
<i>veach-bidir</i>	0,094	0,039	0,033	<b>0,166</b>

Tabuľka 4.2: Trvanie načítania scén v sekundách

## Testovanie tepelných máp

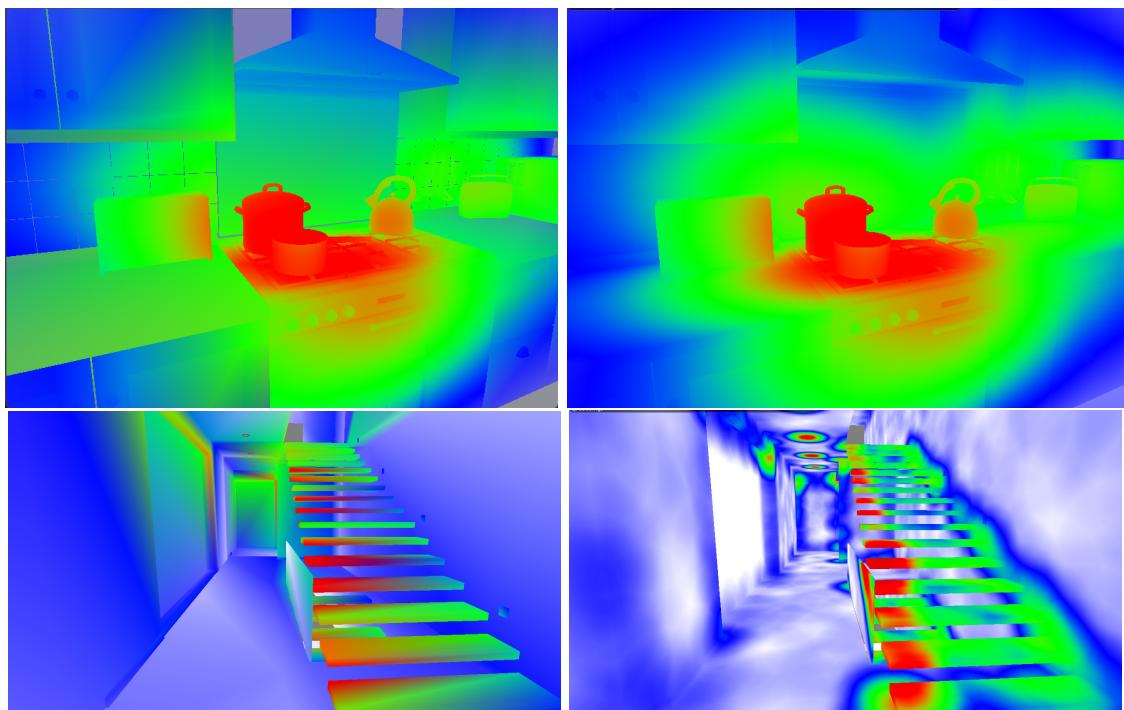
Tabuľka 4.3 znázorňuje počet snímkov za sekundu pri vykreslovaní scén pri žiadnej tepelnej mape ( $\emptyset$ ), pri tepelnej vykreslenej na vrcholoch (V) a pri tepelnej mape vykreslenej na fragmentoch (F) na rôznych rozlíšeniach. Snímky za sekundu boli vypočítané počítaním vykreslených snímkov behom piatich sekúnd a následným vydelením počtu snímkov s prejdenným časom. Všetky hodnoty boli počítané na rovnakom mieste v scéne a s rovnakými

Scéna	$\emptyset$	V	F(1920x1080)	F(1280x720)	F(800x600)
kitchen	183,338	18,767	10,441	14,616	27,869
cornel box	788,406	821,493	169,259	353,969	499,285
living room	249,655	89,472	9,739	19,91	39,224
spaceship	393,77	17,18	6,995	13,668	21,642
staircase	650,74	715,719	42,779	90,189	174,087
veach-ajar	679,519	198,158	12,776	27,654	50,598
veach-bidir	760,449	768,991	25,927	58,198	105,662

Tabuľka 4.3: Snímky za sekundu pri rôznych módoch tepelnej mapy

nastaveniami tepelnej mapy. Tepelná mapa vykreslená na fragmentoch má viac políčok v tabuľke pretože väčšie rozlíšenie znamená viac fragmentov pre ktorých je potrebné tepelnú mapu vypočítať. Čím viac fragmentov je teda na obrazovke, tým je výpočet mapy náročnejší. Pri vykreslení na vrcholoch sa počet vrcholov v scéne nemení a preto nie je potrebné mať viac políčok v tabuľke. Výkon tohto typu tepelnej mapy sa takmer nemení pri zmene rozlíšenia.

Tepelná mapa farbená na fragmentoch je presnejšia ako tepelná mapa farbená na vrcholoch. Zároveň je však náročnejšia na grafiku pri vyšších rozlíšeniach. Výkon sa pri tepelnej mape na vrcholoch sa rozlíšením nemení a preto je vhodnejšia pri vyšších rozlíšeniach. Na obrázkoch 4.13 sú znázornené rozdiely medzi módmi vykreslenia tepelnej mapy. Čím je počet vrcholov na objektoch vyšší, tým je presnejšia aj tepelná mapa vykreslená na vrcholoch avšak tepelná mapa na fragmentoch bude vždy presnejšia. Ak sa objekt skladá z mála trojuholníkov tepelná mapa na vrcholoch nie je presná vôbec, keďže sa farby interpolujú



Obr. 4.13: Rozdiel medzi farbením tepelnej mapy na vrchol (vľavo) a na fragment (vpravo)

medzi fragmentami na celom objekte. Takisto z týchto interpolovaných farieb môžu vznikať ostatné farby ako fialová alebo žltá, ktorá by sa nemala vykreslovať.

### Testovanie hemisférických funkcií

V tabuľke 4.4 sú znázornené počty snímkov za sekundu pri vykreslovaní HSH funkcií na rôznych scénach a na rôznych úrovniach zanorenia HSH funkcií. Počítanie snímkov za sekundu prebiehalo podobne ako pri tepelných mapách. Úroveň zanorenia znázorňuje počet koeficientov použitých pri počítaní HSH funkcie. Čím viac koeficientov HSH funkcie je použitých pri počítaní, tým je render funkcie presnejší. HSH funkcie boli vykreslované na pologuli s 15 stĺpcami a 15 riadkami. Obrázok 4.14 ukazuje rozdiely medzi jednotlivými úrovňami zanorenia funkcie. Hodnoty sú zlava doprava od 1 do 6.

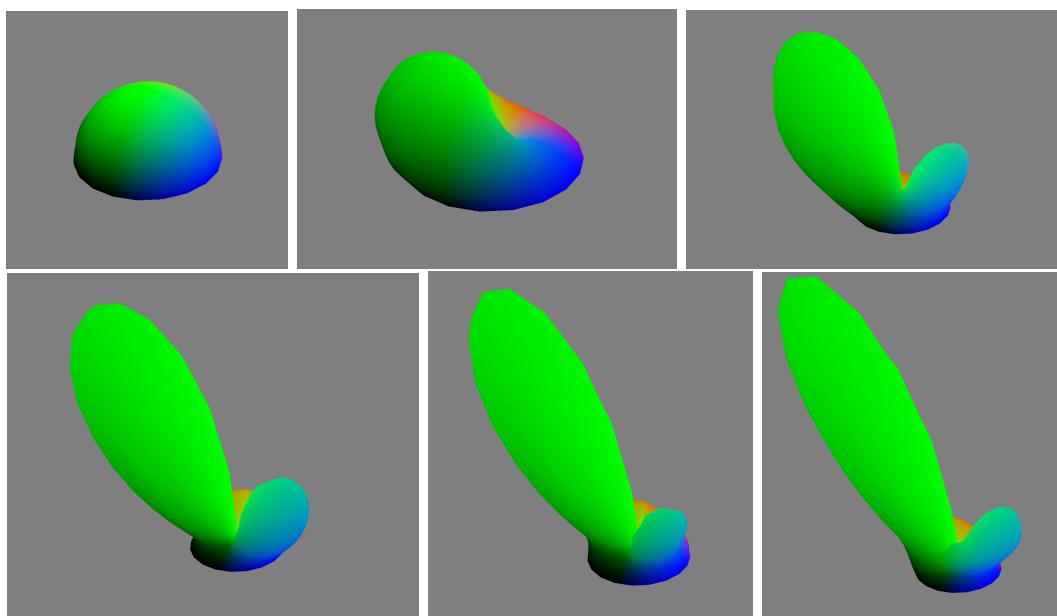
Scéna	$l \leq 1$	$l \leq 2$	$l \leq 3$	$l \leq 4$	$l \leq 5$	$l \leq 6$
kitchen	180,671	176,672	146,179	56,643	27,051	15,113
cornel box	790,585	790,165	654,988	257,625	123,251	68,038
living room	249,673	248,603	219,576	85,372	41,187	23,091
spaceship	247,067	182,011	79,332	29,844	14,257	7,879
staircase	657,413	649,275	645,888	311,391	149,704	81,638
veach-ajar	702,713	573,983	250,216	94,586	46,221	25,38
veach-bidir	759,331	727,195	604,242	222,989	106,85	58,195

Tabuľka 4.4: Snímkы za sekundu pri úrovniach zanorenia hemisférických funkcií

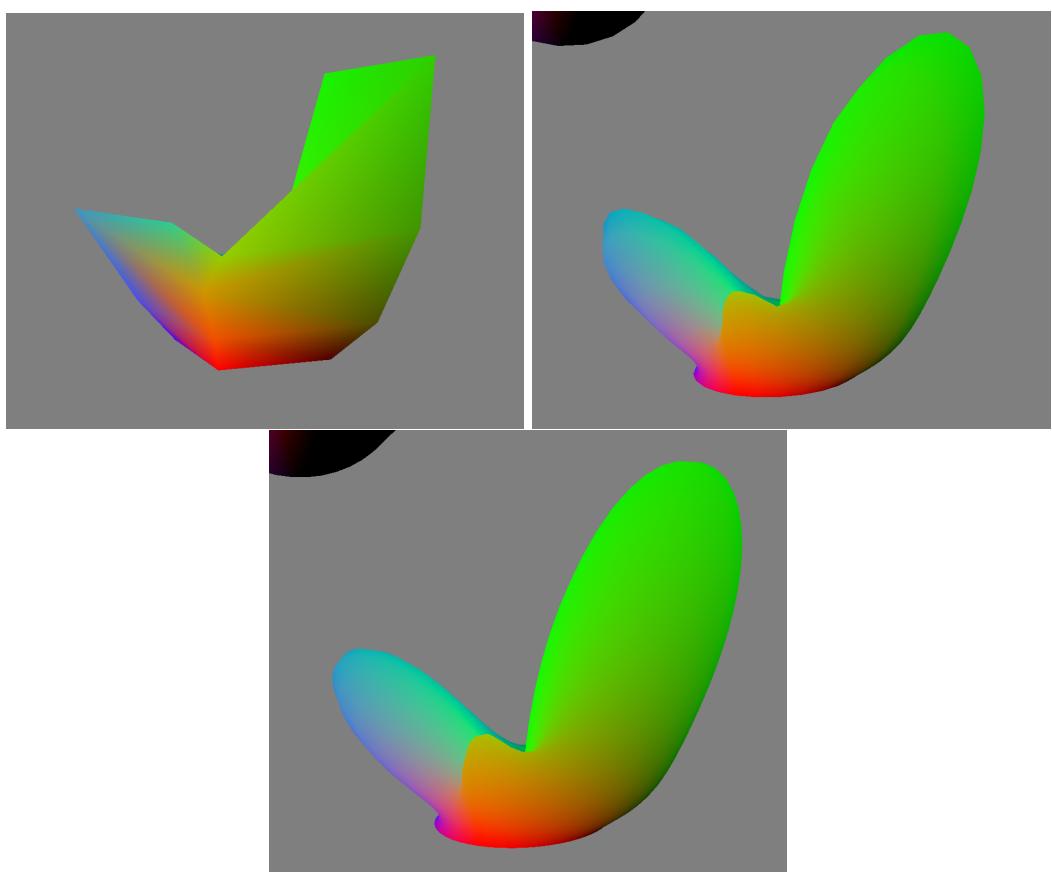
V ďalšej tabuľke 4.5 sú zobrazené snímkы za sekundu pri rôznych počtoch stĺpcov a riadkov hemisféry na ktorej sa zobrazuje HSH funkcia. Počet riadkov a stĺpcov určuje počet vrcholov objektu s ktorými sa bude počítať HSH funkcia. Čím väčší počet vrcholov, tým je výpočet na grafiku náročnejší ale výsledná vykreslená funkcia presnejšia. Na obrázku 4.15 sú zobrazené tri nastavenia kvality hemisféry. Prvé je minimálna hodnota  $5 \times 5$ , druhé je  $25 \times 25$  a posledné je maximálna hodnota  $50 \times 50$ .

Scéna	<b>10 × 10</b>	<b>20 × 20</b>	<b>30 × 30</b>	<b>40 × 40</b>	<b>50 × 50</b>
kitchen	181,195	87,326	40,840	23,843	15,518
cornel box	809,878	396,454	185,53	108,111	69,351
living room	243,091	133,124	62,772	35,995	23,395
spaceship	159,322	46,893	21,652	12,411	8,112
staircase	668,287	483,074	227,052	128,49	82,758
veach-ajar	491,597	147,819	68,692	39,561	26,058
veach-bidir	786,9	348,775	162,905	92,307	59,616

Tabuľka 4.5: Snímkы za sekundu pri rôznych kvalitách hemisférového objektu



Obr. 4.14: Úrovne zanorenia HSH funkcií



Obr. 4.15: Rozdiely medzi kvalitou hemisférového objektu

# Kapitola 5

## Záver

Cieľom práce bolo načítať grafické dátá používané vo vedeckej oblasti ako mračná bodov alebo hemisférické harmonické funkcie a zobraziť ich nad scénami zo vstupných súborov pre Mitsuba renderer. Následne bolo cieľom zobraziť závislosť medzi mračnom bodov a objektami v scéne pomocou tepelných máp. Tento cieľ bol splnený.

Teoretické vedomosti ohľadom dátových typov a operácií nad nimi boli naštudované a opísané v kapitole 2. Následný spôsob vizualizácie týchto dát a operácií bol navrhnutý v sekciách 3.1 a 3.2. Implementácia návrhov bola detailne popísaná v kapitole 4.

Aplikácia dokáže načítať a zobraziť mračno bodov z vysoko konkrétneho formátu súboru, ktorý nieje verejne používaný. Následne vie načítať scénu zo vstupného formátu pre Mitsuba renderer a uložiť a zobraziť objekty formátu Wavefront .obj. Aplikácia vie zobraziť hemisférické harmonické funkcie nad bodmi z mračna. Takisto vie zobraziť tepelné mapy nad objektami podľa počtu bodov v blízkosti objektu. Tieto operácie sú ovládané a upravovateľné používateľom vdaka užívateľskému rozhraniu.

Počas práce som sa naučil spôsoby ukladania a renderovania objektov na grafickej karte. Následne som sa naučil o mračnách bodov a o shadovacích technikách pri farbení tepelných máp. Veľa som sa naučil aj o hemisférických harmonických funkciách a o ich použití v grafickom odbore.

Rozšírenia práce by mohli zahrňovať viaceré funkcie, napríklad analýzu paprskov svetla v scéne. Takisto by sa mohlo optimalizovať vykreslovanie tepelných máp. Upraviť by sa mohlo aj filtrovanie bodov pre jednotlivé objekty, ktoré by mohlo lepšie určovať body v blízkosti vrcholov objektu a tým znížiť zataženie grafických kariet pri renderovaní tepelných máp. Ďalším vylepšením by mohlo byť pridanie možností nastavenia operácií v užívateľskom rozhraní, alebo optimalizácia načítania bodov v mračne pri mračnách s väčším objemom dát.

# Literatúra

- [1] AKENINE MÖLLER, T., HAINES, E., HOFFMAN, N., PESCE, A., IWANICKI, M. et al. *Real-Time Rendering*. 4. vyd. 2018. ISBN 1138627003.
- [2] BOJKO, A. A. Informative or Misleading? Heatmaps Deconstructed. In: *Human-Computer Interaction. New Trends*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, sv. 5610, č. 1, s. 30–39. Lecture Notes in Computer Science. ISBN 3642025730.
- [3] CELES, W. a ABRAHAM, F. Texture-Based Wireframe Rendering. In: *2010 23rd SIBGRAPI Conference on Graphics, Patterns and Images*. IEEE, 2010, s. 149–155. ISBN 9781424484201.
- [4] GAUTRON, P., KRIVANEK, J., PATTANAIK, S. a BOUATOUCH, K. A Novel Hemispherical Basis for Accurate and Efficient Rendering. In: Goslar, DEU: Eurographics Association, 2004, s. 321–330. EGSR’04. ISBN 3905673126.
- [5] GEEKSFORGEEKS. *Search and Insertion in K Dimensional tree* [online]. 2023 [cit. 2023-04-17]. Dostupné z: <https://www.geeksforgeeks.org/search-and-insertion-in-k-dimensional-tree/>.
- [6] JAKOB, W., SPEIERER, S., ROUSSEL, N., NIMIER DAVID, M., VICINI, D. et al. *Mitsuba renderer 3 Documentation* [online]. 2022 [cit. 2023-04-19]. Dostupné z: <https://mitsuba.readthedocs.io/en/stable/>.
- [7] JAKOB, W., SPEIERER, S., ROUSSEL, N., NIMIER DAVID, M., VICINI, D. et al. *Mitsuba renderer 3 GitHub project* [online]. 2022 [cit. 2023-04-19]. Dostupné z: <https://github.com/mitsuba-renderer/mitsuba3/blob/master/README.md>.
- [8] JO, S., JEONG, Y. a LEE, S. GPU-Driven Scalable Parser for OBJ Models. *Journal of computer science and technology*. New York: Springer US. 2018, zv. 33, č. 2, s. 417–428. ISSN 1000-9000.
- [9] KHARROUBI, A., POUX, F., BALLOUCH, Z., HAJJI, R. a BILLEN, R. Three Dimensional Change Detection Using Point Clouds: A Review. *Geomatics*. 2022, zv. 2, č. 4, s. 457–485. ISSN 2673-7418. Dostupné z: <https://www.mdpi.com/2673-7418/2/4/25>.
- [10] KRASNOPOROSHIN, V. a MAZOUKA, D. Graphics Pipeline Evolution Based on Object Shaders. *Pattern recognition and image analysis*. 1. vyd. Pleiades Publishing. 2020, zv. 30, č. 2, s. 192–202. ISSN 1054-6618.

- [11] LEE, K. W. a BO, P. Feature curve extraction from point clouds via developable strip intersection. *Journal of Computational Design and Engineering*. 1. vyd. 2016, zv. 3, č. 2, s. 102–111. DOI: <https://doi.org/10.1016/j.jcde.2015.07.001>. ISSN 2288-4300. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S2288430015000639>.
- [12] *PyOpenGL for OpenGL Programmers* [online]. [cit. 2023-04-14]. Dostupné z: [https://pyopengl.sourceforge.net/documentation/opengl\\_diffs.html](https://pyopengl.sourceforge.net/documentation/opengl_diffs.html).
- [13] RYS, M. XML Document. In: LIU, L. a ÖZSU, M. T., ed. *Encyclopedia of Database Systems*. New York, NY: Springer New York, 2018, s. 4748–4748. ISBN 978-1-4614-8265-9.
- [14] SALOMON, D. *Transformations and projections in computer graphics*. 1. vyd. London: Springer, 2006. ISBN 1-84628-392-2.
- [15] SCHMEDER, A. W. *Spherical directivity resonance synthesis* [Conference paper]. 2010 [cit. 2023-04-19]. Dostupné z: [http://ambisonics10.ircam.fr/drupal/files/Proceedings/presentations/013\\_46.pdf](http://ambisonics10.ircam.fr/drupal/files/Proceedings/presentations/013_46.pdf).
- [16] SHREINER, D., SELLERS, G., KESSENICH, J. a LICEA KANE, B. *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. 8. vyd. 2013. ISBN 978-0-321-77303-6.
- [17] SINGH, P. K., VESELOV, G., VYATKIN, V., PLJONKIN, A., DODERO, J. M. et al. Optimization of K-Nearest Neighbors for Classification. In: *Communications in Computer and Information Science*. Singapore: Springer Singapore Pte. Limited, 2021, sv. 1395, s. 205–214. Communications in Computer and Information Science. ISBN 9811614792.
- [18] WIKIPEDIA CONTRIBUTORS. *Heat map — Wikipedia, The Free Encyclopedia* [online]. 2023 [cit. 2023-04-17]. Dostupné z: [https://en.wikipedia.org/w/index.php?title=Heat\\_map&oldid=1147382059](https://en.wikipedia.org/w/index.php?title=Heat_map&oldid=1147382059).

## Príloha A

### Obsah priloženého média

Príloha obsahuje dátovú štruktúru priloženého média a informácie o jeho súboroch a priečinkoch.

- `xduric05.pdf` – PDF verzia bakalárskej prace
- `xduric05/` – Zdrojové kódy pre textovú časť bakalárskej práce
- `obj/` – .obj súbory pre predom určené tvary objektov
- `shaders/` – Shadery pre aplikáciu
- `scenes/` – Demo scény pre program
- `camera.py` – Kamerový modul
- `FileDialog.py` – Modul na vizuálny výber vstupných súborov
- `objLoader.py` – Modul na čítanie .obj súborov
- `read_point_cloud.py` – Modul na parsovanie súborov s mračnom bodov
- `read_xml_scene.py` – Modul na parsovanie XML súborov
- `main.py` – Hlavný kód aplikácie
- `README.txt` – Informácie o aplikácii, inštalácii a spustení
- `requirements.txt` – Prerekvizity programu použité pri inštalácii
- `video.mp4` – Demonštračné video