Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Zobrazovanie viacrozmerných funkcií na GPU

Diplomová práca

2025
Bc. Adrián Kocifaj

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Zobrazovanie viacrozmerných funkcií na GPU

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Mgr. Andrej Mihálik, PhD.

Bratislava, 2025
Bc. Adrián Kocifaj

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Adrián Kocifaj
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Zobrazovanie viacrozmerných funkcií na GPU.
Multidimensional data rendering.

Anotácia: Nie je jednoduché zobraziť objekty v priestore s dimenziou viac ako 3. Keď však uvážime pevný 3D objekt, tak okrem rozmerov v 3D priestore, vykazuje jeho vzhľad aj ďalšie atribúty ako je napríklad farba alebo priehľadnosť. S použitím volumetrického zobrazovania by sa takto dal priestor rozšíriť za hranicu 3D.

Cieľ: Vytvoriť rozhranie na vizualizáciu viacrozmerného priestoru pomocou volumetrického zobrazovania.

Literatúra: Alan Norton, Generation and Display of Geometric Fractals in 3-D, ACM SIGGRAPH Computer Graphics, vol. 16, no. 3, pp. 61–67, 1982.

Vedúci: Mgr. Andrej Mihálik, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 04.11.2024

Dátum schválenia: 08.11.2024 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Predovšetkým musím spomenúť svojho školiteľa, pána Mgr. Andreja Mihálika, PhD., ktorému ďakujem za vytvorenie témy a za odborné vedenie počas celého procesu.

Abstrakt

Nie je jednoduché zobraziť viacrozmerné objekty, pretože ľudské vnímanie je obmedzené na priestorové videnie. Táto práca sa zaoberá vizualizáciou viacrozmerných funkcií do priestoru, pre vyššie dimenzie funkcií využíva iné vlastnosti, ako sú farba a priehľadnosť. V práci sa zaoberáme vizualizáciou volumetrických dát rôznymi technikami, ich výpočtová zložitosť a pridané dimenzie vyžadujú použitie paralelizmu grafických kariet. Navrhli a implementovali sme interaktívny softvérový systém, ktorý umožňuje toto zobrazenie v reálnom čase. Systém poskytuje používateľovi možnosť skúmať viacrozmerné funkcie, analyzovať ich štruktúru a správanie.

Kľúčové slová: viacrozmerné funkcie, vizualizácia dát, volumetrické zobrazovanie, voxel, oktálový strom

Abstract

It is not easy to display multidimensional objects because human perception is limited to spatial vision. This work deals with the visualization of multidimensional functions into space, for higher dimensions, it uses other properties such as color and transparency. In this thesis we deal with the visualization of volumetric data using different techniques, their computational complexity and added dimensions require the use of graphics card parallelism. We have designed and implemented an interactive software system that enables this visualization in real time. The system provides the user with the ability to explore multidimensional functions, analyzing their structure and behavior.

Keywords: multidimensional functions, data visualization, volumetric rendering, voxel, octree

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Kapitola 1

Východiská

1.1 Úvod do problematiky

Zobrazovanie rôznych typov dát a matematických objektov patrí medzi základné úlohy počítačovej grafiky. Bežné grafické systémy nám umožňujú pracovať s dvojrozmernými a trojrozmernými objektmi, ktoré človek dokáže intuitívne spracovať.

Problemy nastávajú, keď sa presunieme na vyšší počet dimenzií. Tento prechod prináša nasledujúce výzvy:

- **Ľudská percepcia** -– Ľudské oko a mozog sú prispôsobené vnímať iba trojrozmerný priestor. Zobrazenie vyšších dimenzií rovnakým spôsobom nie je možné. Vyžaduje sa interpretácia dodatočných dimenzií pomocou iných vizuálnych atribútov, ako sú farba, jas, priehľadnosť a podobne.

- **Výpočtová zložitosť** – S rastúcim počtom dimenzií rastie výpočtová zložitosť exponenciálne. Na zachovanie interaktivity vizualizácie je nutné využiť najnovšie softvérové a hardvérové postupy. Najlepším súčasným riešením sú grafické karty, ktoré dokážu nezávisle spúšťať milióny paralelných výpočtov.

- **Pamäťová zložitosť** – Podobne ako výpočtová zložitosť, aj pamäťová zložitosť rastie exponenciálne. Z tohto dôvodu je nevyhnutné využiť efektívne dátové štruktúry a techniky kompresie.

Dôležitú úlohu zohráva finálna interpretácia výsledkov. Neexistuje jedno správne zobrazenie a rôzne techniky môžu zdôrazňovať odlišné vlastnosti dát. Finálna vizualizácia si vyžaduje správne zvolenie vizualizačnej techniky, interpretácie výsledkov a optimalizačných metód.

1.2 Funkcie

Funkcia alebo zobrazenie z množiny A do množiny B je pravidlo, ktoré každému prvku $x \in A$ priraďuje presne jeden prvok $y \in B$, formálne zapisujeme

$$f : A \to B.$$

Tento prvok y sa nazýva hodnota funkcie f v bode x a zapisuje sa

$$y = f(x).$$

Množina A sa nazýva definičný obor funkcie a označuje sa $D(f)$. Množina všetkých hodnôt $f(x)$ pre $x \in A$ sa nazýva obor hodnôt funkcie a označuje sa $H(f)$.

1.2.1 Viacrozmerné funkcie

Viacrozmerná funkcia je funkcia, ktorej definičný obor tvorí množina vektorov

$$x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n.$$

Hodnota funkcie môže byť buď reálne číslo alebo vektor

$$y \in \mathbb{R},$$
$$y = (y_1, \ldots, y_m) \in \mathbb{R}^m.$$

Formálne zapisujeme

$$f : \mathbb{R}^n \to \mathbb{R}^m, \quad x \mapsto f(x).$$

Viacrozmerné funkcie sú prirodzeným rozšírením jednorozmerných funkcií, kde namiesto jednotlivých čísel môžeme pracovať s vektormi ako vstupmi a výstupmi.

1.3 Volumetrické zobrazovanie

Na rozdiel od povrchového zobrazovania, kde sú objekty reprezentované len svojím vonkajším obalom pomocou polygónov, sa volumetrické zobrazovanie odlišuje tým, že nepracuje len s hranicami telies, ale zobrazuje aj vnútorné dáta objektov [3].

1.3.1 Voxel

Základným prvkom 2D grafiky je pixel a jeho ekvivalentom v priestorovej grafike je voxel. Voxel predstavuje najmenšiu rozlíšiteľnú časť objemových dát v priestore [3].

Každý voxel nesie informáciu o svojej pozícii v priestore so súradnicami (x, y, z). Ďalej môže obsahovať ľubovoľne zvolené atribúty, ako sú napríklad priehľadnosť, farba a podobne.

1.3.2 Reprezentácia dát

Pravidelná mriežka

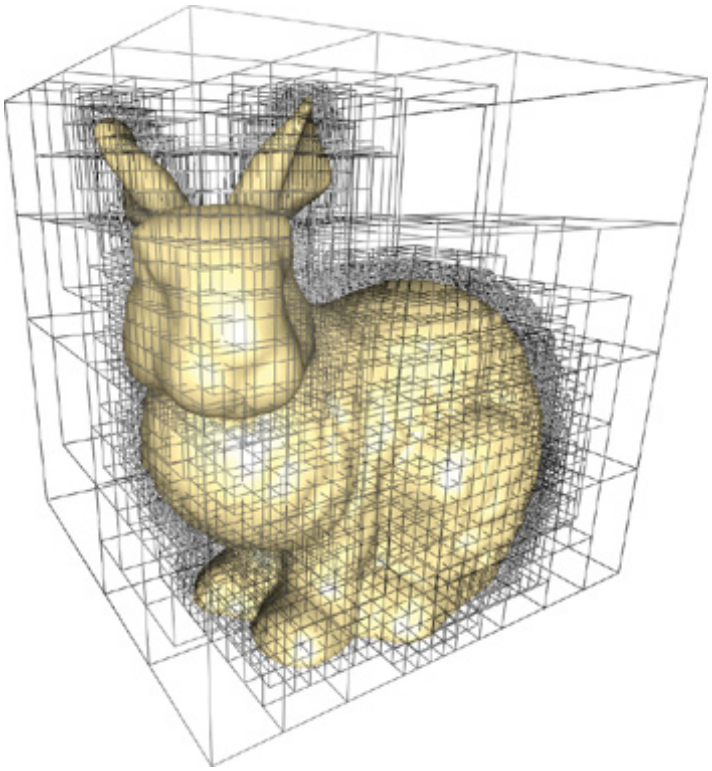
Najjednoduchším spôsobom uloženia je trojrozmerné pole. Dáta sú organizované lineárne, čo umožňuje priamy prístup k hodnotám. Použitím trojrozmerného poľa sú súradnice voxelov definované priamo indexmi poľa.

Výhodou tohto prístupu je jednoduchá implementácia, zatiaľ čo nevýhodou je vysoká pamäťová náročnosť, keďže ukladanie prázdneho priestoru je neefektívne.

Oktálový strom

Pre dáta, kde veľkú časť objemu tvorí prázdny priestor, je efektívne použiť hierarchickú dátovú štruktúru – oktálový strom. Priestor sa rekurzívne delí na osem podriadených oktantov. Delenie pokračuje, kým sa nedosiahne úroveň samostatného voxelu alebo kým nie je oktant homogénny [6].

Táto štruktúra umožňuje preskočiť veľké oblasti prázdneho priestoru, čím výrazne urýchľuje výpočet prechodu štruktúrou a šetrí pamäť.



Obr. 1.1: Vizualizácia delenia priestoru pomocou oktálového stromu [7].

1.3.3 Prechod priestorom

Pri zobrazovaní metódou sledovania lúčov je nutné efektívne nájsť prienik lúča s voxelmi. Naivné vzorkovanie s konštantným krokom môže byť pomalé alebo nepresné, pričom často vznikajú vizuálne artefakty.

Prechod pravidelnou mriežkou

Fast Voxel Traversal Algorithm je štandardná technika používaná pri prechode lúča pravidelnou mriežkou. Ide o rozšírenie algoritmu DDA (*Digital Differential Analyzer*) do trojrozmerného priestoru [1].

Princíp spočíva v inkrementálnom posune lúča z jedného voxelu do susedného. Algoritmus pre efektivitu udržiava pre každú os x, y a z dve kľúčové premenné:

- t_{max} – aktuálna hodnota parametra t, pri ktorej lúč pretne najbližšiu hranicu voxelu v danej osi.
- t_{delta} – konštantná hodnota, o ktorú sa zvýši parameter t pri prechode cez celý voxel v danej osi.

V každom kroku algoritmus porovnáva hodnoty $t_{maxX}, t_{maxY}, t_{maxZ}$ a vyberie os s minimálnou hodnotou. Následne posunie index voxelu v tomto smere a inkrementuje príslušné t_{max} o hodnotu t_{delta}. Tým je zaručené, že lúč navštívi každý pretínaný voxel v správnom poradí bez vynechania alebo duplicity [1].

Prechod oktálovým stromom

Na rozdiel od pravidelnej mriežky s konštantným krokom vyžaduje oktálový strom adaptívny prístup. Najčastejšie sa využíva parametrický algoritmus od autorov *J. Revelles*, *C. Urena*, *M. Lastra*.

Princíp algoritmu spočíva v tom, že poradie návštevy oktantov daného uzlu je vopred určené smerom lúča. Algoritmus na základe parametrického vyjadrenia lúča efektívne identifikuje, ktorými uzlami lúč skutočne prechádza. Toto umožňuje rýchlo preskakovať prázdne oblasti stromu bez zbytočných výpočtov, čo je kľúčové pre zachovanie vysokého výkonu na grafickej karte [9].

1.3.4 Techniky

Raycasting

Raycasting je technika priameho volumetrického zobrazovania, ktorá premieta trojrozmerné objemové dáta do dvojrozmerného obrazu. Pre každý pixel sa vysiela lúč z

kamery a ten pri prechode objemom postupne vzorkuje dáta, tie sa skomponujú do výslednej farby pixelu [8].

Spracovanie jedného lúča je popísané v nasledujúcich krokov:

- **Generovanie lúča** – Pre každý pixel sa vytvorí lúč podľa polohy kamery. Tento lúč určuje, ktorou časťou objemu sa bude prechádzať.
- **Vzorkovanie** – Lúč postupne prechádza objemom a vyberá hodnoty dát.
- **Klasifikácia** – Zo vzorkovaných hodnôt sa určia vizuálne vlastnosti, napríklad farba a priehľadnosť.
- **Kompozícia** – Vzorky pozdĺž lúča sa spájajú do jednej výslednej farby pixelu.

1.4 Kamera v priestore

V priestore je kamera reprezentovaná svojou pozíciou a trojicou ortogonálnych smerových vektorov — *front*, *up* a *right*. Tieto vektory určujú orientáciu kamery, teda smer pohľadu, smer nahor a smer doprava. Spolu vytvárajú lokálny súradnicový systém kamery, ktorý slúži pri generovaní lúčov aj pri transformácii súradníc objektov medzi priestormi.

Ovládanie orientácie kamery sa realizuje pomocou Eulerových uhlov [2]:

- **yaw** — rotácia okolo vertikálnej osi, ktorá mení smer, kam sa kamera otáča doľava alebo doprava,
- **pitch** — rotácia okolo horizontálnej osi, určujúca smer pohľadu nahor alebo nadol,
- **roll** — rotácia okolo osi lúča, ktorá ovplyvňuje náklon obrazu.

1.5 Transparentnosť nezávislá od poradia

Korektné vykresľovanie polopriehľadných objektov predstavuje v počítačovej grafike netriviálny problém. Pri štandardnom prístupe, známom ako *alpha blending*, je výsledná farba pixelu závislá od poradia, v akom sú fragmenty spracované. Pre správny výsledok je potrebné, aby boli transparentné objekty vykresľované v poradí od najvzdialenejších po najbližšie k pozorovateľovi.

Štandardný vzťah pre kompozíciu farieb je:

$$C_{out} = \alpha_{src} C_{src} + (1 - \alpha_{src}) C_{dst},$$

kde:

- C_{src} — farba práve spracovávaného fragmentu,
- α_{src} — priehľadnosť práve spracovávaného fragmentu,
- C_{dst} — aktuálna farba fragmentov ktoré už boli spracované,
- C_{out} — výsledná farba.

Keďže operácia nie je komutatívna, výsledná farba závisí od poradia spracovania fragmentov. Triedenie transparentných objektov je výpočtovo náročné a v prípade vzájomne sa pretínajúcich objektov aj problematické.

1.5.1 Weighted Blended Order-Independent Transparency

Jednou z najefektívnejších moderných metód je technika *Weighted Blended Order-Independent Transparency*, ktorú predstavili autori M. McGuire a L. Bavoil. Na rozdiel od presných metód, ktoré ukladajú všetky fragmenty do zoznamov, toto je aproximačná metóda. Jej hlavnou výhodou je vysoká rýchlosť, čo ju robí vhodnou pre použitie v reálnom čase [5].

Princíp metódy

Každý fragment prispieva do dvoch akumulátorov a do *revealage* textúry podľa vzorcov:

$$C_{\text{acc}} \mathrel{+}= C_i \, \alpha_i \, w_i,$$

$$A_{\text{acc}} \mathrel{+}= \alpha_i \, w_i,$$

$$R \mathrel{\times}= (1 - \alpha_i).$$

Kde jednotlivé veličiny znamenajú:

- C_i – farba práve spracovávaného fragmentu,
- α_i – opacita práve spracovávaného fragmentu,
- w_i – váha práve spracovávaného fragmentu,
- C_{acc} – akumulovaná vážená farba všetkých fragmentov,
- A_{acc} – akumulovaný súčet váh všetkých fragmentov,
- R – celková priepustnosť.

Výpočet výslednej farby

Po akumulácii všetkých fragmentov sa rekonštruuje výsledná farba a priehľadnosť:

$$C_{\text{transp}} = \frac{C_{\text{acc}}}{A_{\text{acc}} + \varepsilon}, \qquad \alpha_{\text{transp}} = 1 - R,$$

kde ε je malá konštanta zabraňujúca deleniu nulou.

1.6 Zobrazovací kanál

Zobrazovací kanál, po anglicky známy ako *rendering pipeline*, je základný proces v počítačovej grafike, ktorý zabezpečuje transformáciu 3D scén na 2D obrazy. Tento proces zahŕňa niekoľko etáp, ktoré postupne spracúvajú geometrické dáta, materiálové vlastnosti a ďalšie informácie. Konkrétne etapy a ich poradie sa môžu líšiť v závislosti od použitého softvéru a zvoleného spôsobu zobrazovania.

1.6.1 OpenGL

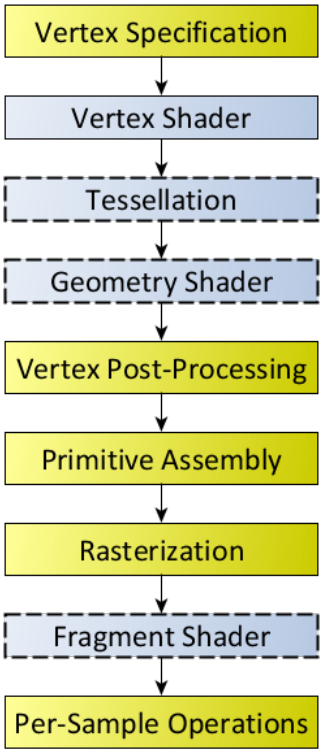
OpenGL je štandardizované rozhranie API (*Application Programming Interface*) pre 2D a 3D grafiku, ktoré umožňuje vývojárom na rôznych platformách vytvárať graficky náročné aplikácie. Využíva sa v mnohých odvetviach, ako sú vývoj hier, virtuálna realita, CAD systémy a vizualizácia vedeckých dát.

Hlavnou výhodou je schopnosť priamej spolupráce s grafickými kartami a využívanie ich akceleračných funkcií, čo umožňuje dosiahnuť vyšší výkon [4]. OpenGL zobrazovací kanál a jeho jednotlivé etapy možno vidieť na obrázku 1.2.

Shader programy

V OpenGL sú shader programy neoddeliteľnou súčasťou zobrazovacieho procesu. Sú to programy napísané v jazyku GLSL (*OpenGL Shading Language*), ktoré umožňujú priamu manipuláciu s grafikou na hardvérovej úrovni [4]. Existujú rôzne typy shader programov, pričom každý je zameraný na špecifickú fázu vykresľovacieho procesu:

- **Vertex shader** sa zaoberá spracovaním jednotlivých vrcholov a vykonáva transformácie vrcholov do priestoru po projekcii, môže byť použitý aj pre iné operácie s vrcholmi, ktoré pokračujú do ďalších etáp.
- **Tessellation shader** umožňuje vytvárať vyšší stupeň detailov na geometrických tvaroch bez potreby zvyšovania počtu vrcholov v pôvodnom modeli, a to pomocou rozdeľovania vrcholových dát na menšie primitíva.
- **Geometry shader** umožňuje spracovanie celej primitívy, ako sú trojuholníky alebo čiary, môže zmeniť počet a tvar primitív alebo generovať novú geometriu.



Obr. 1.2: OpenGL zobrazovací kanál [7].

- **Fragment shader** je kľúčový pri vizualizácii finálneho obrazu, pretože na pixelovej úrovni manipuluje s jednotlivými fragmentami a ich vlastnosťami, ako sú farba a hĺbka. Každý fragment je spracovaný osobitne a nezávisle od ostatných, čo umožňuje vysokú úroveň paralelizácie.

- **Compute shader** je špeciálny typ shader programu v OpenGL, ktorý sa používa výhradne na výpočty nezávislé od tradičného renderovania grafiky. Tento shader program je ideálnym nástrojom pre komplexné vedecké výpočty, vďaka jeho efektívnemu spracovaniu dát a rýchlym výpočtom.

Uniformné premenné sú dôležitým prvkom shader programov, ktoré umožňujú prenos konštantných hodnôt z CPU do GPU. Tieto premenné sú definované v shader kóde a ich hodnoty zostávajú nemenné počas vykonávania shader programu pre všetky spracovávané vrcholy alebo fragmenty. Uniformné premenné sa často používajú na prenos maticových transformácií, svetelných parametrov alebo iných globálnych stavových informácií, ktoré sú potrebné na vykreslenie scény.

Proces presunu uniformných premenných z CPU do GPU zahŕňa niekoľko krokov, ako je získavanie lokácií uniformných premenných a nastavenie premenných. Tento proces umožňuje efektívne riadenie zobrazovacieho kanála a dynamické aktualizovanie parametrov renderovania bez potreby prerušovania práce GPU [4].

1.6.2 Vulkan

Vulkan je nízkoúrovňové API pre grafiku, ktoré poskytuje väčšiu kontrolu nad hardvérovými zdrojmi v porovnaní s OpenGL. Táto kontrola umožňuje vývojárom optimalizovať výkon a efektívnejšie spravovať pamäť, čo je kritické pre vysoko náročné grafické aplikácie. Vulkan podporuje širokú škálu platforiem, ako sú Windows, Linux a Android, čo z neho robí ideálne riešenie pre multiplatformový vývoj.

1.6.3 DirectX

DirectX je súbor API od Microsoftu, ktorý sa využíva na vývoj multimediálnych aplikácií, najmä hier, na platformách Windows. Obsahuje rôzne komponenty, z ktorých najznámejším je Direct3D pre 3D grafiku. Direct3D je špecificky navrhnutý pre optimalizáciu na systémoch Windows a poskytuje vývojárom kontrolu nad hardvérovými zdrojmi. Táto nízkoúrovňová kontrola umožňuje zlepšenie výkonu, ale zároveň zvyšuje jeho náročnosť.

Kapitola 2

Návrh

2.1 Výber technológie

2.1.1 Knižnice

2.2 Vytvorenie a konfigurácia projektu

2.3 Architektúra aplikácie

2.3.1 Diagramy

2.4 Používateľské rozhranie

2.4.1 Spracovanie používateľského vstupu

2.5 Manipulácia s dátami

Kapitola 3

Implementácia

3.1 Pohyb v scéne

3.2 Používateľské rozhranie

3.2.1 Vstupy

3.2.2 Nastavenia

3.2.3 Dizajn

3.3 Ukladanie obrázkov

3.4 Optimalizácie

3.4.1 Pamäť

3.4.2 Výpočty

Kapitola 4

Výskum

Kapitola 5

Výsledky

5.1 Testovacie scénare

5.2 Merania

5.3 Porovnanie zobrazovacích techník

5.4 Vizuálne výsledky

Kapitola 6

Diskusia

6.1 Výzvy a obmedzenia

6.2 Rozšírenia a budúce vylepšenia

6.3 Využitia aplikácie

Záver

Text.

Literatúra

[1] John Amanatides, Andrew Woo, et al. A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10. Citeseer, 1987.

[2] James Diebel et al. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15-16):1–35, 2006.

[3] Arie E Kaufman and Klaus Mueller. Overview of volume rendering. *The visualization handbook*, 7:127–174, 2005.

[4] Khronos Group. OpenGL documentation. Dostupné na: `https://www.khronos.org/opengl/`. Navštívené: 01-12-2025.

[5] Morgan McGuire and Louis Bavoil. Weighted blended order-independent transparency. *Journal of Computer Graphics Techniques (JCGT)*, 2(2):122–141, December 2013.

[6] Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982.

[7] NVIDIA. Octree Textures on the GPU. Dostupné na: `https://developer.download.nvidia.com/books/gpugems2/37_octree_03.jpg`. Navštívené: 2025-12-01.

[8] H. Ray, H. Pfister, D. Silver, and T.A. Cook. Ray casting architectures for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):210–223, 1999.

[9] Jorge Revelles, Carlos Ureña, Miguel Lastra, Dpt Lenguajes, Sistemas Informaticos, and E. Informatica. An efficient parametric algorithm for octree traversal. 05 2000.

Príloha A: obsah elektronickej prílohy

V elektronickej prílohe priloženej k práci sa nachádza zdrojový kód programu a súbory s výsledkami experimentov.

Zdrojový kód je zverejnený aj na stránke `https://github.com/kocifajadrian/master-thesis`.