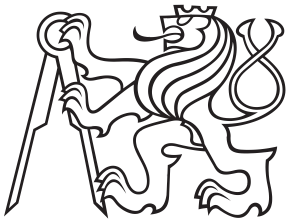
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Realistické metriky 3D scén

Bc. Vladimír Vlček

Vedoucí: Ing. Jakub Hendrich
Obor: Otevřená informatika
Studijní program: Počítačová grafika
Srpen 2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vlček** Jméno: **Vladimír** Osobní číslo: **381836**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačová grafika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Realistické metriky 3D scén

Název diplomové práce anglicky:

Realistic metrics of 3D scenes

Pokyny pro vypracování:

Prostudujte existující realistické metriky pro měření vlastností 3D scény (fatness, density, clutteredness). Vytvořte plugin do systému MeshLab, který umožní uživateli 1) vyhodnocení a vizualizaci metrik na scéně nebo jejím regionu a 2) generování nových scén na základě malých úprav (hodnot metrik) existujících scén. Nástroj otestujte na alespoň pěti scénách různé složitosti. V rendereru Embree změřte a prozkoumejte závislost algoritmů stavby a traverzace BVH daných scén na hodnotách metrik.

Seznam doporučené literatury:

De Berg, Mark, Katz, Matthew J., van der Stappen, A. Frank, and Vleugels, Jules. Realistic Input Models for Geometric Algorithms. Algorithmica, 2002, 34.1: 81-97.
Cazals, Frédéric, and Sbert, Mateu. Some Integral Geometry Tools to Estimate the Complexity of 3D Scenes. Inria, 1997.
Feixas, Miquel, del Acebo, Esteve, Bekaert, Philippe, and Sbert, Mateu. An Information Theory Framework for the Analysis of Scene Complexity. Computer Graphics Forum, 1999, 18.3: 95-106.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Jakub Hendrich, Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **12.02.2020** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2021**

Ing. Jakub Hendrich podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Rád bych poděkoval Ing. Jakubu Hendrichovi za užitečné rady, věcné připomínky a četné konzultace během vypracování diplomové práce. Obzvláště si cením toho, že mi dal příležitost v době, kdy by nade mnou už většina lidí dávno zlomila hůl.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 14. srpna 2020

Abstrakt

Práce se zabývá analýzou složitosti scén ve 3D prostoru pomocí metrik. Hodnoty těchto metrik jsou porovnávány s výkonem algoritmů pro renderování scén. Výstupem práce je aplikace v jazyce C++, která dokáže nad zadanou scénou vypočítat hodnoty metrik (tyto statistiky uloží do prostého textového souboru) a zároveň na základě jejich hodnot vizualizovat scénu v interaktivním prostředí. Uživatel má možnost nastavení několika atributů, které přidávají další funkcionalitu. Také je k dispozici nástroj, který danou scénu inkrementálně mění. Pro podporu hromadného testování různých kombinací metrik, transformovaných scén a způsobu jejich renderování je připraven skript, který možnost takového testování zajišťuje a zároveň kompaktně ukládá výstupní data. Dále umožňuje vytváření grafů podle potřeb uživatele, aby usnadnil vizuální vyhodnocení výsledků. Na závěr je vyhodnocen vztah mezi hodnotami metrik a algoritmy stavby a traverzace hierarchie obalových těles.

Klíčová slova: Výpočetní geometrie; Renderování; Hierarchie obalových těles

Vedoucí: Ing. Jakub Hendrich

Abstract

The thesis deals with analysing the complexity of 3D scenes via metrics. Values of the metrics are compared to the performance of algorithms for rendering scenes. The result of this thesis is an application written in C++ that can calculate the metrics (and save the statistics into a plain text file) and also supports visualising the scene based on the values of the metrics in an interactive environment. The user has the option to select other attributes that extend the functionality. There is also a tool available that can make incremental changes in a scene. To support testing various combinations of metrics, transformed scenes and their methods of rendering, a script has been created, which provides this functionality and also saves the output data compactly. There is also an option to automatically create graphs based on the user's need to make the visual evaluation of the results more comfortable. In the end, an evaluation is made about the relation between values of metrics and algorithms for constructing and traversing bounding volume hierarchies.

Keywords: Computational geometry; Rendering; Bounding volume hierarchy

Title translation: Realistic metrics of 3D scenes

Obsah

Obrázky

Tabulky

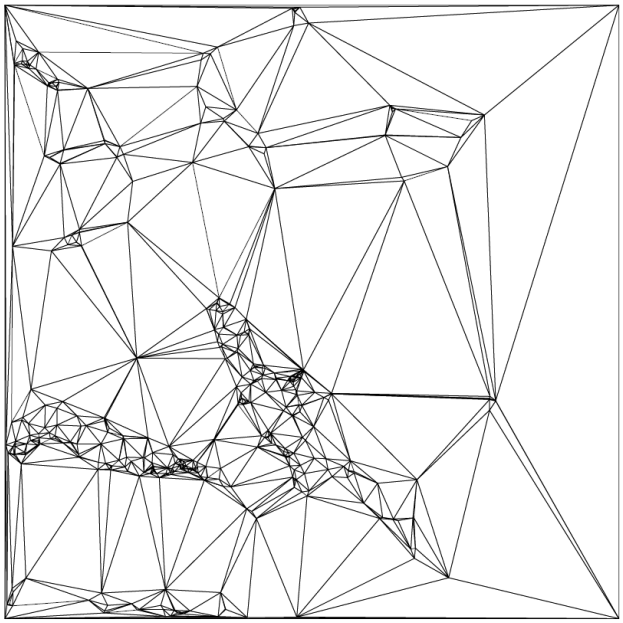
Kapitola 1

Úvod

Stěžejním úkolem počítačové grafiky je práce s komplexními 3D scénami a operacemi nad nimi jako výpočet viditelnosti, detekce kolizí či určení objektů, které se nachází v určitém regionu. Mnoho algoritmů řešící tyto úlohy je ovšem velice obecných a z toho důvodu výpočetně příliš náročných. V potaz se totiž obvykle bere pouze výpočetní složitost pro nejméně příznivé vstupy, které jsou často uměle vytvořené. Avšak tím můžeme zanedbat průměrné, reálné vstupy, pro které algoritmus vykazuje dobré výsledky, a odhadnout jeho efektivitu příliš pesimisticky pro případy, na kterých v praktickém použití opravdu záleží [11].

Pro popis vlastností vstupních dat bylo vytvořeno několik realistických modelů metrik [9]. Každá metrika přiřazuje scéně obvykle jeden parametr. Díky využití těchto modelů můžeme určit charakter scény a zvolit tak vhodný algoritmus pro danou aplikační oblast, ať už se jedná o geografická, architektonická či jiná data. To lze poté experimentálně ověřit. Pro renderování scény, např. pomocí metody sledování paprsků (*ray tracing*) [16], by tak mohla být zvolena vhodná akcelerační struktura a konkrétní způsob její stavby. Tato relace ovšem může být využita i jinak – např. změny geometrie či topologie ve scéně, které ovlivní hodnoty metrik, budou mít zároveň dopad na efektivitu zpracování scény při jejím renderování (ať už kladným či záporným směrem).

Tato práce vychází z modelů popsaných v článku Realistic Input Models for Geometric Algorithms [9], kde je položen základ realistickým metrikám. Autoři se dále zaměřují na případovou studii dat generovaných geografickými informačními systémy. Jedná se o 2D data skládající se z nepravidelné sítě trojúhelníků, scéna tedy obsahuje pouze konvexní a nepřekrývající se objekty. Příklad lze vidět na obrázku 1.1 – jedná se aproximaci terénu reálného prostředí.



Obrázek 1.1: Vizualizace testovacích dat: Údolí smrti, Kalifornie. Převzato z [9].

1.1 Struktura práce

Práce je rozdělena do následujících kapitol. V kapitole 2 se věnuji teorii realistických metrik, zmiňuji příbuzné studie a jejich náhled na danou problematiku. Kapitola 3 se zaměřuje na adaptaci modelů metrik do 3D prostoru společně s konkrétními algoritmy pro jejich výpočet. Součástí práce je také několik nástrojů, zejména pro testování, kterým se věnuji v kapitole 4. Následující kapitola 5 se zaměřuje na vizualizaci scén v závislosti na hodnotách metrik. V kapitole 6 lze najít výsledky testování na různých scénách a vyhodnocení výstupních dat. Poslední kapitola 7 obsahuje závěr práce a možné směry jejího rozvoje v budoucnu.

Kapitola 2

Rešerše

V první části této kapitoly se věnuji teoretickému pozadí jednotlivých metrik, jejich významu, formální definici a vztazích mezi nimi. Definice metrik použité v této práci jsou založeny na modelech, které zavedl de Berg a kol. [9]. Modelů metrik existuje více a mohou mezi nimi existovat pouze malé odchylky či se diametrálně zcela odlišovat. Definice berou v úvahu dimenzi prostoru scény, ale přesně neurčují formu objektů v ní obsažených. Dále je třeba si uvědomit, že metrika má ve výsledku přiřazovat jedinou hodnotu celé scéně. To u některých metrik nemusí platit, jelikož mohou být lokální pro každý objekt (jejich výpočet nebere v potaz ostatní objekty, a tedy je i nezávislý na jejich umístění), a proto každý objekt ve scéně má svoji vlastní hodnotu metriky. Pak je třeba zvolit vhodný způsob, jak z těchto lokálních hodnot získat jediný údaj pro celou scénu, například aritmetickým průměrem či minimální/maximální hodnotou.

V druhé části je zmíněno několik studií, které se zabývají stejnou či příbuznou problematikou. Důraz kladu především na příbuznost k realistickým metrikám (pokud existuje) a využitelnost pro účely této práce.

2.1 Realistické modely

Notace pojmů použitých v této sekci:

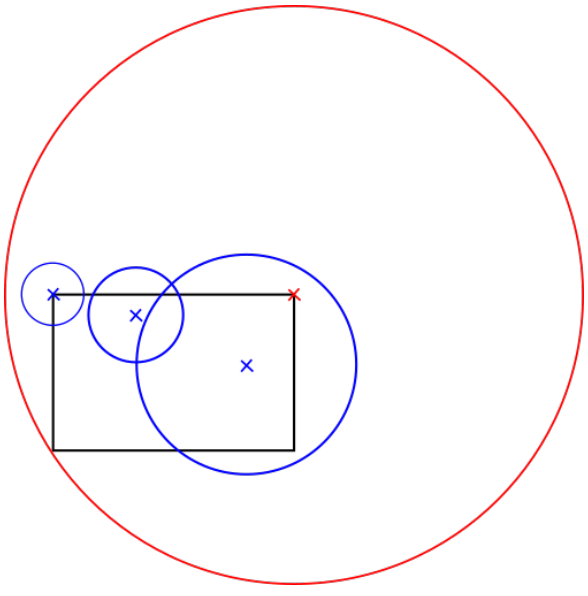
- $\mathbb{E}^d$ – Eukleidovský prostor dimenze d
- $\mathbb{P}$ – objekt v prostoru $\mathbb{E}^d$
- $V(\mathbb{P})$ – objem objektu $\mathbb{P}$
- $\rho(\mathbb{P})$ – minimální poloměr koule, která obsahuje celý objekt $\mathbb{P}$
- $AABB(\mathbb{P})$ – osově zarovnaný kvádr minimální velikosti (dále pouze obálka), který obsahuje celý objekt $\mathbb{P}$

- $\mathbb{S}_i$ – množina objektů $\{\mathbb{P}_1, \ldots, \mathbb{P}_i\}$
- $r(\mathbb{B})$ – poloměr d-rozměrné koule $\mathbb{B}$

První metrikou je tloušťka (*fatness*). Tloušťku objektu lze chápat velice intuitivně – pokud je objekt protáhlý, úzký, pak není tlustý. Naopak nejtlustší konečný objekt ve scéně o třech dimenzích představuje koule.

Definice 2.1. Mějme objekt $\mathbb{P} \subseteq \mathbb{E}^d$ a konstantu $\beta \in \mathbb{R}$, pro kterou platí: $0 \leq \beta \leq 1$. Definujme množinu všech koulí $\mathbb{U}(\mathbb{P})$, jejichž střed leží v objektu $\mathbb{P}$ a jehož hranice zároveň protínají. Objekt $\mathbb{P}$ je β-tlustý, pokud pro všechny koule $\mathbb{B} \in \mathbb{U}(\mathbb{P})$ platí: $\mathrm{V}(\mathbb{P} \cap \mathbb{B}) \geq \beta \cdot V(\mathbb{B})$. Tloušťka objektu $\mathbb{P}$ je definována jako maximální hodnota β, pro kterou je objekt $\mathbb{P}$ β-tlustý.

Ilustrace výběru kruhu s výslednou hodnotou β obdélníku ve 2D lze pozorovat na obrázku 2.1. Pomocí tohoto kruhu je možné dle definice snadno vypočítat hodnotu tloušťky pro tento objekt. Pro zjištění tloušťky objektu nás tedy zajímá pouze jeden kruh, který maximalizuje hodnotu β, ale zároveň splňuje podmínky z definice (tedy jeho střed leží v objektu, jehož hranice zároveň protíná). V praxi to znamená, že hledání tohoto kruhu, popř. koule ve 3D, záleží na tvaru objektu. U obdélníků lze pozorovat, že tento kruh leží na jeho hranici, konkrétně na jednom ze čtyřech vrcholů, a jeho poloměr je rovný úhlopříčce obdélníku.

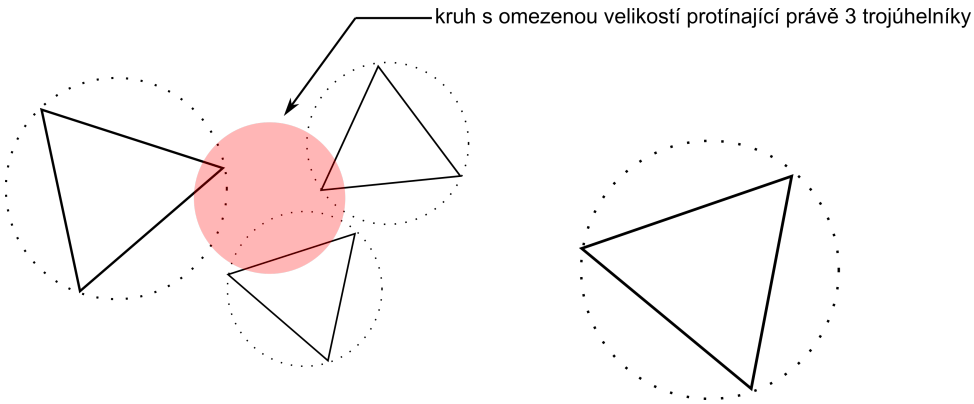


Obrázek 2.1: Ilustrace výběru kruhu pro výpočet tloušťky ve 2D: testovaný objekt – obdélník (černě), několik kruhů z množiny $\mathbb{U}(\mathbb{P})$ (modře), kruh s určující hodnotou β (červeně).

Další metrikou je hustota (*density*). Ta říká, že kterákoliv imaginární koule $\mathbb{B}$ ve scéně protíná pouze omezené množství objektů srovnatelné velikosti. Pokud počet protnutých objektů je malé číslo, pak lze tuto scénu označit jako scénu s malou hustotou. Lze ji tedy vidět jako prostor, ve kterém jsou objekty od sebe dostatečně vzdáleny vzhledem k jejich velikostem, resp. $\rho(\mathbb{P})$.

Definice 2.2. Mějme d-dimenzionální scénu $\mathbb{S} := \{\mathbb{P}_1, \ldots, \mathbb{P}_n\}$ a parametr $\lambda \in \mathbb{N}$, pro který platí: $\lambda \geq 1$. Scéna $\mathbb{S}$ má hustotu λ, pokud pro každou kouli $\mathbb{B}$ platí, že počet objektů $\mathbb{P}_i \in \mathbb{S}$ s $\rho(\mathbb{P}_i) \geq r(\mathbb{B})$, které protíná, je nanejvýše λ. Hustota scény $\mathbb{S}$ je definována jako nejmenší možná hodnota λ.

Scéna ve 2D prostoru skládající se ze čtyřech trojúhelníků je ilustrována na obrázku 2.2. Červený kruh znázorňuje jeden z případů, kdy počet protínajících objektů je právě tři. Tomuto číslu je rovna i hustota celé scény, jelikož neexistuje žádný kruh, který by protínal objektů více. Nesmíme zapomenout, že maximální poloměr tohoto kruhu je omezen: $r(\mathbb{B}) \leq \rho(\mathbb{P}_i)$.

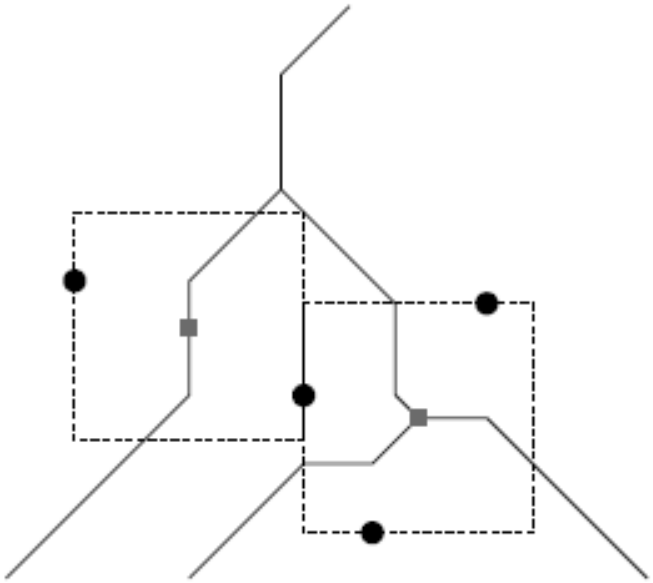


Obrázek 2.2: Ilustrace výběru kruhu pro výpočet hustoty ve 2D se čtyřmi trojúhelníky. Kruhy s poloměrem $\rho(\mathbb{P}_i)$ jsou u jednotlivých trojúhelníků znázorněny tečkovaně.

Třetí metriku lze nazvat nepřehlednost (*clutteredness*). Podobně jako hustota, tak i nepřehlednost scény je závislá na poloze objektů v ní obsažených. Její motivace je však odlišná – snaží se identifikovat, zda ve scéně existuje změť objektů. Tu si v tomto kontextu lze představit jako skupinu blízkých objektů, jejichž obálky jsou si velikostně podobné.

Definice 2.3. Mějme d-dimenzionální scénu $\mathbb{S}$ a parametr $\kappa \in \mathbb{N}$, pro který platí: $\kappa \geq 1$. Scéna $\mathbb{S}$ má nepřehlednost κ, pokud každá hyperkrychle, jejíž vnitřní prostor neobsahuje žádný vrchol z obálek všech objektů scény $\mathbb{S}$, protíná maximálně κ objektů. Celková nepřehlednost scény je minimální hodnota κ.

S ohledem na definici se snažíme najít takové krychle, které neobsahují žádný vrchol z obálek všech objektů scény (dále budu takovéto krychle nazývat prázdné krychle) a zároveň protínají maximální počet objektů. K tomu nám poslouží $\mathbb{L}_\infty$ Voroného diagram, který dekomponuje prostor na oblasti, které jsou nejblíže k diskrétní množině bodů. Pokud vytvoříme Voroného diagram nad vrcholy všech obálek, získáme umístění středů prázdných krychlí. Ty se nacházejí na vrcholech či hranicích tohoto diagramu. Ukázku lze vidět na obrázku 2.3.



Obrázek 2.3: Ilustrace prázdných krychlí pro výpočet nepřehlednosti pomocí $\mathbb{L}_\infty$ Voroného diagramu. Převzato z [9].

Poslední metrikou je tzv. prosté pokrytí (*simple-cover complexity*). Formální definice tohoto modelu zde není uvedena, jelikož pro účel této práce není jednoduše aplikovatelný. Pro úplnost alespoň uvedu základní koncept. Motivací metriky je určit co nejmenší počet koulí, jejichž sjednocení obsahuje obálku scény $\mathbb{S}$ a zároveň každé koule protíná maximálně δ objektů. Metrika má tedy ve výsledku dva parametry, které jsou navzájem svázány. I když by to vzhledem k účelu práce mohla být užitečná a zajímavá metrika, rozhodl jsem se ji vynechat ze dvou důvodů. Prvním je její obrovská časová náročnost. Původní autoři této metriky [9] uvádějí, že nejlepší algoritmus pro její výpočet běží v exponenciálním čase ve 2D prostoru. To je pro větší scény ve 3D, kde složitost ještě naroste, naprosto nepřijatelné. Druhým problémem je výběr parametru δ. Výsledná hodnota v tomto případě totiž není absolutní, ale

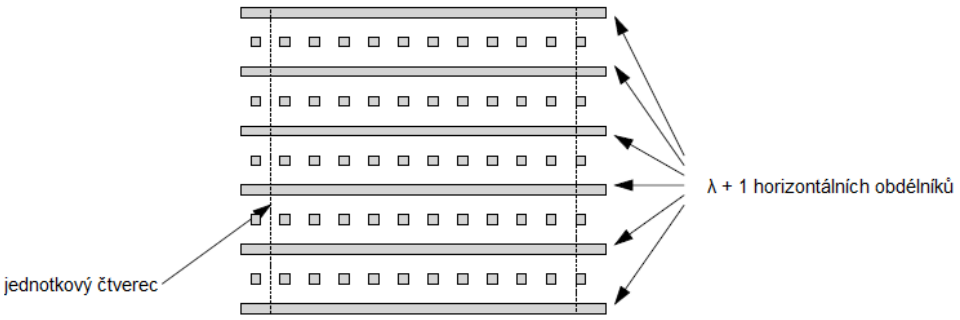
pouze relativní k hodnotě δ.

2.1.1 Vztahy mezi metrikami

Na závěr je třeba také zmínit vztahy mezi jednotlivými modely metrik. Ty jsou důležité nejen k lepšímu pochopení samotných metrik, ale i významu jejich vlastností. Relace mezi modely je následující: velká tloušťka implikuje malou hustotu, které implikuje malou nepřehlednost. Podrobný důkaz těchto vztahů poskytuje původní studie [9].

Je třeba si uvědomit, že vztahy druhým směrem nefungují – například malá hustota neimplikuje velkou tloušťku. Uveďme si názorný příklad. Ve scéně ve vyskytují objekty, které jsou od sebe velice vzdáleny (hustota taková scény je tedy velice malá). Tato vzdálenost je natolik veliká, že i když transformujeme jednotlivé objekty tak, že měníme jejich tloušťku (avšak poloha a celková velikost zůstává stejná), hustota scény se nemění. Neexistuje tedy v tomto případě žádný vztah mezi malou hustotou scény a tloušťkou objektů v ní obsažených.

Na dalším příkladě, tentokrát i s vizualizací (obrázek 2.4), si ukažme, že ani malá nepřehlednost neimplikuje malou hustotu. V této ukázkové scéně se nachází $\lambda + 1$ úzkých horizontálních obdélníků, které mezi sebou mají stejnou vzdálenost a jejíž hodnota je zhruba $1/(\lambda + 1)$. Tato scéna nemá malou hustotu, ale ani její nepřehlednost není 1. Pokud ovšem mezi každou sousední dvojici těchto obdélníků vložíme řadu malých čtverců, nepřehlednost má nyní hodnotu 1. Horizontální vzdálenost mezi dvěma po sobě jdoucími čtverci musí být ovšem větší než horizontální vzdálenost mezi dvěma po sobě jdoucími obdélníky.



Obrázek 2.4: Scéna s hodnotu nepřehlednosti 1, která ovšem nemá malou hustotu. Převzato z [9].

2.2 Jiné přístupy

V této sekci zmiňuji několik studií zaměřených na související problematiku, které se zabývají analýzou složitosti scény. Zajímat mě budou nejen navržené postupy či vyvozené závěry, ale i motivace a cíle, jelikož ty hrají významnou roli pro orientaci a vymezení výzkumu.

Jako první bych chtěl zmínit studii F. Cazalse a M. Sberta [11]. Ti si také uvědomují problém složitého výpočtu operací nad 3D scénami a výsledků při nich použitých algoritmů, u kterých se často opomíjí jejich výkon pro průměrné případy. Těžištěm jejich práce je analyzovat scénu pomocí geometrických a statistických nástrojů, díky nimž lze charakterizovat vlastnosti typické pro určité druhy scén, jako například scény přírody či architektury.

Jako nástroj využívají náhodně generované přímky, roviny a tělesa a jejich interakce se scénou. Velice dobrou shodu mezi teorií a praxí pak názorně ukazuje jejich případová studie. Zajímavá je z mého pohledu statistika spojená s tloušťkou objektu, která může být srovnávána s modelem metriky tloušťky, jež se využívá v této práci. Tu autoři definují pomocí množiny přímek jako délku projekce objektu na tyto přímky.

Další relevantní studie pochází od M. Feixase a kol. [13], jejichž výstup je v rámci teorie informace zavést metriku pro míru složitosti scény, pro kterou je potřeba přesně vyhodnotit viditelnost objektů a metodu globálního osvětlení, radiozitu. Veličiny, které zde navrhují, mohou být interpretovány jako korelace nebo závislost mezi všemi body či ploškami ve scéně.

Kapitola 3

Metriky ve 3D

V této kapitole jsou modely metrik aplikovány na 3D prostor a definovány objekty ve scéně, čímž vzniknou konkrétní algoritmy pro výpočet jednotlivých metrik. Je třeba také brát v úvahu, pro jaký účel jsou metriky počítány – snaha této práce je najít závislost algoritmů stavby a traverzace BVH (*bounding volume hierarchy*) [1] na výsledných hodnotách metrik. Tu lze ověřit rozsáhlým testováním. Pokud však závislost nebude existovat, bude stát za zamyšlení, zde nám alespoň výsledky nenapoví, jak stávající metriky modifikovat, aby vznikl těsnější vztah.

Při výpočtu metrik mě zajímalo několik kritérií. V počítačové grafice se pracuje s velkým množstvím dat, proto je asymptotická časová složitost algoritmů významným faktorem. Náročnost většiny geometrických výpočtů roste exponenciálně s počtem dimenzí, a proto může být vhodné použít aproximaci, která výpočet zjednoduší, ale pouze v takové míře, aby příliš neznehodnotila výslednou hodnotu. V poslední řadě je potřeba zohlednit složitost implementace, aby nepřesahovala rámec rozsahu této práce.

Adaptace metrik pro 3D scény je klíčovou částí. Narazil jsem při ní na několik nepříjemných problémů, kterým jsem musel čelit. S vyšší dimenzí roste nejen výpočetní náročnost algoritmů, ale i komplexita datových struktur.

V této kapitole používám pojmy (s dimenzí 3), jejichž notaci lze najít v sekci 2.1.

3.1 Definice objektu ve scéně

Nejdříve je zapotřebí se rozhodnout, jak budou definovány objekty pro výpočet ve 3D scéně. Jako první možnost se nabízí vzít jednotlivé celky, např. židli či dům. Tyto objekty mohou být definovány při generaci samotných scén, což podporuje například soubor ve formátu OBJ (*Wavefront*) ve formě izolovaných sítí. V tomto případě by ovšem musel uživatel zaručit, že vstupní scény

budou splňovat tuto podmínku. Další možnost je použít algoritmy, které dokáží objekty ve scéně identifikovat, když nejsou předem specifikovány. Jako vhodnější variantu pro danou problematiku jsem ovšem zvolil jinou možnost. Tou je chápat objekt jako základní grafický prvek, což je pro triangulované 3D scény trojúhelník. Hlavní důvod, proč jsem se takto rozhodl, je fakt, že i samotný renderer pracuje se základními primitivy a o sémantice scény nemusí mít žádné údaje.

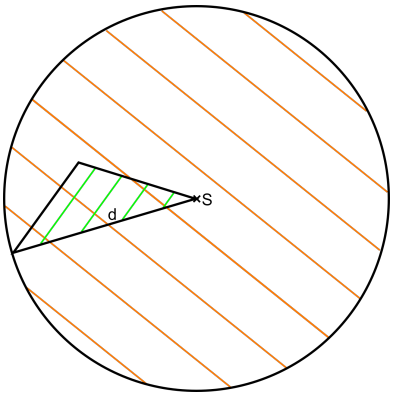
3.2 Tloušťka

Z teoretické definice tloušťky 2.1 uvedené v předchozí kapitole vyplývá nutnost hledání koule, které maximalizuje hodnotu β. Pro trojúhelník ve 3D prostoru je situace velice podobná jako pro obdélník ve 2D, který je jako příklad uveden také v přechozí kapitole. A jelikož všechny tři vrcholy trojúhelníku leží v jedné rovině, je výpočet ekvivalentní ve 2D a 3D prostoru (a stejně tak je kruh analogie koule). Hledaný nejnepříznivější kruh (tedy s nejmenším poměrem obsahu části objektu vyťaté kruhem ku obsahu kruhu) má poloměr délky nejdelší hrany trojúhelníku a jeho střed na jednom z vrcholů této hrany.

Ve výsledku tedy hodnotu tloušťky β objektu $\mathbb{P}$ vypočítáme pomocí následujícího vzorce:

$$\beta(\mathbb{P}) = \frac{S(\mathbb{P})}{S(\mathbb{B})} = \frac{S(\mathbb{P})}{\pi \cdot d^2} \tag{3.1}$$

Jelikož objekt $\mathbb{P}$ je trojúhelník, pak $S(\mathbb{P})$ je obsah tohoto trojúhelníku a d je poloměr hledaného kruhu $\mathbb{B}$ (což je hodnota délky nejdelší hrany tohoto trojúhelníka). Názornou vizualizaci lze pozorovat na obrázku 3.1.



Obrázek 3.1: Ilustrace metriky tloušťky pro obecný trojúhelník jako poměr obsahu zeleně a oranžově vyšrafované části.

Tloušťka nabývá maximální hodnoty pro rovnostranný trojúhelník a naopak konverguje k nule pro tupoúhlé trojúhelníky, jejichž největší vnitřní úhel se blíží 180°. Po dosazení do vzorce 3.1 lze získat hodnotu tloušťky pro několik reprezentativních druhů trojúhelníka:

- Rovnostranný (maximálně tlustý objekt): $\beta = \frac{\sqrt{3}}{4\pi} \approx 13,79 \cdot 10^{-2}$
- Rovnoramenný pravoúhlý: $\beta = \frac{1}{4\pi} \approx 7,96 \cdot 10^{-2}$
- Tupoúhlý s vnitřním úhlem blížící se limitně k 0°: $\beta \to 0$

Algoritmus pro výpočet hustoty se nakonec ukázal jako jeden jednoduchý vzorec. Jeho asymptotická časová složitost je lineární, tedy $\mathcal{O}(n)$, kde n je počet trojúhelníků ve scéně.

3.3 Hustota

Algoritmus pro výpočet hustoty ve 3D prostoru je výrazně složitější než pro výpočet tloušťky – nyní totiž výpočet pro každý objekt ovlivňují jiné objekty nacházející se v jeho bezprostředním okolí.

Podle definice 2.2 v předchozí kapitole se hledá koule, která protíná maximální počet objektů. Víme však, že tato koule nemůže mít větší průměr než průměr jakéhokoliv objektu, který protíná. Pokud je tedy její střed od objektu vzdálen méně než její průměr, musí ho nutně protínat. Lze vytvořit Minkowského sumu koule a objektu, tedy trojúhelníku, která vytyčí prostor, kde se může nacházet střed koule, která tento objekt protíná. Ve 3D prostoru je objekt takovéto sumy velice těžko reprezentovatelný, a ještě hůře by se s ním pracovalo (např. při určování jejich společného průniku). Přistoupil jsem tedy k následujícímu nahrazení. Trojúhelník reprezentuje jeho obálka. Minkowského suma s koulí je aproximována rozšířením této obálky o hodnotu průměru koule ve všech dimenzích, jak na minimální, tak maximální hranici (takovouto obálku nazývám rozšířená obálka).

1 seřaď sestupně všechny objekty ve scéně podle hodnoty jejich ρ
2 $\lambda = 1$
3 **for** $i = 2$ **to** n **do**
4 najdi množinu objektů S z množiny $\mathbb{S}_{i-1}$, jejíž obálky mají společný průnik s obálkou objektu $\mathbb{P}_i$ rozšířenou o hodnotu $2\rho_i$
5 pro každý objekt z množiny S vytvoř novou obálku jejím rozšířením o hodnotu ρ_i, ale zároveň musí ležet uvnitř rozšířené obálky $\mathbb{P}_i$
6 **if** *pokud existuje neprázdný průnik počtu* λ *těchto rozšířených obálek* **then**
7 $\lambda = \lambda + 1$
8 **end**
9 **end**
10 **return** λ

Algoritmus 1: Výpočet hodnoty hustoty.

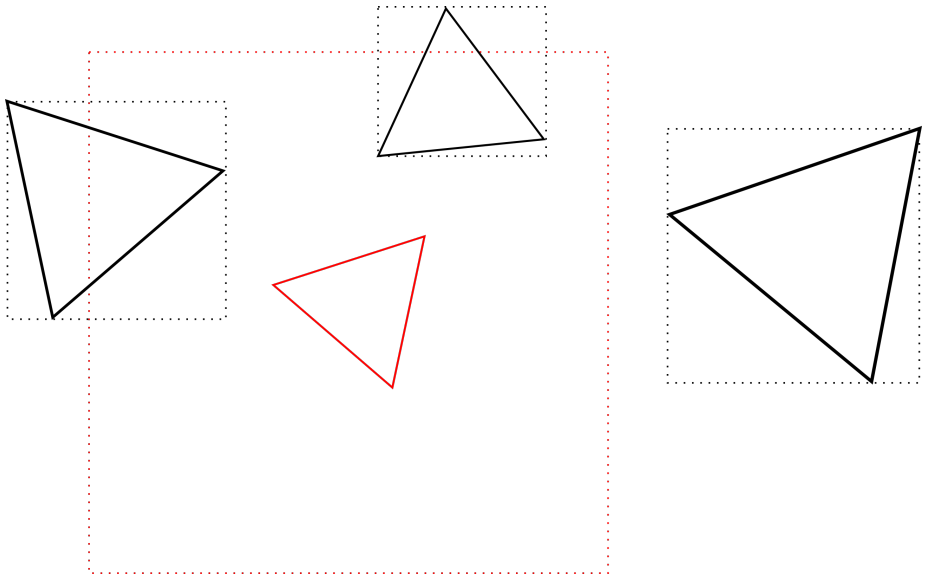
Aby byl algoritmus pro výpočet hustoty efektivní, pracuje na iterativní bázi. Nejdříve je třeba seřadit objekty sestupně podle hodnoty jejich ρ. V každé iteraci i se bere v potaz objekt $\mathbb{P}_i$ poprvé. Jelikož pro zvýšení současné hustoty musí být tento objekt protnut koulí $\mathbb{B}$, jejíž poloměr je ρ_i, můžeme brát v úvahu pouze objekty v tomto okolí. Poté je třeba z této množiny najít maximální počet obálek, které mají společný průnik. Podrobný pseudokód 1 je popsán výše.

Pro dva nejdůležitější kroky algoritmu jsem připravil ukázkovou scénu spolu s vizualizací. Na obrázku 3.2 je červeně vyobrazen právě zpracovávaný trojúhelník $\mathbb{P}_i$ s jeho rozšířenou obálkou o hodnotu $2\rho_i$. Tato rozšířená obálka protíná obálku dvou trojúhelníků ve scéně, které proto tvoří množinu S. Ta je vyobrazena na obrázku 3.3 červeně společně s jejich obálkami, které jsou rozšířeny o ρ_i. Zeleně je zde vyšrafován prostor, kde tato množina obálek vytváří maximální průnik o hodnotě dva. Pokud má nyní λ tuto hodnotu, algoritmus ji inkrementuje o 1.

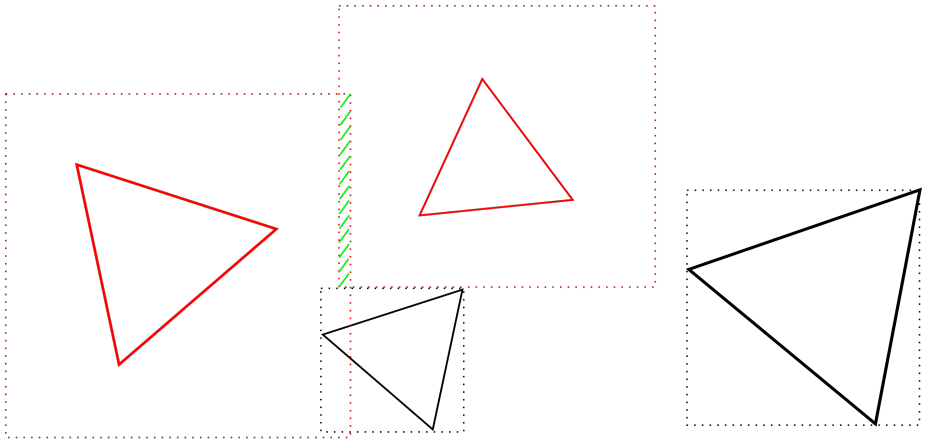
V následujících podsekcích podrobněji popisuji problémy, které přinášejí jednotlivé kroky pseudokódu 1 a efektivní způsoby, jak jsem se s nimi vypořádal.

3.3.1 Akcelerační struktura

V kroku 4 algoritmu 1 je třeba najít všechny objekty, jejichž obálku protíná rozšířená obálka objektu $\mathbb{P}_i$. Při použití naivního algoritmu (otestuj průnik se všemi ostatními obálkami) je asymptotická časová složitost lineární ($\mathcal{O}(n)$)



Obrázek 3.2: Ilustrace kroku 4 algoritmu pro výpočet hustoty: právě zpracovávaný trojúhelník $\mathbb{P}_i$ (červeně) s jeho rozšířenou obálkou.



Obrázek 3.3: Ilustrace kroku 6 algoritmu pro výpočet hustoty: množina trojúhelníků $\mathbb{P}$ (červeně) s rozšířenými obálkami a jimi vytyčený prostor s maximálním množstvím průniků (zeleně).

vzhledem k počtu objektů ve scéně. To je poměrně náročná operace, zejména pokud vezmeme v potaz, že se musí provést pro každý krok vnějšího cyklu – celková složitost algoritmu by tedy byla kvadratická ($\mathcal{O}(n^2)$). Složitost této operace lze vylepšit na logaritmickou ($\mathcal{O}(\log n)$) pomocí akceleračních datových struktur. Jedná se totiž o operaci, která je pro tyto struktury typická – vyhledávání objektů v určitém rozsahu (*range searching*).

Jako první mě napadlo využít kd-strom [4]. Datová struktura, která pod-

poruje vyhledávání v rozsahu, ovšem pouze pro bodová data. Museli bychom tedy naše obálky redukovat na body (např. reprezentovat obálku jejím středem). Takto vzniklý strom by ovšem poskytoval pouze aproximované výsledky. Použitelný pro některé druhy aplikací, ne ovšem v tomto případě.

Lepší varianta se ukázala datová struktura R-strom [15]. Ten se používá pro prostorová data, a proto je pro obálky ideální. Nevýhodou akceleračních struktur je jejich náročná konstrukce – nejsou tedy ideální pro data, která se v průběhu mění, jelikož aktualizace celé struktury je často relativně náročná. To ovšem není překážka pro použití v tomto algoritmu, jelikož postačí tuto strukturu vytvořit pouze jednou, někdy před spuštěním hlavního cyklu, a poté v ní pouze vyhledávat.

3.3.2 Hledání maximálního průniku množiny obálek

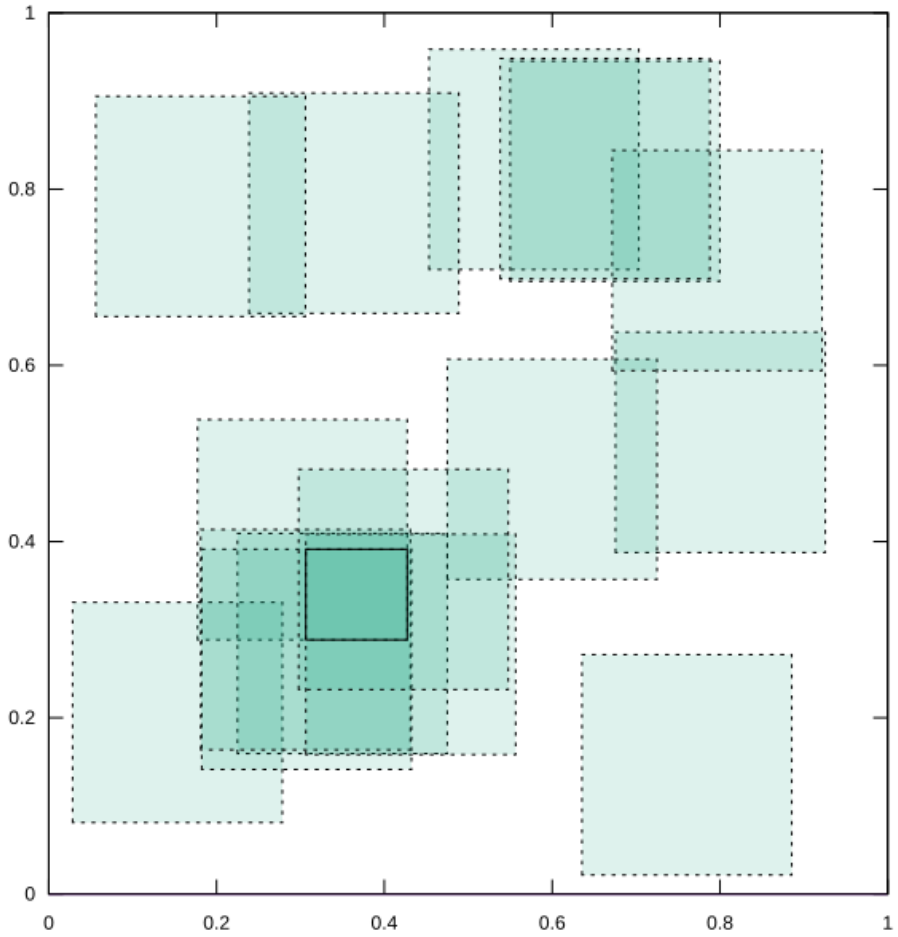
Dalším zajímavým podproblémem algoritmu 1 je krok 6. Zde je potřeba hledat maximální průnik množiny obálek S (*arrangements*). Ukázkový příklad ve 2D prostoru lze vidět na obrázku 3.4.

Nejlepší algoritmus, který se podařilo realizovat, má kubickou asymptotickou složitost ($\mathcal{O}(n^3)$). To se dosáhlo pomocí rozdělení prostoru na regiony vytyčením rovin ze všech stran všech obálek ve scéně (každá obálka rozděluje prostor v jedné dimenzi dvakrát – minimální a maximální hranicí). Pokud poté budeme takto rozdělené regiony procházet v seřazeném pořadí, může se změnit současný počet obálek, které se v tom regionu nachází, pouze o 1. Jako pomocná datová struktura je využito bitové pole o velikosti n pro každou dimenzi. Prvek pole na pozici i je nastaven na hodnotu *true*, pokud se jeho obálka nachází v momentálně procházeném regionu. To nám dovoluje se v konstantní časové složitosti rozhodnout, zde nový region obsahuje o jednu obálku více či méně.

3.3.3 Optimalizace

I když není hlavním kritériem rychlost algoritmů pro výpočet metrik, je třeba dosáhnout alespoň přijatelné meze. To algoritmus pro výpočet hustoty pořád nesplňoval, a proto došlo k dalším optimalizacím.

Výrazným zrychlením jsem dosáhl po aplikaci paralelních výpočtů pomocí procesorových vláken. Ty si mezi sebou rozdělí úlohy, které vznikají v kroku 3 algoritmu 1. Jednotlivé iterace tohoto cyklu na sobě totiž nejsou závislé (místo zvyšování λ o 1 se hledá maximální hodnota), a proto je možné tuto práci rovnoměrně rozdělit. Jelikož všechny vlákna vykonávají stejnou činnost, implementace byla velmi přímočará.



Obrázek 3.4: Ukázková scéna: hledání regionu obsahující maximální množství obálek. Převzato z prototypu aplikace vytvořené vedoucím práce.

Další optimalizační postup je znám z metody větví a mezí (*branch and bound*) [2]. Při prohledávání stavového prostoru lze vyloučit větve, ve kterých se nemůže vyskytnout optimální řešení. Pokud například algoritmus zjistí, že počet objektů v množině S je menší než hodnota λ, nemá smysl pokračovat, jelikož tato množina nemůže navýšit hodnotu λ. I při hledání regionu s maximálním počtem protínajících se obálek (viz předchozí podsekce) lze v některých případech skončit dříve. Pokud je zbývající počet regionů v jedné dimenzi menší než rozdíl maximální a současné hodnoty, nemůže už algoritmus tuto spodní mez překonat.

3.3.4 Časová složitost

Výsledná asymptotická složitost celého algoritmu je v téměř všech částech nejhůře linearitmetická. To je velice přijatelná časová složitost. Poslední úsek má ovšem kubický růst (viz podsekce 3.3.2), není však závislý na velikosti vstupních dat, ale na počtu prvků množiny S, pro jejichž rozšířené obálky se hledá maximální průnik. To znamená, že algoritmu je značně pomalejší pro scény s velkým množstvím relativně blízkých objektů. Takové scény mají mimochodem potenciál pro velkou hustotu. Další důležitým závěrem je, že největší dopad na celkový výkon mají optimalizace právě v této části algoritmu, proto tedy každá malé vylepšení (viz podsekce 3.3.3) má na některé scény obrovský vliv.

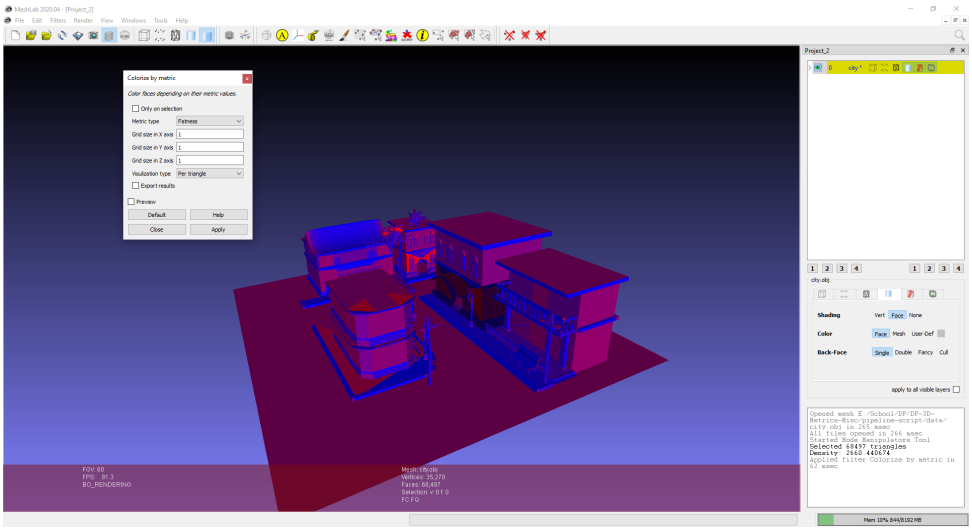
Kapitola 4

Sada testovacích a pomocných nástrojů

V této kapitole podrobněji popisuji jednotlivé nástroje vyvinuté či použité v zájmu potřeb a cílů práce. Při jejich výběru jsem hlavní důraz kladl na efektivitu, snadnou rozšiřitelnost, mé zkušenosti s nástrojem a nezávislost na operačním systému uživatele. Pro počítání metrik a interaktivní vizualizaci jejich hodnot jsem použil aplikaci s názvem MeshLab [5]. Pro renderování scén a měření výkonu stavby a traverzace BVH systém Embree [3]. Za účelem hromadného testovaní jsem vytvořil skript, který zprostředkuje běh MeshLabu a Embree a extrahuje relevantní data do výstupních souborů (popř. dále zpracuje tato data a automaticky vytvoří grafy). Jako poslední jsem napsal malý program, který generuje nové scény na základě malých úprav existujících scén.

4.1 MeshLab

MeshLab je multiplatformní software s otevřeným zdrojovým kódem za účelem zpracování a editace 3D scén tvořených triangulovanými sítěmi. Vytvořila ho skupina výzkumných pracovníků z italského Institutu pro vědu a informační technologie [12]. Poskytuje sadu nástrojů pro úpravu, náhled, texturování či konverzi modelů [10]. I když jsem využil pouze malý zlomek z této funkcionality, důležitá pro mě byla zejména podpora interaktivního náhledu scény a možnost výběru jednotlivých objektů pro zpracovaní uživatelem. Z programátorského hlediska je příjemná podpora pro geometrické objekty, 3D datové struktury či načítání scén. Celý systém je napsán velice efektivně pomocí jazyka C++. Snadnou rozšiřitelnost zajišťuje možnost přidat vlastní zásuvný modul (plugin). Jako poslední bonus byla má zkušenost s tímto programem při vytváření prototypovacího nástroje na počátku této práce, i když pouze z uživatelského hlediska. Pro ukázku aplikace a ilustraci uživatelského rozhraní přikládám screenshot 4.1.



Obrázek 4.1: Ukázka aplikace MeshLab.

Po základním seznámení se zdrojovým kódem bylo mým prvním úkolem integrovat do systému MeshLabu novou funkcionalitu metrik ve formě samostatného pluginu. Bohužel jsem nikde nenašel podrobnější dokumentaci, jak tohoto docílit. Inspiraci jsem tedy našel u již implementovaných modulů, kterých MeshLab obsahuje poměrně velké množství. Naštěstí architektura systému je velice elegantní, a proto se nakonec jednalo o velice přímočarý proces. Nový plugin přidává do aplikace možnost použití filtru, který aplikuje na scénu algoritmus pro výpočet metrik a náležitě ji vizualizuje na základě výsledku.

Obecně může filtr měnit různé parametry scény – polohu vrcholů, barvu, normály, triangulovat geometrii apod. Každá třída, která implementuje takovýto filtr, musí dědit ze třídy MeshFilterInterface, a tedy implementovat minimálně několik abstraktních metod. Já k tomu z různých důvodů přetížil chování ještě několika dalších. Základ mého filtru FilterMetricsPlugin tvoří následující metody:

- MeshFilterInterface::filterName – jméno filtru (zobrazí se v nabídce uživatelského rozhraní)
- MeshFilterInterface::filterInfo – podrobnější popis akcí, které filtr provede
- MeshFilterInterface::getRequirements – speciální požadavky na model
- MeshFilterInterface::getPreConditions – atributy modelu potřebné pro aplikaci filtru

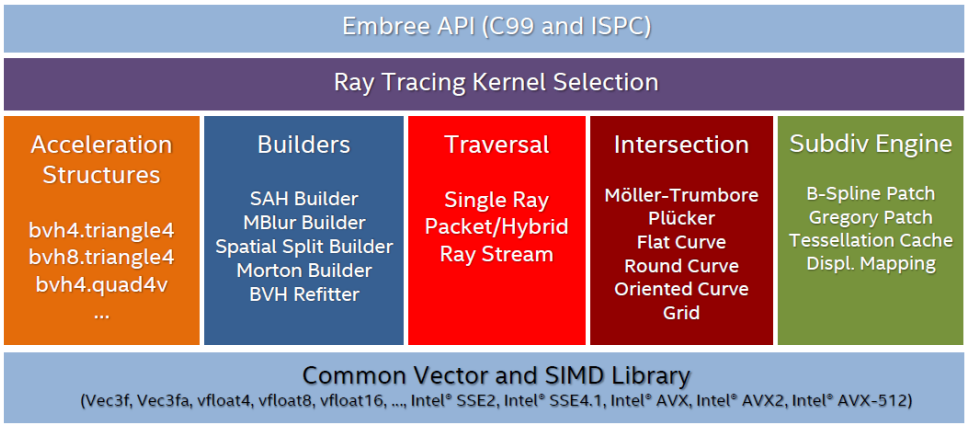
- MeshFilterInterface::postCondition – atributy modelu, které filtr změní při jeho aplikaci
- MeshFilterInterface::initParameterSet – seznam parametrů upravující chování filtru, které může uživatel modifikovat
- MeshFilterInterface::applyFilter – aplikuje filtr na model na základě parametrů

Filtr metrik podporuje pomocí vstupních parametrů i další funkce. Uživatel má možnost vybrat si zájmový region scény. Pouze ten se bude brát v potaz při výpočtu metrik, zbylá scéna bude ignorována. Dále má možnost rozdělit scénu do pravidelné mřížky, nad jejímiž buňkami se budou počítat jednotlivé metriky. To má opět za cíl vymezit jednotlivé oblasti scény, o které má uživatel zájem. Tato mřížka může současně urychlit výpočet některých metrik, jelikož značně redukuje velikost vstupních dat, což má velký vliv u nepříznivé asymptotické složitosti algoritmu (lineární a vyšší). Kromě vizualizace si může uživatel zvolit export statistik metrik do jednoduchého textového souboru.

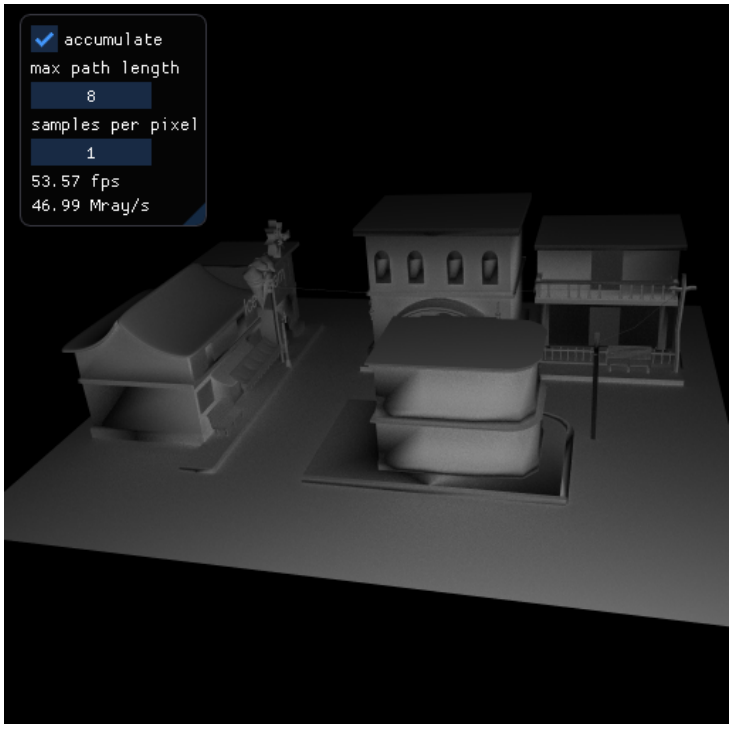
4.2 Embree

Embree je vysoce výkonná knihovna pro sledování paprsků, kterou vyvíjí firma Intel. I tento software má kompletně otevřený zdrojový kód. Navíc na něm autoři aktivně pracují a podporuje tedy nejnovější techniky. Knihovna je optimalizována pro procesory Intel s podporou SSE, AVX, AVX2, and AVX-512 instrukcí pro paralelizaci SIMD (*Single Instruction, Multiple Data*) operací. Hlavní funkcionalitu tvoří efektivní nalezení nejbližšího (či jakéhokoliv) objektu protínající vyslaný paprsek, podpora pro svazky paprsků či paralelní algoritmy pro stavbu BVH [8]. Embree systém je rozdělen do několika jader – viz schéma 4.2.

Embree je koncipováno jako knihovna, kterou má každý možnost integrovat a rozšiřovat pro své účely. Součástí je i sada minimalistických aplikací, jejímž účelem je ukázat několik stěžejních funkcionalit. Jedna z těchto aplikací se jmenuje Path Tracer a implementuje algoritmus sledování cest (*path tracing*) [16] pro renderování scény. Ta podporuje celkem šest algoritmů pro stavbu BVH a dva algoritmy pro její traverzaci. Aplikace má i uživatelské rozhraní (viz screenshot 4.3), které poskytuje zejména náhled na syntetizovanou scénu a několik základních statistik.



Obrázek 4.2: Přehled systému Embree. Převzato z [8].



Obrázek 4.3: Ukázka aplikace Path Tracer postavené na systému Embree.

4.3 Testovací řetězec

Důležitou částí této práce je testování. I při relativně malém množství scén máme mnoho vstupních kombinací, ať už se jedná o výběr metriky, transformace scény či nastavení parametrů Embree. Vytvořil jsem proto testovací řetězec, který tuto činnost automatizuje a zároveň poskytuje výsledná data

jak v textové, tak grafické podobě (formou grafů). Jako technologii jsem zvolil programovací jazyk Python. Jedná se o vysokoúrovňový skriptovací jazyk, který je ideální pro mé účely. Obzvláště prospěšnou považuji knihovnu Matplotlib pro generování grafů.

Jak jsem již zmínil v sekci 4.1, MeshLab je aplikace s uživatelským rozhraním. Naštěstí autoři tohoto nástroje neopomněli možnost dávkového zpracování. Součástí systému je i konzolová aplikace pro tyto účely – MeshLab Server. Uživatel má možnost definovat sadu filtrů a jejich parametrů a uložit je ve formátu MLX (*MeshLabXML*). Tento soubor se poté spolu se scénou předá jako vstupní argument a MeshLab Server následně tuto sadu filtrů aplikuje na danou scénu. Takový přístup se zdá jako velice flexibilní. Ovšem pokud chceme měnit parametry ve filtru, musíme pro každou novou kombinaci vytvořit nový soubor MLX. To není z pohledu automatického zpracování příjemné, a proto jsem se rozhodl vytvořit šablonu pro filtr (viz 4.1), který aplikuje na scénu výpočet metrik. Do této šablony se poté vloží hodnoty jednotlivých parametrů, které si uživatel definuje při spuštění testovacího skriptu. To dává možnost tyto hodnoty parametrizovat bez potřeby mít připravené velké množství MLX souborů.

```
<Param name="MetricType" value="$1" enum_val0="Fatness"
type="RichEnum" isxmlparam="0" enum_cardinality="2"
tooltip="The type of the metric to plot."
description="Metric type" enum_val1="Density"/>
```

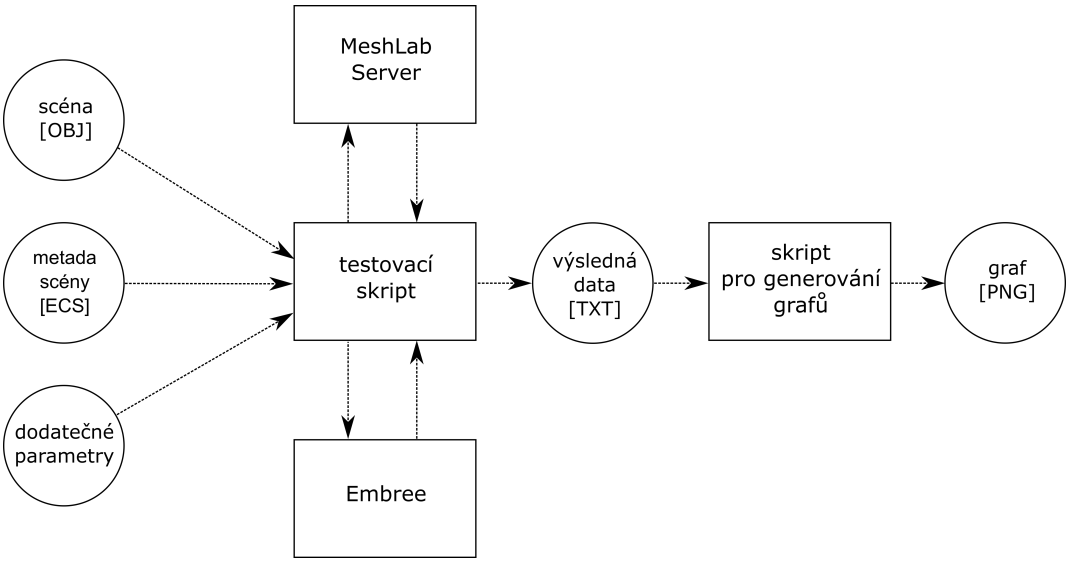
Výpis 4.1 : Záznam z generické šablony MeshLab Serveru.

Aplikace Path Tracer založená na systému Embree se ukázala jako dostačující, bez potřeby dodatečných změn, pro potřeby testování v této práci. Jedná se o konzolovou aplikaci s velkým množství vstupních parametrů. Užitečný je zejména benchmark mód, který poskytuje detailní údaje o časovém trvání jednotlivých kroků algoritmu a jiných statistikách a zároveň nevytváří uživatelské prostředí či jakoukoliv jinou interakci s uživatelem, tudíž je ideální k hromadnému testování.

Jelikož Embree je systém, pro nějž je výkon hlavní priorita, snaží se při svém běhu omezit všechny nepodstatné operace. Tím se považují i mnohé statistické údaje, které mohou být pro testování velice zajímavé. Je proto třeba shromažďování a výpis těchto údajů explicitně zapnout před kompilací aplikace (např. při vytváření projektu) pomocí makra EMBREE_STAT_COUNTERS.

Vytvořený testovací skript tedy zprostředkovává spuštění aplikací MeshLab Serveru a Embree – viz diagram 4.4. Běhy obou aplikací jsou na sobě nezávislé. I když by teoreticky bylo možné je spouštět najednou (paralelně), systém je

při chodu obou programů natolik zatížen, že bychom výsledky nezískali dříve. Naopak bychom dostali zkreslené výkony, jelikož by soupeřily o prostředky, a tak se vzájemně sabotovaly. Příklad spuštění testovacího skriptu spolu s několika parametry lze vidět na výpisu 4.2.



Obrázek 4.4: Diagram testovacího řetězce.

```
python testing-script.py -of city -m fatness -gx 3
-mls "-i data/city.obj" -emb "-c data/city.ecs -o preview.tga
-spp 64 -benchmark 1 5 -verbose 2 -rtcore tri_builder=sah"
```

Výpis 4.2 : Ukázka spuštění testovacího skriptu.

Výstupem jak MeshLabu, tak Embree jsou textové soubory se statistickými údaji. V případě MeshLabu jde o přímo mnou vybrané údaje, které jsou relevantní a zároveň v jednoduchém formátu pro další zpracování. Aplikace Embree generuje velice obsáhlé textové záznamy, avšak většina z nich není z pohledu této práce příliš zajímavých. Testovací skript tedy vybírá pouze malou část, kterou uloží v podobném formátu jako výstupy z MeshLabu. Příklad výstupních textových souborů lze vidět na výpisu 4.3 a 4.4.

```
metric type: fatness
grid resolution: 3 x 1 x 1
min (per triangle): 2.06764e-06
max (per triangle): 0.13761
avg (per triangle): 0.0388448
histogram (size=10): <0-0.0137832> 13997, <0.0137832-0.0275664> 14174 ...
data: 0.0491147 0.0406696 0.0277376
```

```
normalized data: 0.356337 0.295066 0.201242
```

Výpis 4.3 : Ukázka výstupních dat MeshLabu.

```
triangles:
  accel              = default
  builder            = sah
  traverser          = default

build-time: 3.51 [msec]
build-sah: 8.84

render-avg: 0.14 [fps]
render-mrayps-avg: 6.81

traversal stats:
  #normal_travs    = 231.31M
    #nodes         = 922.15M
    #nodes_xfm     = 0.00M
    #leaves        = 245.16M
    #prims         = 293.24M
```

Výpis 4.4 : Ukázka výstupních dat Embree.

Samotná data v textové podobě jsou sice přesnou reprezentací výsledných hodnot, avšak při hledání souvislostí mezi jednotlivými veličinami poměrně nepřehledná. Z toho důvodu je součástí testovacího řetězce i automatická tvorba grafů. Ty lze názorně vidět v kapitole 5 a zejména 6, kde hrají významnou roli při vyhodnocení výsledků.

4.4 Transformace scén

Jeden z bodů práce je generování nových scén na základě malých úprav. Porovnání původních a nově vygenerovaných scén nám poté dává možnost pozorovat změnu hodnot metrik a jejich chování, popř. potvrdit naše očekávaní (viz kapitola 6). Pro tento účel jsem vytvořil malou konzolovou aplikaci, která načítá triangulované scény ve formátu OBJ (*Wavefront*), náležitě je transformuje a následně uloží ve stejném formátu.

Aplikace je napsána v jazyce C++ a pro snížení rozsahu implementace jsem použil dvě knihovny třetích stran. Pro načítání scén jsem uplatnil minimalistickou knihovnu Tiny OBJ Loader [7], jejíž výhoda není jen malá velikost či výpočetní rychlost, ale zejména možnost využití callback funkcí. Pomocí nich si může uživatel ukládat jednotlivé objekty a grafická primitiva přímo do

vlastních datových struktur. Dále jsem využil matematickou knihovnu GLM [6], která mi pomáhá například se základní prací s vektory či maticemi a nad nimi definovanými operacemi.

Nástroj podporuje následující transformace:

1. Rotace: otočení celé scény kolem osy Y o daný úhel. Nenastává žádná změna geometrie či topologie scény.
2. Dělení velkých objektů: příliš velké trojúhelníky se rozdělí na čtyři menší. Vzniknou vytyčením spojnic středů hran a jsou identické. Nastává změna topologie scény.
3. Exploze: nekonstantní posun objektů od středu scény v závislosti na jejich vzdálenosti od tohoto středu (vzdálenější objekty se posunují více). Nastává změna topologie i geometrie scény.

Každá transformace má transformační parametr v, který určuje její míru. Význam tohoto parametru je jiný při každé operaci:

1. Rotace: v úhel rotace ve stupních.
2. Dělení velkých objektů: práh pro určení velikosti trojúhelníku, který se při jeho překročení bude dělit. Matematické vyjádření:

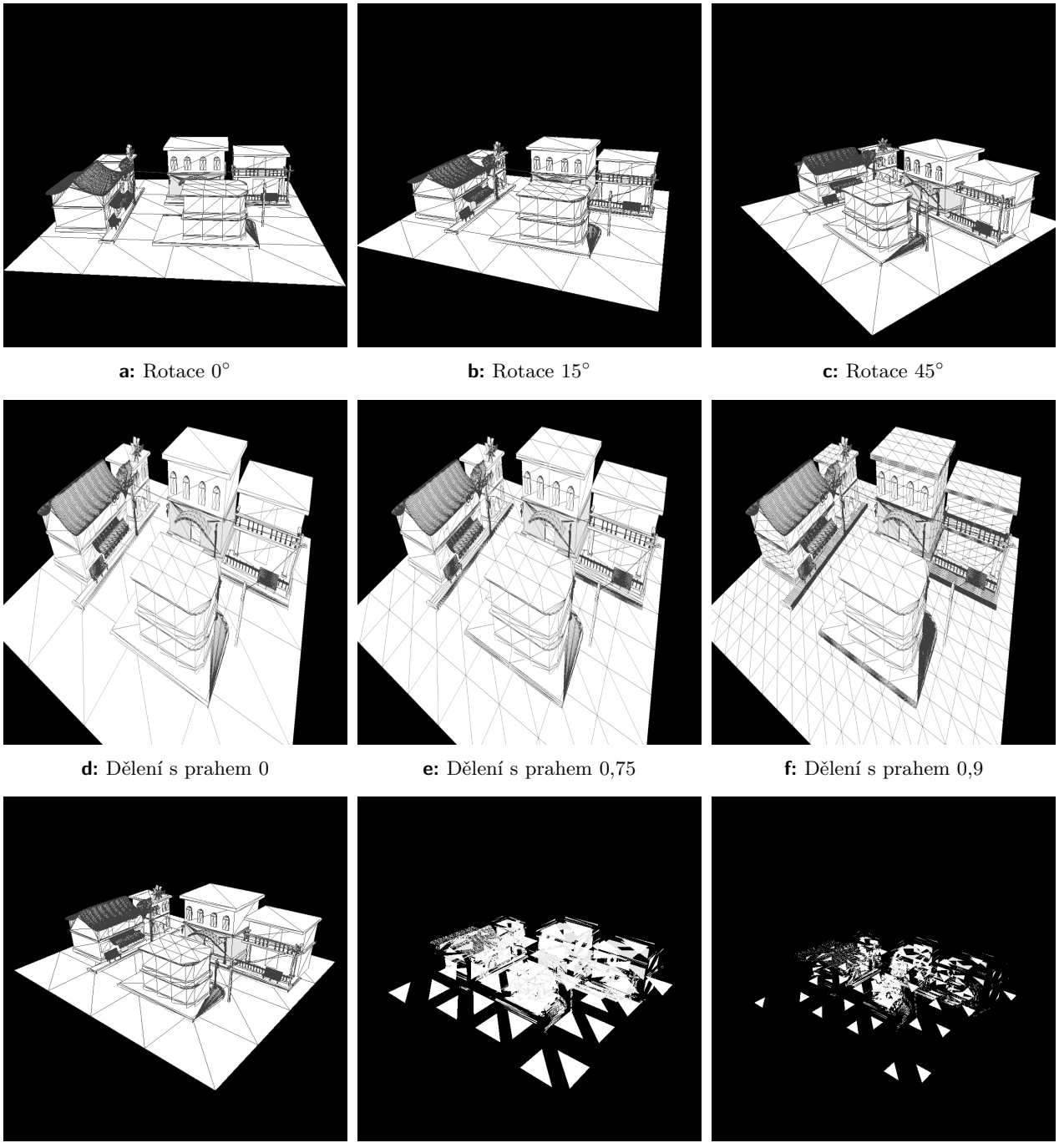
$$průměr\ trojúhelníku > průměr\ scény\ (1 - v)$$

   Při hodnotě $v = 0,7$ tedy budou rozděleny všechny trojúhelníky, které mají průměr větší než 30% průměru scény. Dělení probíhá rekurzivně – pokud je nově vzniklý trojúhelník stále nad prahem, je dělen dále, dokud tuto podmínku nesplňuje.

3. Exploze – exponenciálního hodnota poměru vzdálenosti od středu a vzdálenosti nejvzdálenějšího objektu ve scéně. Matematické vyjádření (vzdálenost je v tomto vzorci myšlena vzdálenost těžiště trojúhelníku od středu scény):

$$vzdálenost = vzdálenost\ (1 + \frac{vzdálenost}{vzdálenost\ nejvzdálenějšího\ objektu})^v$$

Pro ukázku aplikace jsem připravil vizualizaci scén na obrázku 4.5 s jednotlivých transformacemi a různými transformačními parametry.



a: Rotace 0° **b:** Rotace 15° **c:** Rotace 45°

d: Dělení s prahem 0 **e:** Dělení s prahem 0,75 **f:** Dělení s prahem 0,9

g: Exploze s exponentem 0 **h:** Exploze s exponentem 1 **i:** Exploze s exponentem 2

Obrázek 4.5: Vizualizace transformací na scéně city pomocí aplikace MeshLab.

Kapitola 5

Vizualizace

V první části kapitoly se zabývám vizualizací scén na základě hodnot metrik a prezentuji ji na sadě scén, které jsem použil i pro testování v následující kapitole 6. V druhé části díky informaci o distribuci dat představuji několik histogramů a uvádím, jak tyto data mohou indikovat určitý typ scény.

5.1 Vizualizace scén

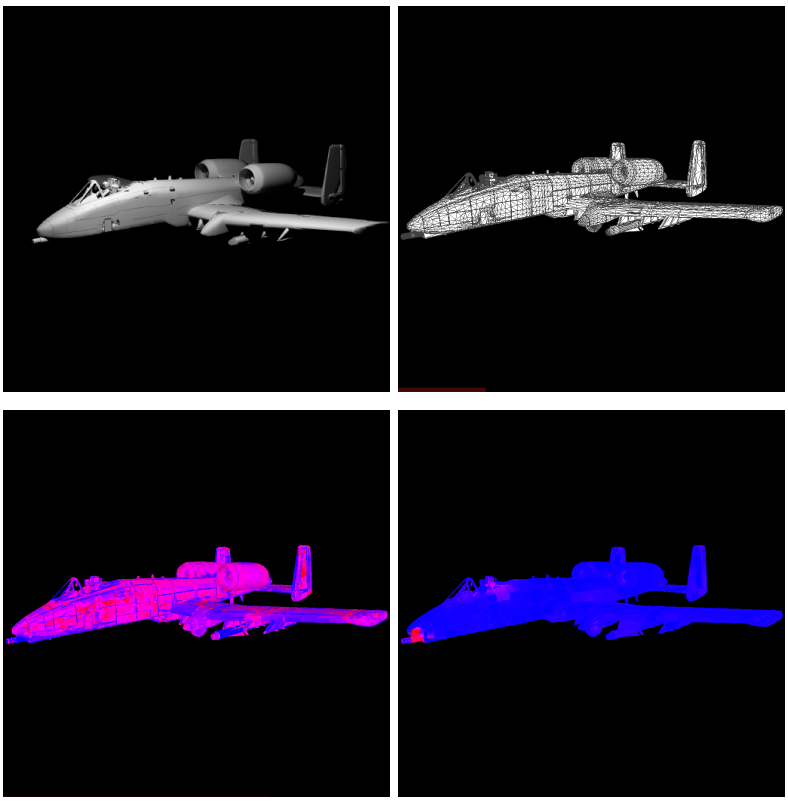
Vizualizace má v dnešní době mnoho využití, zejména v medicíně, vědě či pro vzdělávací účely. Zprostředkovává prostřednictvím zraku jak abstraktní představy, tak konkrétní údaje. V případě této práce se jedná o data, která představují hodnotu metrik pro jednotlivé objekty ve scéně (v tomto kontextu trojúhelníky). Poskytuje tak uživateli možnost náhledu na scénu v interaktivním prostředí, kterou může využít pro lepší porozumění scény (konkrétní informace záleží na samotné metrice). Také může například identifikovat zájmové oblasti, se kterými bude dále pracovat. Z mého pohledu byla vizualizace velice užitečná i při vývoji algoritmů pro výpočet metrik, zvláštně pro dostatečně malé, často uměle vytvořené, scény. Nesmíme zapomenout, že vizualizace je jen pomůcka a přesná data pro vyhodnocení výsledků či dalšího zpracovaní jsou dostupná i v textové podobě.

Postup pro vizualizace scén je velmi přímočarý: každý trojúhelník ve scéně je obarven v závislosti na hodnotě jeho metriky. Jako první je potřeba výsledné hodnoty metrik normalizovat, aby bylo možné tyto hodnoty předat do přechodové funkce a transformovat je tak na barvu. V případě tloušťky je zřejmé, že lze používat maximální dosažitelnou hodnotu β (viz sekce 3.2). Pro hustotu žádná taková horní hranice neexistuje, je potřeba tedy provést vizualizaci až po vypočtení této metriky pro celou scénu a získání její maximální hodnoty. Přechodová funkce pro normalizované hodnoty využívá prostou lineární interpolaci mezi dvěma barvami – studené (modré) a teplé

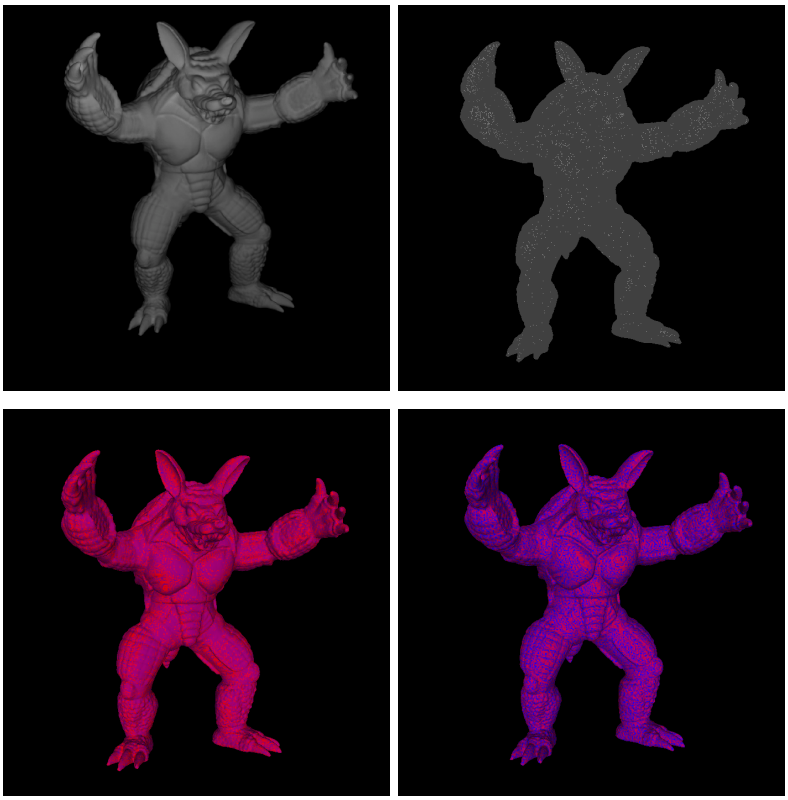
(červené), od nejnižších hodnot po nejvyšší. Původně používaná barevná škála byla pestřejší, což ovšem ve finále dělalo vizualizaci o to nesrozumitelnější pro uživatele.

Vizualizace jednotlivých objektů může být problémová pro metriky, které mají globální charakter a nepřiřazují žádné hodnoty samotným objektům. To platí i v případě hustoty. Naštěstí ovšem v tomto případě lze vizualizaci realizovat díky iterativnímu charakteru algoritmu, který vždy zkoumá okolí určitého objektu. Na druhou stranu, pro zrychlení výpočtu metrik zanedbáváme objekty, které už nemohou ovlivnit koncový výsledek (viz podsekce 3.3.3). To by nám mohlo znehodnotit případnou vizualizaci, jelikož nemáme přesné výsledky metrik pro každý objekt. Z tohoto důvodu jsem se nechal inspirovat Embree aplikací a zavedl do mé implementace kompilační makro BENCHMARK_METRICS, které povoluje tyto optimalizace. Lze tak vytvořit dvě verze aplikace – jednu pro přesnou vizualizaci a druhou, která maximalizuje rychlost výpočtu metrik.

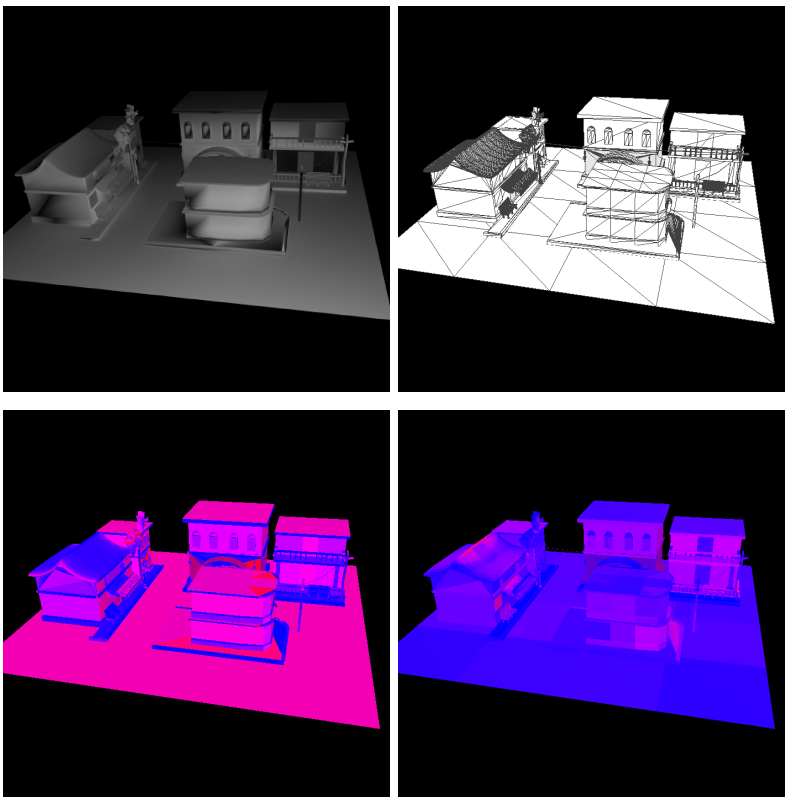
Na dalších několika stránkách následuje vizualizace každé scény 5.1–5.6 ve čtyřech formách (zleva-doprava a seshora-dolů): render z aplikace Path Tracer systému Embree (za použití 1024 vzorků pro jeden pixel), náhled drátěného modelu scény v MeshLabu, vizualizace tloušťky v MeshLabu a vizualizace hustoty v Meshlabu.



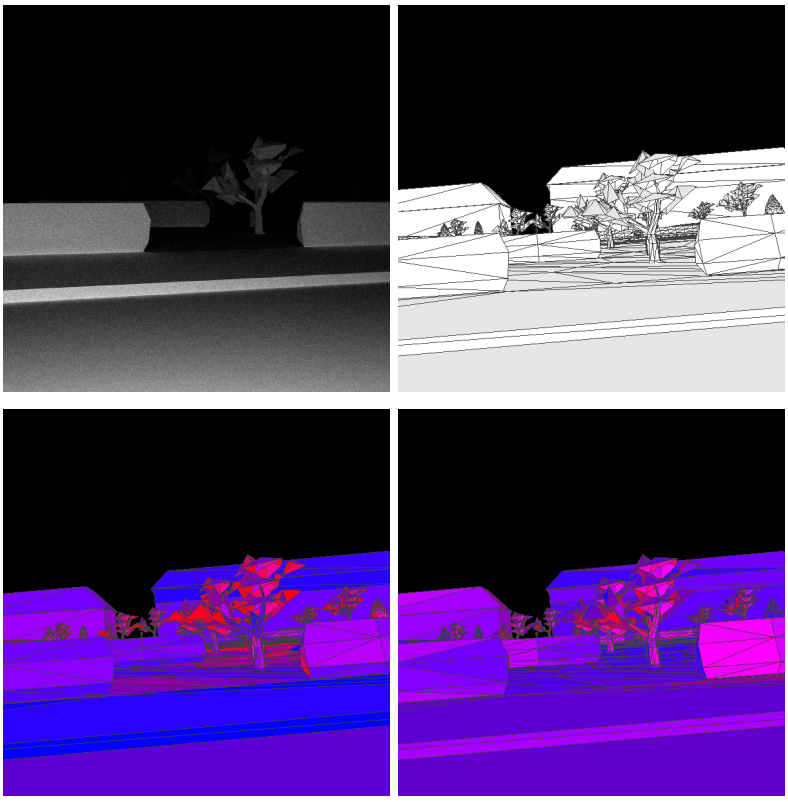
Obrázek 5.1: Vizualizace scény A10.



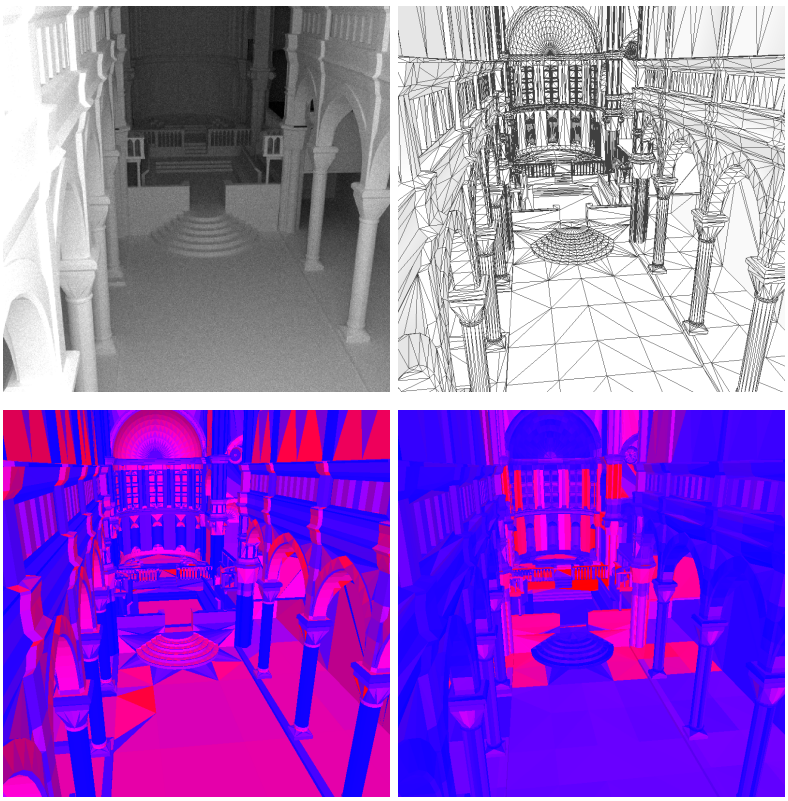
Obrázek 5.2: Vizualizace scény Armadillo.



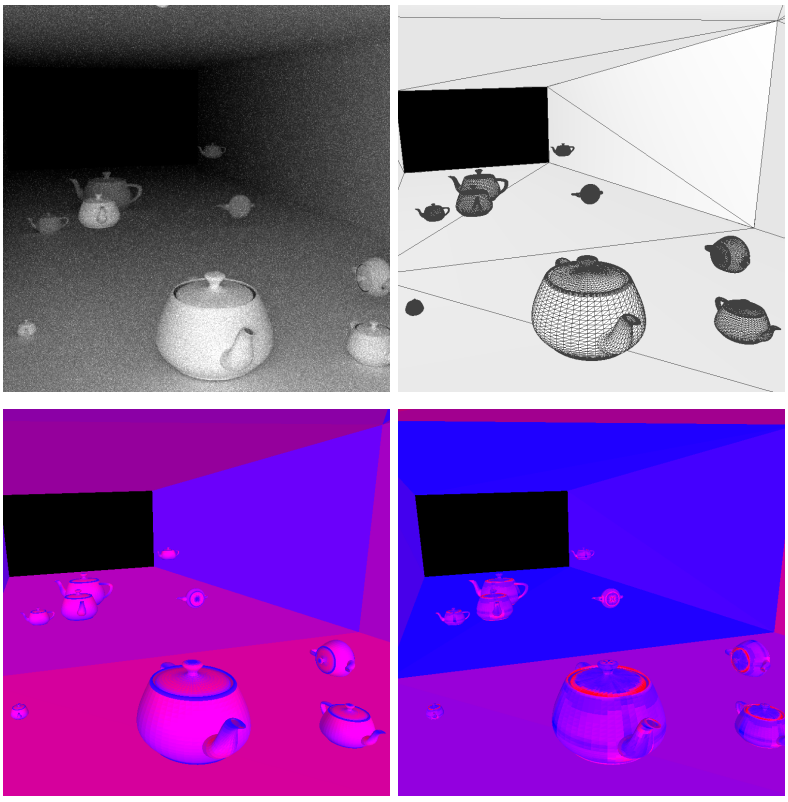
Obrázek 5.3: Vizualizace scény city.



Obrázek 5.4: Vizualizace scény park.



Obrázek 5.5: Vizualizace scény sibenik.



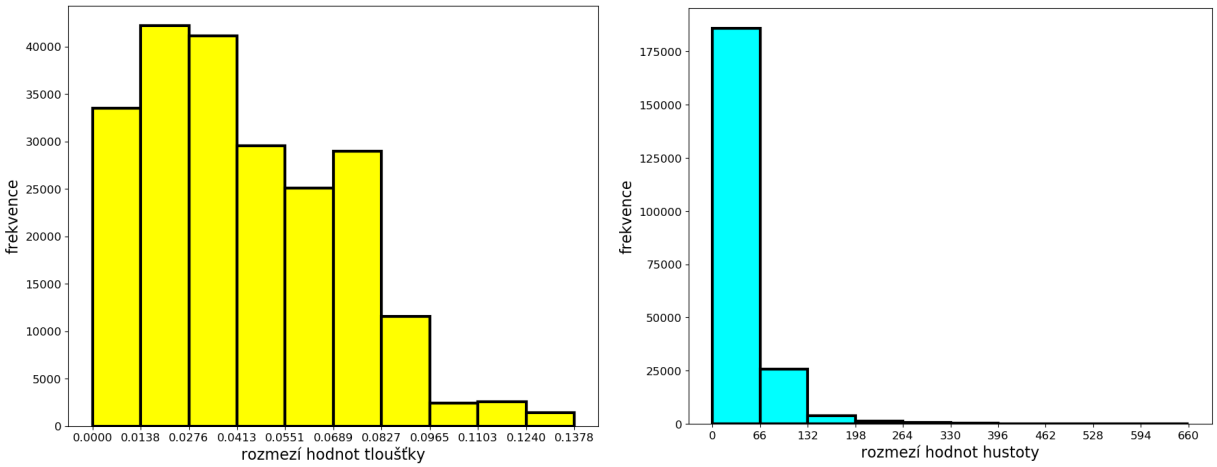
Obrázek 5.6: Vizualizace scény teapots.

5.2 Vizualizace distribuce dat

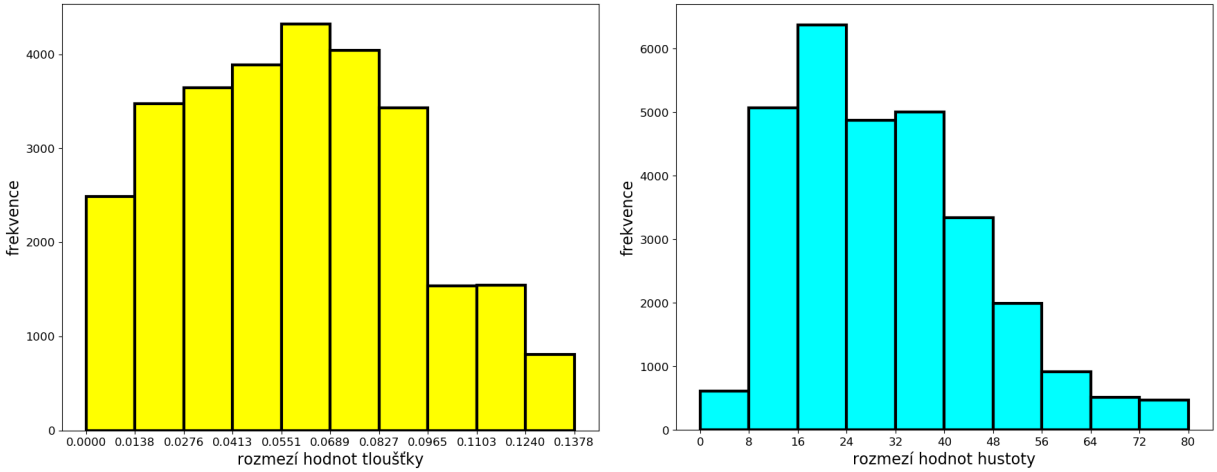
K vizualizaci distribuci dat výsledných metrik nám poslouží histogramy. Informační hodnota tohoto grafické znázornění spočívá v demonstraci rozložení dat a může nám pomoct k vyvození určitých rysů pro scénu, pokud bude distribuce vykazovat nějaký charakteristický vzor. Pro ukázku jsem připravil dva histogramy ze scény A10 (obrázek 5.7) a park (obrázek 5.8). Intervaly jsou vymezeny rovnoměrně každých 10% z maximální hodnoty ve scéně (je jich tedy celkově 10).

Obě scény mají poměrně vyrovnané rozložení tloušťky. Ta platí obecně pro většinu scén, jelikož jsou zpravidla tvořeny různorodě tlustými trojúhelníky. Výjimku tvoří modely, které jsou velice monotónní (např. Armadillo), kde jsou si jednotlivé tvary, a tudíž i trojúhelníky, velice podobné – není tedy potřeba extrémních tvarů (např. úzké, jež mají velice nízkou tloušťku). Také stojí za povšimnutí fakt, že obě scény mají poměrně malé množství objektů, jejichž tloušťka se blíží k maximální hranici. Tento objekt představuje ve scéně rovnostranný trojúhelník. Důvodem může být, že takovýto tvar je pro modelování reálných objektů až příliš "dokonalý", a proto tedy zřídka používaný.

Na druhou stranu histogramy pro hustotu se diametrálně liší. Scéna A10 má většinu objektů s velice malou hustotou. Existují zde ale regiony, kde se shlukuje velké množství trojúhelníků o podobném rozměru a zvyšují tak celkovou hustotu scény (více v sekci 6.1). Tímto regionem může být na architektonických scénách velice detailní objekt (např. lavička ve městě) či komplikované části na samotném modelu (např. kulomet na stíhačce). Pokud ovšem scéna takovéto objekty nemá, může pak histogram hustoty vypadat jako u scény park.



Obrázek 5.7: Histogram dat scény A10: tloušťka (nalevo) a hustota (napravo).



Obrázek 5.8: Histogram dat scény park: tloušťka (nalevo) a hustota (napravo).

Kapitola 6

Výsledky a vyhodnocení testování

V této kapitole prezentuji výsledky testování na několika různorodých scénách. Ty můžeme klasifikovat do 3 typů: architektonické scény (city, sibenik), osamocené modely (A10, Armadillo) a scenérie (teapots, park). Velikost scén se pohybuje v řádu desítek až stovek tisíc trojúhelníků. Kapitola je rozdělena do dvou sekcí – v první se zajímám pouze o metriky a v druhé k tomu přidávám data z rendereru a hledám mezi nimi souvislosti.

Pro testování jsem použil následující konfiguraci:

- CPU: Intel Core i7-8700, 4,4GHz, 12 vláken
- RAM: 16GB
- OS: Windows 10 64 bit
- Kompilátor: The Visual C++ Compiler 19.23

6.1 Metriky

Pro výpočet metrik není použita žádná mřížka a všechny scény jsou obsaženy celé (žádná selekce zájmového regionu).

V tabulce 6.1 jsou uvedeny hodnoty metriky tloušťky pro jednotlivé scény, konkrétně minimální a aritmeticky průměrná hodnota. Minimální hodnota tloušťky je ve většině případů přibližně o 3 řády menší než její průměr. To může být problém pro algoritmy, které zajímá pouze nejméně tlustý objekt ve scéně. Vždy je možné tyto extrémní případy zanedbat (např. nebrat v úvahu 5 procent trojúhelníků s nejmenší tloušťkou), což ale v našem případě není vhodné aplikovat. Představme si ve scéně protáhlý trojúhelník s malou tloušťkou. Takový objekt je z pohledu stavby BVH velice problematický a určitě bychom se neradi o tuto informaci připravili. Aritmetický průměr je spočítán vzhledem k celkovému počtu trojúhelníků ve scéně. Jak lze pozorovat,

mezi scénami může být až dvojnásobný rozdíl této metriky. Na jedné straně Armadillo má největší průměrnou tloušťku, což má vzhledem ke vztahu s metrikou hustoty (viz sekce 2.1.1) implikovat malou hustotu. To lze potvrdit z tabulky 6.2, kde je vidět, že tato scéna má nejmenší maximální hustotu. Na druhé straně je scéna city s malou průměrnou tloušťkou. Tady ovšem již vztahy mezi metrikami aplikovat nelze, jelikož malá tloušťka nic neimplikuje.

scéna	počet trojúhelníků [v tisících]	minimum	aritmetický průměr
A10	219	$3{,}74 \cdot 10^{-7}$	$4{,}32 \cdot 10^{-2}$
Armadillo	346	$9{,}63 \cdot 10^{-5}$	$8{,}87 \cdot 10^{-2}$
city	68	$2{,}07 \cdot 10^{-6}$	$3{,}88 \cdot 10^{-2}$
park	29	$10{,}19 \cdot 10^{-5}$	$5{,}97 \cdot 10^{-2}$
sibenik	80	$3{,}61 \cdot 10^{-9}$	$4{,}33 \cdot 10^{-2}$
teapots	201	$35{,}43 \cdot 10^{-4}$	$5{,}09 \cdot 10^{-2}$

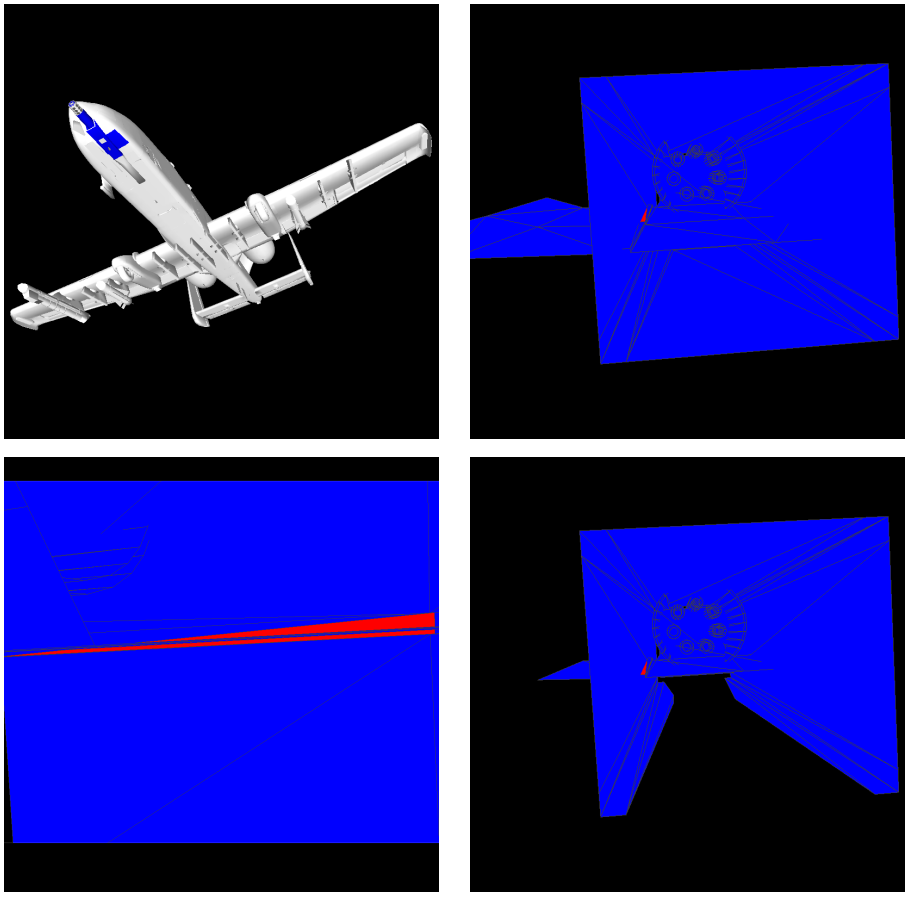
Tabulka 6.1: Hodnoty metriky tloušťky pro jednotlivé scény.

V tabulce 6.2 jsou uvedeny hodnoty metriky hustoty pro jednotlivé scény. Veliké maximální hodnoty hustoty naznačují, že se ve scéně nachází oblast se značným shlukem objektů na malém prostoru, viz scéna A10. Pak rozdíl mezi maximální a průměrnou hodnotou může naznačit, jak moc je scéna pravidelná vzhledem k hustotě jejich oblastí. Důležité je opět připomenout, že hustota je globální metrika, tj. její výsledná hodnota se týká celé scény, ne jednotlivých objektů v ní. Průměrná hodnota je tedy poměrně uměle vytvořená statistika, které nemá základ v definici metriky, ale díky povaze algoritmu, jakým je počítána.

scéna	počet trojúhelníků [v tisících]	maximum	aritmetický průměr
A10	219	660	41,64
Armadillo	346	24	12,05
city	68	304	46,65
park	29	80	29,83
sibenik	80	189	37,25
teapots	201	68	26,48

Tabulka 6.2: Hodnoty metriky hustoty pro jednotlivé scény.

Některé scény vykazují relativně velké údaje o maximální hustotě (např. A10 či city). Jediné referenční hodnoty, které mám k dispozici, jsou ze studie geografických scén ve 2D prostoru z původního článku o realistických metrikách [9]. U nich vychází hustota v řádu desítek, avšak mé výsledky se pohybují i v řádu stovek. Takto vysoké hodnoty mají však své opodstatnění. Prvním je zvýšení dimenze prostoru. Tím dochází k exponenciálnímu nárůstu eventuálně blízkých objektů, a tudíž i exponenciálnímu nárůstu hodnoty hustoty. Druhým důvodem je aproximace dat (viz sekce 3.3). Tím dochází také ke zvýšení této hodnoty, i když by nemělo být markantní. Přesná čísla je ovšem těžké posoudit. Jako příklad uvádím sadu obrázků 6.1 scény A10 se zaměřením na region s maximální hustotou.



Obrázek 6.1: Scéna A10 v různých pohledech na region s maximální hustotou. Vlevo nahoře – celý objekt ze spodního pohledu s modře vyznačenými trojúhelníky (část předního kulometu), které tvoří množinu dostatečně blízkých trojúhelníků vzhledem k právě zpracovávanému a mají alespoň stejně velký průměr. Vlevo dole – detail na právě zpracovávaný trojúhelník (červeně). Vpravo nahoře – detail shluku ze zadního pohledu, ostatní objekty jsou skryté. Vpravo dole – detail shluku ze zadního pohledu, modře viditelné pouze trojúhelníky, jejichž obálky tvoří maximální průnik, a tedy určuje i výslednou hodnotu hustoty.

6.2 Vztah mezi hodnotami metrik a výkonem Embree

Renderování scén pomocí Embree je v rozlišení 512×512 a pro každý pixel je vygenerováno 64 vzorků. Maximální počet odrazů primárních paprsků je 8 (výchozí hodnota). Při měření se zanedbává první snímek z důvodu neustáleného stavu renderovacího procesu, ale měří se následujících 30 snímků. Tento velký počet dokáže s větší pravděpodobností eliminovat nepřesnosti zapříčiněné úlohami běžícími na pozadí operačního systému. Navíc jsem se snažil nastavit systém a naplánovat úlohy tak, aby k tomu nedocházelo. Pro statistiky, které jsou založené na čase (např. FPS) jsem použil verzi Embree s vypnutými dodatečným informacemi. Naopak, pokud jsem chtěl, aby systém tyto údaje měřil a vypsal do výstupního souboru (např. počet traverzačních kroků), využil jsem sestavení, kde je toto chování explicitně povolené. Tato verze ale negativně ovlivní výkon celého systému.

Následující strany obsahující grafy pro všechny scény 6.2–6.7 na něž jsou aplikovány 3 různé typy transformace – rotace, dělení a exploze s různými transformačními parametry. Každý sloupec grafu představuje jednu tuto transformaci. Po řádcích jsou zachyceny postupně tyto údaje: maximální hustota scény, SAH (*surface area heuristic*) [14], počet traverzačních kroků primárního paprsku a počet snímků za vteřinu, tj. FPS (*frames per second*). Poslední dva řádky grafů představují vztah mezi hodnotou SAH a maximální hustotou scény a vztah mezi počtem traverzačních kroků a maximální hustotou scény (scény bez transformace mají v grafu diamantový tvar). Za zmínku stojí, že metriku tloušťky jsem v tomto případě vynechal, jelikož její hodnota je invariantní k všem těmto transformacím (s výjimkou dělení, ovšem zde se mění průměrná hodnota pouze z důvodu, že rozdělené objekty dávají výsledku větší váhu, jelikož se jejich počet navýší, ale jejich tloušťka zůstane stejná).

Rotace scény nemá v téměř všech scénách velký vliv na maximální hodnotu hustoty. To dává smysl, jelikož se nemění topologie ani geometrie scény. Jelikož ovšem hustota pracuje s obálkami ve formě osově zarovnaných kvádrů, může v některých případech rotace scény ovlivnit tuto metriku (viz scéna sibenik). Výrazné zvýšení hustoty ovšem způsobuje dělení scény, jelikož tak vzniká velké množství stejně velkých objektů (které jsou těsně pod prahem dělení). Takto na to reagují všechny scény. Naopak exploze zvyšuje vzdálenosti mezi objekty, a tudíž i snižuje hustotu celé scény. To je opět vidět ve všech testovaných případech. Hodnota SAH reaguje podobně na transformace jako hustota. Velkou roli hraje fakt, že také pracuje s obalovými tělesy.

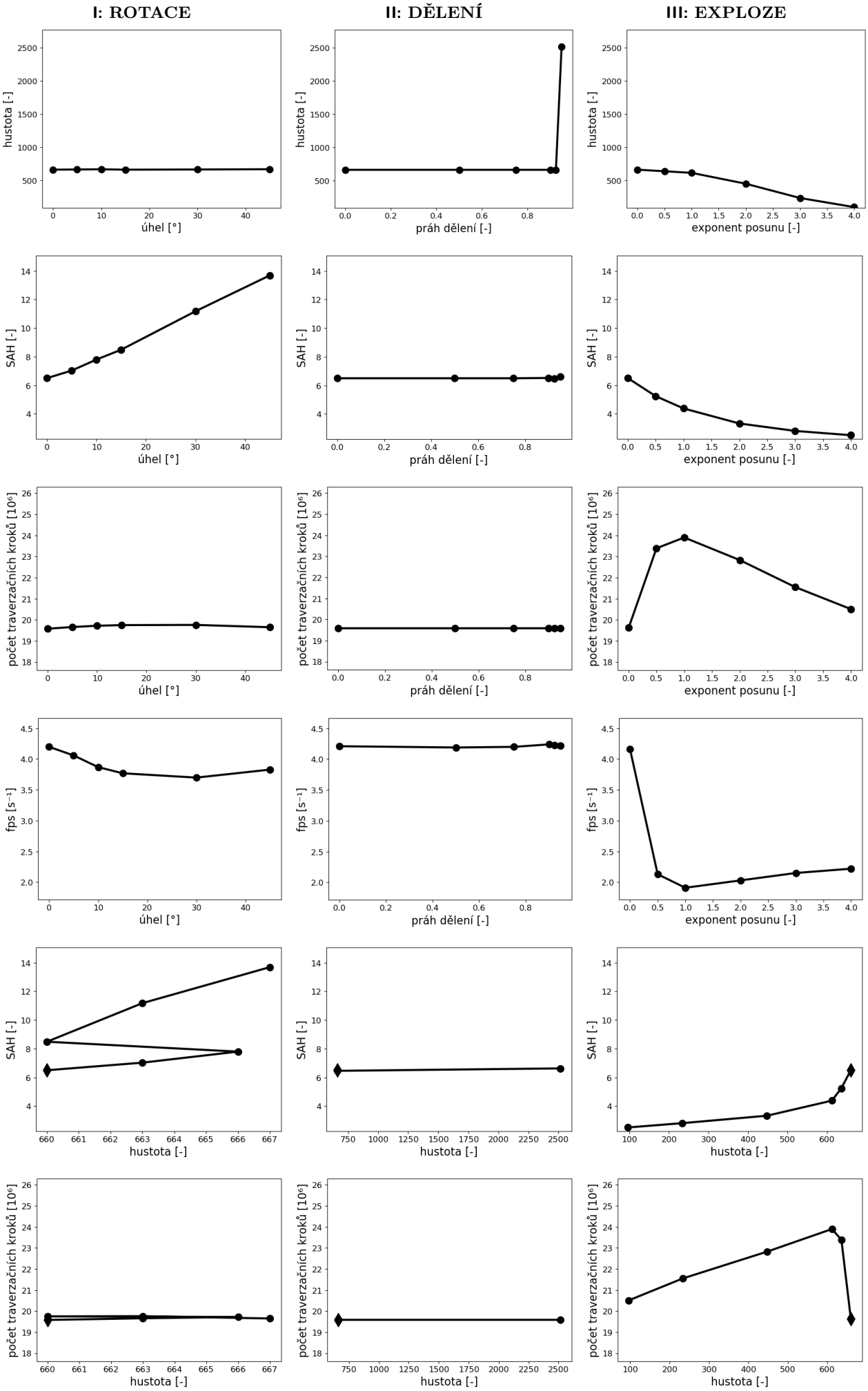
Počet traverzačních kroků téměř ve všech případech neovlivňuje jak rotace, tak dělení. U rotace tvoří výjimku scény sibenik a teapots – to lze být vysvětleno tím, že kamera se nachází uprostřed scény a po jejím otočení se tedy může dostat mnoho objektů mimo její zorné pole. Mnohem zajímavější efekt má ovšem na počet traverzačních kroků exploze. Ta při nízkých hodnotách transformační exponentu působí výrazné zvýšení. To může být vysvětleno tím, že scéna má sice relativně původní tvar, ale detaily všeho jsou velice "rozostřené"(tj. jsou všude malé mezery) a tak se paprsek odráží ve velkém množství. Při zvýšení exponentu exploze se ovšem objekty dostávají dostatečně daleko od sebe, že paprsek má větší šanci vyletět ze scény, a proto počet traverzačních kroků dramaticky klesá.

Hodnota FPS by měla být nepřímo úměrná počtu traverzačních kroků. Tento trend lze velice dobře pozorovat při srovnání grafů těchto veličin na všech scénách.

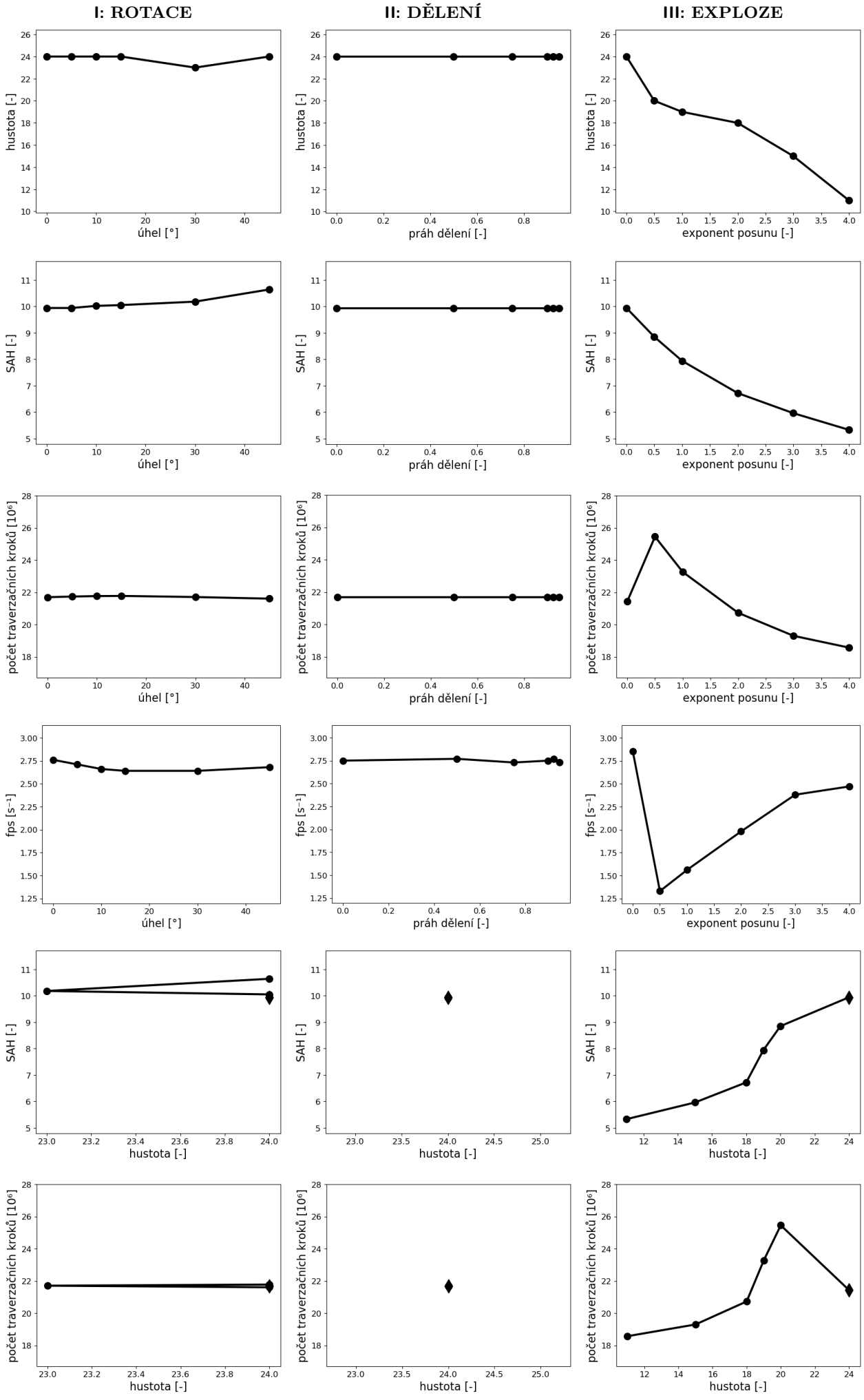
Co se týká vztahu hustoty a SAH, začnu u transformace exploze, jelikož ta má velice zřejmý efekt na všech scénách. Se "sílou"exploze klesá jak hustota scén, tak hodnota SAH (lze tedy vytvořit efektivnější BVH). Náznak tohoto vztahu lze pozorovat i u rotace, jelikož se ovšem hodnoty obou veličin příliš nemění, je to na grafu znázorněné v mnohem menším měřítku, a tedy hůře pozorovatelné. Podobná situace nastává i při dělení. Naštěstí zde existuje několik scén (např. teapots), kde je tento vztah opět vidět ve větším rozsahu.

S klesající hustotou klesá i počet traverzačních kroků při explozi. Jen tam lze opět vidět abnormalitu ohledně počtu kroků při malých explozích. Rotace nemá velký vliv na počet traverzačních kroků (až na speciální případy – viz výše) ani na hustotu scény, takže nám tyto grafy moc informací neposkytují. Stejně jako u SAH, dělení objektů příliš neovlivňuje počet traverzačních kroků. To se ovšem nedá říct o hustotě, jelikož při kritické hodnotě dělení vznikají podobně veliké trojúhelníky, které velmi navyšují hustotu celé scény.

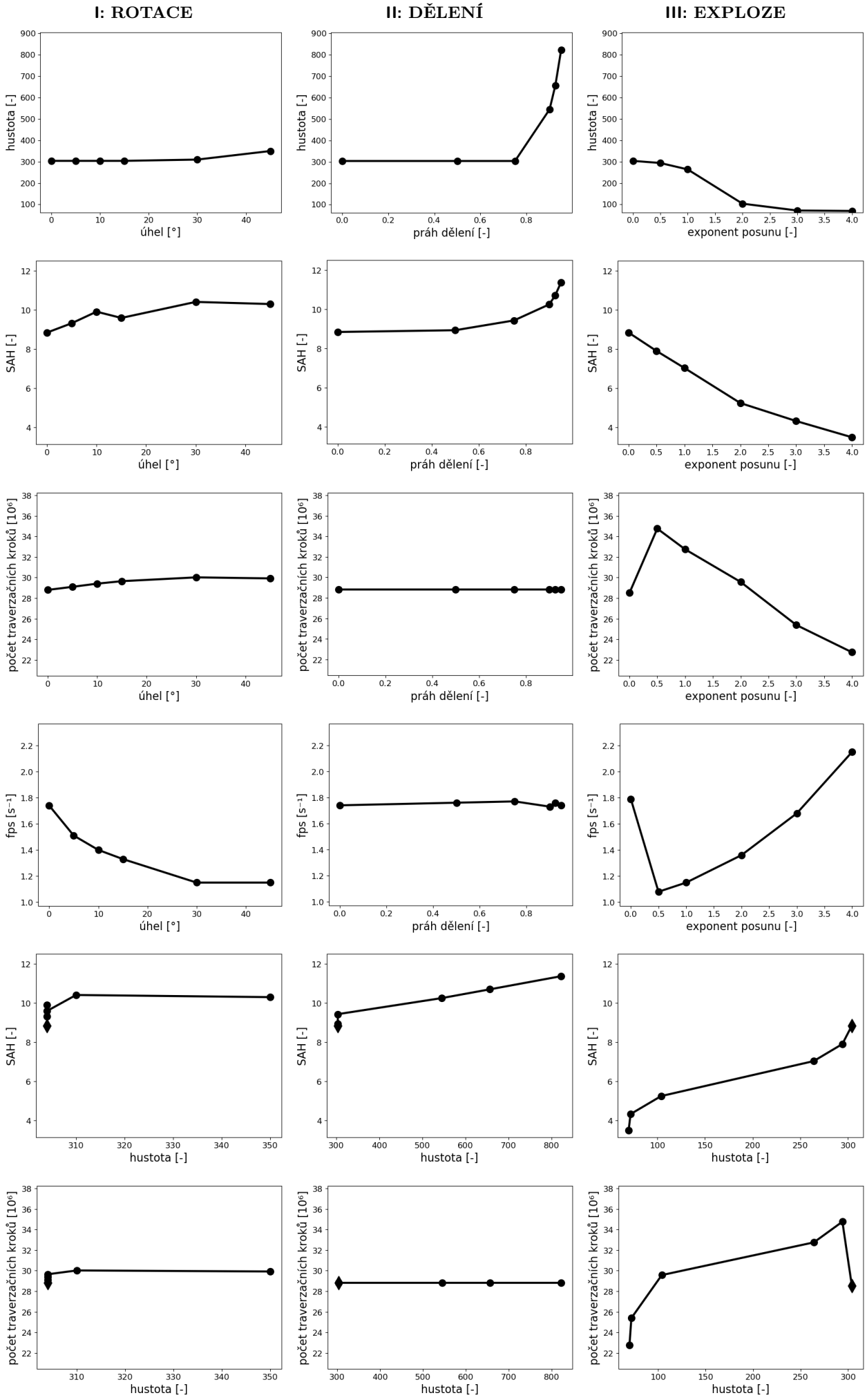
Z vyhodnocení tedy vyplývá, že nejspolehlivější relace existuje mezi metrikou hustoty a hodnotou SAH (určuje kvalitu stavby BVH). To lze pozorovat u všech scén a i všech typů transformací, pouze se mění míra závislosti.



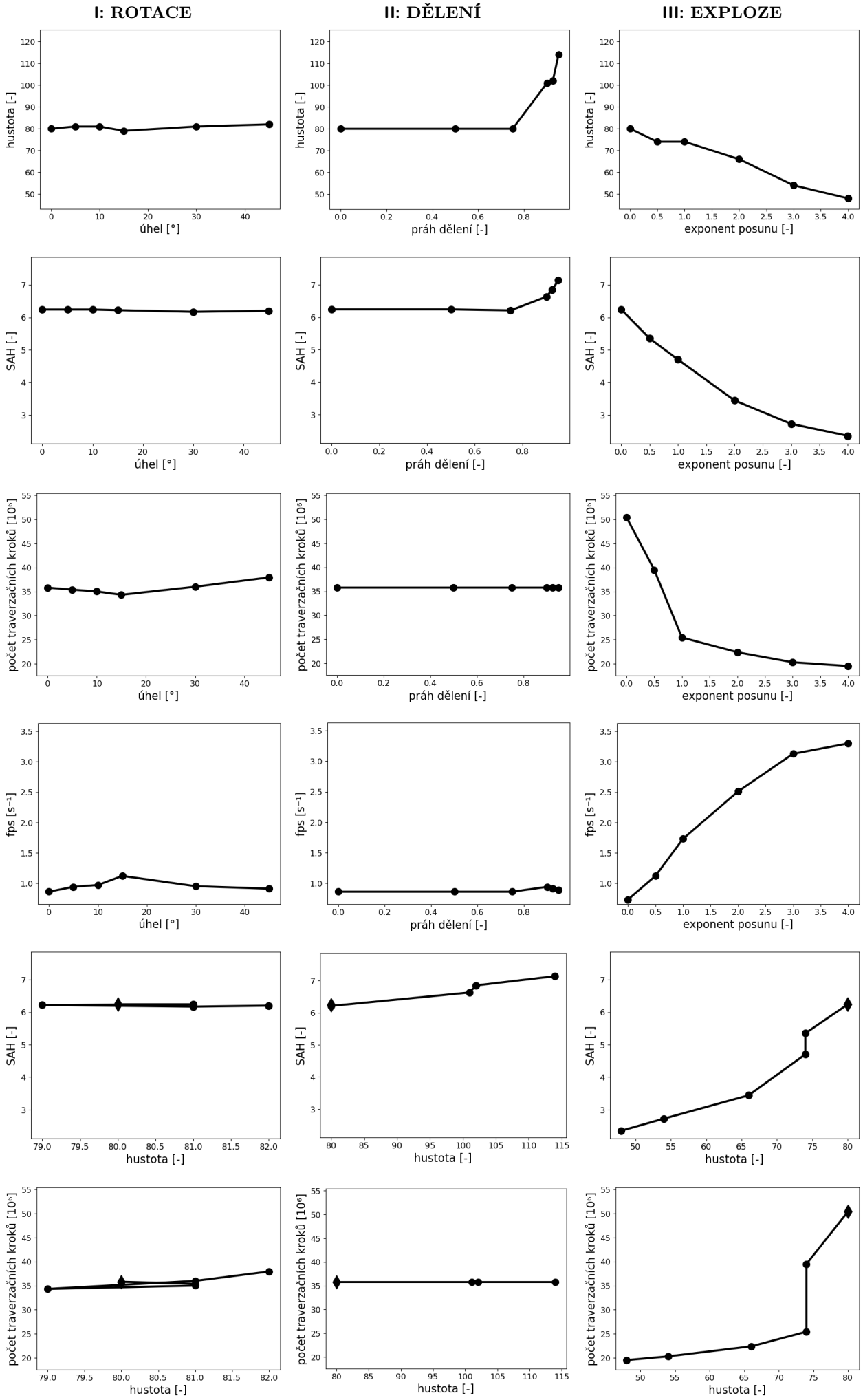
Obrázek 6.2: Naměřené hodnoty pro scénu A10.



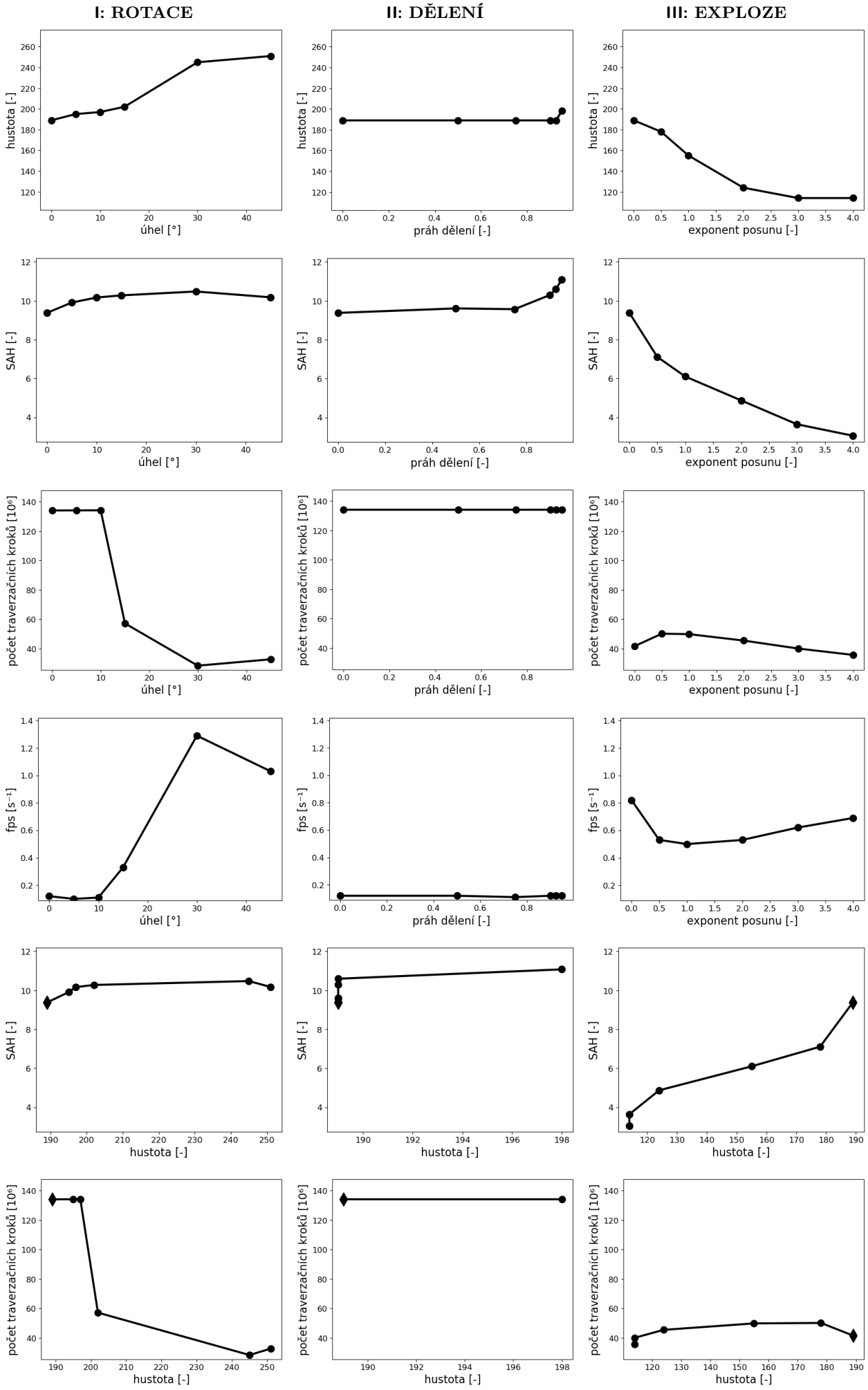
Obrázek 6.3: Naměřené hodnoty pro scénu Armadillo.



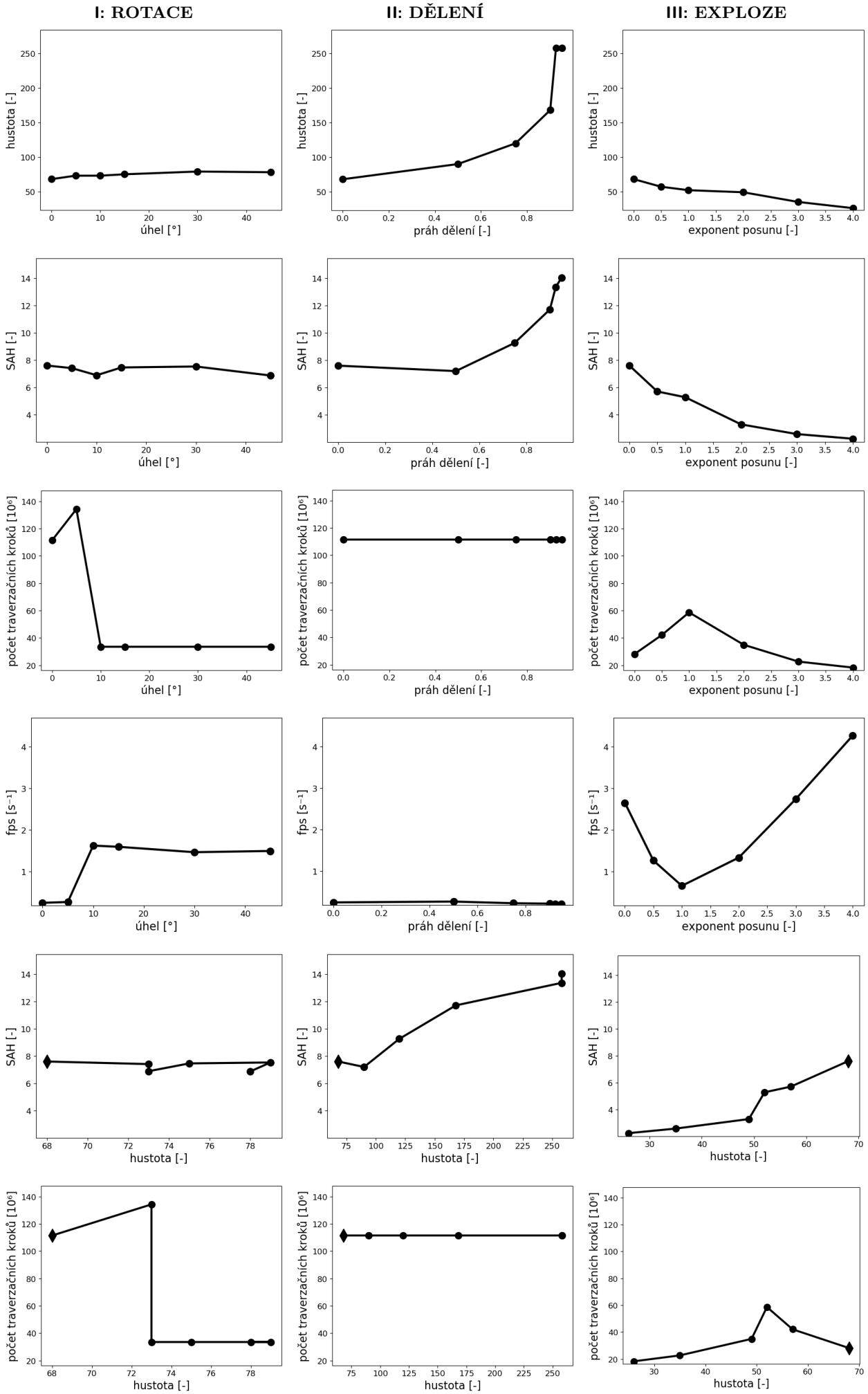
Obrázek 6.4: Naměřené hodnoty pro scénu city.



Obrázek 6.5: Naměřené hodnoty pro scénu park.



Obrázek 6.6: Naměřené hodnoty pro scénu sibenik.



Obrázek 6.7: Naměřené hodnoty pro scénu teapots.

Kapitola 7

Závěr

Cílem práce bylo zavést ve 3D prostoru realistické metriky, které by uživateli poskytly užitečné informace o zkoumané scéně. Úspěšně se mi podařilo dvě takovéto metriky adaptovat pro 3D scény a zároveň na jejich hodnotách umožnit vizuální reprezentaci v interaktivním prostředí. K dispozici jsou samozřejmě i přesně naměřená data, která mohou být potenciálně využita k dalšímu zpracování jako prostředek o údajích scény.

Metriky jsem otestoval na šesti scénách různé složitosti a povahy. Výsledky jsem náležitě vyhodnotil a také jsem zkoumal jejich vztah vzhledem k algoritmům pro konstrukci a traverzaci BVH při renderování scény. Jako prostředek pro diferenciální změny scény jsem napsal program, které podporuje tři transformace. To mi pomohlo tento vztah náležitě prozkoumat, pokud se scéna začne postupně měnit. V testování jsem ukázal, že při zvýšení metriky hustoty se navýší i SAH hodnota postaveného BVH, ačkoliv tento vztah není lineární.

Ve výsledku byl největší důraz kladen na sestavení kompletního výzkumného řetězce. Ten začíná u výpočtu metrik, přes vizualizaci, po zpracování a syntaktickou analýzu dat, až po automatické vytváření grafů na bázi veličin vybraných uživatelem. Tato sada nástrojů je nyní připravena a měla by poskytnout stěžejní oporu pro případné přidávání nových metrik a jejich testování.

Téma práce se ukázalo jako velice bohaté. V průběhu studie jsem narazil na mnoho možných směrů vývoje, které nebyly prozkoumány a určitě stojí za další úsilí. Například algoritmus pro výpočet metriky hustoty, kde jsem se snažil držet teoretického pozadí, ale často dávalo smysl se od původní definice odchýlit. Testování by pak definitivně prokázalo, jestli algoritmus s takovými ústupky poskytuje prokazatelnější výsledky. I samotný vývoj algoritmů pro výpočet metrik byl ztížen faktem, že neexistovala žádná referenční data, takže často se každá část algoritmu musela detailně testovat. To nakonec vedlo

k tomu, že k poslední metrice, nepřehlednosti, jsem se v této práci v kontextu 3D prostoru nevěnoval. Také by stálo za to prozkoumat různé metody stavby a traverzace BVH a prověřit, jestli k nim neexistuje těsnější relace vzhledem k hodnotám metrik.

Literatura

[1] Bounding Volume Hierarchy [online]. [vid. 2019-10-20]. Dostupné z: https://en.wikipedia.org/wiki/Bounding_volume_hierarchy

[2] Branch and Bound [online]. [vid. 2020-08-04]. Dostupné z: https://en.wikipedia.org/wiki/Branch_and_bound

[3] Embree [online]. [vid. 2020-08-05]. Dostupné z: https://www.embree.org

[4] K-d tree [online]. [vid. 2020-08-02]. Dostupné z: https://en.wikipedia.org/wiki/K-d_tree

[5] MeshLab [online]. [vid. 2020-08-05]. Dostupné z: https://www.meshlab.net

[6] OpenGL Mathematics [online]. [vid. 2020-08-05]. Dostupné z: https://glm.g-truc.net

[7] Tiny OBJ Loader [online]. [vid. 2019-10-20]. Dostupné z: https://github.com/tinyobjloader/tinyobjloader

[8] BENTHIN, C.: Embree Ray Tracing Kernels 3.X: Overview and New Features [online]. 2018, presentation. Dostupné z: https://www.embree.org/data/embree-siggraph-2018-final.pdf

[9] de BERG, M., KATZ, M. J., van der STAPPEN, A. F., aj.: Realistic Input Models for Geometric Algorithms. *Algorithmica*, 7 2002: s. 81–97.

[10] CALLIERI, M.: A MeshLab Primer [online]. 2019, presentation. Dostupné z: http://vcg.isti.cnr.it/corsi/G3D_InfoUma/Slides_2020/03_MESHLAB_INTRO.pdf

[11] CAZALS, F., SBERT, M.: Some Integral Geometry Tools to Estimate the Complexity of 3D Scenes. Technická zpráva, INRIA, 1997.

[12] CIGNONI, P., CALLIERI, M., CORSINI, M., aj.: MeshLab: An Open–Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference*, editace V. SCARANO, R. D. CHIARA, U. ERRA, The Eurographics Association, 2008, ISBN 978-3-905673-68-5.

[13] FEIXAS, M., del ACEBO, E., BEKAERT, P., aj.: An Information Theory Framework for the Analysis of Scene Complexity. *Computer Graphics Forum*, 1999.

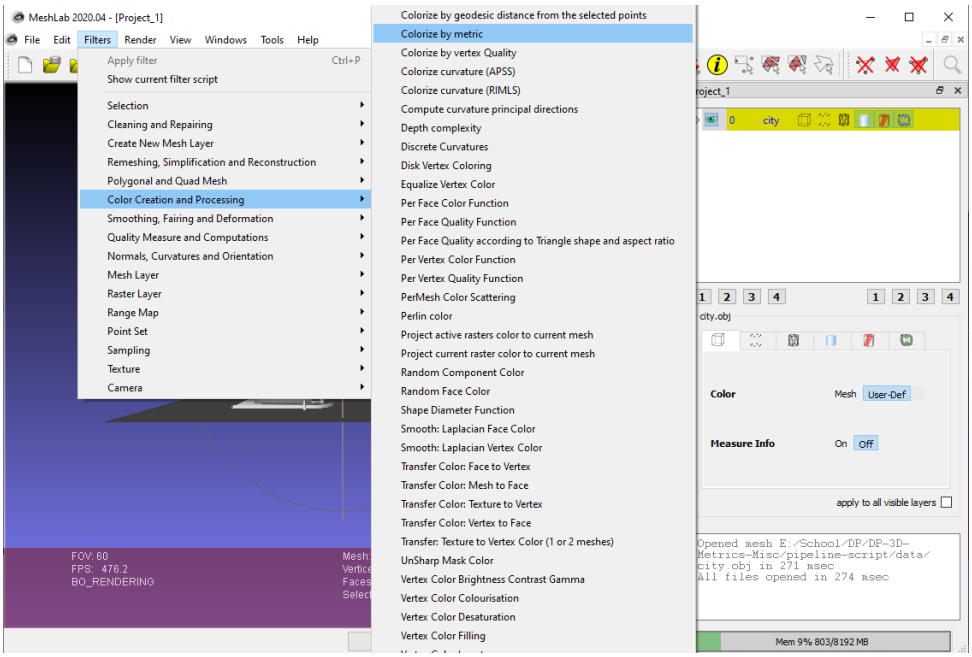
[14] GOLDSMITH, J., SALMON, J.: Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications*, 5 1987: s. 14–20.

[15] GUTTMAN, A.: R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, Association for Computing Machinery, 1984, ISBN 978-0-89791-128-3, s. 47–57.

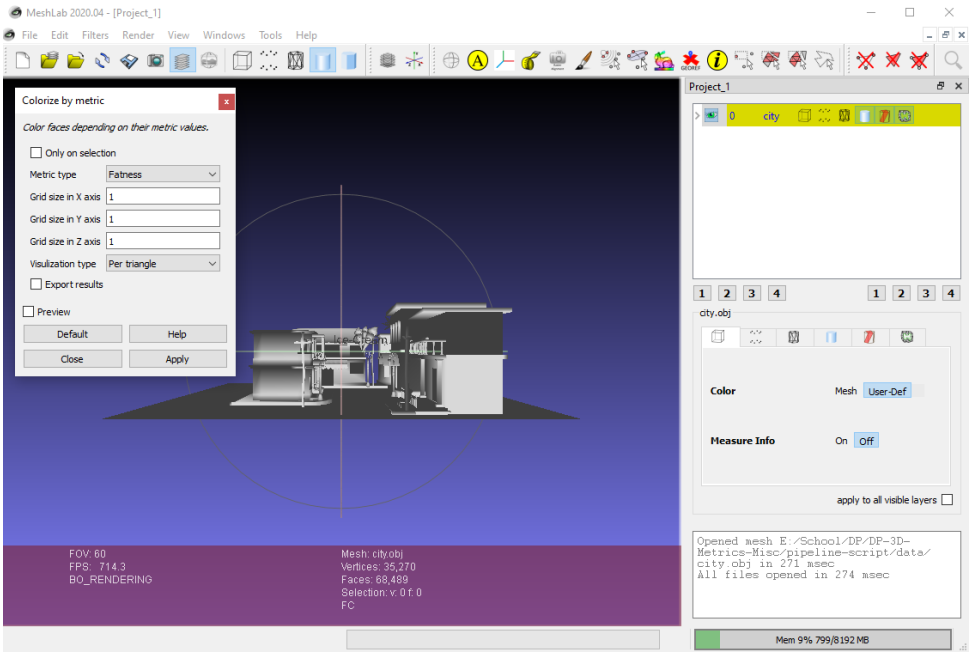
[16] ŽÁRA, J., BENEŠ, B., SOCHOR, J., aj.: *Moderní počítačová grafika*. Computer Press, 2010, ISBN 80-251-0454-0.

Příloha A

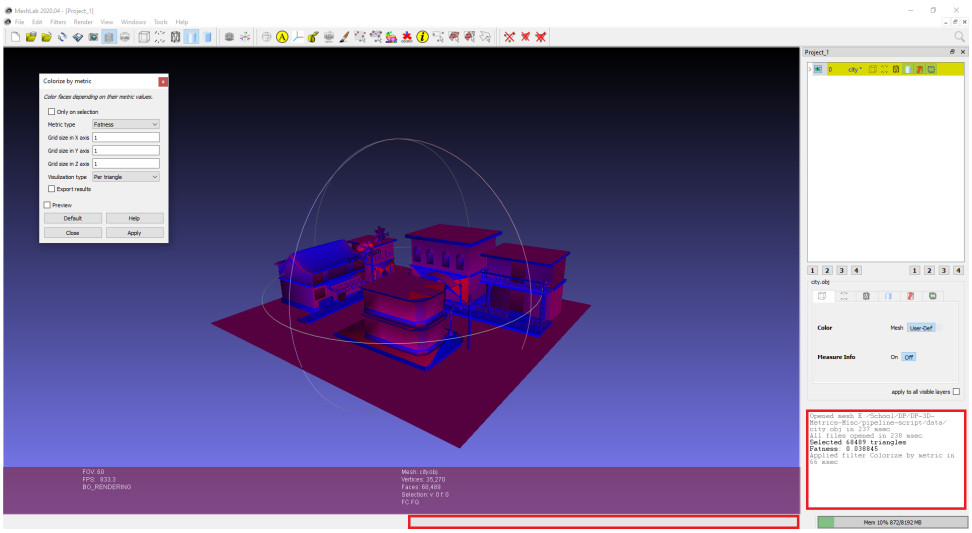
Uživatelský manuál: Aplikace MeshLab filtru



Obrázek A.1: Vybrání filtru pro výpočet metrik.



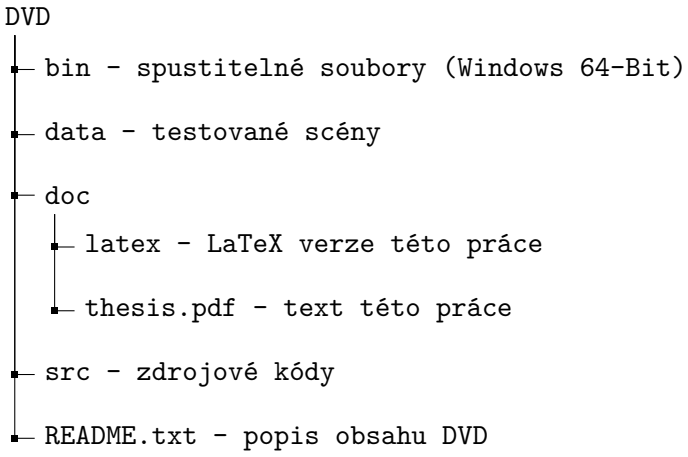
Obrázek A.2: Nastavení parametrů filtru.



Obrázek A.3: Výsledná vizualizovaná scéna. Za povšimnutí stojí ukazatel průběhu výpočtu a okno s informačními záznamy (označeno červenými rámečky).

Příloha B

Obsah přiloženého datového nosiče



Pro vývoj práce jsem využil následující GitLab repozitáře:

- DP-3D-Metrics – Projekt aplikace MeshLab. URL: `https://gitlab.fel.cvut.cz/vlcekvl2/DP-3D-Metrics`
- DP-VCGlib – Projekt knihovny VCG (MeshLab je závislý na této knihovně). URL: `https://gitlab.fel.cvut.cz/vlcekvl2/DP-VCGlib`
- DP-3D-Metrics-Misc – Různorodý obsah (testovací skript, aplikace pro transformaci scén, text DP) . URL: `https://gitlab.fel.cvut.cz/vlcekvl2/dp-3d-metrics-misc`