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

ZOBRAZENÍ ROZSÁHLÝCH VOLUMETRICKÝCH DAT NA CPU

VISUALIZATION OF LARGE VOLUMETRIC DATA ON CPU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. DRAHOMÍR DLABAJA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL ŠPANĚL, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání diplomové práce

Student: **Dlabaja Drahomír, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Počítačová grafika a interakce
Název: **Zobrazení rozsáhlých volumetrických dat na CPU**
Visualization of Large Volumetric Data on CPU
Kategorie: Počítačová grafika

Zadání:

1. Seznamte se s principy přímého zobrazení volumetrických dat (tzv. Volume Rendering).
2. Prostudujte současné přístupy a existující knihovny pro práci s velkými volumetrickými daty a podporu jejich zobrazení.
3. Vyberte vhodné postupy a technologie a navrhněte řešení pro efektivní práci s velkými volumetrickými daty, které umožní jejich 3D zobrazení.
4. Navržené řešení implementujte formou knihovny a vytvořte experimentální demo aplikaci (CPU renderer) pro zobrazení velkých volumetrických dat.
5. Experimentujte s vaší implementací, zhodnoťte dosažené výsledky a diskutujte možnosti budoucího vývoje.
6. Vytvořte stručný plakát nebo video prezentující vaši práci, její cíle a výsledky.

Literatura:

- Wald *et al.*, "OSPRay - A CPU Ray Tracing Framework for Scientific Visualization," in *IEEE Transactions on Visualization and Computer Graphics*, 2017 (https://dl.acm.org/doi/10.1109/TVCG.2016.2599041).
- Beyer *et al.*, "A Survey of GPU-Based Large-Scale Volume Visualization", in *EuroVis - STARs*, 2014 (https://diglib.eg.org/handle/10.2312/eurovisstar.20141175.105-123).

Při obhajobě semestrální části projektu je požadováno:

- Splnění prvních tří bodů zadání.

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

Vedoucí práce: **Španěl Michal, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 18. května 2022
Datum schválení: 2. listopadu 2021

Abstrakt

Tato práce řeší problém zobrazení volumetrických dat na CPU, které svým datovým rozsahem přesahují operační paměť stroje. Práce popisuje návrh vizualizačního schématu, které sestává z datové struktury pro rozsáhlá volumetrická data, a algoritmu, který takto zpracovaná data vizualizuje. Navržená hierarchická datová struktura akceleruje vzorkování a umožňuje redukci celkového množství dat, které je při vizualizaci nutné načíst do fyzické paměti. Vizualizace zpracovaných dat je docíleno metodou vrhání paprsků s využitím existujících optimalizačních technik, jako je přeskakování prázdného prostoru nebo předčasné ukončení paprsku. Datová struktura umožňuje až 12× rychlejší vzorkování v porovnání s vzorkováním surových rozsáhlých volumetrických dat, která jsou serializována po řádcích. Využitím datové hierarchie bylo dosaženo až 150× rychlejší vizualizace rozsáhlých volumetrických dat v téměř bezeztrátovém režimu v porovnání s plně bezeztrátovým režimem. Zobrazovací schéma je implementováno formou knihovny v jazyce C++20. Implementace využívá akceleraci pomocí vektorizace a umožňuje snadnou paralelizaci ze strany uživatele. Knihovna poskytuje nástroje pro zpracování a zobrazení rozsáhlých volumetrických dat na CPU.

Abstract

This thesis deals with the problem of displaying volumetric data that exceeds the operating memory capacity of the machine. The work describes the design of a visualization pipeline, which consists of a data structure for large volumetric data and an algorithm that visualizes such data. The proposed hierarchical data structure accelerates sampling and allows the reduction of the total amount of data that needs to be loaded into physical memory during visualization. Visualization of processed data is achieved by the ray casting method with existing optimization techniques, such as empty space skipping and early ray termination. The data structure allows up to 12× faster sampling compared to the sampling of raw large volumetric data serialized by rows. Up to 150× faster visualization of large volumetric data in near-lossless mode has been achieved compared to the fully lossless mode by utilizing the data hierarchy. The display scheme is implemented in the form of a library in C++20 language. The implementation uses acceleration by vectorization and allows easy parallelization by the user. The library provides tools for processing and visualization of large volumetric data on the CPU.

Klíčová slova

Volumetrická data, rozsáhlá data, bloková dekompozice, vrhání paprsku, vektorizace, datová hierarchie, oktalový strom.

Keywords

Volumetric data, large data, block decomposition, ray casting, vectorization, data hierarchy, octree.

Citace

DLABAJA, Drahomír. *Zobrazení rozsáhlých volumetrických dat na CPU*. Brno, 2022. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Španěl, Ph.D.

Zobrazení rozsáhlých volumetrických dat na CPU

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Michala Španěla, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Drahomír Dlabaja
10. května 2022

Poděkování

Rád bych poděkoval Ing. Michalovi Španělovi, Ph.D. za poskytnutou odbornou pomoc.

Obsah

Kapitola 1

Úvod

Vizualizace volumetrických dat se uplatňuje v medicíně, biologii, fyzice i inženýrství. Oproti běžné povrchové reprezentaci jsou volumetrická data reprezentována diskrétně vzorkovaným trojrozměrným skalárním polem. Tím umožňuje studii její vnitřní struktury a závislostí v datech. Pro vizualizace volumetrických dat již byla vyvinuta řada zobrazovacích metod a akceleračních technik. Většina existujících metod však není snadno škálovatelná s rostoucím rozlišením dat a naráží na limity v podobě paměťové kapacity. Naivně lze tento problém řešit podvzorkováním dat nebo navýšením paměti stroje. První varianta však vede k viditelnému snížení kvality vizualizace, druhá varianta omezuje zobrazení volumetrických dat pouze na specializované pracovní stroje.

Tato práce se věnuje návrhu a implementaci zobrazovacího schématu, které škáluje pro data nad rámec operační paměti stroje. V rámci práce je navržena a vyhodnocena datová struktura pro volumetrická data, která optimalizuje přístupy do paměti a umožňuje efektivní vzorkování dat. Nad touto strukturou je sestavena hierarchie rozlišení, kterou je možné využít k redukci celkového množství paměti potřebné pro vizualizaci objemu z jednoho pohledu. Paralelně k hierarchické reprezentaci je sestaven oktalový strom, který slouží k přeskakování prázdného prostoru. Tyto datové struktury jsou určeny k sestavení ve fázi předzpracování. V rámci práce byl navržen vizualizér, který hierarchickou datovou strukturu a oktalový strom mapuje z úložiště do virtuální paměti. Vizualizér umožňuje volbu úrovně zanoření paprsku do stromu dle volitelného predikátu, a tím i úroveň hierarchie, ze které jsou data vizualizována. To vede na celkovou redukci dat, které je pro vizualizaci nutné načíst do fyzické paměti, a tím i k akceleraci samotné vizualizace. Tato práce experimentuje s predikátem, který volí úroveň hierarchie podle aktuální saturace paprsku.

Kapitola 2 popisuje volumetrická data a metody pro jejich zobrazení. Důraz je kladen zejména na metodu vrhání paprsků a možnosti její optimalizace a akcelerace, neb právě tato metoda je v rámci této práce adaptována na rozsáhlá data. Kapitola 3 se věnuje popisu problematiky rozsáhlých volumetrických dat. Jsou zde popsány problémy, kterým se kvůli rozsáhlosti dat čelí, a jejich řešení. Tato kapitola popisuje existující metody a knihovny, které s rozsáhlými volumetrickými daty pracují. V kapitole 4 je popsáno navržené schéma pro vizualizaci rozsáhlých volumetrických dat. Je zde popsána datová struktura pro efektivní práci s těmito daty. Na tuto strukturu je adaptován algoritmus vrhání paprsků a jeho existující akcelerační techniky. V kapitole 5 je popsána implementace vizualizačního schématu formou knihovny v jazyce C++20. Je vysvětleno rozhraní knihovny, její správné použití a technologie, které byly využity pro její implementaci. Kapitola 6 navržené a implementované vizualizační schéma komplexně vyhodnocuje. Jsou zde vyhodnoceny jednotlivé aspekty navržené datové struktury a vizualizéru.

Kapitola 2

Volumetrická data a jejich zobrazování

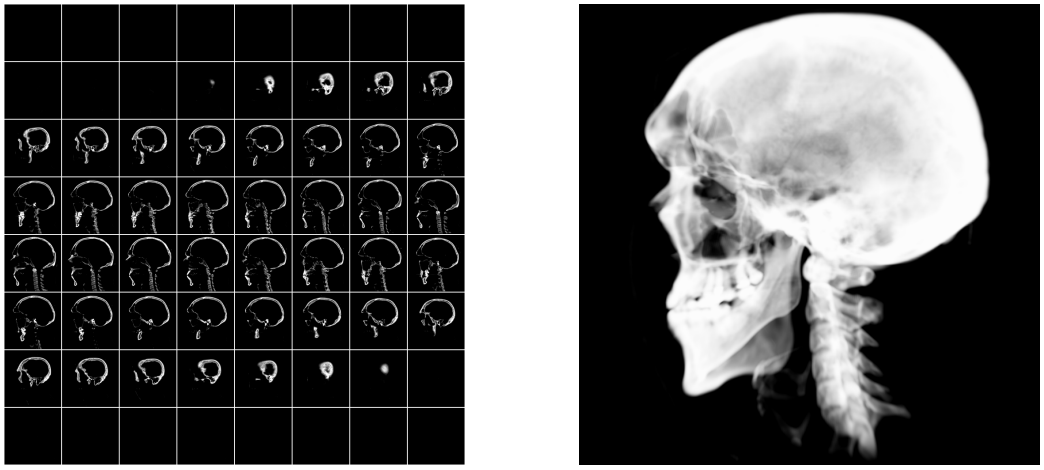
Volumetrická data obecně reprezentují scénu jako funkci, která každému bodu v prostoru přiřazuje hodnotu. V praktickém pojetí jsou volumetrická data reprezentována množinou vzorků této funkce v regulární trojrozměrné mřížce. Volumetrická data lze získat a zpracovat celou řadou metod. V medicíně jde nejčastěji o data zachycená pomocí magnetické rezonance nebo výpočetní tomografie. Data lze v těchto případech interpretovat jako sérii klasických dvourozměrných lékařských snímků s uniformními rozestupy. Oproti dvourozměrným snímkům však umožňují vizualizaci dat v trojrozměrném prostoru, a tím i jejich snazší interpretaci. Ve fyzice se volumetrická data používají pro simulaci a zkoumání složitých jevů. Můžou být produktem prostorových simulací a celulárních automatů. Tímto způsobem lze například simulovat a vizualizovat chemické reakce. V inženýrství se volumetrická data využívají pro simulaci a vizualizaci namáhání mechanických součástí a struktur.

Tato kapitola se zaměřuje na popis volumetrických dat a metod jejich zobrazování. Sekce 2.1 podrobně popisuje volumetrickou reprezentaci prostorových dat. Sekce 2.2 se zabývá formáty pro uložení volumetrických dat. Jsou zmíněny možnosti alternativních reprezentací a komprese. V sekci 2.3 je popsána segmentace a klasifikace volumetrických dat. Je zde vysvětlen význam přenosových funkcí v kontextu vizualizace. Sekce 2.4 popisuje metody nepřímého zobrazování volumetrických dat. V sekci 2.5 jsou popsány základní metody pro přímé vykreslování volumetrických dat. V sekci 2.6 je popsána přímá vizualizace volumetrických dat metodou vrhání paprsků. Právě tato metoda je v rámci této práce adaptována pro rozsáhlá data.

2.1 Volumetrická reprezentace prostorových dat

V praktickém pojetí jsou volumetrická data reprezentována regulárním skalárním trojrozměrným polem. Jeden volumetrický element se nazývá *voxel*. Každý voxel reprezentuje jednu skalární hodnotu zaznamenané vlastnosti. Běžné volumetricky zaznamenávané vlastnosti jsou barva, hustota, teplota nebo tlak. Lze se také setkat s volumetrickými daty reprezentujícími binární hodnoty.

Na volumetrická data lze nahlížet jako na sérii klasických dvourozměrných obrazových snímků. Tento pohled se často uplatňuje v medicínských datech a formátech pro ně. Způsoby interpretace volumetrických dat jsou vizualizovány v obrázku 2.1. Velikost voxelu obecně



(a) (b)

Obrázek 2.1: Volumetrická data lze chápat jako sekvenci dvourozměrných snímků (a). Tyto snímky lze interpretovat a zobrazit jako trojrozměrný objem (b).

nemusí být ekvivalentní ve všech dimenzích. V medicínských datech se například vzdálenost jednotlivých řezů může lišit od velikosti pixelu v rámci jednoho řezu.

2.2 Uložení volumetrických dat a komprese

Naivním způsobem lze volumetrická data ukládat jako trojrozměrné pole hodnot. To však není efektivní kvůli paměťovým nárokům, které jsou o jednu dimenzi vyšší než u klasických dvourozměrných dat. Ve většině případů však volumetrická data obsahují prostorovou redundanci ve všech třech dimenzích, které lze chytře využít pro kompresi.

Triviálně lze kompresi volumetrických dat zjednodušit na kompresi série dvourozměrných obrázků. To umožňuje použití existujících a odladěných metod pro kompresi obrazových dat. Na dvourozměrné snímky lze použít například metodu JPEG v případě ztrátové komprese, metodu PNG v případě bezeztrátové komprese a metodu JPEG 2000 pro ztrátovou i bezeztrátovou kompresi. Takové formáty však využívají redundanci pouze ve dvou dimenzích a nevyužívají kompletní kompresní potenciál volumetrických dat. Výhoda takového přístupu je možnost dekomprese jednoho řezu bez nutnosti dekomprimovat celý objem.

Hybridním přístupem lze na volumetrická data nahlížet opět jako na sérii dvourozměrných obrázků, avšak třetí dimenzi interpretovat jako temporální. Na takto interpretovaná data lze použít libovolnou odladěnou metodu pro kompresi videa. V době psaní této práce jsou považovány za state-of-the-art metody H.266/VVC a AV1. Takto postavená komprese využívá intra kompresi pro odstranění redundance ve dvou dimenzích a inter kompresi pro odstranění redundance ve třetí dimenzi. Oproti kompresi videa zde není požadavek na nízkou latenci při dekompresi, lze proto využít nejvyšší nastavení kodeků a dosáhnout až o 44 % lepšího kompresního poměru v porovnání s doporučeným DICOM enkodérem [11].

Třetím přístupem je rozšíření metod pro kompresi dvourozměrných obrázků do tří dimenzí. Zpravidla to znamená provedení trojrozměrné diskrétní kosinové nebo vlnkové transformace nad daty a následné entropické kódování. Tyto metody nejsou příliš rozšířené, ale ukazují potenciál velmi dobrého využití redundance ve všech třech dimenzích. Vlnková

transformace umožňuje lokální inverzní transformaci v různých úrovních kvality, díky tomu je vhodnou reprezentací jak pro kompresi, tak i vizualizaci rozsáhlých volumetrických dat [12].

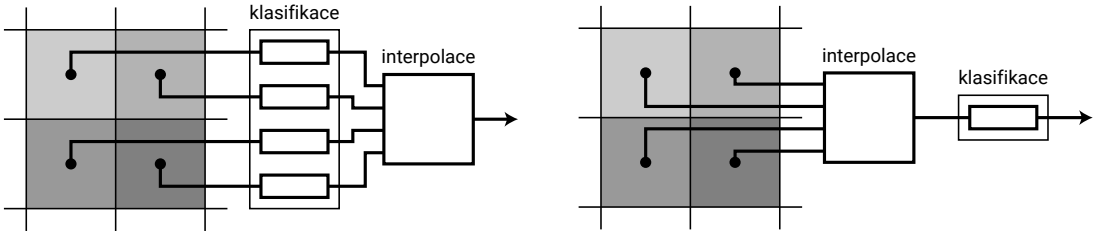
2.3 Klasifikace a segmentace

Volumetrická data jsou ve svém běžném pojetí pouze trojrozměrným polem zaznamenávajícím prostorovou vlastnost. Z pohledu vizualizace je však přínosné tato data interpretovat dle jejich obsahu. Této problematice se věnuje segmentace a klasifikace volumetrických dat.

Klasifikace

Klasifikace se zabývá interpretací skalárních hodnot objemu. Z pohledu vizualizace je důležitá klasifikace barvy a průhlednosti. Ta se provádí pomocí přenosové funkce. Přenosová funkce je libovolná funkce, která mapuje hodnotu vzorku objemu na hodnotu RGBA. Přenosová funkce může mít libovolný charakter a je velmi specifická pro zobrazovaná data. Existuje celá řada nástrojů a metod, jak přenosovou funkci vytvořit. Uživatelské rozhraní pro návrh přenosové funkce je často součástí vizualizačního softwaru, který aktuální funkci rovnou zobrazuje v reálném čase.

Klasifikaci na základě přenosové funkce lze provádět buď před nebo po interpolaci hodnot voxelů. Aplikace přenosové funkce před interpolací dosahuje realistického zobrazení za cenu větší náročnosti, neboť v případě trilineární interpolace je nutné provádět klasifikaci nad osmi sousedními voxely. Oproti tomu klasifikace na interpolovaných datech klasifikuje hodnotu vzorku pouze jednou, avšak výsledek pouze aproximuje realitu. V praxi se častěji používá druhá varianta. Metody klasifikace jsou demonstrovány obrázkem 2.2.

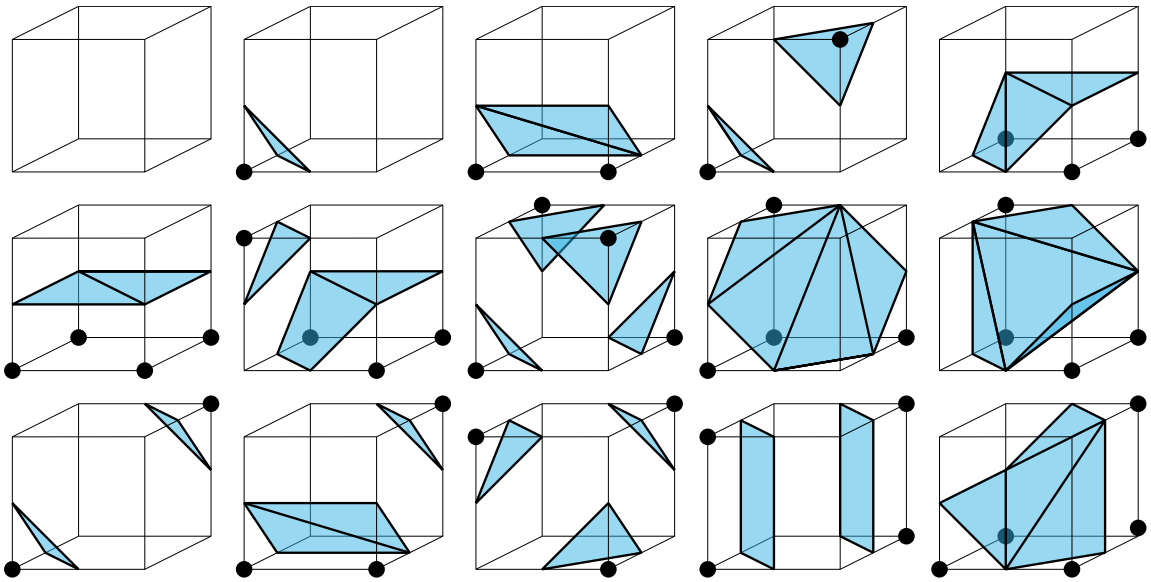


Obrázek 2.2: Klasifikace před interpolací (a) dosahuje přesnějšího výsledku za cenu většího množství operací v porovnání s klasifikací po interpolaci (b).

Vstupem přenosové funkce je hodnota vzorku, často však může být vstupem i jiná lokální informace v okolí vzorku, např. gradient. Vzniká tak vícerozměrná přenosová funkce. Ta umožňuje klasifikaci vizualizovaných dat, která zvýrazňuje požadované vlastnosti. S dimenzionalitou přenosové funkce však roste také náročnost klasifikace.

Segmentace

V některých případech vizualizace volumetrických dat je žádoucí zobrazit pouze určitou část dat, která reprezentuje jeden objekt ve scéně. Může jít například o extrakci a vizualizaci shluku nádorových buněk v medicínských datech. Tuto činnost často nelze dělat pouze pomocí klasifikace, protože hustota nádorové tkáně může odpovídat hustotě jiné



Obrázek 2.3: Základní konfigurace vnitřních a vnějších voxelů u algoritmu Marching cubes.

tkáně v daném snímku. Do přenosové funkce je proto zapojena i informace o pozici vzorků v objemu.

Segmentaci lze provádět jak manuálně, tak i pomocí automatizovaných metod. Manuální metody využívají možností 2D a 3D štětců, tato činnost je však časově velmi náročná. Automatizované metody můžou fungovat s asistencí i bez asistence uživatele. Velmi známým algoritmem pro segmentaci s asistencí uživatele je region-growing algoritmus [14]. Pro plně automatickou segmentaci lze použít například algoritmy umělé inteligence [25].

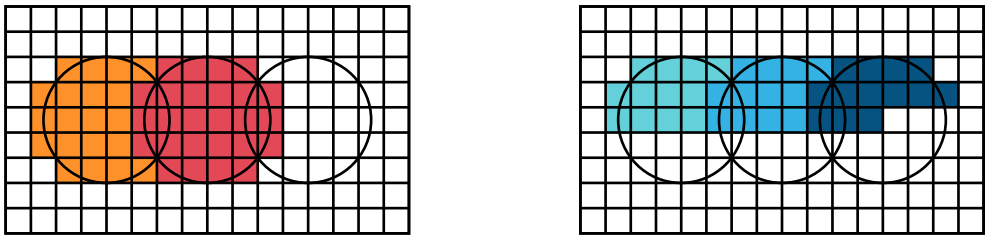
2.4 Nepřímé zobrazování volumetrických dat

Nepřímé metody vizualizace volumetrických dat se zaměřují na převod volumetrických dat do reprezentace, která je snadno vizualizovatelná pomocí konvenčních metod. Nejčastěji jde o extrakci povrchu v místech, kde má objem zvolenou hraniční hodnotu – isopovrch. Výstupem je povrch reprezentovaný klasickou trojúhelníkovou sítí, který je snadno zobrazitelný rasterizací na grafickém akcelerátoru.

Výhodou nepřímých metod je snadné zobrazování pomocí konvenčních algoritmů. Za výhodu lze také považovat dostupnost získaného povrchu, nad kterým lze dělat libovolné operace dostupné pro trojúhelníkové sítě a lze jej využít i k jiným účelům než k vizualizaci. Nevýhodou tohoto přístupu je nutnost provádět extrakci povrchu při každé změně hraniční hodnoty, což je časově náročná operace. Další nevýhodou je, že metody tohoto typu z celého objemu zobrazují pouze povrchy a může tak dojít ke ztrátě důležité informace hlouběji v datech. Nejznámější metodou pro extrakci povrchu je metoda Marching cubes [21]. Tato metoda funguje na principu generování trojúhelníků na základě konfigurace vnitřních a vnějších voxelů. Tyto konfigurace jsou vizualizovány v obrázku 2.3.

2.5 Přímé zobrazování volumetrických dat

Přímé metody se zabývají zobrazováním volumetrických dat bez převodu do pomocné reprezentace. Fungují na principu transformace vzorků z lokálního prostoru objemu do prostoru obrazovky. Podobně jako v problematice zobrazování polygonových modelů, i zde existují *object-order* přístupy, které zobrazují data v pořadí po jednotlivých elementech, i *image-order* přístupy, které naopak pro každý pixel výsledné vizualizace získávají výslednou hodnotu [8]. Tyto přístupy demonstruje obrázek 2.4.



(a) Object-order. (b) Image-order.

Obrázek 2.4: Přístupy zobrazování dat. Data jsou zobrazována buď po elementech (a), nebo po pixelech (b).

Oba přístupy však spojuje to, že aproximativně vyhodnocují *integrál podél projekčního paprsku* vycházející z daného pixelu obrazovky. Tato rovnice integruje barvy vzorků a jejich průhlednost podél paprsku, který přísluší danému pixelu. Integrál podél projekčního paprsku je definován rovnicí 2.1.

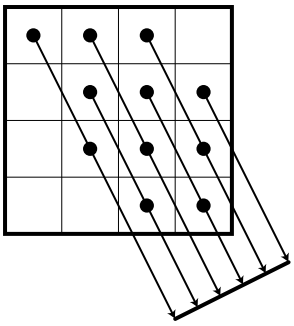
$$I = \int_0^{\inf} c\left(\mathbf{x}\left(\lambda\right)\right) \exp\left(-\int_0^{\lambda} \alpha\left(\mathbf{x}\left(\lambda'\right)\right) \mathrm{d}\lambda'\right) \mathrm{d}\lambda \tag{2.1}$$

Paprsek $\mathbf{x}$ je parametrizován vzdáleností k pozorovateli λ. Barva c a průhlednost α může být získána pro každý bod v prostoru. Barva emitovaná v každém bodě $\mathbf{x}$, $c\left(\mathbf{x}\right)$, je zastíněna průhledností vzorků $\alpha\left(\mathbf{x}\right)$ integrovaných od pozorovatele k bodu emise.

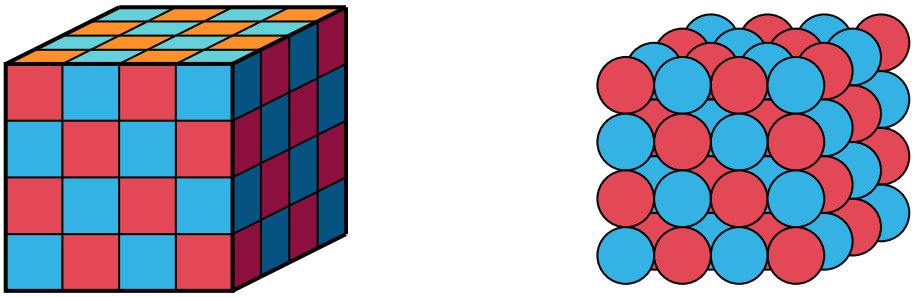
Voxel splatting

Voxel splatting je object-order metoda, která pro každý voxel objemu nalezne jeho pozici v prostoru obrazovky a na nalezeném místě vykreslí 2D primitivum s požadovanou barvou a průhledností [29]. Vlastnosti vykreslovaného primitiva, jako je například jeho tvar a velikost, je nutné odladit v případě každé aplikace algoritmu pro dosažení požadovaného grafického efektu. U této metody je nutné zajistit, aby byly jednotlivé voxely vykreslovány v pořadí dle vzdálenosti od pozorovatele a došlo tak ke korektnímu skládání průhledností a barev.

Výhodou tohoto algoritmu je jeho jednoduchost. Nevýhodou je nerealistické zobrazení objemu. Oproti ostatním metodám tato metoda umožňuje dosažení zajímavých grafických efektů. Tento algoritmus lze akcelerovat pomocí paralelizace i vektorizace. Princip voxel splattingu je vizualizován ve dvou rozměrech na obrázku 2.5 a ve třech rozměrech na obrázku 2.6.



Obrázek 2.5: Princip voxel splattingu ve dvou dimenzích. Každý voxel je promítnut a vyobrazen do pozorovací roviny.

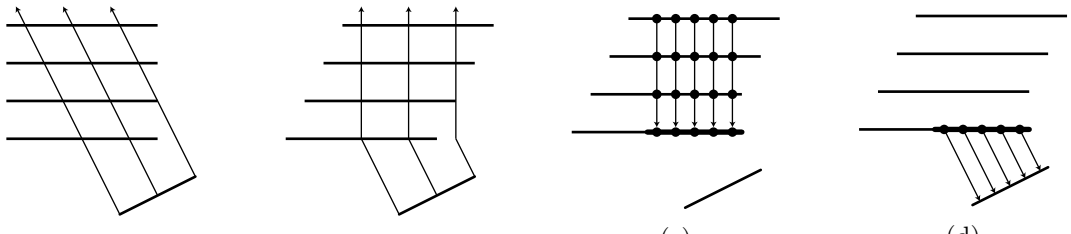


(a) (b)

Obrázek 2.6: Demonstrace metody voxel splatting. Pro každý voxel zobrazovaného objemu (a) je vygenerován disk s příslušnou barvou (b). Disky jsou vizualizovány v pořadí od nejvzdálenějšího k nejbližšímu.

Shear-warp

Shear-warp algoritmus je object-order metoda pro vizualizaci volumetrických dat, která interpretuje objem jako sérii dvourozměrných snímků, které jsou seřazeny dle vzdálenosti od pozorovatele. Tyto snímky jsou transformovány do tzv. *sheared-object* prostoru – na každý snímek je aplikována 2D translace a v případě perspektivní projekce také 2D škálování. Jednotlivé snímky jsou poté skládány a blendovány do jednoho pomocného obrázku, který zobrazuje celý objem v sheared-object prostoru. Tento obrázek je poté transformován a převzorkován do prostoru obrazovky [19]. Metoda shear-warp je pro ortografickou projekci vizualizována v obrázku 2.7.



(a) (b) (c) (d)

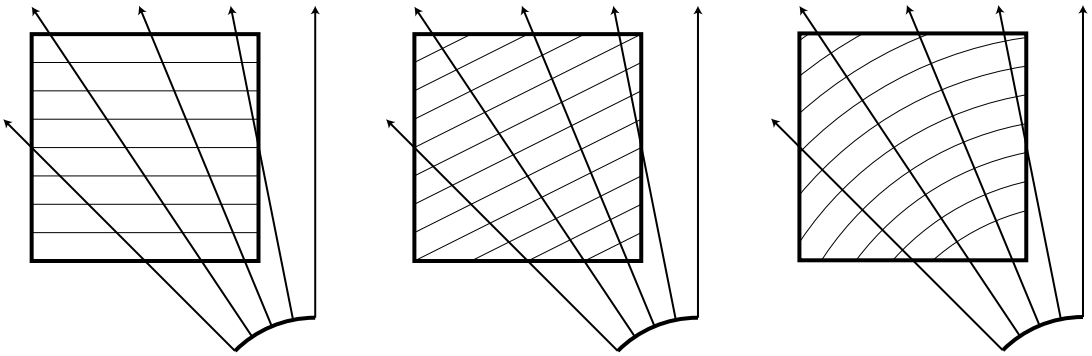
Obrázek 2.7: Algoritmus shear-warp podle aktuální projekce (a) transformuje snímky objemu do sheared-object prostoru (b). Takto transformované snímky jsou složeny pomocí alpha blendingu do mezivýsledku (c), který je promítnut do roviny pozorovatele (d).

Tato metoda velmi dobře využívá prostorové lokality – dochází při ní pouze k bilineární interpolaci. Transformaci jednotlivých snímků lze velmi dobře paralelizovat a akcelerovat pomocí běžných metod pro práci s obrazem. Nevýhodou můžou být vyšší paměťové nároky, neb je nutné pro zachování efektivního přístupu do paměti pro libovolnou pohledovou transformaci udržovat v paměti tři kopie objemu, kde každá je paměťově organizovaná podle jedné prostorové osy. Při vykreslování se poté volí objem uspořádaný podle osy, která nejvíce směřuje k pozorovateli. Shear-warp algoritmus dosahuje lepší kvality vizualizace než voxel splatting.

Vizualizace volumetrických dat založená na texturách

Vizualizace volumetrických dat založená na texturách je image-order metoda zobrazení volumetrických dat, která je specificky navržena pro maximální využití grafického akcelerátoru. Metoda silně využívá texturovací jednotky pro rychlou trilineární interpolaci a využití rasterizátoru pro převod souřadnic z prostoru obrazovky do lokálního prostoru objemu. Pro pozici pozorovatele jsou na CPU vygenerovány polygony reprezentující řezy ohraničujícím tělesem vykreslovaného objemu. Tyto řezy jsou běžně paralelní s obrazovou rovinou, avšak můžou být zarovnané s daty nebo tvořit soustředné oblouky v centru s pozorovatelem. Vzdálenost řezů od sebe, resp. jejich množství řídí výkon vizualizace a kvalitu vykreslovaného obrázku. Polygony reprezentující řezy ve svých vertexech obsahují atribut souřadnic v lokálním prostoru objemu [3, 24, 28]. Metody řezu objemem jsou ve dvou dimenzích vizualizovány na obrázku 2.8.

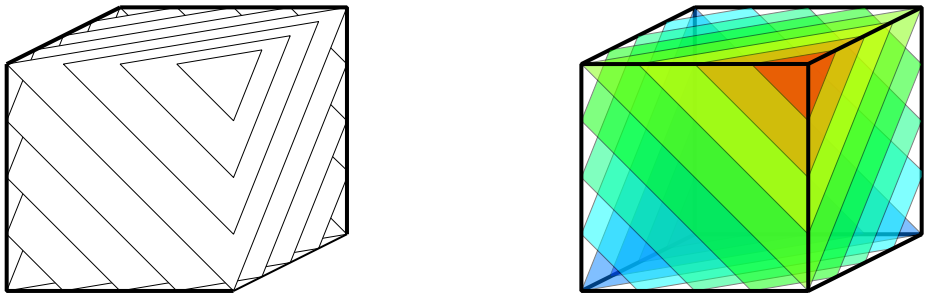
Připravené polygony se pošlou na rasterizaci do GPU. Vykreslování a blending se provádí ve fragment shaderu. Interpolované lokální souřadnice se použijí pro přístup do 3D textury objemu, na získanou hodnotu se aplikuje přenosová funkce a výsledek se uloží do frame bufferu pomocí alpha blendingu. V případě, že 3D textury nejsou hardwarově podporované, lze objem interpretovat na GPU jako sérii 2D textur a provádět trilineární interpolaci manuálně ze dvojice bilineárně interpolovaných vzorků. Metoda je ve třech dimenzích vizualizována na obrázku 2.9.



(a) (b) (c)

Obrázek 2.8: Princip vizualizace volumetrických dat založené na texturách. Řezy objemem můžou být buď paralelní s daty podobně jako v metoda shear-warp (a), paralelní s rovinou pozorovatele pro lepší kvalitu (b) nebo můžou tvořit soustředné kružnice od pozorovatele pro nejlepší kvalitu při perspektivní projekci (c).

Tato metoda velmi dobře utilizuje klasický vykreslovací řetězec. Metoda naráží na limit v podobě maximální dostupné paměti grafického akcelerátoru a maximálního rozměru tex-



(a) (b)

Obrázek 2.9: Vizualizace objemu založená na texturách. Pro objem jsou vygenerovány polygony, které reprezentují řezy ohraničujícím tělesem (a). Tyto řezy jsou vizualizovány běžným grafickým řetězcem a skládány pomocí alpha blendingu (b)

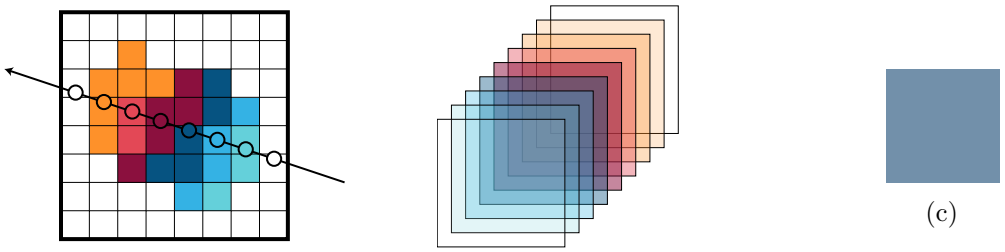
tur. V případě, že vykreslené těleso tyto limity přesahuje, je nutné objem vykreslit jako množinu menších objemů. To má za následek větší režii, což vede na delší dobu vykreslování. Metoda je navržena specificky pro grafický akcelerátor, není proto vhodná v případech, kdy akcelerátor není dostupný.

2.6 Metoda vrhání paprsků

Metoda vrhání paprsků je považována za nejpřesnější metodu pro vizualizaci volumetrických dat, avšak za cenu nejvyšší výpočetní náročnosti [5, 15, 20]. Jde o přímou implementaci diskretizace integrálu podél projekčního paprsku. Tato diskretizace je popsaná rovnicí 2.2.

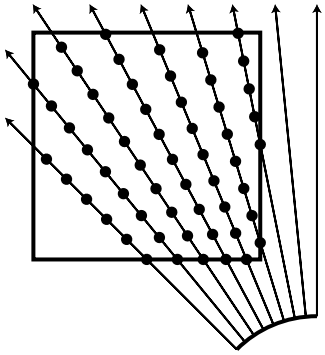
$$I = \sum_{i=0}^{N} \left(c\left(\mathbf{x}\left(i \cdot t\right)\right) \alpha\left(\mathbf{x}\left(i \cdot t\right)\right) \prod_{j=0}^{i-1} \left(1 - \alpha\left(\mathbf{x}\left(j \cdot t\right)\right)\right) \right) \tag{2.2}$$

Tato metoda pro každý pixel výsledného obrázku vrhá paprsek $\mathbf{x}$. Podél paprsku vzorkuje N hodnot objemu s krokem t. Na tyto hodnoty aplikuje přenosovou funkci, na základě které vypočítá barvu vzorku c a průhlednost α. Výsledné barevné hodnoty jsou akumulovány na základě průhlednosti předcházejících vzorků. Kroky metody vrhání paprsků jsou vizualizovány obrázkem 2.10. Demonstrace vzorkování množinou paprsků je vizualizována obrázkem 2.11.



(a) (b) (c)

Obrázek 2.10: Kroky metody vrhání paprsků. Data jsou z objemu vzorkována podél paprsku od nejbližšího po nejvzdálenější (a). Nasbírané vzorky jsou skládány pomocí alpha blendingu (b). Složené vzorky dohromady tvoří výslednou barvu pixelu (c).



Obrázek 2.11: Vizualizace metody vrhání paprsků s perspektivní projekcí ve dvou dimenzích. Metoda se principiálně podobá vizualizaci volumetrických dat založené na texturách, avšak v tomto případě si pozici vzorků řídí sám paprsek.

Metodu lze snadno paralelizovat i vektorizovat, je proto vhodná pro implementaci jak na CPU, tak i na grafickém akcelerátoru. Pro vrhání paprsků existuje celá řada optimalizačních technik [18]. Tyto techniky jsou popsány dále v této sekci. Metoda vrhání paprsků včetně optimalizačních technik je součástí zobrazovacího řetězce navrženého v kapitole 4.

Trilineární interpolace

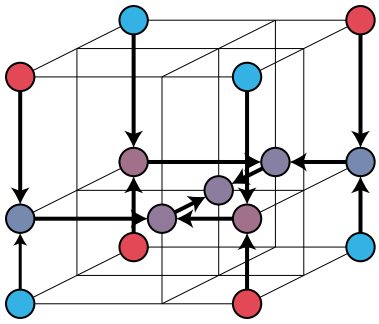
Trilineární interpolace se používá pro aproximaci hodnoty vzorku objemu na souřadnicích, které leží mezi souřadnicemi jednotlivých voxelů. Oproti zaokrouhlení souřadnice na nejbližší voxel je dosaženo hladké a kontinuální reprezentace diskrétních volumetrických dat. Trilineární interpolace je vyjádřena rovnicí 2.3.

$$\begin{aligned} s(x,y,z) = {} & v(\lfloor x\rfloor,\lfloor y\rfloor,\lfloor z\rfloor)\cdot\bar{x}\cdot\bar{y}\cdot\bar{z} \\ & + v(\lfloor x\rfloor+1,\lfloor y\rfloor,\lfloor z\rfloor)\cdot(1-\bar{x})\cdot\bar{y}\cdot\bar{z} \\ & + v(\lfloor x\rfloor,\lfloor y\rfloor+1,\lfloor z\rfloor)\cdot\bar{x}\cdot(1-\bar{y})\cdot\bar{z} \\ & + v(\lfloor x\rfloor+1,\lfloor y\rfloor+1,\lfloor z\rfloor)\cdot(1-\bar{x})\cdot(1-\bar{y})\cdot\bar{z} \\ & + v(\lfloor x\rfloor,\lfloor y\rfloor,\lfloor z\rfloor+1)\cdot\bar{x}\cdot\bar{y}\cdot(1-\bar{z}) \\ & + v(\lfloor x\rfloor+1,\lfloor y\rfloor,\lfloor z\rfloor+1)\cdot(1-\bar{x})\cdot\bar{y}\cdot(1-\bar{z}) \\ & + v(\lfloor x\rfloor,\lfloor y\rfloor+1,\lfloor z\rfloor+1)\cdot\bar{x}\cdot(1-\bar{y})\cdot(1-\bar{z}) \\ & + v(\lfloor x\rfloor+1,\lfloor y\rfloor+1,\lfloor z\rfloor+1)\cdot(1-\bar{x})\cdot(1-\bar{y})\cdot(1-\bar{z}) \end{aligned} \tag{2.3}$$

Kde je každá souřadnice $n \in \{x,y,z\}$ rozdělena na svou celou část $\lfloor n\rfloor$ a zbytkovou část $\bar{n} = n - \lfloor n\rfloor; \bar{n} \in \langle 0,1)$. Výsledná hodnota vzorku s na souřadnicích x,y,z je získána vhodnou kombinací osmi voxelů objemu v. Jelikož je pro získání jediné hodnoty vzorku nutné číst osm hodnot voxelů z objemu, je tato operace **kritickým bodem** implementace řetězce pro vizualizaci volumetrických dat a je nutno věnovat patřičné úsilí pro její optimalizaci, případně celkové redukci aplikací této operace. Trilineární interpolaci lze implementovat pomocí sedmi jednorozměrných lineárních interpolací. To je vizualizováno obrázkem 2.12.

Předčasné ukončení paprsku

Metoda předčasného ukončení paprsku ukončí vzorkování objemu podél daného paprsku v momentě, kdy průhlednost akumulované hodnoty klesne pod hraniční hodnotu. V takovém



Obrázek 2.12: Schéma výpočtu lineární interpolace ve třech dimenzích pomocí sedmi lineárních interpolací v jedné dimenzi. Osm voxelů je interpolováno v jedné dimenzi do čtyř mezivýsledků. Mezivýsledky jsou interpolovány do dvou mezivýsledků v druhé dimenzi. Ty jsou nakonec interpolovány do výsledné hodnoty.

případě již další vzorky nepřinesou žádnou změnu výsledné barevné hodnoty pixelu a další vzorkování je proto zbytečné. Nastavení hraniční hodnoty ovlivňuje výsledný výkon a kvalitu vizualizace objemu. Tato technika je snadno implementovatelná v případě metody vrhání paprsků. Vzhledem k vektorové povaze výpočtu grafických akcelerátorů je v případě vizualizace volumetrických dat založené na texturách náročné tuto techniku efektivně implementovat. Tato technika funguje velmi dobře v případě zobrazování isopovrchů – paprsek může být ukončen po nalezení prvního neprůhledného povrchu [18].

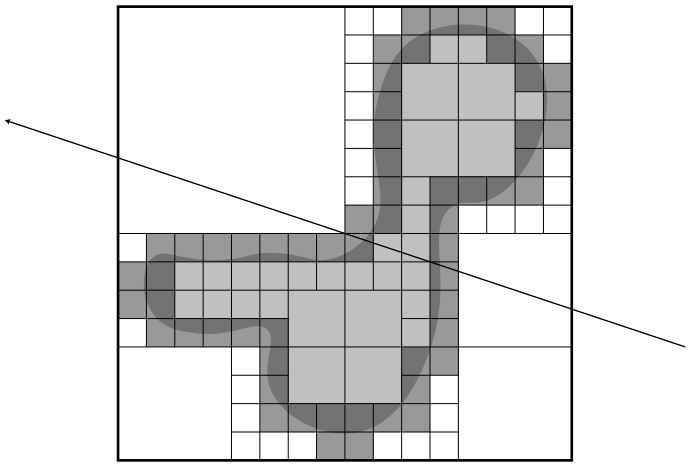
Dělení objemu a přeskakování prázdných regionů

V praxi se může stát, že některé regiony objemu jsou homogenní, nebo nejsou při zvolené přenosové funkci relevantní. Pro využití těchto vlastností dat se používají dodatečné datové struktury pro dělení objemu, které tyto lokální vlastnosti zaznamenávají. Může být využit například oktalový strom, který v každém uzlu agreguje statistické informace od svých potomků, tj. minimální a maximální hodnotu objemu v regionu, který je pokrýván daným uzlem. Tuto datovou strukturu lze využít pro rychlé zjištění informace o regionu a v případě prázdnosti i přeskočení celého regionu [18]. Dělení objemu a přeskakování prázdných regionů je vizualizováno obrázkem 2.13.

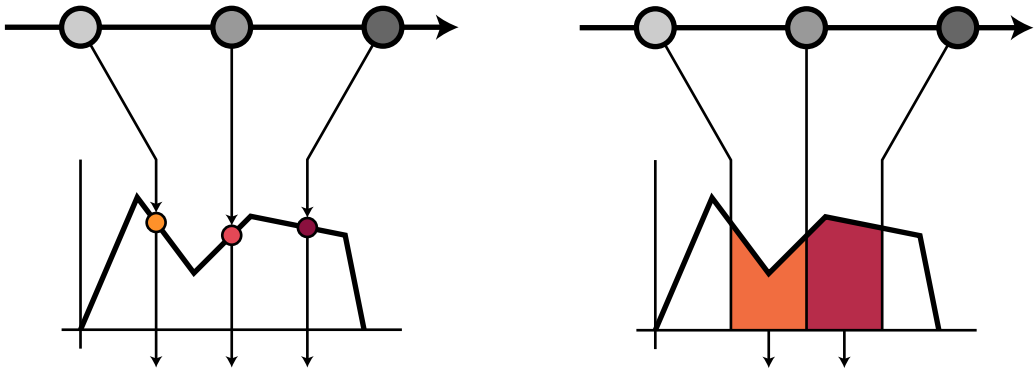
Před-integrovaná přenosová funkce

Vzorkování objemu v diskrétních bodech může vést na tzv. prstencové artefakty, které jsou způsobeny nedostatečným vzorkováním vysokých frekvencí v objemu a přenosové funkci. Tento problém lze vyřešit vykreslováním úseků paprsků namísto bodů. Integrál segmentu na projekčním paprsku je funkce hodnot objemu na vstupu a výstupu segmentu a délky tohoto segmentu. Před-integrováním přenosové funkce pro každou kombinaci těchto hodnot je možné vytvořit texturu, ze které lze v konstantním čase získat hodnotu integrálu podél projekčního paprsku pro daný segment [7, 24]. Tím je aproximována kontinuální integrace objemu mezi dvěma vzorky bez nutnosti zvýšení vzorkovací frekvence. Tyto principy jsou vizualizovány na obrázku 2.14.

V případě, že vzdálenost mezi segmenty je vždy konstantní, lze tabulku redukovat na dva rozměry. Tato technika značně zlepšuje kvalitu výsledného zobrazení při stejné vzorkovací frekvenci, avšak za cenu nutnosti integrovat přenosovou funkci při každé její změně. To může být problém v případech, kde se může přenosová funkce rapidně měnit, například při



Obrázek 2.13: Pro přeskakování prázdných regionů je sestaven oktalový strom, který pro každý region ukládá informace nutné k determinaci jejich prázdnosti. Obrázek demonstruje volumetricky reprezentovaný binární objekt. Bílé regiony mimo objekt jsou považovány za prázdné a při průchodu paprskem jsou přeskočeny. Světlé regiony uvnitř objektu jsou homogenní data, která lze bez ztráty kvality integrovat v jednom kroku. Vzorkovat data je nutné pouze v tmavých regionech na rozhraní objektu, což značně urychluje průchod paprsku objemem.



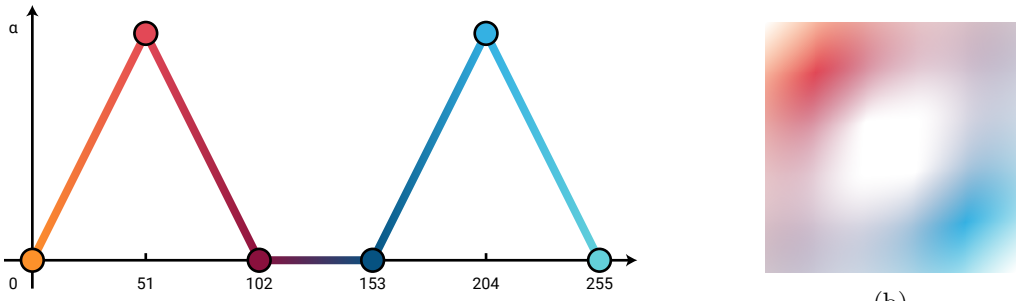
(a) (b)

Obrázek 2.14: Srovnání klasifikace vzorků podél paprsku běžným způsobem (a) s klasifikací na základě vstupního a výstupního vzorku segmentu (b). Druhá varianta vrací průměrnou hodnotu přenosové funkce na intervalu, který je definován vstupy.

ladění přenosové funkce zároveň při zobrazování. Před-integrovanou přenosovou funkci lze používat pro všechny image-order vizualizační metody. Před-integrace přenosové funkce je demonstrovaná obrázkem 2.15.

Progresivní vrhání paprsků

Za optimalizační techniku by se dala taky považovat varianta metody vrhání paprsků, která umožňuje progresivní zlepšování kvality v čase. Toho lze velmi dobře využít pro interaktivní zobrazení volumetrických dat. Progresivního zobrazování je docíleno rozdělením vzorkování objemu od klasifikace a agregace. Vzorky jsou průběžně ukládány a seřazeny podél paprsků



(a) (b)

Obrázek 2.15: Ukázka jednorozměrné přenosové funkce a její před-integrace do dvourozměrné textury. Jednorozměrná přenosová funkce mapuje hodnotu vzorku na hodnoty barvy a průhlednosti (a). Před-integrovaná přenosová funkce je textura, která dvojici hodnot x a y přiřazuje integrál jednorozměrné přenosové funkce, který začíná na hodnotě x a končí na hodnotě y (b).

do jednorozměrných polí. Zobrazování akumulovaných vzorků probíhá průchodem tohoto pole a postupnou aplikací přenosové funkce, shadingu a blendingu. Algoritmus v každé následující iteraci posbírá více vzorků z objemu a zařadí je mezi již nasbírané vzorky. Kvalita se tímto v každé další iteraci zlepšuje. V případě změny transformace se nasbírané vzorky každého paprsku zahodí [26].

Tento algoritmus mimo dobrý kompromis kvality a interaktivity funguje velmi dobře se změnou přenosové funkce a shadingu. V takovém případě se totiž nemusí objem znovu vzorkovat a provede se pouze průchod seřazenými vzorky. Nevýhodou této techniky je velká paměťová náročnost.

Kapitola 3

Přístupy pro zobrazení rozsáhlých volumetrických dat

Za rozsáhlá volumetrická data lze považovat taková data, na která nelze snadno aplikovat konvenční metody pro vizualizaci kvůli tomu, že se objem nevejde do operační paměti nebo paměti grafického akcelerátoru. Naivně lze tento problém řešit pod-vzorkováním objemu, což však vede na snížení výsledné kvality při zobrazení. Tento problém lze také řešit navýšením paměťové kapacity stroje, což však není škálovatelné řešení.

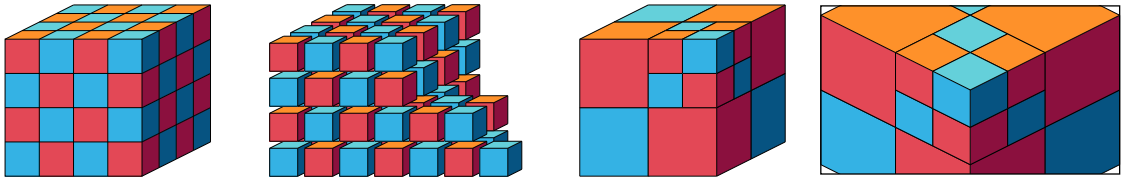
Cílem problému vizualizace rozsáhlých volumetrických dat je nalezení schématu, které agreguje objem do kompaktní reprezentace, kterou lze načíst do paměti a vizualizovat s maximální kvalitou za minimální čas. U algoritmů pro vizualizaci rozsáhlých volumetrických dat je požadována škálovatelnost. To znamená, že úsilí pro zobrazení objemu je proporcionální k požadovanému výstupu a ne k rozlišení samotných objemových dat

V této kapitole jsou popsány přístupy a strategie pro zobrazení rozsáhlých volumetrických dat. Tato kapitola volně zpracovává informace získané z [1, 27] a publikací, které jsou v nich citovány. Sekce 3.1 popisuje základní strukturu obecného řetězce pro zobrazení rozsáhlých volumetrických dat. V sekci 3.2 jsou popsány přístupy, jak rozsáhlá volumetrická data reprezentovat za účelem vizualizace. Sekce 3.3 popisuje nejčastější přístupy a strategie pro zobrazení rozsáhlých volumetrických dat. V sekci 3.4 jsou popsány metody překladu souřadnic z trojrozměrného prostoru na adresu dat v paměťovém prostoru. Sekce 3.5 popisuje strategie tvorby pracovní množiny. Sekce 3.6 obsahuje popis a vyhodnocení existujících řetězců pro zpracování a vizualizaci rozsáhlých volumetrických dat.

3.1 Obecný vizualizační řetězec

Vizualizační řetězec popisuje systém toku dat, jehož cílem je zobrazení těchto dat. Po získání volumetrických dat pomocí měření nebo simulace se první polovina vizualizačního řetězce skládá obvykle z kroků předzpracování dat. Druhou polovinu řetězce tvoří samotná vizualizace. Řetězec definuje, jak jsou data reprezentována, a jak, kdy a kde jsou zpracována. V kontextu rozsáhlých volumetrických dat je důležité, aby byly všechny uzly řetězce škálovatelné, jinak se s rostoucí velikostí objemu stanou kritickým místem.

Vizualizační řetězec pro rozsáhlá volumetrická data je často řízen samotnou vizualizací. To znamená, že vizualizér si sám určuje, která data jsou načtena, zpracována a vizualizována. Například v případě paprskem-řízených metod to prakticky umožňuje tvorbu pra-



(a) (b) (c) (d)

Obrázek 3.1: Řetězec pro vizualizaci rozsáhlých volumetrických dat. Zdrojový objem (a) je ve fázi předzpracování transformován do datové struktury pro efektivní vizualizaci (b). Z této reprezentace se vytvoří a načte do paměti pracovní množina pro jeden specifický pohled (c), ze které je získána výsledná vizualizace (d).

covní množiny přímo při provádění metody vrhání paprsků, tedy s reálnými informacemi o viditelnosti jednotlivých vzorků. Obecný vizualizační řetězec je demonstrován obrázkem 3.1.

3.2 Reprezentace rozsáhlých volumetrických dat

Efektivní reprezentace rozsáhlých volumetrických dat je klíčová pro jejich škálovatelné zobrazení. V kontextu rozsáhlých dat musí datová struktura reprezentující objem umožňovat efektivní tvorbu pracovní množiny. V kontextu zobrazování dat na CPU je nutné explicitně řešit uspořádání dat pro maximalizaci prostorové lokality, a tím i efektivitu přístupu k těmto datům. Obecným požadavkem je, aby struktura reprezentovala data kompaktně a s minimální redundancí.

Pro rozsáhlá volumetrická data se používají tři druhy dekompozice objemu:

- *Bloková dekompozice* – objem je rozdělen na krychlové bloky o stejné velikosti.
- *Hierarchická dekompozice* – objem je rekurzivně podvzorkován do hierarchie rozlišení.
- *Vlnková dekompozice* – objem je rekurzivně rozdělen na vysokofrekvenční a nízkofrekvenční složku. Tyto složky tvoří hierarchii.

V kontextu zobrazování dat na CPU je v rámci této práce experimentováno se třemi způsoby linearizace dat:

- Linearizace *po řádcích* – triviální linearizace, která zajišťuje prostorovou lokalitu pouze v jedné dimenzi.
- Linearizace diskrétní *Hilbertovou křivkou* – zajišťuje prostorovou lokalitu ve třech dimenzích a sousednost voxelů.
- Linearizace diskrétní *Mortonovou křivkou* – zajišťuje prostorovou lokalitu ve třech dimenzích bez sousednosti voxelů.

Přístupové vzory při vizualizaci volumetrických dat

V případě čtení jednorozměrných dat z operační paměti je prostorová lokalita zajištěna implicitně. Jelikož je paměťový prostor lineární, znamená přístup k sousedním datovým jednotkám také přístup k sousedním jednotkám v paměti. Stejný předpoklad však neplatí pro vícerozměrná data. Jelikož neexistuje implicitní mapování z trojrozměrného datového

prostoru na jednorozměrný paměťový prostor, je potřeba toto mapování zajistit explicitně vhodnou *linearizací*.

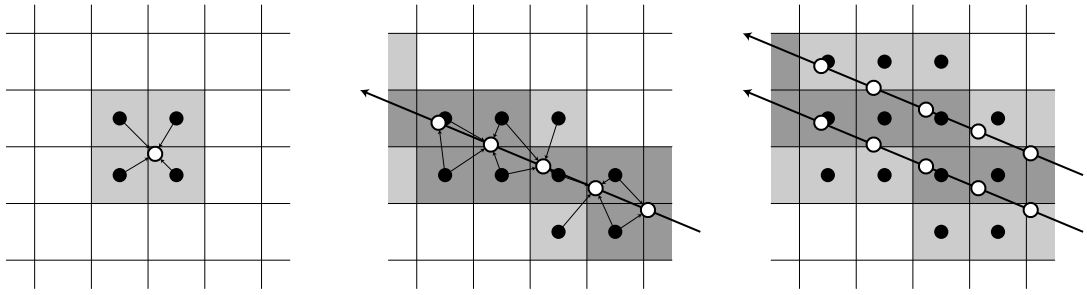
Linearizací je v kontextu volumetrických dat myšleno mapování, které každému voxelu v trojrozměrném prostoru přiřazuje adresu v jednorozměrném paměťovém prostoru. Vhodná linearizace odráží požadované přístupové vzory k datům a maximalizuje pro ně prostorovou lokalitu. Přístupové vzory při metodě vrhání paprsků jsou vizualizovány na obrázku 3.2.

Při zobrazování volumetrických dat se používá několik přístupových vzorů k datům, které lze v kontextu lokality využít. V kontextu této práce je nejvýznamnějším přístupovým vzorem trilineární interpolace. Tento přístupový vzor čte osm voxelů sousedících v trojrozměrném prostoru pro vygenerování jednoho vzorku. Dá se předpokládat, že čtení těchto osmi voxelů se provádí sekvenčně. Vhodná linearizace by proto měla zajistit, že tyto voxely budou v paměťovém prostoru poblíž sebe.

Dalšími paměťovými přístupy, které přímo pracují se sousedností voxelů, je například počítání gradientů či tri-kubická interpolace. Gradienty počítají s 27 voxely sousedícími v trojrozměrném prostoru. Tri-kubická interpolace se počítá z 64 sousedních voxelů. Vzorkování lze provádět také zaokrouhlením souřadnic na souřadnice nejbližšího voxelu. Tato metoda se sousedními voxely nepracuje, avšak za cenu vysokofrekvenčních artefaktů ve výsledné vizualizaci.

Druhý důležitý přístupový vzor, který je specifický pro metodu vrhání paprsků, je vzorkování podél paprsku. Dá se předpokládat, že jednotlivé vzorky od sebe nebudou vzdáleny víc než několik jednotek voxelů. Z pohledu linearizace je tedy opět nutné, aby byly voxely sousedních vzorků poblíž sebe.

Třetím vzorem, který se v této práci využívá, je koherence sousedních paprsků. Vizualizéry využívající metodu vrhání paprsku se vyznačují tím, že vrhají svazek koherentních paprsků. To znamená, že paprsky jsou vyslány z jednoho bodu podobným směrem nebo z blízkých bodů stejným směrem. U koherentních paprsků lze předpokládat, že vzorky sousedních paprsků budou poblíž sebe. Datová struktura by to opět měla odrážet.



(a) (b) (c)

Obrázek 3.2: Přístupové vzory při vzorkování volumetrických dat. Interpolace vzorků pracuje se sousedními voxely (a). Vzorky podél paprsku sdílí při interpolaci některé voxely (b). Sousední paprsky ve svazku koherentních paprsků sdílí některé voxely (c).

Bloková dekompozice

Nejčastěji používanou datovou strukturou pro rozsáhlá volumetrická data je bloková dekompozice. Ta dělí objem na menší, krychlové pod-objemy, které mají stejnou velikost. V případě, že velikost objemu není násobkem rozměru bloku, je objem zarovnán fiktivními daty,

například opakováním okrajových voxelů objemu. Okrajové hodnoty je potřeba brát v potaz při vizualizaci, kde by se fiktivní voxely mohly projevit chybnou vizualizací.

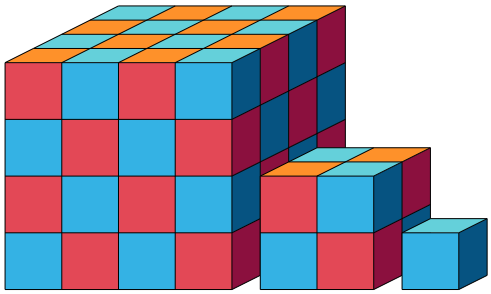
Bloková dekompozice je vhodná datová struktura pro zobrazení rozsáhlých volumetrických dat kvůli tomu, že jednotlivé bloky lze nahrávat do paměti a vizualizovat samostatně, neb reprezentují určenou část objemu. U blokově reprezentovaných dat je potřeba brát v potaz okraje jednotlivých bloků, protože sousední voxely nejsou triviálně dostupné. Se sousedností voxelů se pracuje například v případě trilineární interpolace nebo počítání gradientů. Problém lze řešit determinací bloku pro každý čtený voxel. To však znamená osminásobnou režii při adresování bloků a načítání těchto bloků do paměti, což zvyšuje latenci a případně i paměťovou náročnost. Teoreticky efektivnějším přístupem je do jednotlivých bloků zakomponovat překryv, tj. vzorky ze sousedních bloků. Tento přístup přináší do datové reprezentace redundanci, která je nepřímo úměrná velikosti bloku.

Důležitým parametrem blokové dekompozice je volba velikosti bloku. Ta záleží na použité metodě zobrazování. Menší bloky umožňují přesnější reprezentaci pracovní množiny. Na druhou stranu je objem reprezentován větším počtem bloků a v případě překryvu přinášejí větší paměťovou redundanci. Víceprůchodové zobrazovací techniky zpravidla fungují s většími bloky pro redukci počtu průchodů. Metody vrhání paprsků pracují lépe s menšími bloky, kvůli jemnější granularitě, a tím minimalizaci paměťových operací.

Hierarchická dekompozice

Hierarchická dekompozice reprezentuje volumetrická data hierarchií rozlišení. Výhodou této reprezentace je možnost vzorkovat objem v rozlišení podle požadavků na kvalitu a latenci. Není tedy nutné přistupovat k datům pouze v plném rozlišení. To může značně redukovat množství dat, které je pro zobrazení objemu nutné načíst. Zároveň může sloužit k redukci aliasu při nízkofrekvenčním vzorkování.

Tato reprezentace se v praxi používá s blokovou dekompozicí pro vytvoření hierarchické blokové dekompozice. Tu si lze představit jako blokovou dekompozici každé úrovně hierarchie samostatně. Taková struktura umožňuje nejen volbu rozlišení při vzorkování, ale také efektivní nahrávání regionů objemů do paměti, které poskytuje bloková dekompozice. U tohoto typu dekompozice je nutné dávat si pozor na interpolaci okrajů bloků – zejména v případě, že sousední bloky patří do různé úrovně hierarchie. Hierarchická dekompozice je vizualizována obrázkem 3.3.



Obrázek 3.3: Hierarchická dekompozice objemu. Objem je rekurzivně podvzorkován do reprezentace s poloviční rozlišením v každé dimenzi.

Vlnková dekompozice

Vlnková dekompozice reprezentuje objem jeho vlnkovou transformací. Vlnkovou reprezentaci si lze představit jako hierarchickou dekompozici, kdy každá úroveň hierarchie obsahuje pouze informaci pro zvýšení kvality té předchozí. Díky tomu disponuje výhodami hierarchické reprezentace, avšak s minimální redundancí. Takto reprezentovaná data lze navíc při vhodné volbě vlnky ještě dále komprimovat. Díky tomu je dosaženo vysoké kompaktnosti dat. Zobrazení takto reprezentovaných dat sestává z lokální inverzní transformace objemu až na požadovanou úroveň dekompozice.

Linearizace bloků

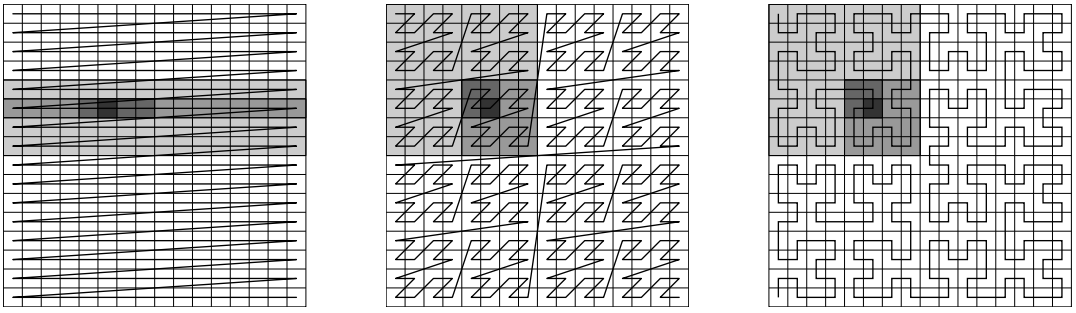
V datových strukturách pro zobrazení rozsáhlých volumetrických dat je důležité řešit nejen dekompozici na bloky, ale také serializaci jednotlivých voxelů v bloku. Čtení malých kousků dat na náhodných pozicích je kvůli lokalitě časově mnohem náročnější než čtení kontinuálních bloků paměti. Proto datové struktury určené pro vizualizaci dat často využívají pokročilé prostor-vyplňující křivky. Nevýhodou těchto křivek je netriviální převod souřadnic z trojrozměrného prostoru do linearizovaného jednorozměrného prostoru.

Prostor-vyplňující křivky zajišťují linearizaci obecně n-dimenzionálních prostorů do jednodimenzionálního prostoru. V kontextu linearizace dat jsou důležité pouze křivky, které pracují s diskrétními prostory, avšak obecně jsou tyto křivky definovány nad prostory kontinuálními. Existuje celá řada křivek, které se pro linearizaci $(n > 1)$-dimenzionálních dat používají.

Nejjednodušším příkladem linearizační křivky je linearizace po řádcích. Pro diskrétní volumetrická data v mřížce si tuto linearizaci lze představit jako rozdělení trojrozměrného objemu na dvourozměrné řezy o jednotkové tloušťce. Tyto řezy jsou rozděleny na jednodimenzionální sekvence, které jsou následně serializovány. Linearizace tímto způsobem je přirozená a překlad souřadnic voxelu na adresu je velmi jednoduchý, avšak prostorové lokality dosahuje pouze v jedné dimenzi. Tento způsob linearizace je vhodný pro náhodné čtení voxelů, případně sekvenční zpracování.

V kontextu prostorové lokality ve všech třech dimenzích je zajímavá Hilbertova křivka a Mortonova křivka. Obě křivky jsou si v několika ohledech podobné. Tyto křivky linearizují prostor rekurzivní aplikací sebe sama na oktanty trojrozměrného prostoru. To znamená, že tyto křivky implicitně tvoří hierarchickou strukturu. Obě tyto křivky zajišťují, že voxely sousedící v trojrozměrném prostoru budou s velkou pravděpodobností poblíž sebe v lineárním paměťovém prostoru. Hierarchická struktura křivek se velmi dobře mapuje na hierarchii paměti, aniž by bylo nutné znát velikosti jednotlivých úrovní paměti. Tento princip se obecně označuje jako *cache-oblivious*.

Tyto křivky jsou však v několika bodech odlišné. Tyto odlišnosti jsou determinující pro použití ve vizualizéru rozsáhlých volumetrických dat. Mortonova křivka oproti Hilbertově poskytuje velmi jednoduchý algoritmus převodu souřadnic z lineárního do trojrozměrného prostoru, který používá pouze bitové operace. Hilbertova křivka na druhou stranu zajišťuje, že sousedící vzorky v linearizovaném proudu spolu vždy sousedí i v prostoru. Sousednost voxelů je vlastnost, kterou lze velmi dobře využít pro kontextově-adaptivní kompresi linearizovaného proudu, avšak v kontextu vzorkování volumetrických dat je irelevantní. Pro volumetrická data je proto vhodná linearizace pomocí trojrozměrné Mortonovy křivky, a to kvůli svému jednoduchému algoritmu pro převod souřadnic. Linearizační křivky jsou zobrazeny v obrázku 3.4.



(a) Po řádcích. (b) Mortonova křivka. (c) Hilbertova křivka.

Obrázek 3.4: Metody linearizace bloku. Zašedlé regiony demonstrují prostorovou lokalitu pro různé paměťové jednotky, které můžou představovat například úrovně vyrovnávací paměti. Serializace Mortonovou (b) a Hilbertovou (c) křivkou oproti linearizaci po řádcích (c) zajišťuje prostorovou lokalitu ve všech dimenzích škálovatelným způsobem.

3.3 Strategie zpracování a zobrazení

Primární strategií pro zobrazení rozsáhlých volumetrických dat je redukce množství dat, které musí vizualizér načíst z paměti a zobrazit. Tu je nutno odlišit od redukce dat za účelem komprese. Zatímco komprese je redukce dat s cílem kompaktně reprezentovat data jako taková, redukce dat za účelem vizualizace si klade za cíl kompaktně reprezentovat volumetrická data pouze v kontextu jednoho specifického vizualizačního průchodu. Technik pro redukci dat je celá řada, od podvzorkování až po různé možnosti abstrakce a filtrování.

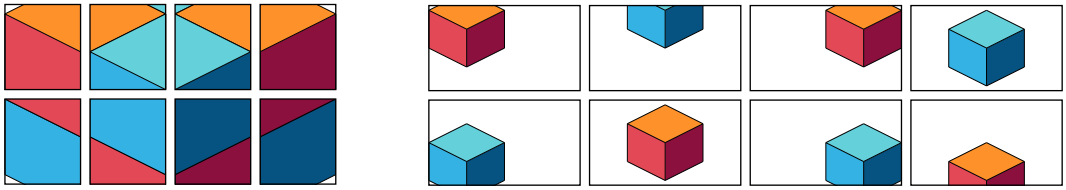
Tato sekce popisuje existující strategie vizualizace rozsáhlých volumetrických dat ve třech kategoriích:

- *Dekompoziční* strategie – rozdělení problému vizualizace rozsáhlých volumetrických dat na podproblémy.
- Strategie plánování *v čase* – volba časových bodů, ve kterých je s daty manipulováno.
- *Paměťové* strategie – volba úrovní paměťové hierarchie, ve které se data nachází.

Dekompoziční strategie

Dekompoziční strategie redukují problém zpracování rozsáhlých volumetrických na menší problémy, které lze snáze řešit. Tyto strategie jsou také označovány za *rozděl a panuj* strategie. Dělení práce umožňuje redukovat paměťové nároky a složitost. Umožňují také paralelizaci nebo distribuci práce. Metody dekompozice jsou vizualizovány obrázkem 3.5.

V kontextu vizualizace lze práci dělit buď v doméně vizualizace nebo v doméně dat. V doméně vizualizace si lze dekompozici představit jako rozdělení výsledného obrázku na dlaždice, které jsou vizualizovány nezávisle na sobě. Příkladem této dekompozice je metoda vrhání paprsků, která výsledný obrázek dekomponuje na jednotlivé pixely. V praxi se však vrhá koherentní svazek paprsků, který se zpracovává společně. Oproti dekompozici v doméně dat lze tímto způsobem implementovat vizualizací řízené optimalizační techniky, např. předčasné ukončení paprsku. Tento přístup očekává přístup ke kompletnímu datasetu, ze kterého si do vlastní paměti nahrává pro každou dlaždici jeho podmnožinu, je vhodný pro paralelizaci na architektuře PRAM.



(a) (b)

Obrázek 3.5: Demonstrace dekompozice problému vizualizace volumetrických dat v doméně vizualizace (a) a v doméně dat (b).

Dělení v doméně dat je provedeno dekompozicí na bloky, které jsou vizualizovány samostatně. Částečné vizualizace pod-objemů jsou poté zkombinovány do výsledné vizualizace objemu. Výhodou tohoto přístupu je jeho plná škálovatelnost. Pracovní jednotky pracují s disjunktními podmnožinami dat, je tedy možné s nimi operovat nezávisle na sobě. Díky tomu je tento přístup vhodný pro distribuované architektury.

Strategie plánování v čase

Množství dat, se kterými je manipulováno, lze také řídit vhodnou volbou časového bodu, kdy se s těmito daty pracuje. Pro zpracování a vizualizaci rozsáhlých volumetrických dat lze využít a kombinovat tři strategie.

První možností je předzpracování dat. Ve fázi předzpracování se provádí časově nejnáročnější operace, dochází k výpočtu akceleračních struktur a metadat. Vstupní data jsou transformována do vhodné reprezentace, tvoří se například hierarchická struktura. Tato fáze se provádí pouze jednou, zpravidla před zahájením samotné vizualizace. Předzpracování redukuje množství operací při samotné vizualizaci, avšak často za cenu větších nároků na úložiště.

Druhou možností je zpracování na požádání. To znamená, že data jsou získávána, až když si o ně následující uzel vizualizačního řetězce zažádá. Jde tedy o využití *pull* modelu. Příkladem tohoto přístupu jsou paprskem řízené metody, které si žádají načtení dat do paměti až v momentě, kdy je nutné jej zobrazit. Data, která nejsou k vizualizaci potřeba, nejsou proto zpracovávána a ani načtena do operační paměti. Při zpracování na požádání se můžou provádět různé lokální operace, například trilineární interpolace, výpočet gradientů, klasifikace, segmentace a filtrování.

Třetí možností je streamování dat. Tento přístup zpracovává data při vizualizaci, avšak oproti zpracování na požádání funguje na *push* i *pull* modelu. Vizualizér v tomto případě zpracovává a vizualizuje data, která jsou mu přidělena předcházejícím uzlem vizualizačního řetězce. Vizualizér má stále možnost vznést požadavek na načtení nebo zpracování určitých dat, avšak na výsledek nečeká, neb mu bude přidělen v budoucnu. Tento přístup umožňuje zobrazení nekompletních dat a temporální adaptaci na aktuální podmínky. Příkladem operace, která se hodí pro *push* model, je načtení dat vyššího rozlišení z hierarchické dekompozice objemu, zatímco se zobrazují data s nižším rozlišením.

Paměťové strategie

Data se v každém uzlu vizualizačního řetězce můžou nacházet v různé úrovni paměťové hierarchie. Vhodnou paměťovou strategií lze adaptovat množství dat v jednotlivých úrovních hierarchie na jejich kapacitu. Data, která jsou přímo vizualizována, se nacházejí v ope-

rační paměti nebo v paměti grafického akcelerátoru. Techniky, které pracují s kompletními daty v operační paměti, se nazývají *in-core*. V kontextu rozsáhlých volumetrických dat data z definice přesahují tuto úroveň paměti, proto se používá pouze pro uložení pracovní množiny a metadat.

Techniky, které pracují s daty mimo operační paměť, se nazývají *out-of-core*. Data v této úrovni můžou být uložena buď v lokálním nebo síťovém úložišti. Tyto techniky musí řešit strategie načítání dat z úložiště, údržbu metadat ohledně načtených dat a překlad adres z prostoru objemu do prostoru pracovní množiny.

Do této sekce taky patří strategie vizualizace *in-situ*. Při této strategii běží generování objemových dat paralelně s jejich vizualizací. Data tímto způsobem není nutné ukládat. Tuto strategii lze dobře kombinovat s *push* i *pull* modelem. Vizualizér buď zobrazuje data, která jsou mu přidělována, nebo přímo žádá simulaci či měřidlo, aby mu data potřebná pro vizualizaci získala.

Za paměťovou strategii by se také dal označit mechanismus distribuovaných architektur pro zobrazování rozsáhlých volumetrických dat. Každý uzel distribuovaného systému vizualizuje podmnožinu dat. Výsledné vizualizace jsou poté zkomponovány do jednoho snímku.

3.4 Překlad adres

Společnou vlastností technik pro zobrazování rozsáhlých volumetrických dat je, že v operační paměti se nachází pouze pracovní množina, která je nutná pro zobrazení těchto dat. Z pohledu vizualizéru je vhodné objem adresovat a vzorkovat v normalizovaných souřadnicích z intervalu $\langle 0, 1\rangle^3$. Je proto nutné zajistit překlad normalizovaných souřadnic objemu na fyzické adresy voxelů z pracovní množiny. Tento problém se zpravidla řeší tabulkou, která je v rámci determinace pracovní množiny aktualizována podle dostupných dat.

Pro různé reprezentace volumetrických dat existuje řada přístupů, kterými lze překlad adres řešit. Koncepčně jsou to přístupy pro překlad adres nehierarchických a hierarchických dat. Hierarchickými daty je v tomto případě myšlena taková reprezentace objemu, která jej reprezentuje hierarchií rozlišení. Překlad adres může být navíc proveden pomocí jednoúrovňových nebo víceúrovňových tabulek. Jednoúrovňové tabulky mapují souřadnice přímo na data. Víceúrovňové tabulky oproti tomu tvoří separátní hierarchii, čímž je dosaženo škálovatelnosti nejen dat, ale také samotné překladové tabulky.

Přímý přístup

Tento přístup se používá v případě, že se celý objem nachází v operační paměti. Není zde potřeba žádná tabulka, neboť existuje přímé mapování z normalizovaných souřadnic do souřadnic objemu. V kontextu zobrazování rozsáhlých volumetrických dat není tento přístup použitelný, neboť data přesahují operační paměť stroje.

Jednoúrovňové tabulky pro nehierarchická data

Tento přístup je prvním přístupem, který lze aplikovat na zobrazování rozsáhlých volumetrických dat. Ve své podstatě jde o tabulku, která mapuje souřadnice vzorku na fyzickou adresu. V případě rozdělení objemu do bloků tabulka mapuje souřadnice bloku na adresu, kde jsou fyzicky uloženy voxely daného bloku. Překlad normalizovaných souřadnic sestává z determinace bloku, do kterého daná souřadnice patří, přečtení hodnoty z tabulky a deter-

minace souřadnic voxelu uvnitř tohoto bloku. Za implementaci tohoto přístupu lze označit virtuální textury bez mip map, které se používají na GPU.

Jednoúrovňové tabulky pro hierarchická data

Jednoúrovňové tabulky lze triviálním způsobem rozšířit na hierarchická data. To si lze představit jako vytvoření jednoúrovňové tabulky pro každou úroveň hierarchie dat. Z hlediska adresování normalizovaných souřadnic je navíc nutné determinovat úroveň hierarchie dat, ze kterých má být hodnota voxelu získána. Tento přístup je výhodný díky tomu, že v pracovní množině lze snadno kombinovat bloky z různých úrovní hierarchie.

Stromové struktury

Pokud hierarchická data tvoří stromovou strukturu, lze tuto strukturu také implementovat a interpretovat jako víceúrovňovou překladovou tabulku. Ta každému uzlu stromu přiřazuje adresu dat, který daný uzel pokrývá. Determinace fyzické adresy se poté provádí paralelně s průchodem stromové struktury při zobrazování dat. Tento přístup má nevýhodu v tom, že úrovně překladové tabulky jsou spojeny s úrovní rozlišení dat. V praxi se pro hierarchická data používají oktalové stromy. To znamená, že každá úroveň hierarchie má v každé dimenzi poloviční rozlišení než ta předchozí. Z pohledu zobrazování je tento rozdíl rozlišení dat velkým skokem ve výsledné kvalitě vizualizace. Z pohledu překladu adres je však hloubka těchto stromů příliš velká a pro rozsáhlá data je potřeba velké množství kroků pro nalezení fyzické adresy.

Víceúrovňové tabulky

V případě extrémně rozsáhlých volumetrických dat v kombinaci s dělením na bloky malých rozměrů jsou jednoúrovňové tabulky nepraktické z hlediska uložení samotných tabulek. Řešením tohoto problému jsou víceúrovňové tabulky stránek. Tyto tabulky stránek neadresují data přímo, ale tvoří samostatnou hierarchii. Data adresuje až nejnižší úroveň hierarchie. Překlad probíhá průchodem této hierarchie. Oproti jednoúrovňovým tabulkám je překlad náročnější, avšak škálovatelnost těchto tabulek je mnohonásobně větší. Hierarchické tabulky jsou ortogonální k hierarchickým datům. Pro každou úroveň hierarchie dat je zvlášť vytvořena hierarchická tabulka pro překlad adres. Oproti překladu v rámci datové hierarchie lze při tomto přístupu odladit větvení stromu překladové tabulky nezávisle na větvení hierarchie dat.

3.5 Determinace pracovní množiny

Determinace pracovní množiny je důležitý krok v každém řetězci pro vizualizaci rozsáhlých volumetrických dat, bez pracovní množiny by totiž nebylo možné s těmito daty vůbec pracovat. Volba pracovní množiny ovlivňuje kvalitu a latenci výsledné vizualizace. V ideálním případě pracovní množina neobsahuje žádná data, která se na vizualizaci nepodílejí. Toho však nelze v realitě docílit a využívají se heuristické přístupy. Existuje celá řada přístupů, které se liší svou efektivitou, výpočetní náročností a flexibilitou. Různé přístupy berou v potaz různé vstupy.

Volba podle pohledového kuželu

První aspekt, který je brán v potaz při tvorbě pracovní množiny, je pohledový kužel. Bloky objemu, které leží kompletně mimo pohledový kužel, nemají vliv na vizualizaci. Toho lze využít a takové bloky do pracovní množiny vůbec nezahrnovat. Tím je dosaženo redukce množství bloků v pracovní množině. Tento přístup není sám o sobě škálovatelný a je potřeba jej kombinovat s ostatními přístupy.

Volba podle klasifikace a segmentace

Klasifikace a segmentace objemu jsou další aspekt, který lze brát v potaz při tvorbě pracovní množiny. Segmentaci lze využít v případě, že je potřeba vizualizovat pouze vybraný segment objemu. Logicky pak není potřeba, aby pracovní množina obsahovala bloky, které se na vybraných segmentech nepodílí. Z pohledu klasifikace lze využít přenosové funkce pro determinaci neviditelných bloků. K tomuto účelu jsou pro každý blok objemu vypočítány minimální a maximální hodnoty voxelů. Při determinaci pracovní množiny se zjišťuje, jestli přenosová funkce v tomto intervalu generuje viditelná data. Tento přístup však počítá s dostupností minimálních a maximálních hodnot bloků, což může znamenat značnou režii. Informaci o viditelnosti v závislosti na klasifikaci a segmentaci lze předpočítat, nebo ji lze získávat za běhu. Přístupy se liší ve své dynamičnosti vzhledem ke změně přenosové funkce a v časové i prostorové náročnosti.

Volba podle zastínění

Dalším aspektem, který lze brát v potaz při determinaci pracovní množiny, je zastínění jednotlivých bloků. To znamená, že bloky, které do výsledné vizualizace přispívají nejvíce, jsou voleny z nejnižší úrovně hierarchie, a tím i s největší kvalitou. Zastíněné bloky jsou poté voleny ve vyšších úrovních hierarchie s nižší kvalitou. Tento přístup často vede na víceprůchodové metody, kdy jedním z průchodů je odhad okluze v prostoru, na základě kterého se pracovní množina tvoří. Determinaci tímto způsobem je také možné dělat temporálně. Paprskem řízené metody tvorby pracovní množiny můžou tuto metriku používat rovnou při průchodu objemu paprskem.

Paprskem řízené metody

Paprskem řízené metody využívají přímo metodu vrhání paprsků pro determinaci pracovní množiny. To znamená, že paprsek začíná s prázdnou pracovní množinou, do které si přidává bloky podle potřeby. Díky tomu je implicitně dosaženo volby pracovní množiny na základě pohledového kuželu, jelikož žádný paprsek není vygenerován mimo pohledový kužel. V případě aplikace optimalizačních technik, jako je přeskakování prázdného prostoru a předčasné ukončení paprsku, je dosaženo volby pracovní množiny i na základě přenosové funkce a zastínění. Paprskem řízené metody můžou využívat *pull* model pro získání dat, která nejsou fyzicky dostupná, nebo můžou fungovat temporálně a zažádat si o daný blok v další iteraci vizualizace. V tomto případě můžou nedostupný blok přeskočit, nebo jej můžou v případě hierarchických dat nahradit jejich reprezentací v nižším rozlišení. Mimo determinaci viditelnosti bloků můžou paprskem řízené metody lokálně determinovat také rozlišení bloku, ze kterého chtějí vzorek číst. Toto rozhodnutí může být provedeno buď pro každý vzorek zvlášť, nebo na základě pod-objemu, ve kterém se vzorek nachází.

3.6 Současné přístupy pro zobrazení rozsáhlých volumetrických dat

Guthe et al. [12] využívá hierarchické vlnkové reprezentace pro kompresi a vizualizaci rozsáhlých volumetrických datasetů na GPU. Díky povaze vlnkové transformace je dekomprese prováděna lokálně a adaptivně na základě aktuálních pozorovacích podmínek, čímž je dosaženo interaktivního zobrazení. Tato metoda využívá hardwarové texturovací jednotky.

Kohlmann et al. [17] srovnává lineární a blokové rozvržení dat v kontextu různých přístupových vzorů na rozsáhlých volumetrických datech. Přístupy jsou testovány na implementaci v jazyce Java, která je určena pro použití v medicínských pracovních stanicích.

Gobetti et al. [9] řeší problém zobrazení rozsáhlých volumetrických dekompozicí objemu na bloky, které jsou organizovány a agregovány do pyramidové struktury. Při běhu algoritmu běží na CPU adaptivní loader, který aktualizuje množinu bloků v paměti GPU na základě aktuální transformace, přenosové funkce a zpětné vazby z GPU. Ze strany GPU se provádí vrhání paprsků a průchod neprázdnými bloky v pořadí odpředu dozadu. Tento přístup umožňuje interaktivní zobrazování giga-voxelových datasetů na stolním PC.

Crassin et al. [2] využívá streamování pro dosažení interaktivní vizualizace na rozsáhlých datasetech. Toto řešení využívá adaptivní reprezentaci pracovní množiny, která je aktualizována na základě průchodu paprsku objemem. Datová struktura navržená v této práci využívá faktu, že nejvíce informací v objemu je obsaženo na rozhraní prázdného a neprázdného prostoru. Řešení se dokáže temporálně adaptovat na požadovaný kompromis mezi kvalitou vizualizace a latence.

Engel [6] představuje framework, který zvládá vizualizovat objemy teravoxelových rozměrů na GPU. Tento framework je založený na progresivním přístupu, který využívá hierarchii rozlišení pro dosažení interaktivních snímkových frekvencí. Obsah pracovní množiny je determinován průchodem paprsku stromovou strukturou. Data, která nejsou viditelná, proto nejsou nikdy načtena do paměti. Framework si dokáže poradit s hustě vzorkovanými anizotropními daty.

Knoll et al. [16] řeší problém vizualizace rozsáhlých volumetrických dat na CPU. Objem je rozdělen na bloky, nad kterými je postavena implicitní hierarchie hraničních objemů. Tato hierarchie je při vizualizaci heuristicky ořezávána na základě před-integrované přenosové funkce, což vede na přeskočení prázdného a rychlou integraci homogenního prostoru. Zobrazování probíhá průchodem této hierarchie v rámci vrhání paprsků. Tento algoritmus utilizuje jak paralelizaci, tak i vektorizaci. Výhodou tohoto přístupu je absence víceúrovňové dekompozice objemu. Tento algoritmus dosahuje menší latence v porovnání s GPU přístupy pro rozsáhlé datasety.

Hadwiger et al. [13] navrhl systém vizualizace, který pracuje s kontinuálním tokem dvourozměrných snímků z elektronového mikroskopu. Tento systém využívá hierarchickou překladovou tabulkou pro dosažení škálovatelnosti na úroveň *petascale* dat. Vizualizace je implementována metodou vrhání paprsků. Systém pro determinaci pracovní množiny funguje na principu virtualizace paměti, kdy je přístup k nedostupným datům interpretován jako výpadek vyrovnávací paměti. Systém funguje bez nutnosti předpočítávat hierarchii rozlišení před samotnou vizualizací. Tento systém je plně řízen vizualizací – data, která se nepodílí na výsledné vizualizaci, nejsou zpracovávána.

Reichl et al. [23] využívá techniky pro rozsáhlá volumetrická data pro vizualizaci prostorově kontinuálních 3D polí. Data jsou podvzorkována do hierarchické blokové reprezentace, nad kterou je sestaven oktalový strom. Každý blok v tomto stromě je transformován do vlnkové reprezentace a komprimován. Vzorky z bloků jsou dekomprimovány na

GPU a vizualizovány pomocí metody vrhání paprsků. Tento přístup dosahuje nižší latence v porovnání s přímým zobrazováním polí pouze za cenu mírně vyšších nároků na úložiště.

Wald et al. [27] představuje framework OSPRay, který mimo jiné umožňuje aplikovat metodu vrhání paprsku pro vizualizaci rozsáhlých volumetrických dat na CPU. Problém rozsáhlých volumetrických dat je řešen rozdělením objemu na dvouúrovňovou blokovou strukturu s bloky o velikostech 512^3 a 2^3. Tento framework dokáže pracovat s libovolnými daty, které je možné vzorkovat, není tedy omezen pouze na pravidelnou mřížku.

Kapitola 4

Návrh řetězce pro vizualizaci rozsáhlých volumetrických dat

Cílem této kapitoly je návrh vizualizačního řetězce, který umožní relativně interaktivní vizualizaci rozsáhlých volumetrických dat metodou vrhání paprsků. Vizualizační část řetězce musí být konfigurovatelná hodnotou, která přímo ovlivňuje velikost pracovní množiny. Řetězec je navržen s cílem jeho implementace v jazyce C++. Tato práce se zabývá několika aspekty návrhu tohoto řetězce:

- Blokově-hierarchická datová struktura a uspořádání dat v paměti.
- Oktalový strom pro překlad adres.
- Vizualizace metodou vrhání paprsků a její optimalizace.
- Volba pracovní množiny podle zastínění paprsku.
- Mapování struktur do paměti a stránkování.
- Paralelizace a vektorizace pomocí SIMD instrukcí.

Prvním řešeným krokem je návrh vhodné datové reprezentace. V sekci 4.1 je popsán návrh blokové struktury nad volumetrickými daty. Sekce 4.2 popisuje sestavení hierarchické datové struktury a oktalového stromu nad blokovými daty. Takto reprezentovaná data jsou na nejnižší úrovni optimalizovány pro rychlé a efektivní vzorkování, na vyšší úrovni potom pro efektivní volbu pracovní množiny, a tím i řízení kvality celé vizualizace. Ke každému bloku datové hierarchie jsou v separátní struktuře zaznamenány statistické informace o regionu, který daný uzel pokrývá. Na základě těchto metadat je poté možné přeskakovat prázdné regiony při průchodu paprsku objemem. Tato datová struktura je navržena tak, aby ji bylo možné předpočítat a poté opakovaně využívat pro vizualizaci.

Nad datovou strukturou je navržena adaptace metody vrhání paprsků, která využívá průchodu paprsku oktalovým stromem pro rychlé adresování bloků. Pro metodu vrhání paprsků byla navržena metoda volby pracovní množiny, která je založená na aktuálním zastínění paprsku. Vizualizér k údržbě pracovní množiny využívá mapované soubory a mechanismus stránkování. V rámci práce bylo experimentováno s možnostmi paralelizace a vektorizace navrženého řetězce. Vrhání paprsků v datové struktuře se věnuje sekce 4.3.

4.1 Bloková struktura pro efektivní vzorkování

Nejmenším stavebním blokem struktury pro volumetrická data, která má umožňovat vizualizaci na CPU, je struktura, která umožňuje jejich efektivní vzorkování. V kontextu této práce se autor zaměřuje zejména na vzorkování dat pomocí trilineární interpolace, avšak obecně existuje celá řada metod, kterými lze data vzorkovat. Za zmínku stojí například počítání gradientů z dat, kterým lze aproximovat normály, a tím i osvětlovací model.

Aplikace prostor-vyplňujících křivek na volumetrická data

Mortonova křivka zajišťuje prostorovou lokalitu pro všechny v této práci relevantní přístupové vzory. Jediným problémem její aplikace na volumetrická data je, že je v diskrétním prostoru definovaná pouze pro krychle s hranou o velikosti mocniny dvojky. Pokud by tedy byla aplikovaná na celý objem, musel by být přesahující prostor uměle doplněn prázdnými voxely na požadovanou velikost krychle. To by ve spoustě případů – zejména u rozsáhlých dat – způsobovalo neúměrnou režii, kvůli které by byl takový formát nepoužitelný.

Jako řešení tohoto problému byla navržena reprezentace, která dělí objem do krychlových bloků s hranou o velikosti mocniny dvojky. Každý blok objemu je pomocí Mortonovy křivky linearizován zvlášť. Při vhodné velikosti bloku tento přístup kombinuje všechny výhody linearizace Mortonovou křivkou a minimalizuje paměťovou režii – zarovnány jsou pouze okrajové bloky.

Duplikace okrajů

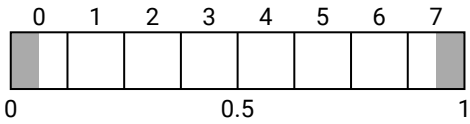
Rozdělením objemu do bloků vzniká další problém, který je potřeba vyřešit. Tento problém je demonstrován na trilineární interpolaci. Trilineární interpolace potřebuje k vyprodukování hodnoty přečíst $2 \times 2 \times 2$ sousedních voxelů. Linearizace Mortonovou křivkou zajistí, že voxely budou s velkou pravděpodobností v nejnižší možné úrovni paměťové hierarchie. Není však zajištěno, že všechny vzorky budou součástí jednoho bloku. V případě trilineární interpolace na hranicích bloků by mohlo dojít ke čtení až z osmi sousedních bloků. To je nevýhodné kvůli režii překladu adres při zjišťování příslušnosti jednotlivých voxelů do bloků.

Z tohoto důvodu byla zavedena duplikace okrajových vzorků v bloku. Blok o velikosti $(2^N)^3$ tak reprezentuje $(2^N - 1)^3$ unikátních vzorků objemu. Zbylé vzorky jsou duplikovány ze sousedních bloků. Tím je zajištěno, že pro libovolný vzorek z objemu existuje právě jeden blok, který obsahuje všechny voxely nutné pro trilineární interpolaci. Na druhou stranu znamená tento překryv větší nároky na paměťové úložiště. Redundanci lze spočítat pomocí rovnice 4.1.

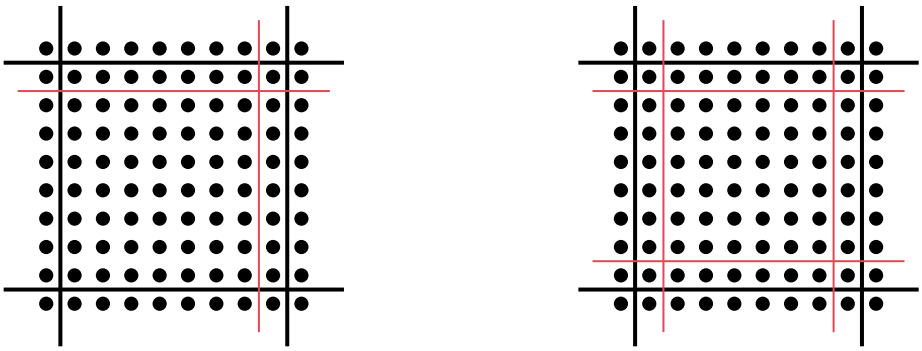
$$R_r = 1 - \frac{(2^N - 1)^3}{(2^N)^3} \tag{4.1}$$

Z rovnice je zřejmé, že zavedená redundance klesá s rostoucí velikostí bloku. S tou však také roste potenciální režie zarovnání dat, která svými rozměry neodpovídají násobkům rozměrů jednoho bloku. Koncept duplikace okrajů je možné rozšířit pro možnost rychlého výpočtu gradientu nebo tri-kubickou interpolaci. V takovém případě je nutno duplikovat více okrajových vzorků a redundance tím roste. Duplikace okrajů je v kontextu sémantického pohledu na blok jako na texturu vizualizována v obrázku 4.1. Obrázek 4.2 demonstruje překryvem způsobenou redundanci.

Alternativou k překryvu sousedních bloků vždy zůstává možnost okraje neduplikovat vůbec a pro každý voxel zvlášť indexovat blok, do kterého náleží. V rámci této práce je však nezávislost bloků důležitá pro principy, které jsou popsány v následující kapitole. Tato alternativa proto není v rámci této práce vyhodnocována. Volbě velikosti bloku se věnuje experimentální vyhodnocení v kapitole 6.



Obrázek 4.1: Demonstrace denormalizovaných (horní hodnoty) a normalizovaných (spodní hodnoty) souřadnic jednorozměrného bloku. Zašedlé regiony reprezentují data mimo podprostor bloku. Pro jejich korektní interpolaci v těchto regionech je nutné data vzorkovat z příslušného sousedního bloku.



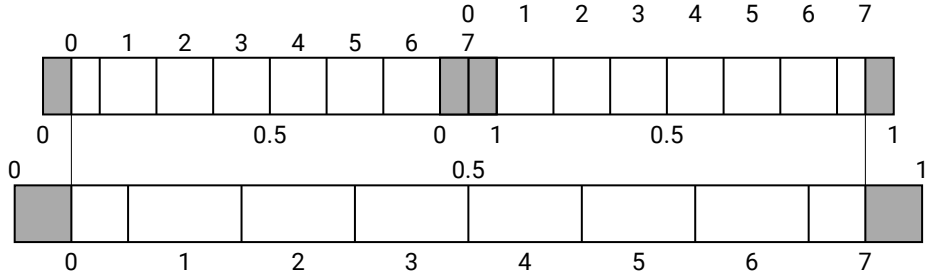
(a) (b)

Obrázek 4.2: Jednoduchá duplikace okrajů v bloku vhodná pro trilineární interpolaci (a) a zdvojená duplikace okrajů vhodná pro počítání gradientů (b).

4.2 Hierarchická reprezentace pro efektivní tvorbu pracovní množiny

Bloková reprezentace volumetrických dat umožňuje rychlé vzorkování podél paprsku, neřeší však samotný problém rozsáhlosti těchto dat a jejich načítání do operační paměti. Tato sekce se věnuje popisu návrhu datové struktury, která umožňuje tvorbu pracovní množiny pokrývající celou viditelnou část objemu. Pracovní množina se vejde do operační paměti a umožňuje tak jednoprůchodovou vizualizaci rozsáhlého objemu.

Toho je docíleno sestavením hierarchické reprezentace dat, kde každá úroveň hierarchie je reprezentována blokovou strukturou popsanou v předchozí sekci. Nad jednotlivými úrovněmi datové hierarchie je sestaven implicitní oktalový strom, který mapuje uzly na datové bloky z hierarchie. Tento strom slouží jako tabulka pro překlad adres bloku na adresu v paměti, zároveň však agreguje informaci o minimální a maximální hodnotě v podprostoru, který daný uzel pokrývá. Takto sestavenou strukturu lze využít v kombinaci s algoritmem průchodu paprsku oktalovým stromem pro rekurzivní přeskakování prázdného prostoru na základě aktuální klasifikace. Zároveň jej lze využít pouze jako tabulku, která umožňuje přístup k libovolnému bloku v libovolné úrovni v konstantním čase bez nutnosti



Obrázek 4.3: Dvě úrovně hierarchické reprezentace jednorozměrných bloků s duplikací okrajů. Sousedící bloky v jedné úrovni hierarchie duplikují jeden společný blok. Hierarchie je tvořena tak, aby blok vyšší úrovně reprezentoval stejný podprostor objemu jako příslušné bloky nižší úrovně.

strom rekurzivně procházet. Díky těmto vlastnostem je navržená datová struktura flexibilním prostředkem pro vizualizaci rozsáhlých volumetrických dat celou řadou metod, které jsou popsány v kapitole 4.3. Hierarchická struktura sestavená nad blokovou dekompozicí v kontextu sémantického pohledu na objem jako na texturu je vizualizována obrázkem 4.3.

Hierarchie bloků

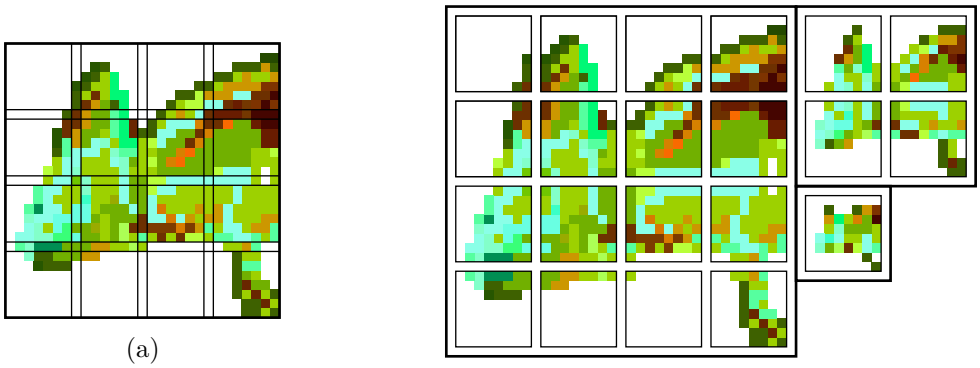
Základem hierarchie bloků je objem zpracovaný do blokové struktury popsané v sekci 4.1. Takto reprezentovaný objem je již rozporcován do samostatných bloků, které tvoří nejnižší úroveň datové hierarchie. Každý blok vyšší úrovně hierarchie agreguje osm sousedících bloků nižší úrovně. Nejvyšší úroveň hierarchie obsahuje jeden blok, který agreguje celý objem. Velikost jednoho bloku je na každé úrovni stejná, každá úroveň tedy reprezentuje objem v poloviční rozlišení v každé dimenzi v porovnání s nižší úrovní.

Při agregaci sousedních bloků je nutno korektně zacházet s okrajovými voxely. Jelikož jsou tyto voxely duplikovány z okolních bloků, je nutné zajistit, aby tato vlastnost byla zachována i pro vyšší úrovně datové hierarchie. Při nesprávném zacházení s okrajovými voxely by docházelo k chybám vizualizace a blokovým artefaktům. Principiálně je zachování této vlastnosti zajištěno získáním okrajových hodnot bloku vyšší úrovně agregací okrajových vzorků v sousedících osmicích v nižší úrovni. Prakticky to znamená, že blok vyšší úrovně neagreguje informaci pouze z osmi, ale z 27 sousedních bloků nižší úrovně. Tento fakt je potřeba brát v potaz při sestavování min-max oktalového stromu. Tvorba hierarchie z blokové struktury je demonstrována obrázkem 4.4.

Důležitá situace, kterou je potřeba v hierarchické struktuře brát v potaz, je zacházení s okrajovými bloky. Při agregaci bloků na okrajích objemu se může stát, že na jeden blok vyšší úrovně připadá méně něž 27 bloků nižší úrovně. V takovém případě se nedostupné voxely nižší úrovně simulují nejbližšími fyzicky dostupnými voxely. Tím je zajištěno, že vzorkování fyzicky dostupných voxelů nezpůsobí artefakty, které by mohlo způsobit například pouhé vyplnění nulami. Z pohledu vizualizace je přesto důležité doplněné voxely nevzorkovat, protože by se mohly na vizualizaci projevit. Zacházení s okrajovými voxely je vizualizováno obrázkem 4.3.

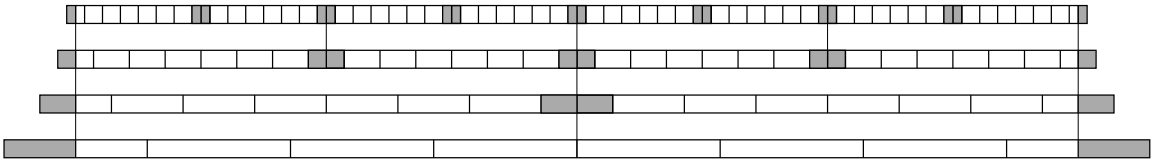
Hierarchicky reprezentovaná data přináší určitou redundanci. Ta však v tomto případě nikdy nepřesáhne hodnotu vypočítanou rovnicí 4.2.

$$R_m = \sum_{n=1}^{\inf} \frac{1}{8^n} = \frac{1}{7} \approx 14,3\% \tag{4.2}$$



(a) (b)

Obrázek 4.4: Bloková dekompozice dvoudimenzionálního obrázku s duplikací okrajů (a), nad kterou je vytvořena hierarchická datová struktura (b).



Obrázek 4.5: Čtyři úrovně hierarchické reprezentace jednorozměrných bloků s duplikací okrajů.

Množství bloků, které jsou fyzicky v datové struktuře obsaženy, lze však na základě sekundární stromové struktury dále redukovat a v extrémním případě lze dosáhnout i komprese.

Oktalový strom a adresace bloků

Samotná hierarchie je pro adresaci bloků teoreticky dostatečná. Bloky mají ve všech úrovních stejnou velikost a rozlišení každé úrovně je známé. Samotná datová hierarchie proto implicitně tvoří oktalový strom.

Taková datová struktura však neumožňuje ukládání metadat o daném podprostoru, které jsou užitečné pro optimalizaci samotného ukládání a následné vizualizace. Zejména důležitá je informace o minimální a maximální hodnotě voxelu v daném podprostoru. Tuto informaci lze použít pro přeskakování prázdného prostoru na základě přenosové funkce, a tím dosáhnout optimalizace při vizualizaci. Zároveň lze tyto hodnoty využít pro identifikaci homogenního prostoru. Ten se vyznačuje tím, že minimální a maximální hodnota mají stejnou hodnotu. V takovém případě není nutné bloky z daného podprostoru vůbec do souboru ukládat, což vede na ušetření paměti.

Z těchto důvodů je při tvorbě hierarchie paralelně sestavován také oktalový strom. Každý uzel tohoto stromu ukládá informaci o minimální a maximální hodnotě podprostoru, který daný uzel pokrývá. K tomu je v každém uzlu obsažena adresa příslušného bloku. Tato adresa je validní pouze v případě, že blok není homogenní, tedy když se minimální a maximální hodnota liší.

V rámci této práce byla zvolena implicitní forma oktalového stromu. Tato forma umožňuje adresování uzlů libovolné úrovně v konstantním čase bez nutnosti průchodu stromu od kořenového uzlu. Zároveň díky tomu není potřeba v uzlech ukládat adresu potomků.

Nevýhodou této formy stromu je nutnost ukládat i prázdné nebo homogenní vnitřní uzly, a to až do listové úrovně stromu.

Uložení dat a metadat

V kontextu rozsáhlých volumetrických dat bude primární datová hierarchie přesahovat kapacitu operační paměti. Je tedy nutné zajistit vhodnou reprezentaci datových struktur na nevolatilním úložišti, ze kterého si tato data bude vizualizační řetězec podle potřeby načítat.

U blokových dat je vhodné zajistit zarovnání bloků na adresy, které jsou fyzickým násobkem svojí velikosti. Jelikož čtení z nevolatilního úložiště probíhá po stránkách, je tímto zajištěn minimální počet čtecích operací pro načtení jednoho bloku do operační paměti. Tuto podmínku lze zajistit využitím dvou separátních souborů – jeden slouží pro uložení dat, druhý pro metadata. Výhoda tohoto přístupu je, že při generování hierarchické reprezentace lze oba soubory generovat paralelně v rámci jednoho průchodu přímo do úložiště.

Soubor pro uložení dat neobsahuje nic než sekvenci serializovaných nehomogenních bloků. Jelikož jsou adresy bloků součástí sekundárního oktalového stromu, není potřeba v tomto souboru zachovávat jakékoliv pořadí bloků. Soubor pro uložení metadat může obsahovat různé informace o objemových datech, primárně však obsahuje oktalový strom, který adresuje bloky v primárním souboru.

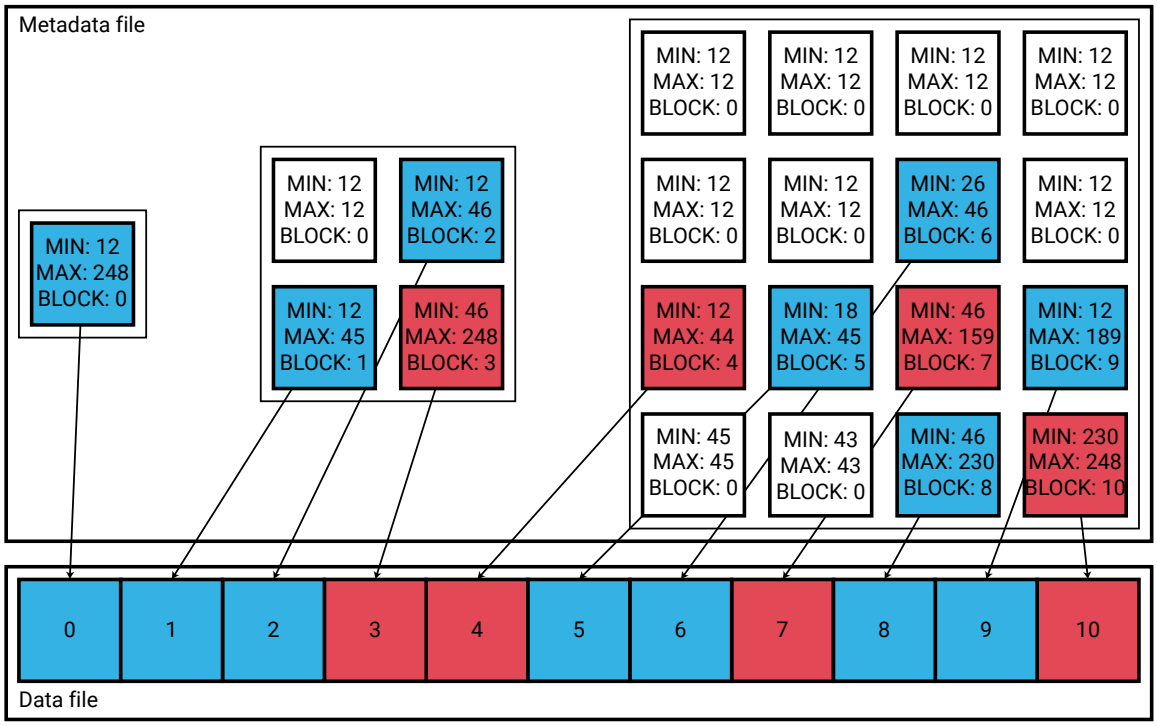
Adresace voxelu v takto připravených strukturách probíhá přečtením oktalového stromu ze sekundárního souboru, vyhledáním uzlu požadované úrovně a kontrolou minimální a maximální hodnoty podprostoru. V případě, že je minimální a maximální hodnota stejná, odpovídá hodnota voxelu této hodnotě. V opačném případě je z primárního souboru přečten datový blok na adrese uložené v uzlu. V tomto bloku dále probíhá adresování specifického voxelu. Je plně úlohou vizualizéru provádět adresaci uzlu, volbu úrovně datové hierarchie a načítaní bloků efektivním způsobem. Soubory se strukturami jsou vizualizovány obrázkem 4.6.

4.3 Vizualizace hierarchicky reprezentovaných dat na CPU

Tato sekce se věnuje vizualizaci rozsáhlých volumetrických dat, která jsou reprezentována navrženou hierarchickou datovou strukturou. V rámci této práce je kladen důraz zejména na metodu vrhání paprsků na CPU. Předpokladem je dostupnost datového i metadatového souboru na lokálním úložišti. Datová struktura však není na tento setup omezena.

Navržená metoda využívá mapované paměti a mechanismu stránkování pro správu pracovní množiny a načítání požadovaných dat do fyzické paměti. Tvorba pracovní množiny je přímo řízena rekurzivním sestupem svazku paprsků oktalovým stromem. Metoda umožňuje omezit velikost pracovní množiny volbou úrovní stromu, do kterých se paprsky v jednotlivých podprostorech objemu zanoří.

V rámci této práce je navržen jednoduchý a efektivní algoritmus, který volbu pracovní množiny provádí na základě aktuální průhlednosti vzorků akumulovaných skrz paprsek. Navržená metoda implementuje běžné optimalizační techniky, které zahrnují přeskakování prázdných regionů, rychlou integraci homogenních regionů a předčasné ukončení paprsku. Klasifikace je prováděna před-integrovanými přenosovými funkcemi.



Obrázek 4.6: Reprezentace navržených datových struktur na nevolatilním úložišti. Nehomogenní uzly oktalového stromu obsahují reference do datového souboru, kde jsou uloženy jednotlivé bloky.

Reprezentace a údržba pracovní množiny

Datová struktura popsaná v této kapitole je navržena jako statická. To znamená, že se předpočítá pouze jednou a při vizualizaci už se nemění. V kontextu vizualizace je tedy nutné zajistit také vhodnou reprezentaci pracovní množiny, která umožňuje dynamické změny.

Ve většině existujících přístupů jsou data načítána explicitní kopií dat z úložiště do fyzické paměti. V takovém případě je nutná manuální správa překladové tabulky, která mapuje trojrozměrný prostor na adresu v pracovní množině. V této práci se autor rozhodl problém údržby pracovní množiny řešit využitím mechanismu mapované paměti a stránkováním. Přístup k datům je získán *mapováním* celé struktury do virtuálního paměťového prostoru.

Vizualizér tímto způsobem získává přístup ke kompletním datům, které může číst a vzorkovat libovolně a podle potřeby. Operační systém zajistí, že přečtené stránky jsou načteny do fyzické paměti a zajišťuje také mapování z virtuální paměti. V případě vyčerpání fyzické paměti se operační systém postará o nahrazení načtených stránek pomocí odladěných stránkovacích algoritmů. Jelikož jsou data mapována pouze pro čtení, v případě zahození některé z mapovaných stránek na rozdíl od swapování nikdy nedochází k zápisu dat zpět na úložiště.

S datovou strukturou namapovanou do virtuální paměti je při vizualizaci nutné brát ohled pouze na množství přečtených bloků v rámci jednoho vizualizačního průchodu objemem, nenavštívené bloky do fyzické paměti nikdy nebudou načteny. Tento přístup využívá veškerou dostupnou fyzickou paměť jako adaptivní cache použitých bloků. Tímto způsobem jsou implicitně optimalizovány koherentní vizualizace objemu v čase – množství výpadků stránek je mezi dvěma po sobě podobnými vizualizacemi minimální. Zároveň lze díky to-

muto přístupu vizualizovat pracovní množinu, která přesahuje dostupnou paměť stroje, avšak za cenu výpadků stránek v rámci jedné vizualizace, a tím i vyšší latence. Dá se předpokládat, že stránky mapované pro čtení mají pro stránkovací algoritmy nízkou prioritu, nemělo by tedy docházet ke konfliktům s interaktivními procesy. Tento přístup správy pracovní množiny je navržen tak, aby byl škálovatelný jak na osobní počítače, tak i na výkonné pracovní stanice.

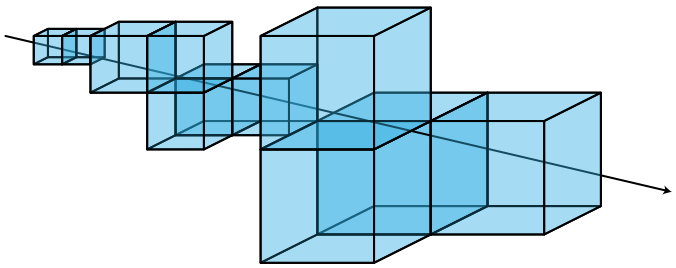
Vzorkování hierarchicky reprezentovaných volumetrických dat

Navržená datová struktura nedefinuje strategii, kterou má být vzorkována. V rámci práce bylo experimentováno s několika způsoby, jak adresování provádět. Tyto způsoby se liší svým výkonem a možnostmi, které poskytují.

Triviálně lze pro každý vzorek v konstantním čase adresovat uzel oktalového stromu v požadované hloubce, ze kterého je získána hodnota vzorku. Každý uzel obsahuje informaci o minimální a maximální hodnotě voxelu v podprostoru, pomocí kterých je minimalizováno množství explicitních vzorkování blokových dat. Tento přístup používá oktalový strom pouze jako chytřejší překladovou tabulku z prostorové souřadnice na virtuální adresu. Tento přístup umožňuje volbu úrovně datové hierarchie *per-sample*. Při tomto přístupu je pro každý vzorek nutné adresovat blok, do kterého patří.

Alternativou k tomuto přístupu je přímé využití oktalového stromu pro průchod paprsku objemem. Algoritmus je aplikován na kořenový uzel stromu. Algoritmus pro každý uzel stromu determinuje oktanty, které paprsek protíná. Poté dochází k rekurzivnímu zanoření algoritmu do uzlů reprezentujících protnuté oktanty, a to v pořadí, které je determinováno směrem vrženého paprsku. Algoritmus v každém uzlu umožňuje provést rozhodnutí, zda se má zanořit do nižší úrovně, blok integrovat, nebo jej úplně přeskočit. Rekurze končí v listových uzlech stromu, které reprezentují objem v nejvyšším možném rozlišení. Průchod paprsku oktalovým stromem je vizualizován obrázkem 4.7.

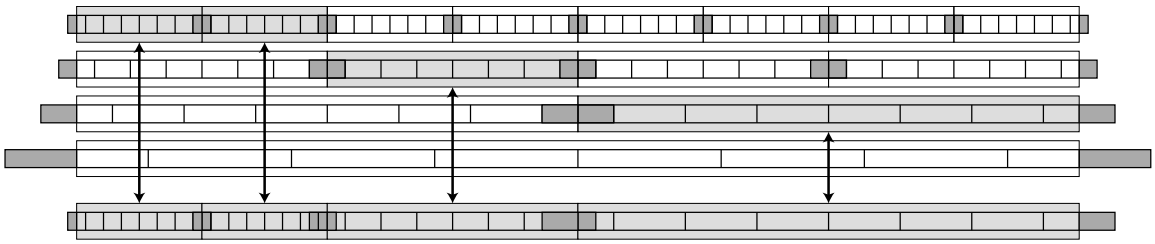
Výhodou tohoto přístupu je, že pokud dojde k fyzickému vzorkování bloku, tak již není potřeba pro každý vzorek blok adresovat. Další výhodu představuje možnost přeskakování prázdného prostoru a rychlá integrace homogenního prostoru, které může vzorkování podél paprsku. Granularita volby kvality je v tomto přístupu na úrovni jednoho uzlu stromu. Při tomto přístupu přináší největší režii právě samotný průchod paprsku oktalovým stromem. Bylo proto věnováno značné úsilí pro jeho optimalizaci a akceleraci.



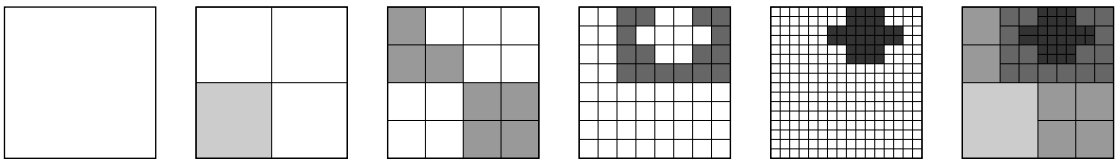
Obrázek 4.7: Průchod paprsku oktalovým stromem s blokovou hierarchií. Paprsek si sám volí bloky, ze kterých chce číst data.

Volba pracovní množiny

Posledním aspektem, který je nutný pro vizualizaci rozsáhlých hierarchicky zpracovaných volumetrických dat vyřešit, je volba pracovní množiny. Aby byla volba efektivní, musí splňovat celou řadu požadavků. Pracovní množina musí kompletně reprezentovat viditelnou část objemu. Kvalita vizualizace pracovní množiny by měla aproximovat vizualizaci objemu v plném rozlišení. Pracovní množina by měla minimalizovat celkové množství dat, které je potřeba načíst do fyzické paměti. Algoritmus determinace pracovní množiny by měl být navíc dostatečně rychlý, aby sám svým časem běhu nepřesahoval latence způsobené přístupy do paměti. Determinace pracovní množiny v hierarchické struktuře je vizualizována obrázky 4.8 a 4.9.



Obrázek 4.8: Selekce bloků ze čtyř-úrovňové hierarchické reprezentace jednorozměrných bloků s duplikací okrajů.



(a) (b) (c) (d) (e) (f)

Obrázek 4.9: Adaptivní selekce bloků z pěti-úrovňové hierarchické reprezentace dvourozměrných bloků. Bloky jsou zvoleny z pěti úrovní hierarchie (a, b, c, d, e) do výsledné pracovní množiny (f).

V praxi je každý algoritmus determinace pracovní množiny kompromisem těchto požadavků. V rámci této práce byl navržen paprskem řízený algoritmus determinace pracovní množiny, který pracuje v kombinaci s průchodem oktalového stromu. Tento algoritmus pro každý uzel oktalového stromu rozhoduje o zanoření v závislosti na aktuální saturaci paprsku podle funkce v rovnici 4.3.

$$L_i = \min\left(-\log_2(1-\alpha) * q, L_{max}\right); \tag{4.3}$$

Kde L_i je požadovaná úroveň hierarchie, ze které budou data vzorkována, α je saturace paprsku a q je volitelný parametr kvality. Maximální úroveň hierarchie L_{max} omezuje výslednou úroveň hierarchie na dostupná data. Podle této funkce každý paprsek začíná vzorkování v nejvyšší možné kvalitě a postupně s průchodem objemu kvalita vzorků klesá. Tento algoritmus je plně řízený samotným paprskem. Výhodou algoritmu je jeho jednoduchost a nenáročnost, provádí se přímo v rámci průchodu paprsku oktalovým stromem. Díky tomu není nutné pracovní množinu ve vizualizéru explicitně reprezentovat a udržovat.

Nevýhodou algoritmu je možnost vzniku blokových artefaktů při nízkých kvalitách vizualizace v případě, že jsou si úrovně hierarchie příliš odlišné. Jako další nevýhodu lze

považovat fakt, že velikost pracovní množiny není tímto algoritmem shora omezena a je závislá na přenosové funkci. To však lze kompenzovat nastavením nižší hodnoty kvality pro taková data. Vizualizér je navržen a implementován tak, aby bylo možné s volbou pracovní množiny experimentovat. V praktickém využití se očekává, že tento algoritmus bude dále laděn na požadovaná data, požadovanou kvalitu a běhové prostředí.

Aplikace optimalizačních technik pro metodu vrhání paprsků

V rámci této práce byly na průchod hierarchickou datovou strukturou aplikovány známé optimalizační techniky pro metodu vrhání paprsků. Tyto techniky zahrnují přeskakování prázdného prostoru, rychlou integraci homogenního prostoru a předčasné ukončení paprsku.

Přeskakování prázdného prostoru slouží k odstřižení podstromů, které nejsou při vizualizaci viditelné. Za prázdný prostor je označen takový prostor, kde zvolená přenosová funkce nemůže vyprodukovat hodnotu s nenulovou průhledností. Tato optimalizační technika je tedy závislá na aktuálně použité přenosové funkci. V rámci průchodu paprsku oktalovým stromem se přeskakování prázdného prostoru aplikuje porovnáním předpočítané minimální a maximální hodnoty voxelů v daném podprostoru s přenosovou funkcí. Pokud přenosová funkce v daném intervalu hodnot neprodukuje viditelný výstup, je uzel včetně podstromu označen za prázdný a v tomto podprostoru neprobíhá další žádný další průchod paprskem. V případě běžně používaných dat a přenosových funkcí vede tento přístup na optimalizaci.

Rychlá integrace homogenního prostoru slouží k odstřižení podstromů, které obsahují neměnná data. Za homogenní prostor je označen takový prostor, který je uniformně vyplněn jednou hodnotou. Tato optimalizační technika není závislá na přenosové funkci. Tento prostor je indikován stejnou minimální a maximální hodnotou voxelů v uzlu stromu. Homogenní prostor je odstřižen již v procesu tvorby hierarchické datové struktury, proto tyto uzly nejsou daty ani podloženy a tato optimalizační technika je proto povinná. Integrace tohoto prostoru sestává z jediného integračního kroku. Z matematického hlediska je tato integrace ekvivalentní integraci po částech, z praktického hlediska je však výpočetně mnohem méně náročná.

Předčasné ukončení paprsku zabraňuje integraci objemu v případě, že je již paprsek saturován. Za saturovaný paprsek je označen takový paprsek, u kterého další vzorkování mění hodnotu jen minimálně. Hraniční hodnota saturace je volena jako kompromis mezi redukcí množství čtených vzorků a artefaktů způsobených nedostatečnou integrací. Tato optimalizační technika je závislá na přenosové funkci. V navrženém vizualizéru je rozhodnutí o ukončení paprsku provedeno na úrovni celých bloků. V případě saturace paprsku je zamezeno dalšímu zanořování při průchodu stromem.

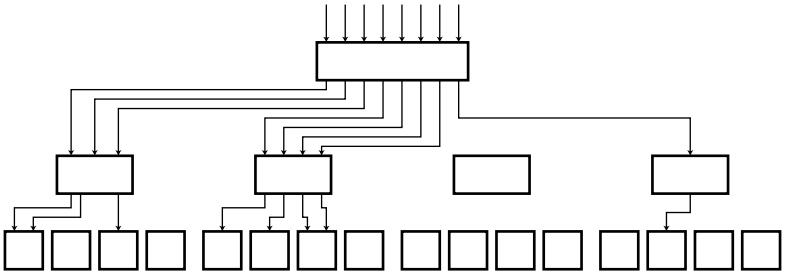
Průchod objemu svazkem koherentních paprsků a akcelerace pomocí vektorizace

Posledním aspektem, kterému se věnuje sekce o návrhu vizualizéru, je využití koherence paprsků pro další optimalizaci přístupů do paměti a k vektorizaci. Vizualizér dosud popsaný v této sekci předpokládá průchod jednoho samostatného paprsku oktalovým stromem. Kompletní vizualizace volumetrických dat touto metodou sestává z vrhání individuálních paprsků pro každý pixel. Tento přístup žádným způsobem nevyužívá koherence sousedních paprsků.

Koherentní svazek paprsků je svazek takových paprsků, u kterých je velká pravděpodobnost, že budou vzorkovat stejná nebo vzájemně blízká data. Vzorkování koherentních paprsků je výhodné v kontextu prostorové lokality. Předpokládá se, že vzorkování více koher-

entních paprsků současně lépe utilizuje paměťovou hierarchii a povede k lepšímu výkonu vzorkování.

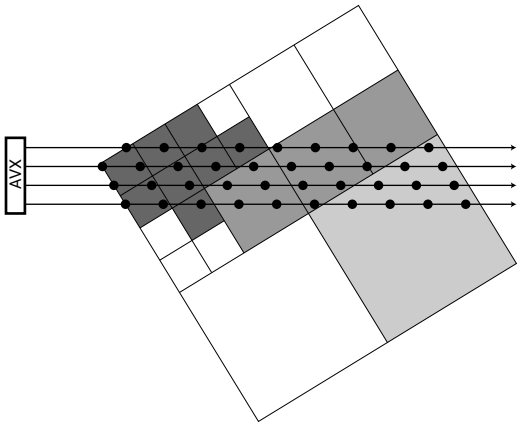
V rámci této práce byl zvolen svazek koherentních paprsků jako kandidát na vektorizaci. Tato volba padla na základě předpokladu, že při průchodu svazku koherentních paprsků objemem se pro všechny paprsky provádí v drtivé většině případů stejné elementární operace, které lze efektivně vektorizovat pomocí SIMD instrukcí. Implementace tohoto přístupu nepředpokládá žádnou specifickou šířku vektoru, proto je tento způsob vektorizace škálovatelný na všechny vektorové instrukční sady. Průchod svazku paprsků oktalovým stromem je demonstrován obrázkem 4.10.



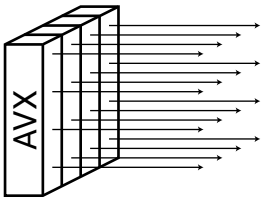
Obrázek 4.10: Průchod svazku paprsků oktalovým stromem do listové úrovně. Paprsky se nejdříve zanořují společně. Na nižších úrovních je menší pravděpodobnost, že sousední paprsky prochází stejnými uzly.

V rámci práce byly provedeny experimenty se dvěma přístupy využití koherence paprsku. Prvním přístupem je přímá aplikace vektorizace na skalární průchod paprsku objemem. V tomto případě svazek sestává z n paprsků reprezentujících n horizontálně sousedících pixelů ve výsledné vizualizaci, kde n je šířka vektorového registru v hodnotách reprezentujících reálná čísla. V případě instrukcí sady AVX jde například v případě použití 32bitové floating point aritmetiky o 8 koherentních paprsků. Tento způsob práce s koherentními paprsky využívá vektorizaci velmi přímým způsobem, avšak celkové množství koherentních paprsků je relativně malé. Paprsky spolu navíc sousedí pouze v horizontální rovině.

Pro maximální využití koherence paprsků bylo experimentováno se strukturou, která je pracovně označována jako *packlet*. Jeden packlet paprsků sestává z $n \times n$ paprsků reprezentujících dlaždici $n \times n$ sousedících pixelů ve výsledné vizualizaci. Z praktického hlediska jde o n vertikálně sousedících vektorů, každý reprezentující n horizontálně sousedících paprsků. Jednotlivé operace jsou implementovány sekvenční aplikací n vektorových operací. Tento přístup rozšiřuje metodu vektorizace přímým způsobem pro větší využití koherence paprsků. Je tomu však za cenu větší režie v případě, že paprsky divergují. U těchto přístupů se navíc obecně očekávají větší nároky na zásobník z důvodu nutnosti pro každý paprsek udržovat jeho stav průchodu. Tato režie roste s množstvím paprsků. Vrhání svazku paprsků je demonstrováno obrázkem 4.11.



Obrázek 4.11: Vrhání vektorového svazku paprsků do hierarchických dat.



Obrázek 4.12: Jeden packlet je reprezentován několika vektory.

Kapitola 5

Implementace

Vizualizační řetězec popsaný v předchozí kapitole byl implementován formou knihovny v jazyce C++20. Sekce 5.1 popisuje architekturu, aplikační rozhraní a použité technologie. V rámci práce byly implementovány nástroje, které tuto knihovnu používají pro zpracování a interaktivní vizualizaci rozsáhlých volumetrických dat. Tyto nástroje jsou popsány v sekci 5.2.

5.1 Knihovna a její aplikační rozhraní

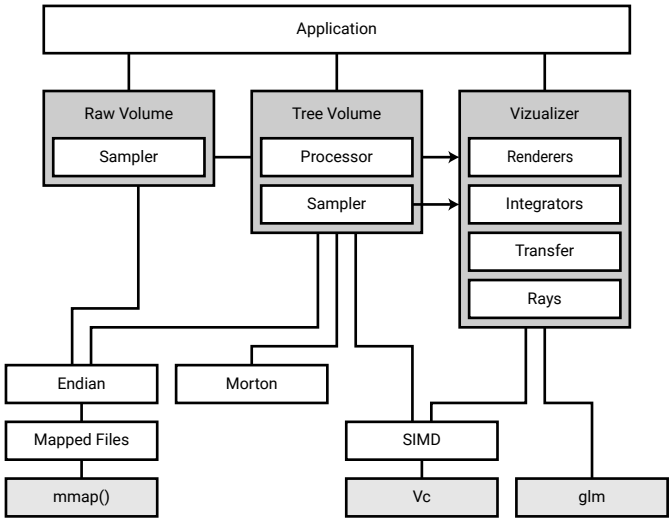
Knihovna je koncipována formou ucelených a nezávislých funkčních bloků, které umožňují uživateli snadno implementovat vizualizér dle přímých požadavků. Knihovna obsahuje vektorové varianty všech typů a funkcí, které se využívají při integraci paprsku. Funkční bloky jsou popsány v tabulce 5.1. Zjednodušený blokový diagram knihovny je vizualizován na obrázku 5.1.

Tabulka 5.1: Funkční bloky implementované knihovny.

Název	**Popis**
Endian	Datové typy s implicitní endian konverzí.
Integrators	Integrace objemu podél paprsku.
Mapped Files	RAII zapouzdření mapovaných souborů.
Morton	Obecná trojrozměrná Mortonova křivka.
Raw Volume	Čtení a vzorkování surových volumetrických dat.
Rays	Reprezentace paprsků a operace nad nimi.
Renderers	Funkce pro kompletní vizualizaci a transformaci objemů.
SIMD	Zapouzdření knihovny Vc.
Tree Volume	Zpracování, čtení a vzorkování hierarchicky reprezentovaných dat.
Transfer	Reprezentace přenosových funkcí a 2D textury.

Drtivá většina těchto modulů využívá šablonového meta-programování. Knihovna se z velké části drží funkcionálního paradigmatu. Velký důraz byl kladen na maximální nezávislost jednotlivých funkčních bloků. Použití knihovny sestává ze skládání poskytnutých funkcí. Knihovna ze strany uživatele umožňuje snadno nahradit libovolný funkční blok za vlastní implementaci. Není tak svým rozhraním nijak omezena.

Knihovna poskytuje dvě třídy pro reprezentaci objemu. Tyto třídy reprezentují surový, resp. hierarchicky zpracovaný objem. Obě třídy využívají mapování paměti do virtuálního



Obrázek 5.1: Zjednodušený diagram architektury jednotlivých modulů navržené knihovny.

adresového prostoru pro čtení dat ze souboru. Třídy poskytují naprosto minimální rozhraní, které umožňuje pouze adresování a čtení voxelů. Jakékoliv další operace, jako je například lineární interpolace, jsou implementovány nad tímto rozhraním.

Knihovna poskytuje kompletní sadu funkcí pro zpracování a vizualizaci rozsáhlých volumetrických dat. Knihovna dokáže vizualizovat jak hierarchicky zpracované rozsáhlé objemy, tak i surová volumetrická data. Vizualizér lze složit kombinací rendereru a integrátoru. Renderer generuje paprsek nebo svazky paprsků dle aktuální transformační matice, projekce a rozlišení výstupu. Tyto paprsky jsou v rámci rendereru transformovány do normalizovaného prostoru objemu. Integrátor provádí integraci objemu podél paprsku nebo svazku paprsků.

Reprezentace surových volumetrických dat

Třída reprezentující datový soubor se surovými volumetrickými daty je definována v hlavičkovém souboru `<raw_volume/raw_volume.h>`. Veřejné rozhraní třídy je následovné:

```
template <typename T>
class RawVolume {
public:
  RawVolume(const char *file_name,
            uint32_t width, uint32_t height, uint32_t depth);

  ...
};
```

Šablonový parametr `T` definuje typ hodnoty voxelu. Jedna instance této třídy napříč celým svým životem reprezentuje jeden celý namapovaný datový soubor, který obsahuje surová volumetrická data. Třída je určena pouze pro čtení těchto dat. Třída předpokládá voxely serializované po řádcích ve formátu *little endian* zvoleného datového typu. Velikost datového souboru na adrese `file_name` musí odpovídat násobku rozměrů objemu a šířky datového typu `width * height * depth * sizeof(T)`. Třída je implementována jako immutable, veškerý stav je definován v momentě konstrukce a v průběhu života se nemění.

Funkce pro vzorkování surových volumetrických dat je definována v hlavičkovém souboru `<raw_volume/sampler.h>` a její vektorová varianta v souboru `<raw_volume/sampler_simd.h>`.

```
template <typename T>
float sample(const RawVolume<T> &volume, float x, float y, float z);

template <typename T>
simd::float_v sample(const RawVolume<T> &volume,
                     const simd::float_v &x,
                     const simd::float_v &y,
                     const simd::float_v &z,
                     const simd::float_m &mask);
```

Tyto funkce jsou parametrizovány vzorkovaným objemem `volume` a trojicí **normalizovaných** souřadnic `x`, `y` a `z`. Tyto souřadnice jsou z intervalu $\langle 0, 1\rangle^3$. Vektorová varianta má navíc parametr `mask`, který definuje maskování jednotlivých hodnot, tj. které hodnoty se mají při čtení paměti přeskočit. Funkce vrací lineárně interpolovaný vzorek objemu, resp. vektor těchto vzorků.

Reprezentace hierarchicky zpracovaných volumetrických dat

Hierarchicky zpracovaná data jsou reprezentována v hlavičkovém souboru `<tree_volume/tree_volume.h>`. Veřejné rozhraní třídy je velmi podobné třídě pro surová data:

```
template <typename T>
class TreeVolume {
public:
  TreeVolume(const char *blocks_file_name,
             const char *metadata_file_name,
             uint32_t width, uint32_t height, uint32_t depth);

  const Info info;
  ...
};
```

Šablonový parametr `T` definuje typ hodnoty voxelu. Jedna instance této třídy napříč celým svým životem reprezentuje jeden soubor dat a metadat, který obsahuje hierarchicky zpracovaná volumetrická data a k nim příslušící oktalový strom. Třída je určena pouze pro čtení těchto dat. Třída je implementována jako immutable, veškerý stav je definován v momentě konstrukce a v průběhu života se nemění. Konstruktor `TreeVolume()` přijímá jako parametr cestu k souboru dat `blocks_file_name` a metadat `metadata_file_name`. Navíc přijímá rozměry volumetrických dat.

`TreeVolume<T>::Info` je datová struktura, která popisuje oktalový strom a data v něm uložená:

```
template <typename T>
struct TreeVolume<T>::Info {
  struct Layer;
```

```
  Info(uint32_t width, uint32_t height, uint32_t depth);

  float width_frac;
  float height_frac;
  float depth_frac;
};
```

`TreeVolume<T>::Info` reprezentuje strukturu oktalového stromu pouze na základě rozměrů objemu. Lze ji tak používat samostatně bez datového souboru. Toho se využívá například při generování samotných hierarchických dat. `Info::Layer` popisuje jednu vrstvu oktalového stromu, která reprezentuje blokově zpracovaná data v jedné úrovni hierarchie rozlišení. Z pohledu klienta jsou důležité zejména položky `width_frac`, `height_frac` a `depth_frac`. Tyto položky definují interval normalizovaných souřadnic $\langle 0, *\_\mathtt{frac}\rangle^3$, který je při vzorkování pokrytý objemovými daty. Vzorky mimo tento interval leží na okrajích, které byly do oktalového stromu doplněny kvůli zarovnání na mocniny dvojky.

Funkce pro generování hierarchických dat a metadat je dostupná v hlavičkovém souboru `<tree_volume/processor.h>`:

```
template <typename T, typename F>
void process_volume(const typename TreeVolume<T>::Info &info,
                    const char *blocks_file_name,
                    const char *metadata_file_name,
                    const F &input);
```

Tato funkce přijímá jako parametr cestu k souboru dat `blocks_file_name` a metadat `metadata_file_name`. Mimo to funkce přijímá informace o hierarchické datové struktuře `info`. Parametr `input` je funktor se signaturou `T(uint32_t x, uint32_t y, uint32_t z)`, který pro souřadnice `x`, `y` a `z` vrací hodnotu voxelu zpracovávaného objemu. Tímto způsobem lze zpracovat libovolně reprezentovaná objemová data, pro která je nutné pouze definovat tento přístupový funktor.

Funkce pro vzorkování hierarchicky reprezentovaných volumetrických dat jsou definovány v hlavičkovém souboru `<tree_volume/sampler.h>` a jejich vektorové varianty v souboru `<tree_volume/sampler_simd.h>`.

```
template <typename T>
float sample(const TreeVolume<T> &volume, float x, float y, float z,
             uint8_t layer);

template <typename T>
simd::float_v sample(const TreeVolume<T> &volume,
                     const simd::float_v &x,
                     const simd::float_v &y,
                     const simd::float_v &z,
                     uint8_t layer,
                     simd::float_m mask);
```

Tyto funkce se od jejich surových variant liší pouze jedním parametrem `layer`, který udává číslo vrstvy, ze které má být vzorek získán. Tyto funkce z normalizovaných souřadnic adresují uzel, ze kterých má být vzorek získán. V případě, že je uzel, do kterého vzorek patří, již známý (například při průchodu paprsku oktalovým stromem), jsou dostupné také varianty, které přímo vzorkují zvolený blok:

```
template <typename T>
float sample(const TreeVolume<T> &volume, uint64_t block_handle,
             float denorm_x, float denorm_y, float denorm_z);

template <typename T>
simd::float_v sample(const TreeVolume<T> &volume,
                     const std::array<uint64_t, simd::len> &block_handle,
                     const simd::float_v &denorm_x,
                     const simd::float_v &denorm_y,
                     const simd::float_v &denorm_z,
                     const simd::float_m &mask);
```

Tyto metody přijímají index bloku `block_handle` a **denormalizované** souřadnice `denorm_x`, `denorm_y` a `denorm_z`. Integrátor může využívat libovolný způsob vzorkování dat.

Renderery

Knihovna obsahuje renderovací funkce pro skalární, vektorové a packletové renderování. Tyto funkce jsou definovány v hlavičkách `<renderers/*.h>`.

```
template <typename F>
void render_scalar(uint32_t width, uint32_t height,
                   float fov, const glm::mat4 &vmt,
                   uint8_t *rgb_buffer, const F &integrator);

template <typename F>
void render_simd(uint32_t width, uint32_t height,
                 float fov, const glm::mat4 &vmt,
                 uint8_t *rgb_buffer, const F &integrator);

template <typename F>
void render_packlet(uint32_t width, uint32_t height,
                    float fov, const glm::mat4 &vmt,
                    uint8_t *rgb_buffer, const F &integrator);
```

Renderery jsou obecné funkce, které pro výřez `width` × `height` generují paprsky s perspektivní projekcí se zvoleným úhlem pohledu `fov`. Na tyto paprsky je aplikována inverze transformační matice `vmt`. Parametr `integrator` je funktor se signaturou `glm::vec4(Ray)` pro skalární renderer, `simd::vec4(simd::Ray)` pro vektorový renderer a `Vec4Packlet(RayPacklet)` pro packletový renderer. Parametr `rgb_buffer` je ukazatel na pole `RGB` hodnot, do kterého jsou ukládány výsledky integrace. Smyčka rendereru je paralelizována pomocí pragmat `OpenMP`[1].

Integrátory

Integrátory jsou funkce, které integrují objem podél paprsku nebo svazku paprsků. Celá řada integrátorů je implementována v hlavičkových souborech `integrators/*.h`. Integrátory implementují vizualizační jádro metody vrhání paprsků. Obsahují veškeré optimalizační techniky popsané v této práci.

[1] OpenMP – https://www.openmp.org/

Pro integraci surových volumetrických dat jsou dostupné integrátory `<integrators/raw_slab.h>` a `<integrators/raw_slab_simd.h>`. Pro integraci hierarchicky zpracovaných volumetrických dat jsou dostupné integrátory `<integrators/tree_slab.h>`, `<integrators/tree_slab_simd.h>` a `<integrators/tree_slab_packlet.h>`. Knihovna obsahuje několik dalších experimentálních integrátorů, které byly implementovány nad rámec této práce. Uživatel má možnost implementovat a použít vlastní integrační jádro, které může například počítat osvětlovací model a vysílat sekundární paprsky do scény. Integrátory jsou definovány následujícími funkcemi:

```
template <typename T, typename F>
glm::vec4 integrate_raw_slab(const RawVolume<T> &volume, const Ray &ray,
                             float step, float terminate_thresh,
                             const F &transfer_function);

template <typename T, typename F>
simd::vec4 integrate_raw_slab_simd(const RawVolume<T> &volume,
                                   const simd::Ray &ray, float step,
                                   float terminate_thresh,
                                   simd::float_m mask,
                                   const F &transfer_function);

template <typename T, typename F, typename P>
glm::vec4 integrate_tree_slab(const TreeVolume<T> &volume,
                              const Ray &ray, float step,
                              float terminate_thresh,
                              const F &transfer_function,
                              const P &integrate_predicate);

template <typename T, typename F, typename P>
simd::vec4 integrate_tree_slab_simd(const TreeVolume<T> &volume,
                                    const simd::Ray &ray, float step,
                                    float terminate_thresh,
                                    simd::float_m mask,
                                    const F &transfer_function,
                                    const P &integrate_predicate);

template <typename T, typename F, typename P>
Vec4Packlet integrate_tree_slab_packlet(const TreeVolume<T> &volume,
                                        const RayPacklet &ray, float step,
                                        float terminate_thresh,
                                        MaskPacklet mask,
                                        const F &transfer_function,
                                        const P &integrate_predicate);
```

Parametr `volume` je integrovaný objem. Parametr `ray` je integrovaný paprsek, resp. svazek paprsků. Parametr `step` je velikost integračního kroku. Parametr `terminate_thresh` je parametr saturace pro předčasné ukončení paprsku. Ve vektorové variantě je parametr `mask`, který slouží pro maskování neaktivních paprsků. Tyto paprsky nejsou integrovány.

Parametr `transfer_function` je funkční objekt se signaturou `glm::vec4(float slab_start, float slab_end)`, resp. `simd::vec4(const simd::float_v &slab_start, const simd::float_v &slab_end)`. Tyto parametry definují hodnotu vzorku na začátku a na konci integračního kroku. Přenosová funkce je libovolná funkce, která splňuje toto rozhraní. Rozhraní předpokládá před-integrovanou přenosovou funkci, která pro daný interval vrací integrál přenosové funkce normalizovaný šířkou tohoto intervalu.

Parametr `integrate_predicate` je funkční objekt se signaturou `bool(const glm::vec3 &cell, uint8_t layer)`, resp. `simd::float_m(const simd::vec3 &cell, uint8_t layer, const simd::float_m &mask)`. Tento predikát řídí průchod paprsku oktalovým stromem. Predikát pro buňku `cell` ve vrstvě `layer` rozhodne, zda má být blok příslušící této buňce integrován paprskem, nebo má dojít k zanoření do nižší úrovně oktalového stromu.

Přenosové funkce

Knihovna poskytuje podporu pro tvorbu po částech lineárních přenosových funkcí. Hlavičkový soubor `<transfer/piecewise_linear.h>` obsahuje metodu, která v mapě klíčových bodů `values` najde dva klíče ohraničující hodnotu `value` a vrátí lineární interpolaci dvou příslušných hodnot.

```
template <typename T>
T piecewise_linear(const std::map<float, T> &values, float value);
```

Knihovna poskytuje podporu pro před-integrování přenosových funkcí v hlavičkovém souboru `<transfer/preintegrate_function.h>`.

```
template <typename F>
Texture2D<float> preintegrate_function(uint32_t size, const F &func);
```

Tato funkce provede integraci libovolného funktoru `func` se signaturou `float(uint32_t)` v intervalu $\langle 0, 1)$. Funkce vrací texturu o rozměrech `size * size`, která mapuje počáteční hodnotu `x` a koncovou hodnotu `y` na integrál přenosové funkce normalizovaný šířkou intervalu. Struktura `Texture2D<T>` implementuje obecnou texturu, která umožňuje čtení a zápis pixelů.

```
template <typename T>
class Texture2D {
public:
  Texture2D(uint32_t width, uint32_t height);

  ...
};
```

Nad touto třídou jsou implementovány funkce pro vzorkování textury. Tyto funkce pro zvolenou texturu `texture` a normalizované souřadnice z intervalu $\langle 0, 1\rangle$, `x` a `y`, vrací lineárně interpolovanou hodnotu vzorku na daných souřadnicích.

```
template <typename T>
float sample(const Texture2D<T> &texture, float x, float y);

template <typename T>
simd::float_v sample(const Texture2D<T> &texture,
                     const simd::float_v &x, const simd::float_v &y,
                     const simd::float_m &mask);
```

Použité technologie

Implementace knihovny využívá knihovnu `glm`[2]. Tato knihovna implementuje typy pro matematické vektory a elementární operace nad nimi. Knihovna `glm` je využita zejména pro implementaci reprezentace paprsků a operací nad nimi.

K vektorizaci pomocí SIMD instrukcí je využita knihovna `Vc`[3]. Tato knihovna implementuje a abstrahuje vektorové instrukce pomocí rozhraní, které definuje obecné vektorové typy a operace nad nimi. Elementární vektorové datové typy jsou využity pro implementaci vektorových variant integračních algoritmů.

5.2 Nástroje pro zpracování a vizualizaci

Nad knihovnou byly implementovány dva nástroje, které umožňují vizualizaci rozsáhlých volumetrických dat. Oba nástroje slouží jako přímá ukázka použití aplikačního rozhraní navržené knihovny. První nástroj umožňuje zpracovat surová volumetrická data do hierarchické reprezentace. Data v této reprezentaci jsou tvořena dvěma soubory, kde jeden soubor reprezentuje oktalový strom a druhý soubor obsahuje datové bloky. Druhý nástroj slouží k interaktivní vizualizaci takto zpracovaných dat. Nástroj implementuje jednoduché uživatelské rozhraní.

Zpracování datové struktury

Zpracování surových dat zajišťuje nástroj `process`, který lze použít následujícím způsobem:

```
./process <raw-volume-file> <width> <height> <depth> <bytes-per-voxel> \
          <data-file> <metadata-file>
```

Tento nástroj na vstupu získá cestu k souboru se surovými volumetrickými daty `<raw-volume-file>` a popis těchto dat `<width> <height> <depth> <bytes-per-voxel>`. Nástroj vytvoří dva soubory. Soubor `<metadata-file>` reprezentuje implicitní oktalový strom tak, jak je popsaný v této práci. Uzly oktalového stromu referují na soubor `<data-file>`, který slouží jako banka bloků.

Vizualizace zpracovaných dat

Vizualizaci zpracovaných dat zajišťuje nástroj `vizualize`, který lze použít následujícím způsobem:

```
./vizualize <data-file> <metadata-file> <width> <height> <depth> \
            <bytes-per-voxel> <transfer-function-file>
```

Nástroj na vstupu získá soubor s bloky `<data-file>` a oktalový strom `<metadata-file>`, společně s popisem volumetrických dat `<width> <height> <depth> <bytes-per-voxel>`. Nástroj k vizualizaci potřebuje definici přenosové funkce, kterou získává ze souboru `<transfer-function-file>`. Data ze souboru s přenosovou funkcí jsou interpretovány jako po částech lineární funkce, mezi klíčovými body se provádí lineární interpolace.

Vizualizační nástroj sestává z výřezu, ve kterém probíhá vizualizace scény, a jednoduchého ladícího uživatelského rozhraní. Pohybu ve scéně lze docílit navigací pomocí kláves

[2] Knihovna `glm` – https://github.com/g-truc/glm
[3] Knihovna `Vc` – https://github.com/VcDevel/Vc

`WASD`. Rotace kamery lze docílit podržením pravého tlačítka myši a tažením. Ladící uživatelské rozhraní mimo libovolné nastavení transformací objemu a kamery umožňuje také nastavení samotného rendereru. Obecně lze konfigurovat délku integračního kroku `Step` a hraniční hodnotu pro předčasné ukončení paprsku `Ray termination threshold`. Volit zde lze z tří navržených rendererů `Scalar Tree`, `Vector Tree` a `Packlet Tree`, které se liší množstvím vyslaných paprsků do scény a vektorizací. Tyto renderery lze řídit parametrem `Quality`, který řídí vliv průhlednosti paprsku na vrstvu hierarchie, ze které jsou získávány vzorky. Mimo to implementace obsahuje další experimentální renderery, které byly implementovány nad rámec této práce.

Reprezentace přenosových funkcí

Nástroj pro vizualizaci přijímá přenosové funkce definované množinou klíčů a hodnot:

```
<number-of-colors>
<key> <red> <green> <blue>
...
<number-of-values>
<key> <alpha>
...
```

Kde `<number-of-colors>` a `<number-of-values>` definují počet klíčových bodů. Klíčové hodnoty `<key>` jsou hodnoty z intervalu $\langle 0, 1 \rangle$, kde 0 odpovídá nejnižší možné hodnotě voxelu v objemu a 1 odpovídá největší možné hodnotě voxelu v objemu. Barevné hodnoty `<red> <green> <blue>` jsou hodnoty z intervalu $\langle 0, 1 \rangle$ které dohromady definují jednu barvu z modelu rgb. Hodnoty průhlednosti `<alpha>` jsou kladné hodnoty, které definují koeficient *neprůhlednosti* vzorku, tj. větší hodnota představuje méně průhledný vzorek.

Použité technologie

Pro tvorbu interaktivní aplikace byla použita knihovna `glfw`[4]. Tato knihovna zajišťuje tvorbu okna, interakci s periferiemi a zajišťuje kontext pro zobrazení vizualizovaných dat.

Pro uživatelské rozhraní byla využita knihovna `Dear ImGui`[5]. Tato knihovna poskytuje nástroje pro tvorbu jednoduchého grafického uživatelského rozhraní.

[4] Knihovna `glfw` – https://github.com/glfw/glfw
[5] Knihovna `Dear ImGui` – https://github.com/ocornut/imgui

Kapitola 6

Experimentální vyhodnocení

V této kapitole jsou popsány experimenty, které vyhodnocují vliv popsaných mechanismů na výkon vizualizéru. Cílem experimentů je průzkum konfigurací datové struktury pro vizualizaci rozsáhlých volumetrických dat metodou vrhání paprsků. Každý experiment je navržen tak, aby testoval jeden aspekt datové struktury bez okolních vlivů. Jelikož by kompletní stavový prostor experimentů přesahoval rámec této práce, jsou zde popsány experimenty se subjektivně největší informační hodnotou. Komplexnější experimenty, které zkoumají vzájemný vliv proměnných, můžou být objektem dalšího zkoumání.

Sekce 6.1 popisuje společné testovací prostředí experimentů. Sekce 6.2 obsahuje sadu experimentů, která měří vzorkovací výkon při zobrazování dat v plném rozlišení. Je zde vyhodnocena samostatná bloková dekompozice a paměťové přístupy. Sekce 6.3 vyhodnocuje navržené vizualizéry na reálných hierarchicky zpracovaných datech s reálnými přenosovými funkcemi. Tato sada experimentů simuluje a vyhodnocuje reálné použití knihovny.

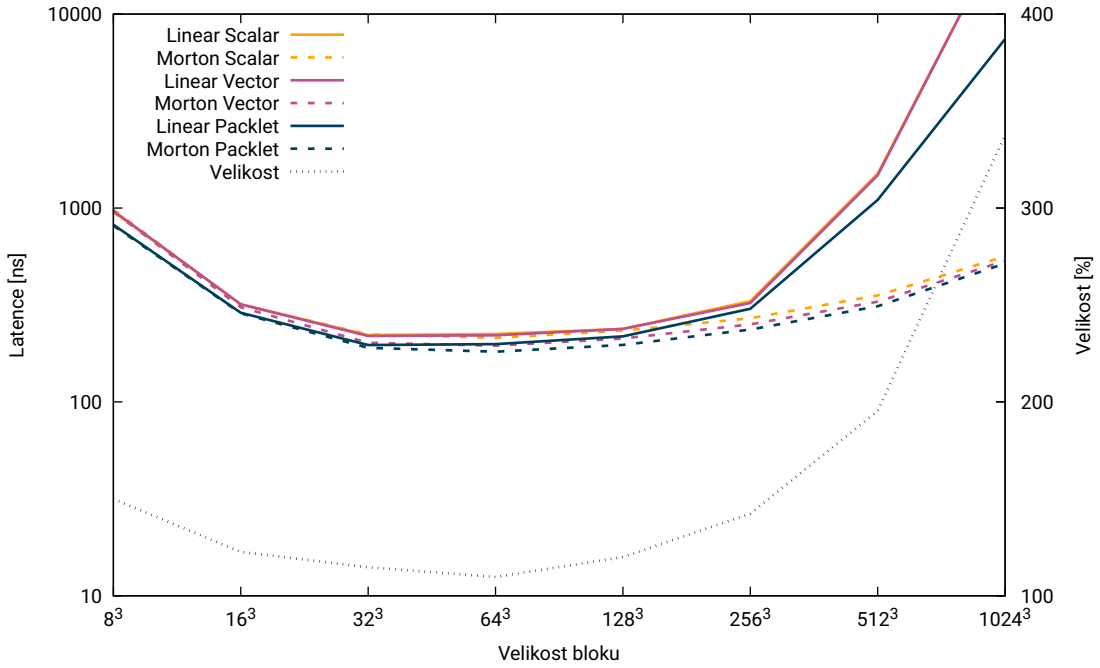
6.1 Testovací prostředí

Experimenty byly provedeny na procesoru **Intel® Core™ i5-8300H CPU @ 2.30GHz**. Paměťová hierarchie testovacího stroje je uvedena v tabulce 6.1. Šířka jednotky vyrovnávacích pamětí je 64 B. Z hlediska pozdějších experimentů s rozsáhlými daty je důležitá také velikost stránky, která je nastavena na běžnou hodnotu 4 KiB.

Tabulka 6.1: Paměťová hierarchie testovacího stroje.

Paměť	**Kapacita**
L1	32 KiB
L2	256 KiB
L3	8 MiB
RAM	8 GiB
SSD	500 GB

Experimenty mimo jiné srovnávají také vliv akcelerace pomocí SIMD instrukcí. V kontextu experimentů je důležitá zejména informace, že byla použita instrukční sada AVX s šířkou vektoru 256 bitů. Veškerá práce s reálnými čísly je prováděna ve 32bitové floating point aritmetice. Šířka vektoru je tedy 8 hodnot.



Obrázek 6.1: Vliv velikosti bloku na průměrnou dobu pro získání jednoho vzorku z rozsáhlých dat 2048^3. Graf srovnává způsoby linearizace. Graf také srovnává skalární, vektorové a packletové vzorkování.

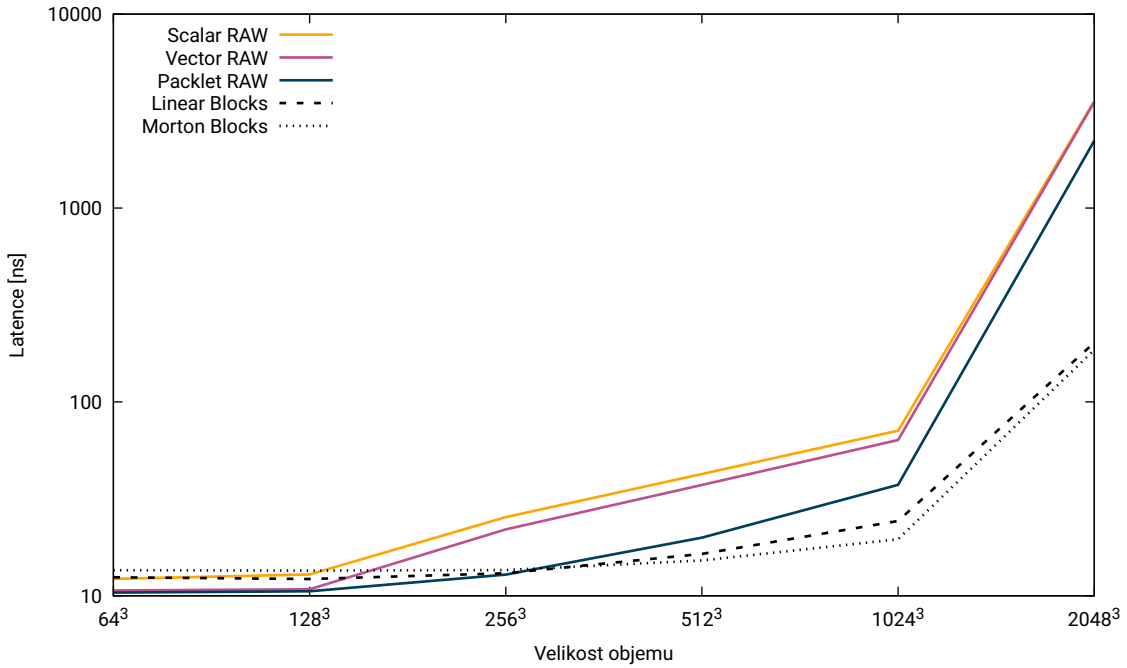
6.2 Vzorkování dat v plném rozlišení

První sada experimentů se věnuje měření vzorkování volumetrických dat v plném rozlišení. V této sekci jsou demonstrovány experimenty nad blokově zpracovanými daty, které měří vliv velikosti bloku, linearizace a vektorizace na vzorkování dat. Poté je demonstrováno škálování rychlosti vzorkování s rostoucí velikostí dat. Veličina měřená v těchto experimentech je čas na získání jednoho vzorku.

Měření probíhalo vizualizací pěti pohledů na objem. Pozice pozorovatele byly zvoleny tak, aby objem pokrýval celý výřez vizualizace. Každý pohled byl vizualizován s rozlišením 256×256 paprsků, pro každý paprsek bylo získáno průměrně 500 vzorků. Veškeré optimalizace byly v rámci měření vypnuty, měřil se pouze čas adresování jednoho vzorku a přístup do paměti. Experiment využívá pouze jedno jádro procesoru. Tato sada experimentů pracuje vždy s daty, které mají šestnáct bitů na voxel.

V rámci prvního experimentu je měřen vliv velikosti bloku na průměrnou dobu pro získání jednoho vzorku objemu 2048^3. Součástí experimentu je také měření vlivu linearizace bloku. Od linearizace Mortonovou křivkou se očekává dosažení lepší prostorové lokality, avšak za cenu náročnějšího adresování voxelů. Výsledek tohoto experimentu je demonstrován na obrázku 6.1.

Z grafu je vidět, že Mortonova křivka mnohem lépe škáluje na rostoucí velikost bloku. Je to tím, že tento způsob linearizace tvoří bloky implicitně uvnitř každého explicitního bloku. Packletový průchod mírně redukuje čas vzorkování, neboť představuje přístup s lepší prostorovou lokalitou. Z grafu je zřejmé, že nejrychlejšího vzorkování je dosaženo s bloky 32^3, 64^3 a 128^3. Velké explicitní bloky kvůli menší granularitě zvyšují režii způsobenou



Obrázek 6.2: Vliv velikosti objemu na průměrnou dobu pro získání jednoho vzorku. Graf srovnává skalární, vektorové a packletové vzorkování surových dat. K tomu srovnává nejlepší dosaženou variantu z předchozího experimentu, což je dělení objemu do bloků 64^3 ve dvou variantách linearizace s packletovým vzorkováním.

zarovnáním. Oproti tomu malé explicitní bloky přináší redundanci z důvodu duplikovaných okrajových voxelů v bloku.

Obrázek 6.2 demonstruje škálování vlivu uspořádání dat na průměrnou dobu pro získání jednoho vzorku. Z grafu je zřejmé, že vzorkování surových dat není škálovatelné na velká či přímo rozsáhlá data. Získání jednoho vzorku v rozsáhlých datech 2048^3 trvá $100\times$ déle než získání jednoho vzorku běžných v datech 512^3. To je způsobeno zejména neefektivním přístupem do paměti. Mírné zlepšení lze sledovat v packletové variantě, která pro největší měřená data zrychlila vzorkování $1.6\times$.

Graf také srovnává vzorkování objemu zpracovaného do blokové reprezentace. Tato reprezentace dělí objem na bloky 64^3 a přidává překryv sousedních voxelů. Z grafu plyne, že pouze díky blokovému uspořádání dat v paměti lze dosáhnout až $12\times$ rychlejšího vzorkování rozsáhlých volumetrických dat $2048^3 \times 16$b. Pro malá data bloková struktura představuje naopak režii navíc. Malé zlepšení přináší také linearizace Mortonovou křivkou.

Tato sekce objektivně vyhodnocuje blokovou datovou strukturu a linearizaci Mortonovou křivkou jako vhodnou pro reprezentaci rozsáhlých volumetrických dat. Toto vyhodnocení je nezávislé na obsahu dat, přenosové funkci, a tím i na optimalizačních technikách. Tato struktura tvoří základ pro datovou hierarchii, která je společně s optimalizačními technikami využita pro redukci celkového množství čtené paměti, a tím i dosažení téměř interaktivní vizualizace.

6.3 Vyhodnocení hierarchické datové struktury

Sada experimentů v této sekci se zabývá vyhodnocením vizualizace hierarchické datové struktury na reálných rozsáhlých volumetrických datech. Ta je vytvořena rekurzivním podvzorkováním volumetrických dat. Každá úroveň hierarchie je zpracována do blokové struktury, která akceleruje vzorkování. Cílem interaktivní vizualizace rozsáhlých volumetrických dat je efektivně volit úroveň hierarchie pro jednotlivé vzorky tak, aby byla maximalizována kvalita vizualizace a zároveň minimalizováno množství paměti, ze kterého je vizualizace provedena, což vede k redukci latence způsobené přístupy do paměti.

Jelikož se hierarchie používá v tandemu s oktalovým stromem, který slouží k adresování bloků a přeskakování prázdného prostoru, jsou v této sekci opět testovány velikosti bloků. Tentokrát však za účelem vyvážit akceleraci vzorkování s průchodem stromu. Dá se očekávat, že mělký strom s velkými bloky bude minimalizovat čas průchodu paprsku stromovou strukturou. Oproti tomu menší bloky umožňují jemnější granularitu pro přeskakování prázdného prostoru.

V této sekci je představen testovací dataset. Dále je popsána metodologie měření a metrika PSNR, která slouží k měření vizuální kvality. Následuje samotné vyhodnocení. U každého objemu je měřena rychlost sestavení hierarchie a velikost datové struktury vzhledem ke zdrojovým datům. Nakonec je srovnána vizualizace navrženou metodou založenou na aktuální průhlednosti paprsku pro různé úrovně kvality s triviálním podvzorkováním.

Dataset

Testovací data byla získána z různých zdrojů pro dosažení maximální různorodosti. Pro data byly vytvořeny přenosové funkce, které mapují barevný gradient na rozsah relevantních hodnot objemu. Přenosové funkce byly navrženy tak, aby vizualizace objemu byla netriviální. Důraz byl kladen na dostatečnou průhlednost přenosových funkcí.

Tabulka 6.2: Testovací dataset.

Název	Rozlišení	Hloubka	Velikost	Zdroj
Backpack	$512 \times 512 \times 373$	16 b	195.6 MiB	Volvis[1]
Stag Beetle	$832 \times 832 \times 494$	16 b	683.9 MiB	Gröller et al. [10]
Kingsnake	$1024 \times 1024 \times 795$	8 b	833.6 MiB	DigiMorph [22]
Hodinky	$959 \times 959 \times 737$	16 b	1.4 GiB	T3D[2]
Chameleon	$1024 \times 1024 \times 1080$	8 b	2.3 GiB	DigiMorph [22]
Scan2	$1999 \times 1999 \times 1680$	16 b	13.4 GiB	T3D[2]
EM30-H-im	$4096 \times 4096 \times 1000$	8 b	16.8 GiB	MitoEM [4]
Woodbranch	$2048 \times 2048 \times 2048$	16 b	17.2 GiB	UZH[3]
Pig Heart	$2048 \times 2048 \times 2612$	16 b	21.9 GiB	CVRTI & SCI[4]

Dataset je popsán v tabulce 6.2. Jednotlivé objemy jsou s jejich navrženými přenosovými funkcemi vizualizovány v obrázku 6.3. Objem Backpack v datasetu reprezentuje malá volu-

[1] Volvis – volvis.org and Kevin Kreeger, Viatronix Inc., USA

[2] T3D – poskytnuto firmou TESCAN 3DIM.

[3] UZH – The Computer-Assisted Paleoanthropology group and the Visualization and MultiMedia Lab at University of Zurich (UZH)

[4] CVRTI & SCI – Cardiovascular Research and Training Institute (CVRTI) and the Scientific Computing and Imaging (SCI) Institute at the University of Utah.

metrická data. Objemy Stag Beetle, Kingsnake, Hodinky a Chameleon představují středně velká až velká data. Z hlediska vizualizace rozsáhlých volumetrických dat jsou relevantní objemy Scan2, EM30-H-im, Woodbranch a Pig Heart, které svou velikostí přesahují kapacitu operační paměti testovacího stroje.

Metrika PSNR

Jelikož je vizualizace založená na volbě úrovně datové hierarchie, je předpokládána ztráta kvality výsledné vizualizace. Pro vyhodnocení vizuální kvality vizualizace bylo srovnání provedeno pomocí špičkového poměru signálu k šumu (PSNR). PSNR lze vypočítat ze střední kvadratické chyby (MSE). Výpočet je popsán rovnicí 6.1.

$$\begin{aligned} \mathrm{MSE} &= \frac{1}{K}\sum_{k=0}^{K-1}(M_k - N_k)^2 \\ \mathrm{PSNR} &= 10 \times \log_{10}\left(\frac{255^2}{\mathrm{MSE}}\right) \end{aligned} \tag{6.1}$$

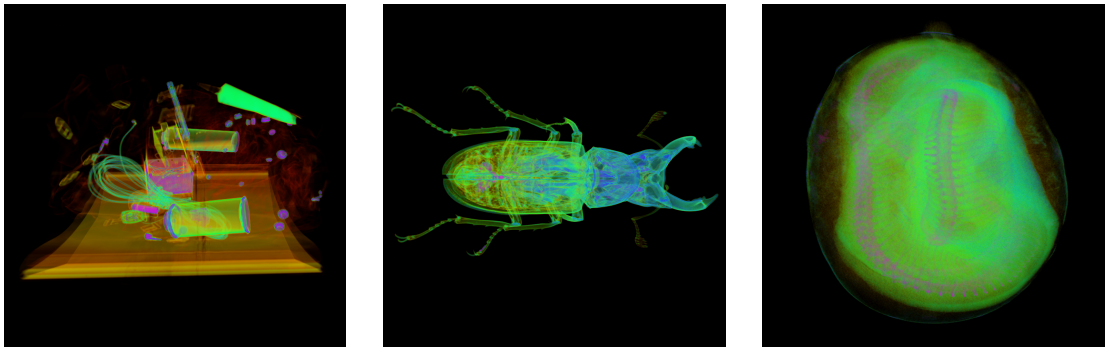
Metrika porovnává dva stejně velké signály M a N, kde každý obsahuje K hodnot. V každém měření je nejdřív provedena vizualizace objemu v nejvyšší možné kvalitě, která se poté využívá jako reference pro vizualizace v nižších úrovních kvality. Střední kvadratická chyba je měřena přímo na hodnotách RGB.

Metodika

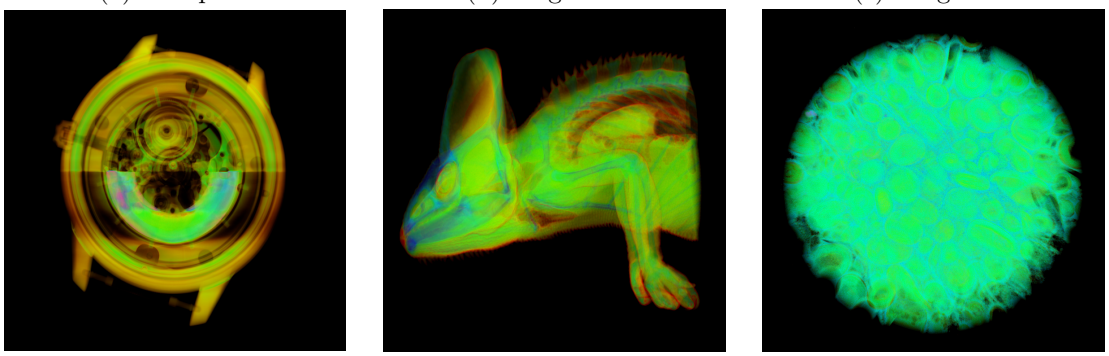
Experimenty pro každý testovaný objem a velikosti bloků z množiny $\{16^3, 32^3, 64^3, 128^3, 256^3\}$ vyhodnocují čas zpracování do hierarchické datové struktury, paměťovou režii struktury a čas vizualizace objemu v plném rozlišení, což odpovídá vizualizaci blokově zpracovaného objemu v nejnižší úrovni hierarchie. Poté je na zpracovaných datech měřen výkon navrženého vizualizéru, který srovnává průměrný čas pro vržení jednoho paprsku a výslednou kvalitu vizualizace. Pro srovnání je stejné vyhodnocení provedeno nad vizualizací po jednotlivých vrstvách datové hierarchie, což odpovídá triviální vizualizaci uniformně podvzorkovaných dat.

Testovaná scéna sestává z objemu a pozorovatele uspořádaného tak, aby objem pokrýval celou vizualizaci. Měřilo se 10 snímků s rozlišením 256×256, které animují rotaci objemu podél své středové osy. To simuluje podmínky interaktivní vizualizace, kde jsou snímky mezi sebou temporálně koherentní. Frekvence vzorkování byla zvolena tak, aby jeden průměrný paprsek reprezentoval průměrně 500 vzorků v objemu. Tato hodnota byla získána empiricky a reprezentuje vizualizaci v maximální kvalitě bez viditelných artefaktů způsobených nízkou vzorkovací frekvencí.

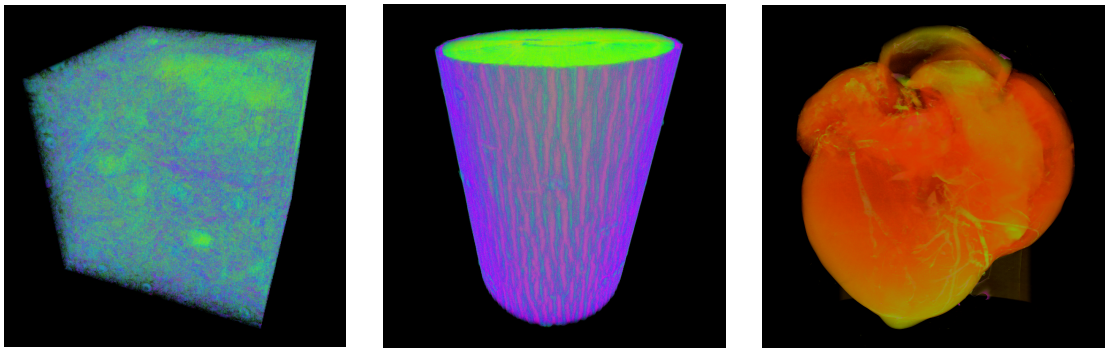
Vyhodnocení bylo provedeno včetně všech v této práci popsaných optimalizačních technik, které redukují průchod oktalovým stromem, a tím i množství čtené paměti. Vyhodnocení bylo provedeno včetně paralelizace na čtyřech fyzických jádrech procesoru se zapnutou technologií Hyper-Threading, kde na každém jádře běží dvě vlákna pro vykrývání latencí paměťových přístupů. Jelikož bylo zjištěno, že průměrný čas pro vržení jednoho paprsku je citlivý nejen na jednotlivé parametry vizualizace, ale i na jejich kombinace, byly tyto experimenty koncipovány tak, aby co nejlépe simulovaly očekávané použití vizualizéru.



(a) Backpack (b) Stag Beetle (c) Kingsnake



(d) Hodinky (e) Chameleon (f) Scan2



(g) EM30-H-im (h) Woodbranch (i) Pig Heart

Obrázek 6.3: Vizualizace testovacího datasetu s navrženými přenosovými funkcemi.

Srovnání s vizualizací v plném rozlišení

Tento experiment demonstruje škálování navrženého vizualizéru vzhledem k velikosti a povaze vizualizovaných volumetrických dat. Experiment srovnává čas bezeztrátové vizualizace objemu v plném rozlišení s časem pro vizualizaci pomocí navržené metody s téměř-bezeztrátovým nastavením kvality. Bezeztrátová vizualizace byla provedena vzorkováním z nejnižší úrovně datové hierarchie. Ztrátová vizualizace volila úroveň dat podle aktuální průhlednosti paprsku. Vyhodnocení bylo v obou variantách provedeno s bloky různých velikostí. Pro každou variantu byla zvolena nejrychlejší velikost bloku.

Výsledky experimentu jsou demonstrovány v obrázku 6.4. Z grafů plyne, že pro data, která se vejdou do operační paměti, představuje datová hierarchie a algoritmus volby úrovně z ní spíše režii navíc. Bloková struktura a oktalový strom pro přeskakování prázdného prostoru, sestavený nad těmito daty, je tedy pro vizualizaci naprosto dostačující. Oproti tomu na rozsáhlých volumetrických datech bylo s navrženým vizualizérem dosaženo až 150× rychlejší vizualizace v téměř-bezeztrátovém režimu. To dokazuje efektivitu hierarchické struktury dat a navrženého vizualizačního algoritmu.

Srovnání s uniformně podvzorkovanými daty

Tento experiment srovnává navržený algoritmus pro volbu pracovní množiny podle aktuální průhlednosti s triviálním algoritmem, který vizualizuje jednu vrstvu datové hierarchie, tedy jednu úroveň podvzorkování dat. Experiment má srovnat výslednou kvalitu a čas ztrátové vizualizace. Měření bylo provedeno na předposlední úrovni datové hierarchie, ve které je rozlišení objemu v každé dimenzi poloviční vzhledem k původnímu objemu. Nad touto vizualizací byla změřena dosažená kvalita a čas. Poté byla s navrženým vizualizérem změřena maximální dosažená kvalita za stejnou nebo nižší dobu.

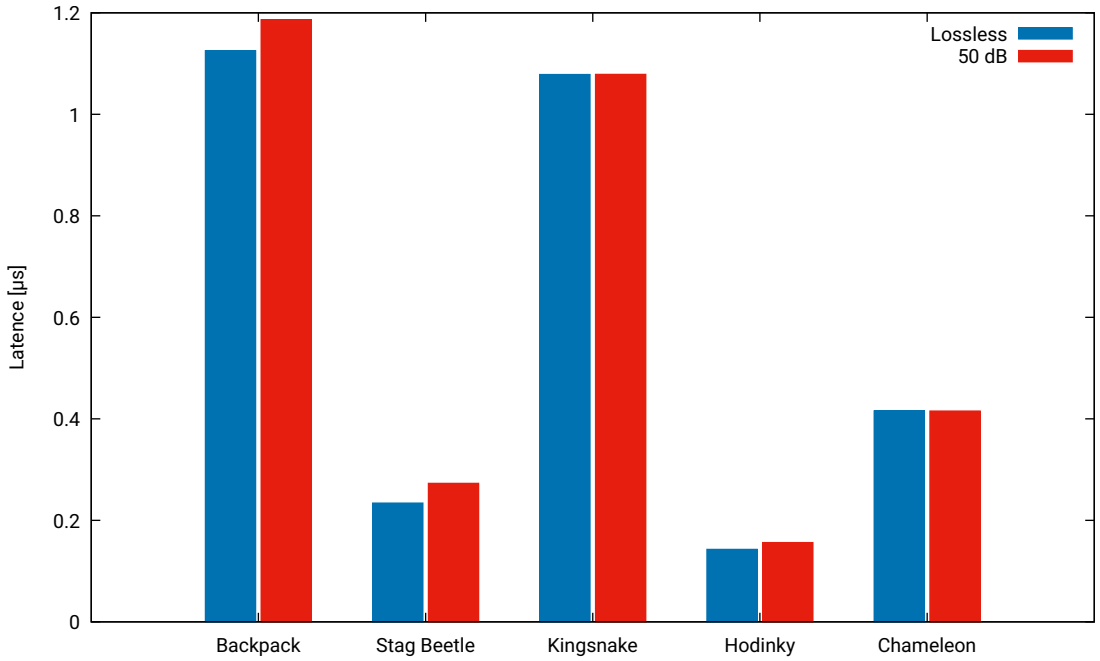
Výsledky experimentu jsou demonstrovány v obrázku 6.5. Z grafů plyne, že na rozsáhlých datech dosahuje navržený vizualizér značně lepší kvality vizualizace za stejný čas ve dvou případech. Nejlepšího rozdílu je dosaženo na objemu EM30-H-im, kde navržený vizualizér dosáhl 52 dB PSNR, což odpovídá téměř bezeztrátové vizualizaci. Vizualizace s podvzorkováním dosáhla kvality 24 dB PSNR, což odpovídá nízké vizuální kvalitě. Vizualizace objemu Pig Heart je v obou případech vizualizována v pásmu téměř bezeztrátové vizualizace, avšak dá se očekávat, že navržený vizualizér bude o něco lépe škálovat na nižší úrovně kvality.

Na objemu Scan2 a Woodbranch oba vizualizéry dosahují za stejný čas srovnatelné hodnoty 38 dB PSNR. Hodnoty odpovídají střední kvalitě vizualizace. Navržený vizualizér má i přesto nespornou výhodu v tom, že dokáže škálovat nad tuto hodnotu až po téměř bezeztrátovou vizualizaci. V kontextu vizualizace podobných dat či přenosových funkcí lze s vizualizérem dále experimentovat a ladit metodu pro volbu zanoření v hierarchii tak, aby bylo dosaženo lepších výsledků.

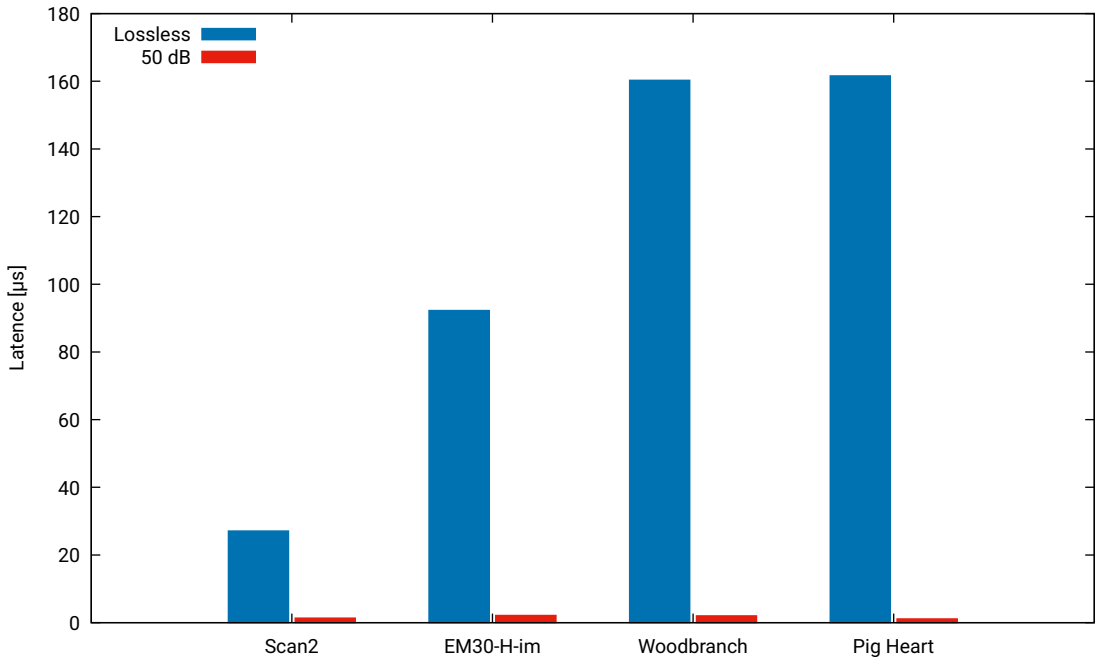
Čas pro sestavení struktury a její velikost

Důležitým aspektem navržené datové struktury je doba sestavení a paměťová režie. Ačkoliv se struktura sestavuje pouze jednou, pro praktické použití je důležité, aby sestavení proběhlo v rozumném čase. Stejně tak je důležité, aby výsledná datová struktura svou velikostí příliš nepřesahovala zdrojová data.

Vyhodnocení paměťové režie je vizualizováno v obrázku 6.6. V grafech jde vidět, že pro malé bloky aproximuje velikost struktury predikovaných 143 %, které jsou způsobeny duplikací sousedních voxelů v blocích. Při takto malých blocích je režie způsobená zarovnáním

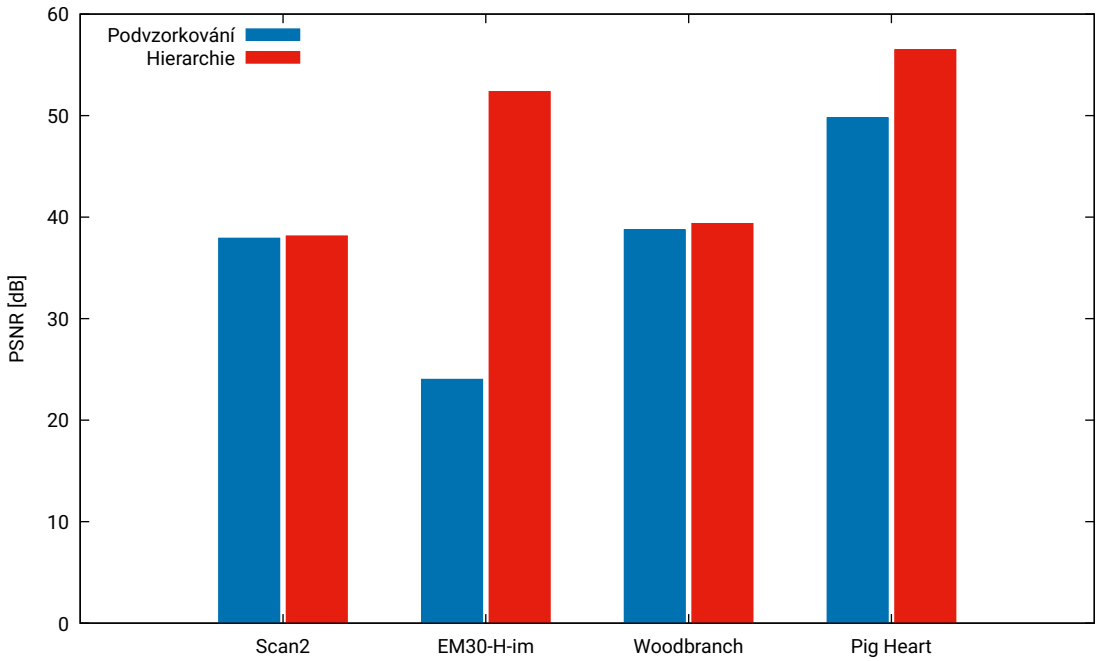


(a) Data, která se vejdou do operační paměti.



(b) Rozsáhlá data.

Obrázek 6.4: Srovnání bezeztrátové vizualizace objemu v plném rozlišení a téměř-bezeztrátové vizualizace navrženým algoritmem na datech, která se vejdou do operační paměti (a) a rozsáhlých datech (b).



Obrázek 6.5: Srovnání kvality vizualizace triviálně podvzorkovaného objemu s vizualizací navrženým algoritmem na rozsáhlých datech.

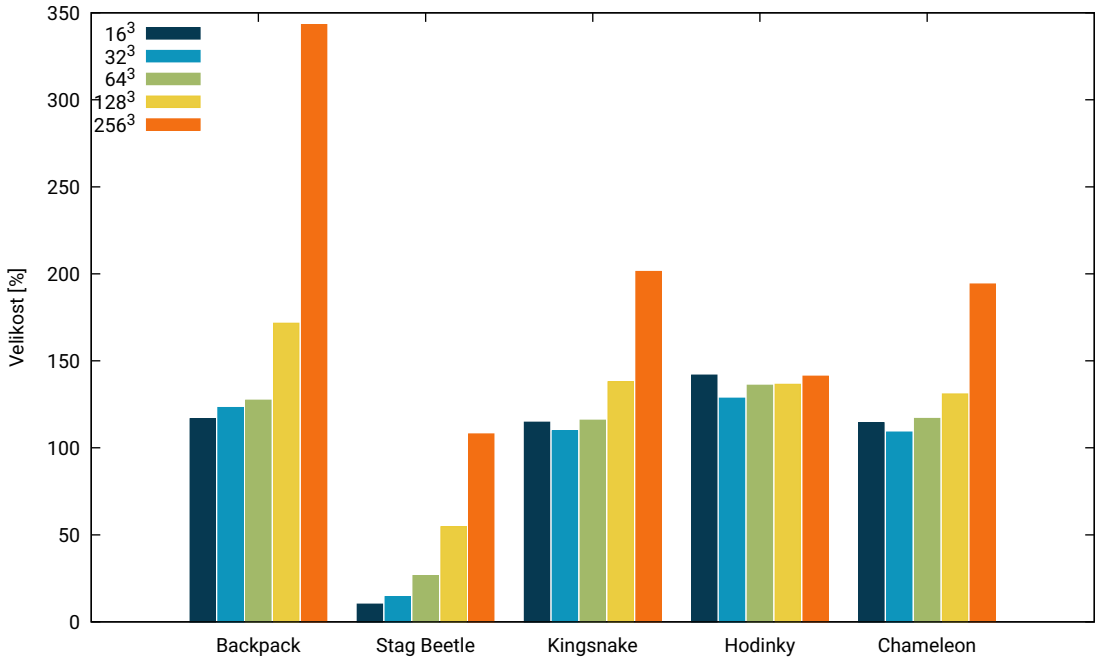
dat na bloky naprosto minimální. Opak platí na druhém konci spektra testovaných velikostí bloku, kde zejména u menších objemu režie z důvodu zarovnání dat na bloky převyšuje velikost původního surového objemu.

S rostoucí velikostí bloku navíc klesá granularita dekompozice. Je tak menší pravděpodobnost, že prostor v bloku bude homogenní a z výsledného datového souboru může být vypuštěn. Tento efekt lze pozorovat na obrázku Stag Beetle, kde právě díky homogenním blokům dochází ke značné kompresi datové struktury až pod úroveň zdrojových surových dat. Tato komprese klesá s rostoucí velikostí bloku.

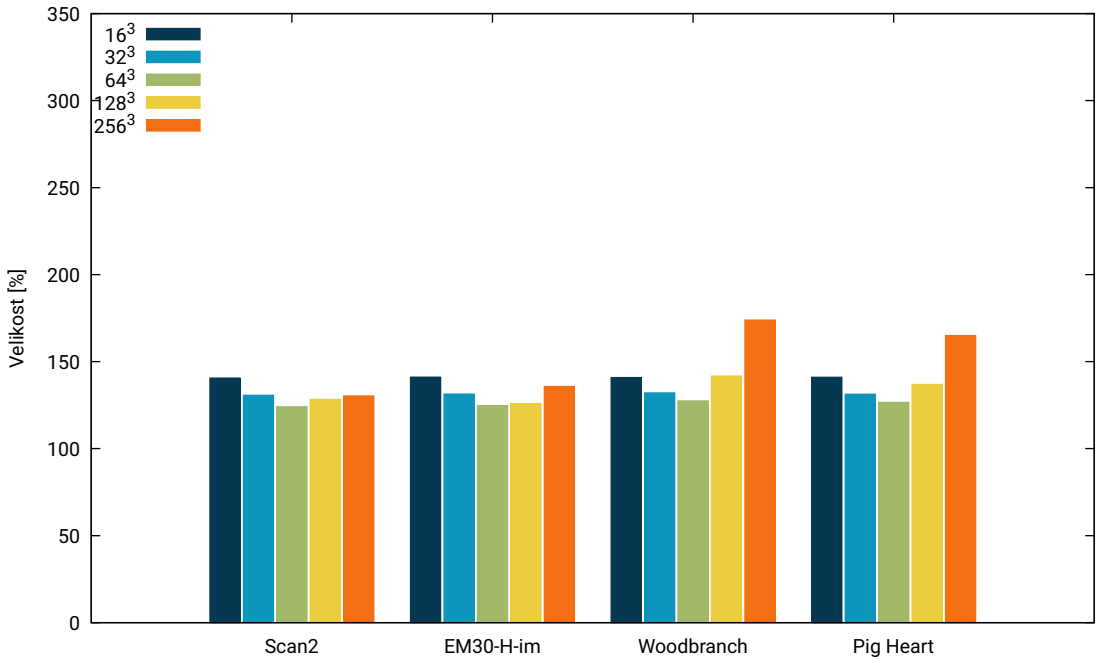
U rozsáhlých dat není režie způsobená zarovnáním objemu na bloky příliš dramatická, v grafech je přesto znatelná. Kvůli režii způsobené zarovnáním jsou pro malá data efektivnější spíše malé bloky 16^3 a 32^3. Pro rozsáhlá data dochází k vyvážení vlivů jednotně při velikosti bloku 64^3.

Vyhodnocení časové režie je vizualizováno v obrázku 6.7. Při srovnání grafů je na první pohled zřejmé, že zpracování dat, která se vejdou do operační paměti probíhá v řádu sekund, kdežto zpracování rozsáhlých dat probíhá v řádu minut. Tento skok je způsoben tím, že při zpracování surových dat do bloků v první iteraci algoritmu není přístup do paměti lokální. Při zpracování rozsáhlého objemu jsou zahazovány stránky, které nejsou kompletně zpracovány do bloků, jsou proto načítány do fyzické paměti více než jednou.

Dá se očekávat, že větší data je nutné déle zpracovávat. Doba zpracování proto pro různé velikosti bloku lehce koreluje s velikostí výsledné datové struktury. Jelikož se při sestavování hierarchie počítají statistiky a již jsou přeskakovány bloky z nižších vrstev, dá se předpokládat, že čas zpracování je závislý přímo na obsahu zpracovávaných dat.

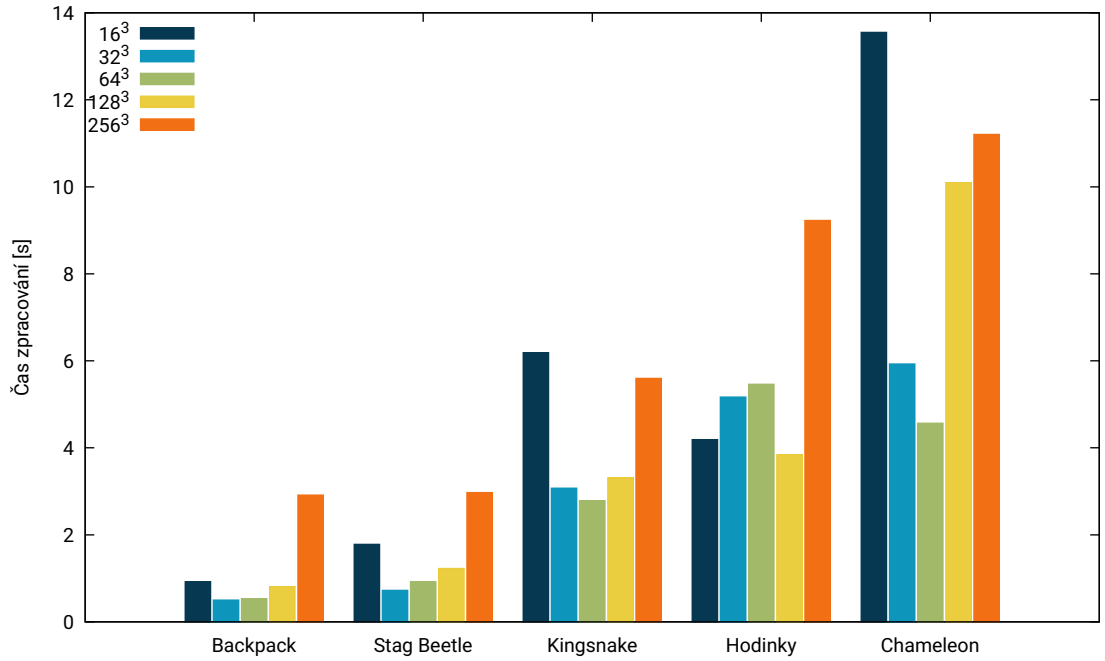


(a) Data, která se vejdou do operační paměti.

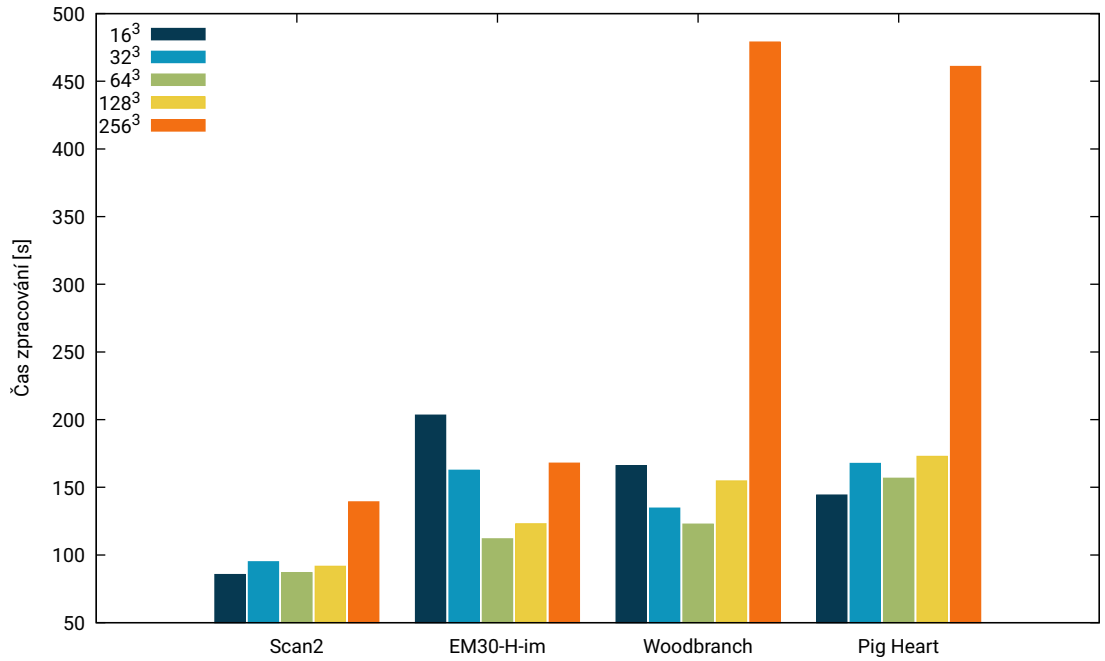


(b) Rozsáhlá data.

Obrázek 6.6: Srovnání velikosti hierarchické datové struktury pro různé velikosti bloků v procentech původních surových dat. Srovnání je provedeno na datech, která se vejdou do operační paměti (a) a na rozsáhlých datech (b).



(a) Data, která se vejdou do operační paměti.



(b) Rozsáhlá data.

Obrázek 6.7: Srovnání doby zpracování hierarchické datové struktury ze surových dat pro různé velikosti bloků na datech, která se vejdou do operační paměti (a) a na rozsáhlých datech (b).

Vyhodnocení velikosti bloku

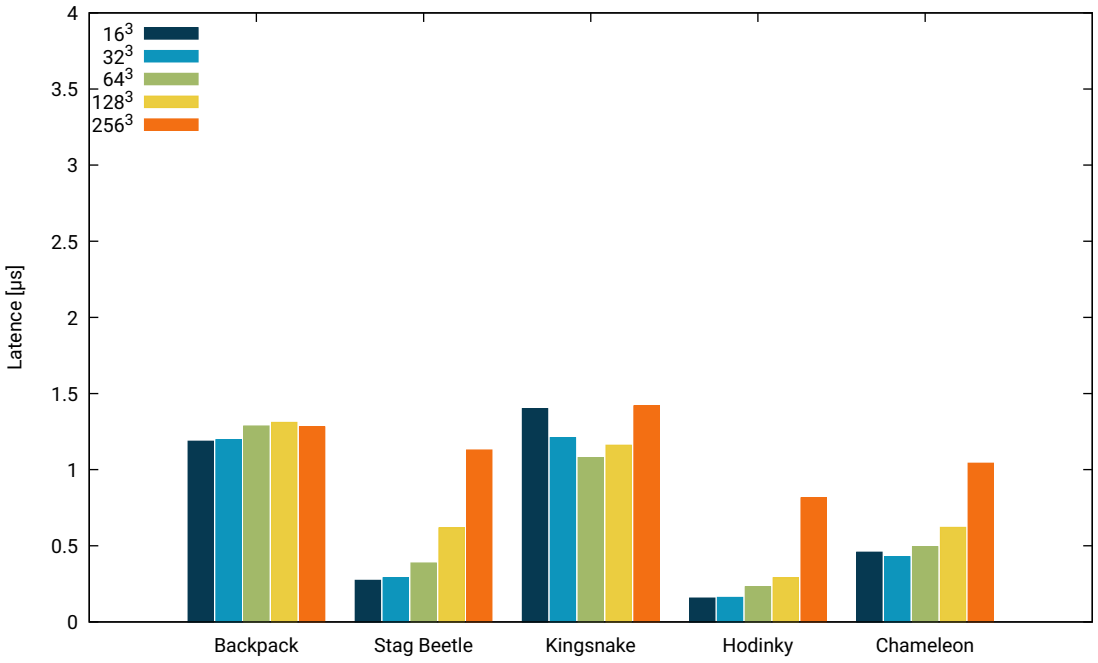
Navržená hierarchická datová struktura využívá bloky k několika účelům. V první řadě bloky slouží k uspořádání dat pro optimalizaci prostorově lokálních paměťových přístupů. V druhé řadě se bloky používají jako uzly oktalového stromu, který se používá k optimalizačním technikám jako přeskakování prázdného prostoru a rychlá integrace homogenního prostoru.

Tento experiment vyhodnocuje vliv velikosti bloku na průměrný čas pro vržení jednoho paprsku objemů z testovacího datasetu. Výsledky měření jsou vizualizovány v obrázku 6.8. Z grafů plyne, že menší data dosahují nejrychlejší vizualizace při menších blocích 16^3 nebo 32^3. U rozsáhlých dat je téměř jednoznačně dosaženo nejlepších výsledků při velikosti bloku 64^3. Výjimkou je objem EM-30-H-im, který nejlepšího výsledku dosahuje s bloky 256^3. Tento výkyv lze zdůvodnit tím, že tento objem neobsahuje žádný prázdný ani homogenní prostor. Optimalizační techniky v tomto případě představují pouze režii navíc a velké bloky ji proto redukují.

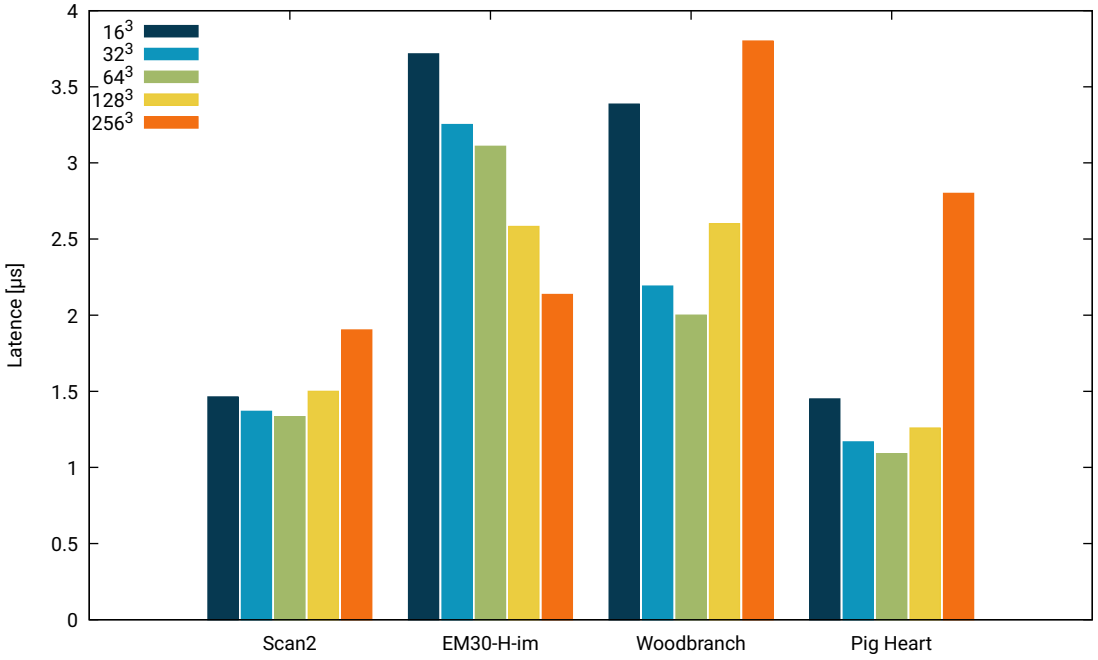
Vyhodnocení vektorizace a packletového průchodu

Vektorizace a packletizace vzorkování se při experimentech s blokovou strukturou ukázala jako efektivní metoda akcelerace. Oproti vzorkování blokové struktury se však při vizualizaci hierarchické struktury provádí průchod oktalovým stromem. Jelikož může každý paprsek dosáhnout jiného zanoření a obecně protíná různé bloky, dá se očekávat, že efektivita vektorizace a packletizace bude při průchodu objemem nižší než při triviálním vzorkování.

Vyhodnocení vektorizace a packletizace je vizualizováno na obrázku 6.9. Z grafů plyne, že pro všechny testované objemy bylo nejlepšího výsledku dosaženo při pouhé vektorizaci. Packletizace průchodu paprsků oktalovým stromem představuje dle naměřených dat spíše režii navíc. To je s největší pravděpodobností způsobeno právě tím, že každý paprsek vzorkuje data z jiné úrovně hierarchie. To znamená, že se v rámci každého paprsku vzorkují různé bloky. Kvůli tomu je při vzorkování velká část packletu vymaskovaná, což snižuje efektivitu vektorizace.

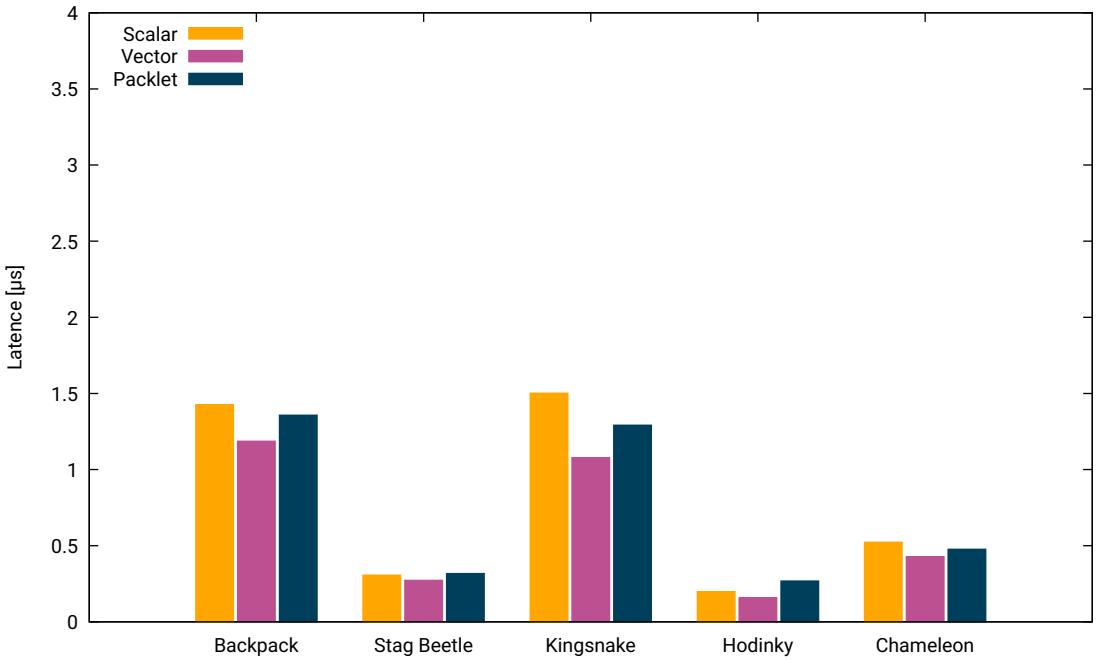


(a) Data, která se vejdou do operační paměti.

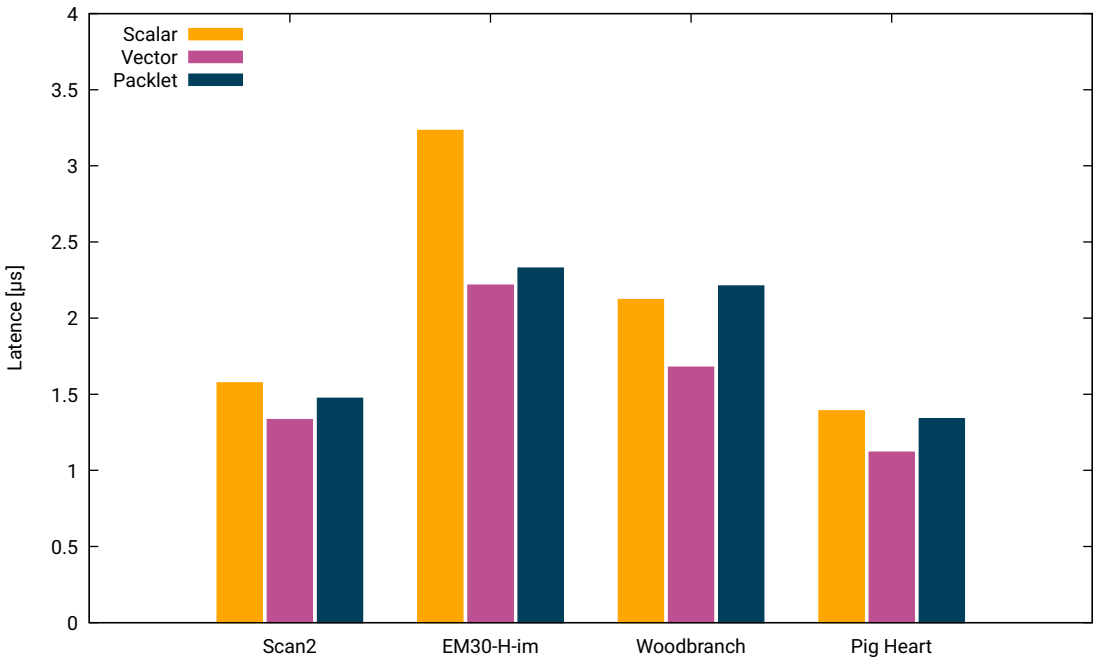


(b) Rozsáhlá data.

Obrázek 6.8: Srovnání průměrného času vržení jednoho paprsku pro různé velikosti bloků na datech, která se vejdou do operační paměti (a) a na rozsáhlých datech (b).



(a) Data, která se vejdou do operační paměti.



(b) Rozsáhlá data.

Obrázek 6.9: Srovnání průměrného času vržení jednoho paprsku pro skalární, vektorový a packletový renderer na datech, která se vejdou do operační paměti (a) a na rozsáhlých datech (b).

Kapitola 7

Závěr

Tato práce si kladla za cíl navrhnout a implementovat schéma pro vizualizaci rozsáhlých volumetrických dat. Toho bylo docíleno návrhem hierarchické datové struktury, která akceleruje vzorkování a umožňuje redukci celkového množství dat, které je při vizualizaci nutné načíst do fyzické paměti. Nad touto strukturou byla navržena a implementována metoda vrhání paprsků včetně existujících optimalizačních technik. Práce popisuje principy přímého zobrazení volumetrických dat. V práci jsou popsány existující přístupy pro práci s rozsáhlými volumetrickými daty a jejich vizualizaci. V návaznosti na existující přístupy bylo navrženo řešení pro efektivní vizualizaci rozsáhlých volumetrických dat. Řešení bylo implementováno formou knihovny v jazyce C++20. Nad touto knihovnou byly implementovány nástroje, které umožňují zpracování a zobrazení rozsáhlých volumetrických dat. Navržené řešení a jeho implementace byla experimentálně vyhodnocena.

V rámci práce bylo experimentováno s uspořádáním dat v paměti pro maximalizaci prostorové lokality při průchodu paprsku objemem. Byla navržena bloková dekompozice a serializace Mortonovou křivkou, díky které bylo dosaženo až 12× rychlejšího vzorkování v porovnání s naivně serializovanými rozsáhlými volumetrickými daty. Nad blokovou strukturou byla sestavena datová hierarchie, díky které bylo dosaženo až 150× rychlejší vizualizace rozsáhlých volumetrických dat v téměř-bezeztrátovém režimu v porovnání s bezeztrátovou vizualizací v plném rozlišení. Práce představuje jednoduchý algoritmus volby úrovně datové hierarchie, který v porovnání s triviálním algoritmem za stejný čas dosahuje stejné nebo lepší kvality vizualizace. Dosažené zlepšení kvality je v případě náročných dat až 28 dB PSNR. Navržená datová struktura je průměrně o 25 % objemnější v porovnání s daty v surovém stavu. V rámci práce bylo experimentováno s vektorizací pomocí SIMD instrukcí, díky které bylo dosaženo až 18% zrychlení.

Během zpracování této práce se autor seznámil nejen s problematikou vizualizace volumetrických dat, ale také obecně s metodami pro práci s rozsáhlými daty, metodami využití paměťové hierarchie, mechanismem mapované paměti a vektorizací pomocí SIMD instrukcí.

Navržená datová struktura je optimalizována pro vzorkování pomocí trilineární interpolace. Na to lze navázat další optimalizací struktury pro vzorkování gradientů, které lze následně využít pro výpočet osvětlovacího modelu. Práce ukazuje, že i jednoduchá metoda volby úrovně hierarchie dosahuje solidních výsledků v porovnání s naivním přístupem. Tuto metodu je však možné dále zdokonalovat a ladit na specifických datech pro docílení lepších vlastností vizualizéru.

Literatura

[1] Beyer, J., Hadwiger, M. a Pfister, H. State-of-the-Art in GPU-Based Large-Scale Volume Visualization. *Computer Graphics Forum*. Květen 2015, sv. 34. DOI: 10.1111/cgf.12605.

[2] Crassin, C., Neyret, F., Lefebvre, S. a Eisemann, E. GigaVoxels : Ray-Guided Streaming for Efficient and Detailed Voxel Rendering. *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*. Únor 2009. DOI: 10.1145/1507149.1507152.

[3] Cullip, T. a Neumann, U. Accelerating Volume Reconstruction With 3D Texture Hardware. In:. 1994.

[4] D. Wei, D. B. et al. MitoEM Dataset: Large-scale 3D Mitochondria Instance Segmentation from EM Images. In: *International Conference on Medical Image Computing and Computer Assisted Intervention*. 2020.

[5] Drebin, R. A., Carpenter, L. a Hanrahan, P. Volume Rendering. In: *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: Association for Computing Machinery, 1988, s. 65–74. SIGGRAPH '88. DOI: 10.1145/54852.378484. ISBN 0897912756. Dostupné z: https://doi.org/10.1145/54852.378484.

[6] Engel, K. CERA-TVR: A framework for interactive high-quality teravoxel volume visualization on standard PCs. In: *2011 IEEE Symposium on Large Data Analysis and Visualization*. 2011, s. 123–124. DOI: 10.1109/LDAV.2011.6092330.

[7] Engel, K., Kraus, M. a Ertl, T. High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*. Červen 2001. DOI: 10.1145/383507.383515.

[8] Fernando, R. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education, 2004. ISBN 0321228324.

[9] Gobbetti, E., Marton, F. a Iglesias Guitián, J. A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer*. Červenec 2008, sv. 24, s. 797–806. DOI: 10.1007/s00371-008-0261-9.

[10] Gröller, M. E., Glaeser, G. a Kastner, J. *Stag beetle*. 2005. Dostupné z: https://www.cg.tuwien.ac.at/research/publications/2005/dataset-stagbeetle/.

[11] Guarda, A. F., Santos, J. M., da Silva Cruz, L. A., Assunção, P. A., Rodrigues, N. M. et al. A method to improve HEVC lossless coding of volumetric

medical images. *Signal Processing: Image Communication*. 2017, sv. 59, s. 96–104. DOI: https://doi.org/10.1016/j.image.2017.02.002. ISSN 0923-5965. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0923596517300176.

[12] Guthe, S., Wand, M., Gonser, J. a Strasser, W. Interactive rendering of large volume data sets. In: *IEEE Visualization, 2002. VIS 2002.* 2002, s. 53–60. DOI: 10.1109/VISUAL.2002.1183757.

[13] Hadwiger, M., Beyer, J., Jeong, W.-K. a Pfister, H. Interactive Volume Exploration of Petascale Microscopy Data Streams Using a Visualization-Driven Virtual Memory Approach. *IEEE Transactions on Visualization and Computer Graphics*. 2012, sv. 18, č. 12, s. 2285–2294. DOI: 10.1109/TVCG.2012.240.

[14] Huang, R. a Ma, K.-L. RGVis: region growing based techniques for volume visualization. In: *11th Pacific Conference onComputer Graphics and Applications, 2003. Proceedings.* 2003, s. 355–363. DOI: 10.1109/PCCGA.2003.1238277.

[15] Kajiya, J. a herzen, B. von. Ray Tracing Volume Densities. *ACM SIGGRAPH Computer Graphics*. Červenec 1984, sv. 18, s. 165–174. DOI: 10.1145/964965.808594.

[16] Knoll, A., Thelen, S., Wald, I., Hansen, C. D., Hagen, H. et al. Full-resolution interactive CPU volume rendering with coherent BVH traversal. *2011 IEEE Pacific Visualization Symposium*. 2011, s. 3–10.

[17] Kohlmann, P., Bruckner, S., Kanitsar, A. a Groller, M. E. Evaluation of a Bricked Volume Layout for a Medical Workstation based on Java. *Journal of WSCG*. jan 2007, sv. 15, 1-3, s. 83–90. ISSN 1213-6972. Dostupné z: https://www.cg.tuwien.ac.at/research/publications/2007/Kohlmann-2007-EBV/.

[18] Kruger, J. a Westermann, R. Acceleration techniques for GPU-based volume rendering. In: *IEEE Visualization, 2003. VIS 2003.* 2003, s. 287–292. DOI: 10.1109/VISUAL.2003.1250384.

[19] Lacroute, P. a Levoy, M. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: Association for Computing Machinery, 1994, s. 451–458. SIGGRAPH '94. DOI: 10.1145/192161.192283. ISBN 0897916670. Dostupné z: https://doi.org/10.1145/192161.192283.

[20] Levoy, M. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*. 1988, sv. 8, č. 3, s. 29–37. DOI: 10.1109/38.511.

[21] Lorensen, W. a Cline, H. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *ACM SIGGRAPH Computer Graphics*. Srpen 1987, sv. 21, s. 163–. DOI: 10.1145/37401.37422.

[22] Maisano, J. High-resolution X-ray CT and the digital library of morphology. *Integrative and comparative biology*. Oxford University Press. 2003, sv. 43, č. 6, s. 930. ISSN 1540-7063.

[23] Reichl, F., Treib, M. a Westermann, R. Visualization of big SPH simulations via compressed octree grids. In: *2013 IEEE International Conference on Big Data*. 2013, s. 71–78. DOI: 10.1109/BigData.2013.6691717.

[24] Röttger, S., Guthe, S., Weiskopf, D., Ertl, T. a Strasser, W. Smart Hardware-Accelerated Volume Rendering. In:. Leden 2003.

[25] Trávníčková, K. *Interaktivní segmentace 3D CT dat s využitím hlubokého učení*. Brno, CZ, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/22315/.

[26] Usher, W., Amstutz, J., Brownlee, C., Knoll, A. a Wald, I. Progressive CPU Volume Rendering with Sample Accumulation. In: Telea, A. a Bennett, J., ed. *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2017. DOI: 10.2312/pgv.20171090. ISBN 978-3-03868-034-5.

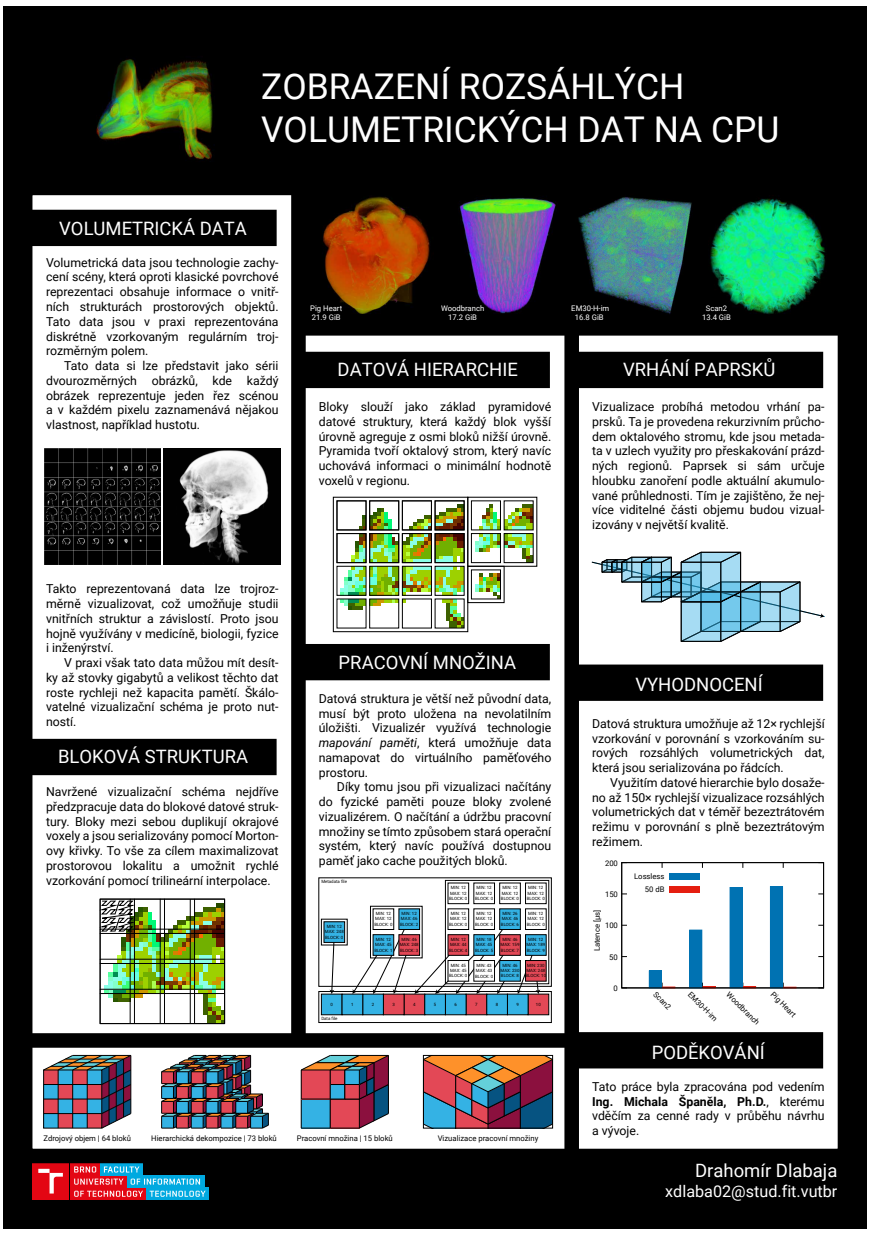
[27] Wald, I., Johnson, G., Amstutz, J., Brownlee, C., Knoll, A. et al. OSPRay - A CPU Ray Tracing Framework for Scientific Visualization. *IEEE Transactions on Visualization and Computer Graphics*. 2017, sv. 23, č. 1, s. 931–940. DOI: 10.1109/TVCG.2016.2599041.

[28] Wang, J., Yang, F. a Cao, Y. A cache-friendly sampling strategy for texture-based volume rendering on GPU. *Visual Informatics*. 2017, sv. 1, č. 2, s. 92–105. DOI: https://doi.org/10.1016/j.visinf.2017.08.001. ISSN 2468-502X. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2468502X1730027X.

[29] Westover, L. A. *SPLATTING: A Parallel, Feed-Forward Volume Rendering Algorithm*. USA, 1991.

Příloha A

Ukázka vytvořeného plakátu



Příloha B

Obsah paměťového média

• `bin/`	Přeložené binární soubory.
• `data/raw/`	Vzorek surových volumetrických dat.
• `data/tf/`	Vzorek přenosových funkcí.
• `doc/`	Zdrojové soubory technické zprávy.
• `src/liblvf/`	Zdrojové soubory knihovny.
• `src/tools/`	Zdrojové soubory nástrojů.
• `src/README.md`	Návod k překladu a zprovoznění.
• `poster.pdf`	Plakát shrnující tuto práci.
• `report.pdf`	Technická zpráva.
• `report-print.pdf`	Tisková verze technické zprávy.