MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Jaroslav Vozár

Vizualizace pro Ray-tracing

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Josef Pelikán

Studijní program: Informatika

Studijní obor: IPSS

Praha 2019

Týmto by som chcel poďakovať svojmu vedúcemu bakalárskej práce za to, že si vždy našiel čas na konzultácie a bol ochotný prediskutovať a podrobne rozobrať akýkoľvek nápad.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V …..... dne .....

podpis

Název práce: Vizualizace pro Ray-tracing

Autor: Jaroslav Vozár

Katedra / Ústav: Katedra softwaru a výuky informatiky (201. • 32-KSVI)

Vedoucí bakalářské práce: RNDr. Josef Pelikán, Katedra softwaru a výuky informatiky

Abstrakt: Předmětem této práce je rozšíření stávajícího programu na vykreslování obrázků pomocí sledování paprsku (ray-tracing). Jelikož se tento existující program používá při výuce počítačové grafiky na MFF UK, rozšiřující nástroje poskytují právě cestu k jednoduššímu pochopení látky. Tím jsou statistiky, analytické mapy a hlavně možnost sledovat vykreslovací paprsek přímo v 3D scéně. Právě poslední zmíněné rozšíření je zásadní a nejvíce užitečné pro pochopení hlubších principů ray-tracingu.

Kromě usnadnění pochopení principů jednoho z hlavních technik vykreslování v počítačové grafice, tato rozšíření pomohou i při tvorbě domácích úkolů na předměty počítačové grafiky.

Klíčová slova: Ray tracing, GrCis, OpenGL, depth map, normal map

Title: Visualization for Ray-tracing

Author: Jaroslav Vozár

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Josef Pelikán, Department of Software and Computer Science Education

Abstract: The aim of this work is to extend the existing image-rendering program based on ray-tracing. As this existing program is used to teach computer graphics at the Faculty of Mathematics and Physics of the Charles University, expanding tools provide the way to a simpler understanding of the substance. This includes statistics, analytical maps and especially the ability to track the rendering ray directly in the 3D scene. The last mentioned extension is essential and most useful for understanding deeper ray-tracing principles.

In addition to facilitating the understanding of one of the main rendering techniques in computer graphics, these extensions will also help create homework for computer graphics subjects.

Keywords: Ray tracing, GrCis, OpenGL, depth map, normal map

Názov práce: Vizualizácia pre Ray-tracing

Autor: Jaroslav Vozár

Katedra / Ústav: Katedra softwaru a výuky informatiky

Vedúci bakalárskej práce: RNDr. Josef Pelikán, Katedra softwaru a výuky informatiky

Abstrakt: Predmetom tejto práce je rozšírenie existujúceho programu na vykresľovanie obrázkov pomocou sledovania lúča (ray-tracing). Nakoľko sa tento existujúci program používa pri výučbe počítačovej grafiky na MFF UK, rozširujúce nástroje poskytujú práve cestu k jednoduchšiemu pochopeniu látky. Tým sú štatistiky, analytické mapy a hlavne možnosť sledovať vykresľovací lúč priamo v 3D scéne. Práve posledné spomenuté rozšírenie je zásadné a najviac užitočné pre pochopenie hlbších princípov ray-tracingu.

Okrem uľahčenia pochopenia princípov jedného z hlavných techník vykresľovania v počítačovej grafike, tieto rozšírenia pomôžu aj pri tvorbe domácich úloh na predmety počítačovej grafiky.

Kľúčové slová: Ray tracing, GrCis, OpenGL, depth map, normal map

Obsah

1 Úvod

Renderovanie (vykresľovanie) pomocou rekurzívneho sledovania lúču (ray-tracing) sa používa v počítačovej grafike od 80. rokov. Je nenahraditeľnou technikou pri zobrazovaní 3D animácií, ako napríklad animované filmy a seriály. Kvôli jeho vysokej výpočtovej náročnosti sa táto technika nikdy vo veľkom nepoužívala v miestach, kde je potrebné renderovanie v reálnom čase. Avšak vďaka technologickým pokrokom v minulých rokoch sa táto technika pomaly, ale isto začína používať aj v počítačových hrách. To prinesie ďalší a nezadržateľný rozvoj ray-tracingu a metód založených na ňom.

Avšak na jeho optimalizáciu a správnu implementáciu je najprv potrebné túto techniku renderovania plne pochopiť. Na to, okrem iného, slúžia predmety počítačovej grafiky na MFF UK. Knižnica nástrojov GrCis vytvorená práve pre tieto účely poskytuje renderer založený na ray-tracingu s interným označením 048rtmontecarlo-script. Ten však umožňoval len vyrenderovanie scén s rôznymi nastaveniami, ale nijako nezobrazoval vnútorný princíp fungovania. Statické obrázky taktiež nie sú na vysvetlenie tejto problematiky vždy vhodné.

Práve toto je dôvod vzniku tejto práce. Doplnenie existujúceho ray-traceru o interaktívne nástroje umožňujúce jednoduchšie pochopenie tejto renderovacej techniky. Okrem priameho použitia pri výučbe na prednáškach a cvičeniach sú interaktívne nástroje užitočné aj pri samoštúdiu a tvorbe domácich úloh na spomínané predmety.

Táto práca pridáva do existujúceho diela 048rtmontecarlo-script interaktívne okno s 3D zobrazením scény spolu s reprezentáciou šírenia rekurzívneho lúča danou scénou. Okrem toho tiež pridáva aj analytické a štatistické nástroje, ktoré pomôžu s hlbším pochopením problematiky a analýzou vlastných scén alebo iných prídavkov do 048rtmontecarlo-script.

Štruktúra práce pozostáva zo začiatočnej kapitoly **Teória**, ktorá čitateľa oboznámi so základnými pojmami potrebnými pre pochopenie zvyšku práce. Kapitola **Analýza problému** bližšie rozoberá dôvod vzniku, knižnicu GrCis ako aj analýzu podobného existujúceho softwaru tretích strán.

Štvrtá (**Užívateľská dokumentácia**) a piata (**Programátorská dokumentácia**) kapitola sa venujú potrebnému rozboru diela. Najprv

z užívateľského pohľadu – ako sa to má používať. A potom z programátorského pohľadu – ako to vo vnútri funguje.

Za nimi sa nachádza **Záverečná analýza** popisujúca a hlavne odôvodňujúca vybrané postupy a techniky, ktoré boli nakoniec zvolené. Taktiež sa tu nachádza porovnanie so starým systémom renderovania – aké priniesol výhody a výkonnostné zlepšenie. V poslednej kapitole **Záver** je už len zhrnutie celej práce a jej praktický prínos.

2 Teória

V tejto kapitole sú vysvetlené základné princípy zobrazovacích techník, ktoré sú nevyhnutné pre pochopenie fungovania celého diela.

2.1 Renderovanie

Pod pojmom renderovanie je možné si obecne predstaviť zobrazenie 3D scény na 2D plátno, najčastejšie monitor. V istom slova zmysle sa renderovanie môže chápať aj z 2D scény na 2D plátno, ale to je pre túto prácu irelevantné.

3D scéna je reprezentácia objektov v trojrozmernom svete. Môže to byť napríklad popis jednoduchých primitív (kváder, valec, guľa ...), kde špecifikácia scény obsahuje polohy, rozmery, rotácie a farbu takýchto primitív. Prípadne môže obsahovať aj binárne operácie nad týmito primitívami (zjednotenie, rozdiel, prienik ...). Táto technika sa nazýva CSG (Constructive Solid Geometry) [9].

Druhou veľmi používanou technikou popisu 3D scény je trojuholníková sieť. Ide o jednoduchý zoznam trojuholníkov (pozícií ich bodov v priestore), z ktorých sú tvorené všetky objekty. Trojuholníky môžu obsahovať informáciu aj o farbe, materiály, normálovom vektore v danom bode apod..

Úlohou renderovania je zobrazenie práve nejakej takejto scény na výstup, ktorým je napríklad monitor užívateľa. S potrebou zobrazenia 3D dát na 2D výstup sa stretneme najčastejšie pri 3D animáciách (animované filmy a seriály) a počítačových hrách. V zásade sa používajú 2 rôzne techniky renderovania, ktoré sú bližšie vysvetlené v nasledujúcich podkapitolách.

2.1.1 Rasterizácia

Pre zobrazenie 3D scény na 2D výstup stačí použiť jednoduchú lineárnu algebru. Na každý bod scény (v prípade trojuholníkových sietí), resp. na každý objekt (v prípade CSG) sa aplikujú transformačné matice. Tie prevedú súradnice bodu v 3D svetovom priestore do 2D priestoru výstupu.

Táto technika je veľmi rýchla, hlavne vďaka možnosti vysokej paralelizovateľnosti, ktorú využívajú grafické karty. Vďaka tomu je rasterizácia obľúbená hlavne v počítačových hrách, kde je potrebné vykresliť snímky (2D výstupy z 3D scény) v reálnom čase (rádovo desiatky za 1 sekundu). Avšak je

potrebné sa zmieriť s viacerými nedostatkami tejto techniky, ako je nemožnosť tvorby plných odleskov a nižšej kvality tieňov.

Medzi hlavné platformy/knižnice používané pre rasterizáciu patria OpenGL [7] a DirectX [8]. Tie okrem množstva nástrojov na priamu komunikáciu s grafickým procesorom obsahujú aj možnosť práce so shadermi, čo sú malé programy upravujúce vykresľovanie obrázku. Hlavne sa starajú o tieňovanie, ale môžu aj meniť geometriu [13].

2.1.2 Ray-tracing

Na druhej strane, ray-tracing (prekladané ako „metóda sledovania lúčov“) využíva fyzikálny princíp, na akom funguje aj videnie živočíchov. Svetelný fotón vznikne v zdroji svetla a z neho sa šíri scénou, pokiaľ nie je pohltený nejakým objektom alebo neskončí v oku. V ňom sa vyvolá vnem, ktorý putuje do mozgu a to vnímame ako videnie. Nakoľko je množstvo vyslaných fotónov extrémne veľké, je výpočtovo nemožné simulovať priechod scénou každého jedného z nich. Preto ray-tracing využíva fakt, že nám stačí sledovať lúče z cieľa (z kamery) do všetkých svetelných zdrojov. Po ceste lúč zbiera informáciu o farbe objektov, od ktorých sa odrazil, prípadne či bol daný lúč pohltený niektorým objektom.

Táto technika prináša realistický a vysokokvalitný výstup, avšak za cenu náročnejšieho výpočtu. Preto sa ray-tracing používa tam, kde rýchlosť renderovania nie je kritická, ako je napríklad renderovanie 3D animácií a 3D modelov. V takýchto prípadoch sa 3D animácia vyrenderuje a uloží vo forme videa (sekvencii 2D obrázkov).

Ray-tracing sa v súčasnosti začal používať aj pri renderovaní v reálnom čase (počítačové hry), avšak len ako pomocná technika spolu s rasterizáciou.

Pod pojmom *primárny lúč* sa myslí lúč, ktorý vychádza z kamery a ide do scény. Popri tom ray-tracer pracuje aj s *tieňovými lúčmi*. Tie začínajú v každom priesečníku primárneho lúču s nejakým objektom scény a končia v zdroji svetla. Pre každý zdroj svetla sa vyšle z každého priesečníka minimálne 1 tieňový lúč. Tým sa zistí, ktoré svetlá prispievajú k farbe objektu v danom priesečníku.

Pod pojmom ray-tracing sa môže myslieť len jej základná verzia. V takom prípade od nej odvodené (v dnešnej dobe prevažujúce) metódy malú špeciálne označenie. Jedná sa o techniky založené na metóde Monte Carlo, ako je napríklad

path-tracing [21]. V tomto texte sa však ray-tracingom bude myslieť akákoľvek renderovacia technika založená na sledovaní lúča.

Viac informácií o ray-tracingu je k dispozícii napríklad v [10].

2.2 Zobrazenie scény

Niekedy je potrebné z technických alebo implementačných dôvodov reprezentovať 3D scény len symbolicky. Na to sa v tejto práci použijú 2 techniky popísané v nasledujúcich podkapitolách.

2.2.1 Bounding box

Jedná sa o kváder, ktorý tesne obkolesuje nejaký objekt. Tento kváder je zvyčajne rovnobežný so svetovými osami scény.

2.2.2 Point cloud

Pod pojmom point cloud sa rozumie množina bodov v 3D priestore. Každý bod môže niesť informáciu o farbe a prípadne ďalších atribútoch ako normálový vektor.

2.3 Normálový vektor

Normálový vektor je zvyčajne vektor kolmý na priamku alebo rovinu. V počítačovej grafike sa tento vektor modifikuje za účelom realistického zobrazenia plôch.

Namiesto vytvárania zložitej geometrie sa len modifikujú normálové vektory, ktoré ovplyvňujú tieňovanie v danom mieste. Tým sa dá dosiahnuť efekt nerovnej geometrie, aj keď povrch samotný je v skutočnosti rovný. Táto technika je predovšetkým užitočná pre zníženie výpočtovej náročnosti ako aj náročnosti tvorby 3D scén umelcami.

3 Analýza problému

V tejto kapitole je opísaný dôvod vzniku a základný popis práce.

3.1 Dôvod vzniku

Primárne existujú 2 dôvody pre vznik tohto diela:

- výučba a samoštúdium princípov ray-tracingu
- ladenie a hľadanie chýb vo vlastných 3D scénach.

Ray-tracing, ako dôležitá renderovacia metóda (resp. základ pre renderovacie techniky odvodené od nej) sa vyučuje aj na predmetoch počítačovej grafiky na MFF UK. K tomu už dlhšie slúži knižnica GrCis (viď 3.2), ktorá obsahuje, okrem iného, aj program na renderovanie 3D scén (zadaných v CSG) pomocou ray-tracingu. Toto dielo nadväzuje na knižnicu GrCis a rozširuje ju, nakoľko v nej chýbajú akékoľvek vizualizačné nástroje na ladenie a demonštráciu.

3.2 Knižnica GrCis

GrCis je knižnica základných algoritmov a dátových štruktúr, ktoré pomáhajú pri výučbe predmetov so zameraním na počítačovú grafiku na MFF UK. Celý popis, ako aj odkaz na repozitár s návodom je dostupný na webovej stránke venovanej tejto knižnici [1].

3.3 Základný popis

Úlohou tohto diela je rozšírenie funkcionality projektu 048rtmontecarlo-script (a tým pádom aj 048rtmontecarlo, ktorý je veľmi podobný) z knižnice GrCis. Tento projekt obsahuje nástroj na renderovanie 3D scén pomocou ray-tracingu. Okrem iného umožňuje rôzne nastavenie ray-tracingu, ako napríklad počet vyslaných lúčov na pixel, odrazy, zalamovanie lúčov atď.

Primárnou úlohou Vizualizácie pro Ray-tracing je pridanie podpory pre zobrazenie prechodu lúču scénou – **RayVisualizer**.

Avšak, okrem toho pridáva aj ďalšie analytické nástroje, ktoré pomôžu pri výučbe ray-tracingu ako aj pri tvorbe vlastných scén a doplnkov. Práve rôzne doplnky do 048rtmontecarlo-script sú častým zadaním domácich úloh z predmetov počítačovej grafiky. Preto bolo potrebné dbať na vysokú modularitu a rozšíriteľnosť.

Medzi doplňujúce súčasti diela pre spomínaný ray-tracer patria:

- **AdditionalViews** – zobrazovač takzvaných máp, čiže obrázkov podobných hlavnému výstupu, ale nesúcich rozdielnu informáciu. Hlavný výstup obsahuje v každom pixely informáciu o farbe lúča, ktorý sa do neho dostal. Farba lúča záleží na zdroji svetla a farbe objektov, od ktorých sa lúč odrazil alebo v ktorých sa zalomil pri prechode scénou. Mapy obsahujú napríklad informáciu o hĺbke v danom pixeli, t.j. vzdialenosti prvého objektu, s ktorým sa lúč pri vyslaní z kamery preťal.
- Distribuované renderovanie po sieti pomocou **RenderClient**. Toto rozšírenie umožňuje renderovanie rovnakej scény na viacerých strojoch. Ray-tracing je časovo veľmi náročná metóda a vyrenderovanie jednej scény môže trvať aj niekoľko desiatok minút alebo hodín. Avšak ray-tracing je možné veľmi dobre paralelizovať. Vďaka tomu je možné spojiť veľké množstvo výpočtových strojov a renderovať rovnakú scénu naraz. Všetko, čo je k tomu potrebné, je prepojenie počítačov sieťou (nemusia byť v lokálnej sieti) a spustenie jednoduchého konzolového klienta – *Render Client*.

3.4 Existujúci software

Existuje celý rad programov na renderovanie 3D scén pomocou ray-tracingu, ako napríklad **Autodesk Maya**, **3dsMax** a **Blender**. Pokročilejšie z nich dokážu do istej miery vytvoriť aj výstup podobný mapám. Avšak sú veľakrát zložité na používanie a nevhodné na výučbu princípov ray-tracingu.

Pri výučbe najviac pomôže vizualizácia lúču samotného. Na toto existuje napríklad projekt **rtVTK** [2] (Ray Tracing Visualization Toolkit), ktorý je však v súčasnosti bez podpory a bez možnosti si ho stiahnuť vo forme spustiteľného súboru. Ďalej existuje projekt **Path Graph** [3]. Ten je však do značnej miery jednoduchý a umožňuje zobrazenie šírenia lúčov len v primitívnych scénach a to v 2D pri pohľade zhora. Taktiež nepodporuje objekty rôznych materiálov a tým pádom rozdielnu mieru odrazivosti a zalamovania vnútri objektov. To má oproti tomuto dielu zásadné a neprehliadnuteľné obmedzenia.

4 Užívateľská dokumentácia

Užívateľská dokumentácia sa stručne zaoberá pôvodných ray-tracerom *048rtmontecarlo-script*, ale hlavne poskytuje vysvetlivky a návod, ako pracovať s rozšíreniami tohoto ray-traceru, ktoré sú predmetom diela.

4.1 Základné informácie o ray-tracery

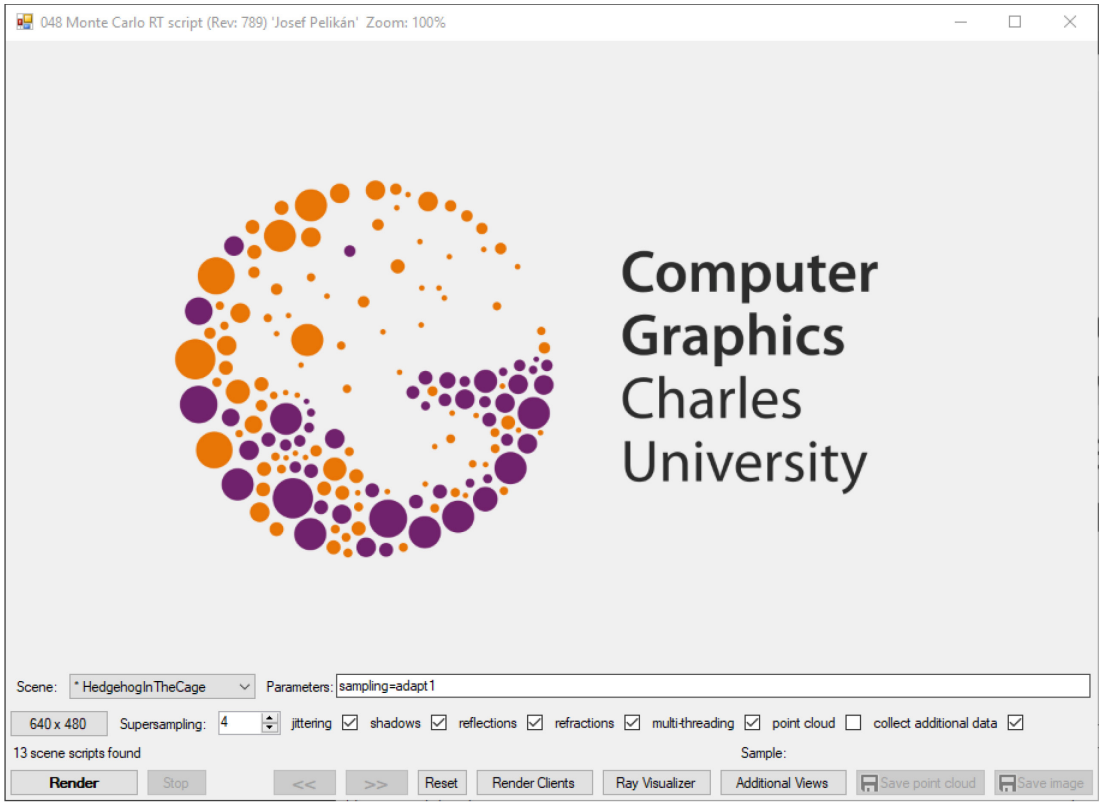
Základný ray-tracer *048rtmontecarlo-script* má veľmi intuitívne ovládanie. Scéna sa zvolí zo zoznamu. Tá môže byť aj vo forme C# script súboru, v priečinku **/data/rtscenes**. Následne sa zvolí úroveň renderovania - supersampling (maximálny počet vrhnutých primárnych lúčov na pixel) a prípadne ďalšie doplňujúce nastavenia pomocou zaškrtávacích políčok (checkboxov).

Rozširujúce nástroje vytvorené v rámci tejto práce sú prístupné cez tlačidlá **Ray Visualizer**, **Additional Views** a **Render Clients** (viď Obrázok 1).

4.1.1 Parametre renderovania

- Rozlíšenie – tlačidlo s aktuálnym rozlíšením (predvolene 640 x 480 na obrázku) umožňuje otvoriť malé okno s nastavením rozlíšenia, teda počtu pixelov na šírku a na výšku výsledného vyrenderovaného obrázku
- Supersampling – udáva maximálny počet lúčov vrhnutých na 1 pixel (skutočný počet môže byť menší, ak je zapnutý adaptívny supersampling)
- Parameters – jednoduchý príkazový riadok, ktorý v základe umožňuje zapnúť adaptívny supersampling (príkazom *„sampling=adapt1“*, resp. *„sampling=adapt0“* pre jeho vypnutie).
- jittering – pri zapnutom jitteringu sa lúče v rámci 1 pixelu vrhajú nerovnomerne (pri rovnomernom vrhaní lúčov, tzv. uniform sampling, sa môžu objaviť obrazové artefakty)
- shadows – zapne renderovanie tieňov
- reflections – zapne renderovanie odrazov
- refractions – zapne zalamovanie lúčov vnútri objektov
- multi-threading – povolí použiť viacero vlákien na výpočet
- point cloud – povolí zbieranie dát pre point-cloud (toto umožní zobrazenie point cloudu v *RayVisualizery* ako aj jeho uloženie)

- collect additional data – povolí zbieranie dát pre *AdditionalViews* (mapy a štatistiky)



Obrázok 1 - Základné užívateľské rozhranie

4.1.2 Vplyv na výkon

Je nutné počítať s tým, že zber dát pomocou checkboxov „point cloud“ a „collect additional data“ majú nepriaznivý vplyv na dobu renderovania. Kým zber dát pre point cloud spomalí renderovanie netriviálne (okolo 25%), zber dát pre *AdditionalView* má na rýchlosť renderovania zanedbateľný vplyv (pod 5%). Pre podrobnejšie výsledky viď podkapitolu 6.10 v záverečnej analýze.

4.2 AdditionalViews

AdditionalViews je súprava máp a štatistických nástrojov pre ray-tracing. Pojmom **mapa** sa myslí bitmapa reprezentujúca aktuálnu scénu v inom štýle, ako je štandardný vyrenderovaný obrázok. Príkladom je napríklad hĺbková mapa farebne reprezentujúca hĺbku scény v danom pixeli.

Štatistické nástroje slúžia na získavanie presnejších údajom z jednotlivých máp, ako napr. priemerný počet poslaných primárnych lúčov na pixel.

Na to, aby mapy a štatistiky v *AdditionalViews* fungovali, treba ešte pred renderovaním označiť checkbox *„collect additional data“* (predvolene označené). Bez toho sa nedá vytvoriť žiadna mapa. Po kompletnom vyrenderovaní sa na každej záložke automaticky vyrenderuje príslušná mapa. Toto môže trvať až niekoľko sekúnd (pri veľkom množstve zozbieraných dát ako napríklad pri veľkom supersamplingu a vysokom rozlíšení). Pri bežnom používaní to však trvá len zlomok sekundy.

Po kliknutí myšou na mapu sa na ľavom boku zobrazia informácie o konkrétnom pixeli a prípadne aj obecné štatistiky. Pri držaní tlačidla *Ctrl* a tlačenom ľavom tlačidle myši je možné kurzorom hýbať bez posunutia obrázku samotného. To je užitočné na skúmanie kontinuálnej zmeny hodnôt. Zobrazia sa jednak súradnice daného pixelu a jednak hodnota mapy v danom bode. Napríklad pre hĺbkovú mapu je to hĺbka scény v danom bode.

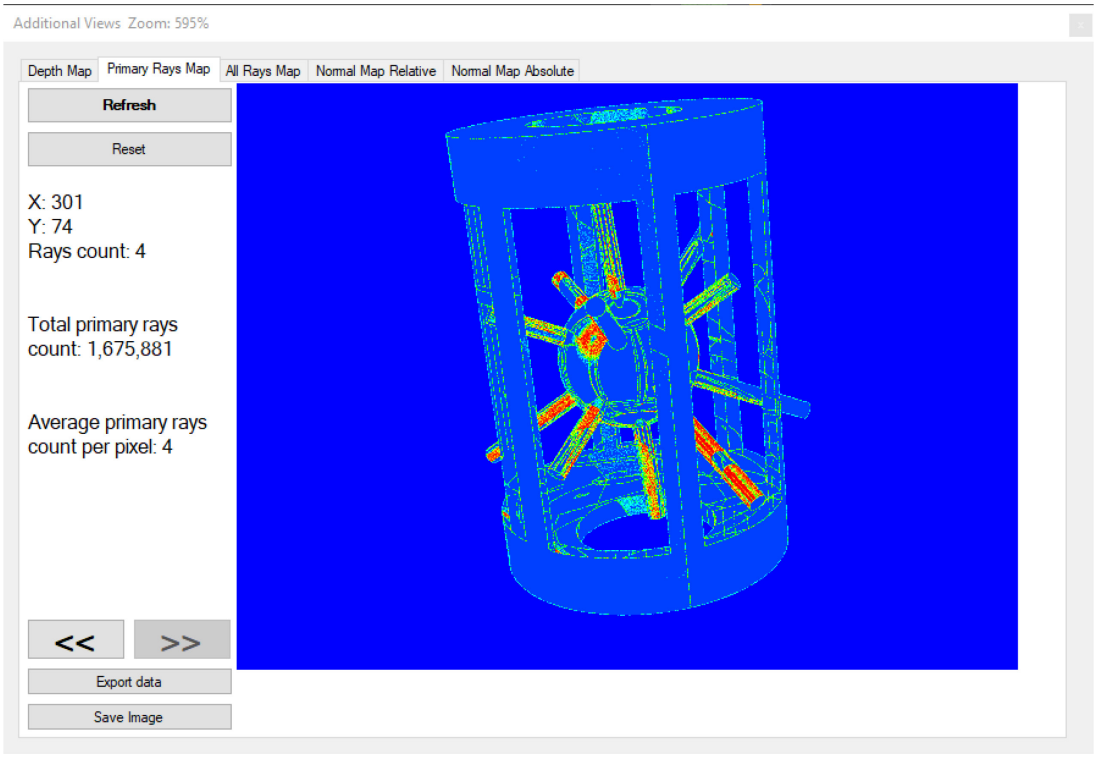
Každú mapu je takisto možné uložiť ako obrázok vo formáte PNG priamo na disk, jednoducho tlačidlom **Save Image** (v ľavom dolnom rohu).

Nad týmto tlačidlom sa nachádza ďalšie veľmi užitočné tlačidlo, a to **Export Data**. To umožní vyexportovať dáta aktuálne zvolenej mapy do formátu CSV (Comma Separated Values) [14] a použiť ich na dodatočnú analýzu. Tento formát je podporovaný prakticky všetkými tabuľkovými editormi. Jeho čítanie je takisto jednoduché na implementáciu do vlastného programu. Pre čo najväčšiu kompatibilitu je použitým oddeľovačom bodkočiarka (**;**).

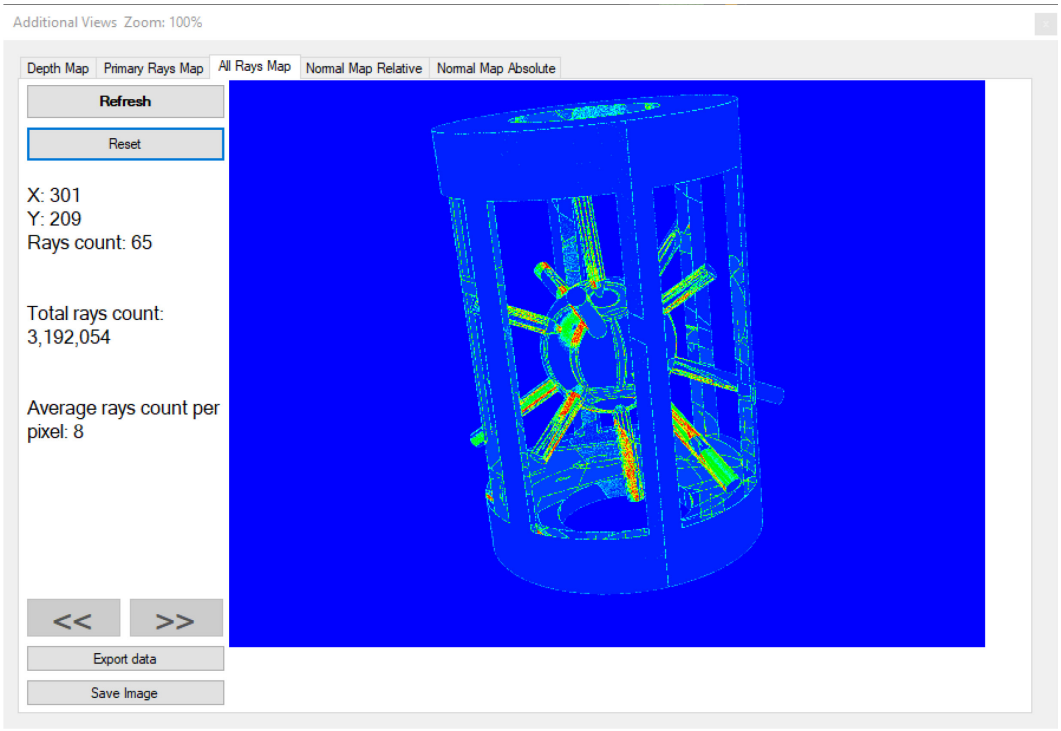
4.2.1 RaysMap

Jedná sa o najjednoduchší typ mapy, ktorý len zbiera počet lúčov vyslaných na daný pixel. Buď primárnych (*PrimaryRaysMap*) – vyslaných z kamery. Alebo všetkých (*AllRaysMap*) – do tohto počtu sa rátajú aj odrazené lúče, tieňové lúče a všetky ostatné druhy. *PrimaryRaysMap* má zmysel len pri adaptívnom supersamplingu (príkaz *„sampling=adapt1“* v príkazovom riadku *Parameters*), čiže keď sa do rozdielnych pixelov posiela rozdielne množstvo lúčov. Inak by *PrimaryRaysMap* v každom pixeli zobrazoval rovnaký počet poslaných lúčov. Lúče, ktoré nepretnú scénu v žiadnom bode sa do *RaysMap* nezarátavajú.

Farba bitmapy je založená na farebnej škále od tmavo modrej (najmenej lúčov), cez zelenú a žltú, po červenú (najviac lúčov). To priamo vidieť na Obrázok 2 a Obrázok 3.



Obrázok 2 - Primary Rays Map

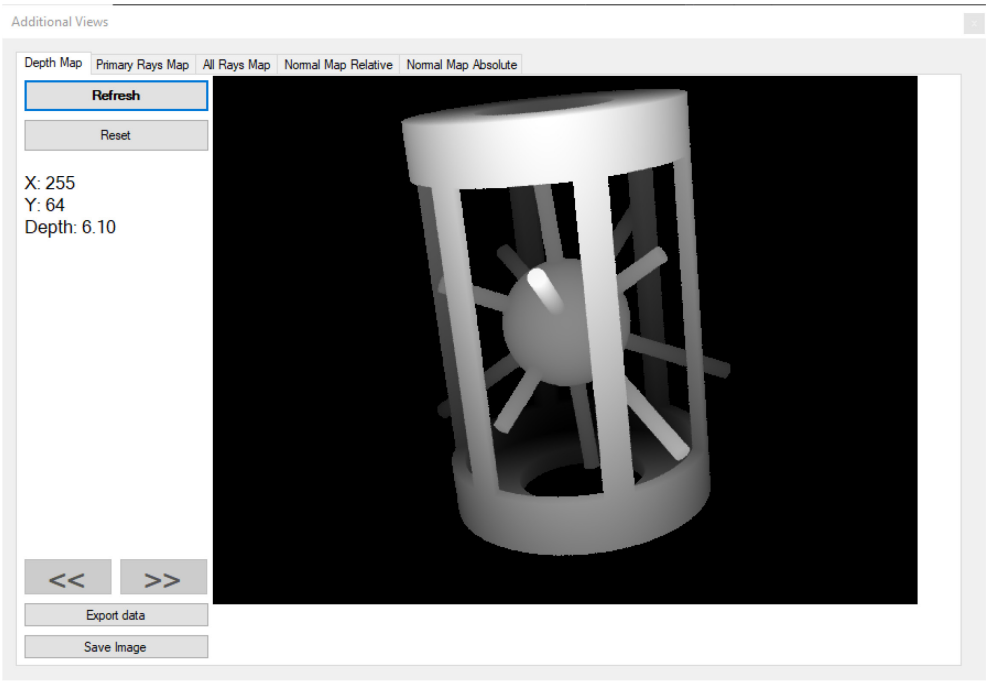


Obrázok 3 - All Rays Map

4.2.2 DepthMap

Táto mapa udáva hĺbku scény v danom pixeli. Lepšie povedané, vzdialenosť prvého priesečníka primárneho lúču so scénou. V prípade viacerých lúčov na pixel, je táto hodnota spriemerovaná. Treba upozorniť, že nie každý lúč pretne scénu. V takom prípade je hĺbka scény *„nekonečno“*.

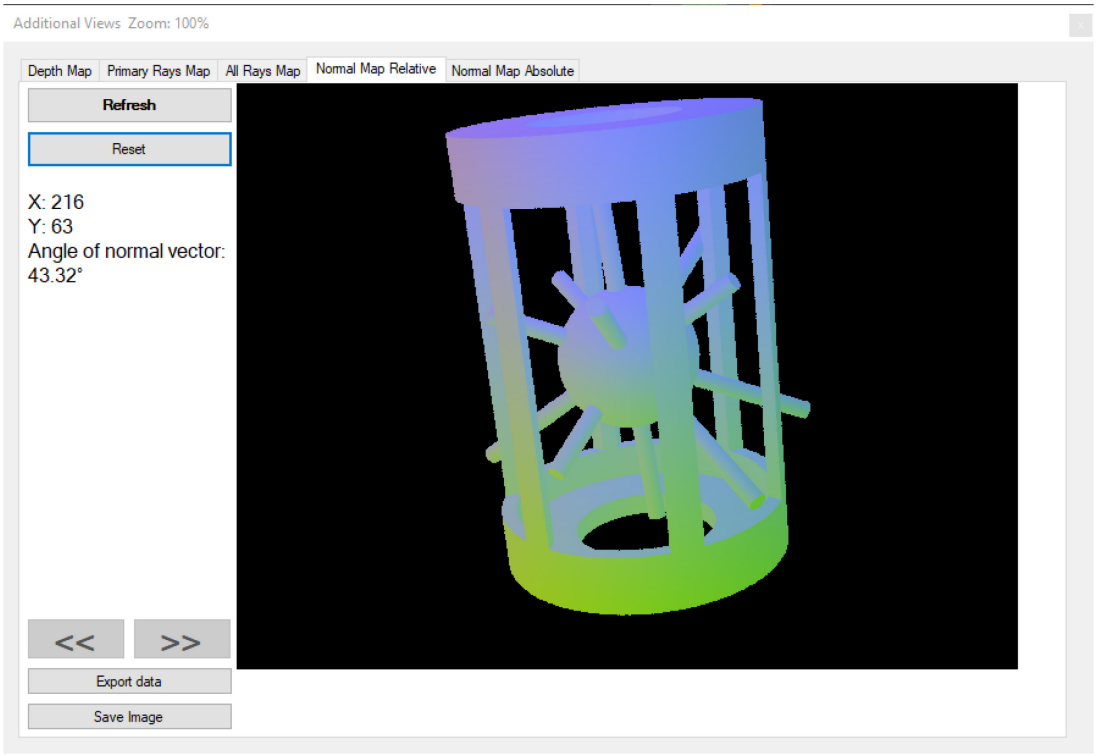
Farba bitmapy je založená na logaritmickej farebnej škále od čisto bielej (blízko) po čisto čiernu (ďaleko) tak, ako vidieť na Obrázok 4.



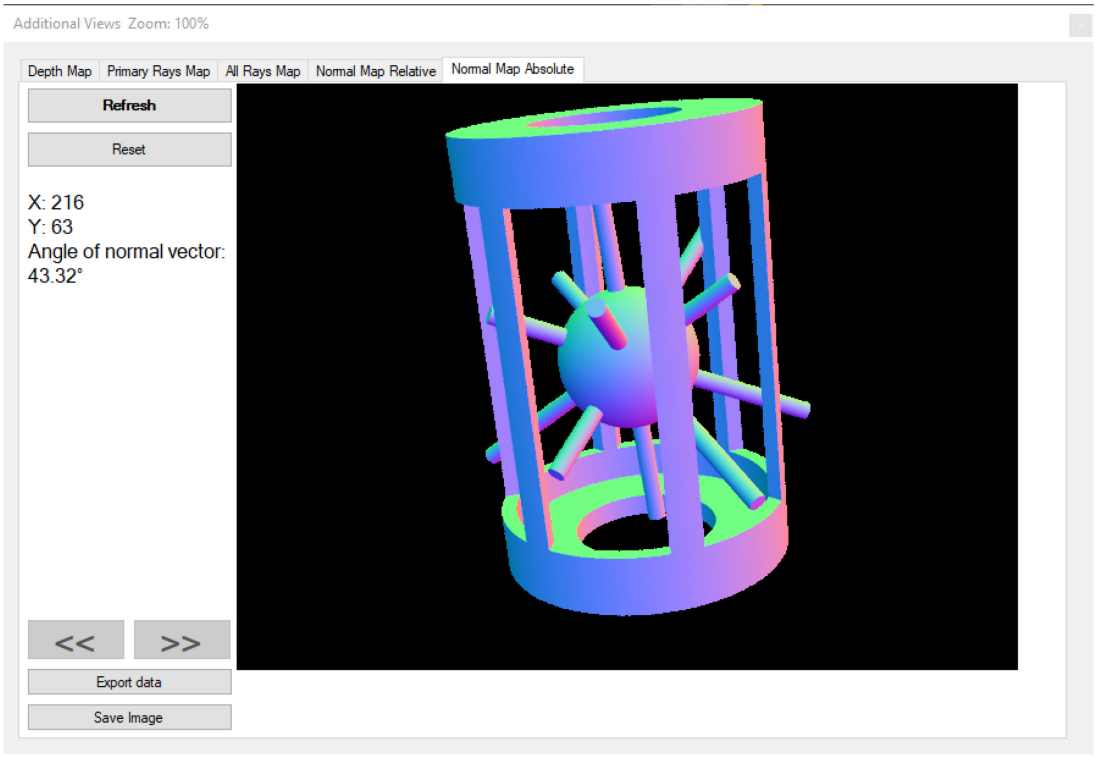
Obrázok 4 - Depth Map

4.2.3 NormalMap

Normálová mapa uchováva informáciu o normálovom vektore v prvom priesečníku primárneho lúču (vyslaného do daného pixelu) so scénou. Existujú 2 verzie – *NormalMapAbsolute* (viď Obrázok 6) a *NormalMapRelative (viď* Obrázok 5*)*, líšiace sa tým, či sa za normálový vektor považuje skutočný vektor na povrchu telesa alebo tento vektor relatívne k pohľadu kamery.



Obrázok 5 - Normal Map Relative



Obrázok 6 - Normal Map Absolute

4.2.4 Štatistiky

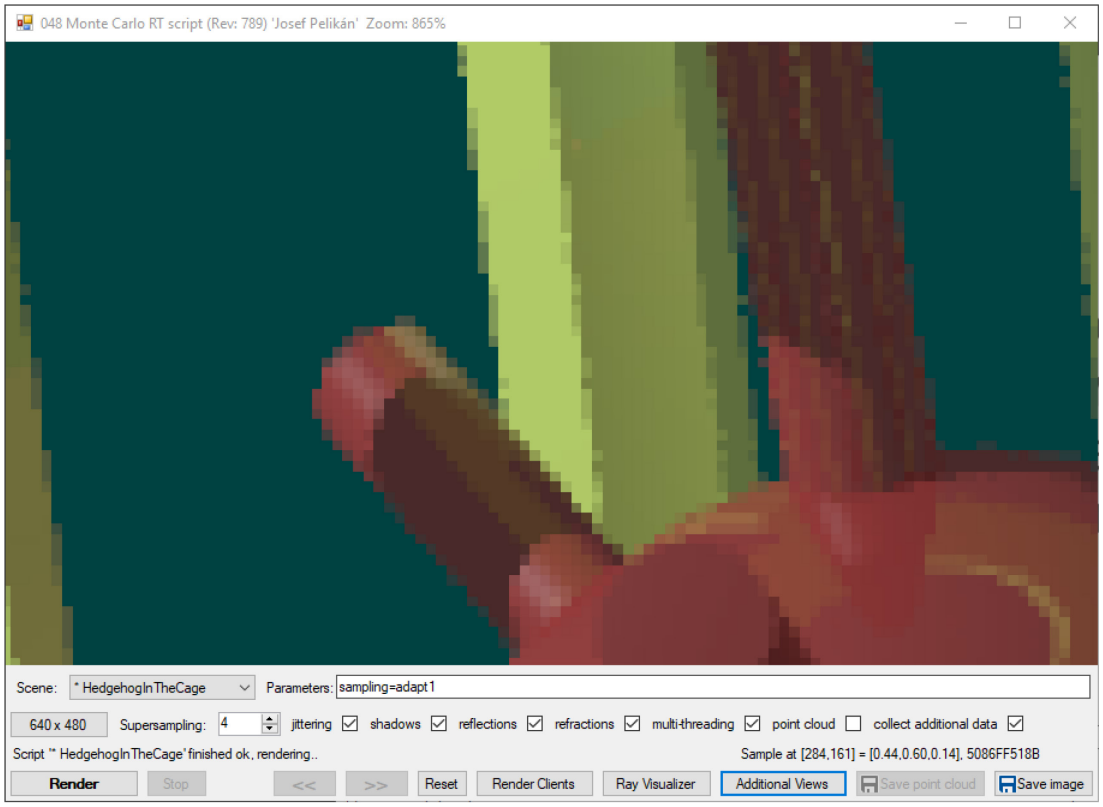
Štatistiky na ľavom boku zobrazujú obecné informácie, ako napríklad počet všetkých primárnych lúčov a ich priemerný počet na pixel. Alebo môžu tiež

zobrazovať informácie ku konkrétnym mapám a ku konkrétnemu zvolenému pixelu mapy. Bližšie informácie sú pri popise jednotlivých máp v predchádzajúcich podkapitolách.

4.3 Pohyb a priblíženie obrázku

Hlavný vyrenderovaný obrázok ako aj mapy umožňujú posun pomocou myši (pohyb myšou pri stlačenom ľavom tlačidle) ako aj priblíženie a oddialenie (kolieskom myši).

Pri priblížení sa zachovávajú skutočné hodnoty pixelov bez akéhokoľvek vyhladzovania, aby sa nestratila informácia v daných pixeloch (viď Obrázok 7).



Obrázok 7 - Priblíženie a posun obrázku

Pre zmenu priblíženia a posunutia na pôvodné hodnoty stačí stlačiť tlačidlo **Reset**. Posúvanie a približovanie je možné aj počas renderovania.

4.4 História obrázkov

Hlavný *Form* (prvé okno, ktoré sa otvorí; obsahuje hlavný vyrenderovaný obrázok) a tiež *AdditionalViews* obsahujú tlačidlá so značkami šípok „<<“ a „>>“. Tie slúžia na prehliadanie histórie obrázkov. Vždy keď je vyrenderovaný nový

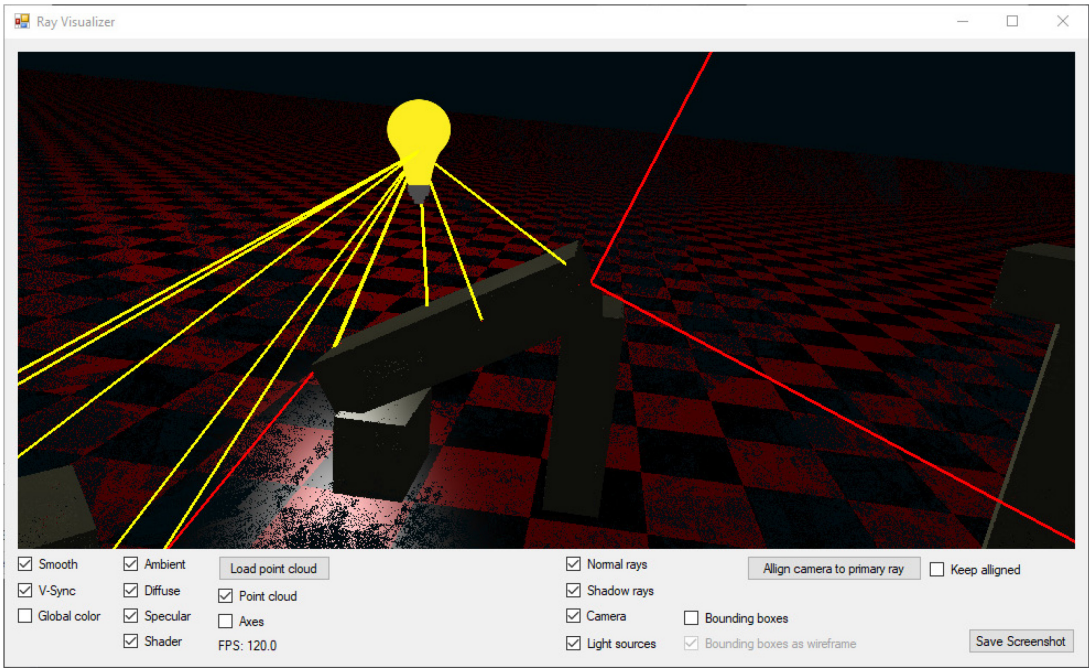
hlavný obrázok, uloží sa do histórie. To isté platí pre mapy. Avšak mapy sú vyrenderované, až keď sa prejde na konkrétnu záložku s nimi. Čiže pri prechode v histórii niektoré mapy nemusia byť k dispozícii.

Vďaka histórii je možné porovnávať rovnakú scénu s rozdielnymi nastaveniami renderovania.

Kvôli pamäťovým nárokom je možné do histórie uložiť 5 variácií obrázku naraz (5 variácií v histórii pre hlavný vyrenderovaný obrázok a 5 pre každú mapu zvlášť). Pri vyrenderovaní nového obrázku a prekročení limitu sa najstarší obrázok z histórie vymaže.

4.5 RayVisualizer

RayVisualizer je plne interaktívne OpenGL okno, ktoré zobrazuje jednoduchú reprezentáciu scény (pomocou bounding boxov alebo point cloudu) a šírenie lúču v nej.



Obrázok 8 - RayVisualizer

4.5.1 Princíp fungovania

Toto okno sa otvára priamo tlačidlom **Ray Visualizer** z hlavného okna ray-traceru. Otvorenie tohto okna po prvýkrát trvá približne 1-3 sekundy. Ďalšie otvorenia sú radovo rýchlejšie. Hneď po začiatku renderovania sa v tomto okne zobrazí jednoduchá reprezentácia scény a je možné:

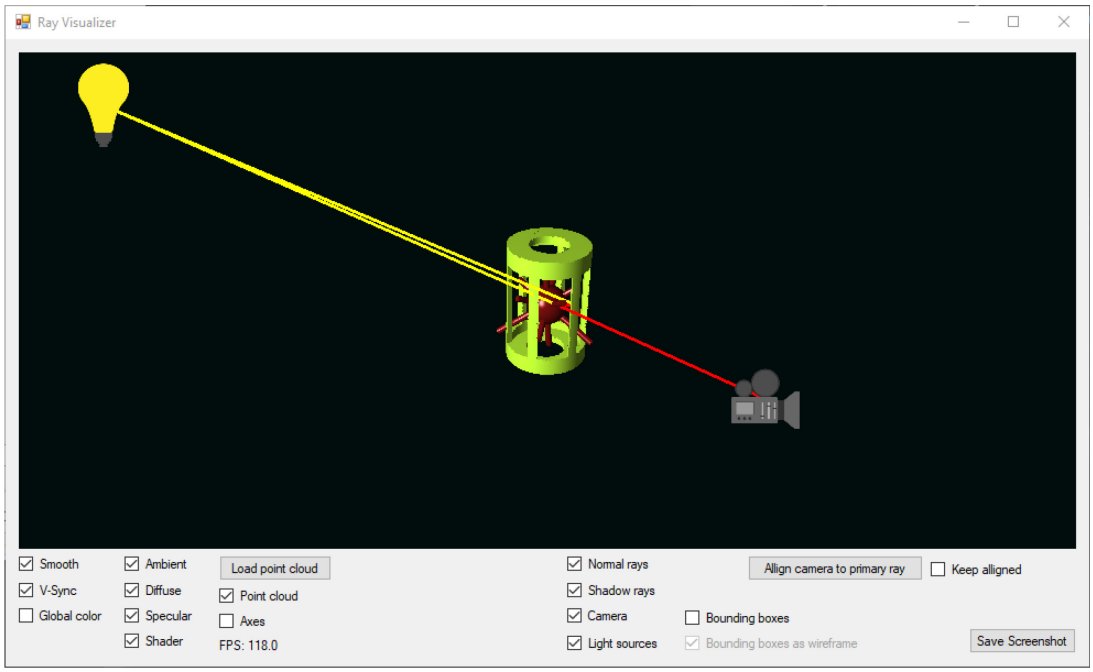
- otáčať kamerou – pohybom myši pri stlačenom ľavom tlačidle myši,
- približovať kameru – kolieskom myši,
- pohybovať sa v scéne – pomocou kláves **W**, **A**, **S**, **D** pre pohyb do strán a kláves **Q**, **E** pre pohyb hore a dole. Tlačidlo **R** slúži na návrat späť na pôvodnú pozíciu kamery.

Po kliknutí na vyrenderovaný obrázok v hlavnom ray-tracery sa v *RayVisualizer*-y objaví reprezentácia vrhnutého lúča. Červeným sú zobrazené normálne renderovacie lúče. Žlté lúče sú tieňovacie (od bodu pretnutia scény k všetkým svetelným zdrojom). Ďalej vidieť v scéne 2D obrázky kamery (ide o začiatok primárneho lúča, nakoľko kamera ako taká nemusí mať pozíciu) a žiaroviek, ktoré reprezentujú svetelné zdroje (viď Obrázok 9).

Na spodnej lište je viacero nastavení. z nich sú najdôležitejšie tie v pravej časti. Jedná sa o možnosť zobraziť/skryť jednotlivé druhy lúčov, reprezentáciu kamery a svetelných zdrojov, prípadne reprezentáciu samotnej scény.

Tlačidlom **Align camera to primary ray** sa pohľad kamery presunie tak, aby bol primárny lúč kolmo na obrazovku, čiže presne vidieť, ako bol daný lúč vyslaný. Toto je predovšetkým užitočné v kombinácii s checkboxom *„Keep aligned“*. Vďaka tomu sa aj po zmene zobrazeného lúča (znovu kliknutím na vyrenderovaný obrázok), kamera posunie tak, aby bola kolmo na primárny lúč. Užitočné je to nie len pri kliknutí na vyrenderovaný obrázok, ale hlavne pri kliknutí, podržaní *Ctrl* a ťahaní kurzoru po danom vyrenderovanom obrázku.

V pravom dolnom rohu sa nachádza tlačidlo **Save Screenshot**, ktoré umožňuje rýchlo a jednoducho uložiť snímok, ktoré je práve viditeľný v *RayVisualizery*. Okrem toho sa na dolnom panely nachádzajú aj checkboxy na zapínanie/vypínanie rôznych nastavení renderovania, ako je napríklad difúzne a okolité osvetlenie alebo vertikálna synchronizácia (V-sync).

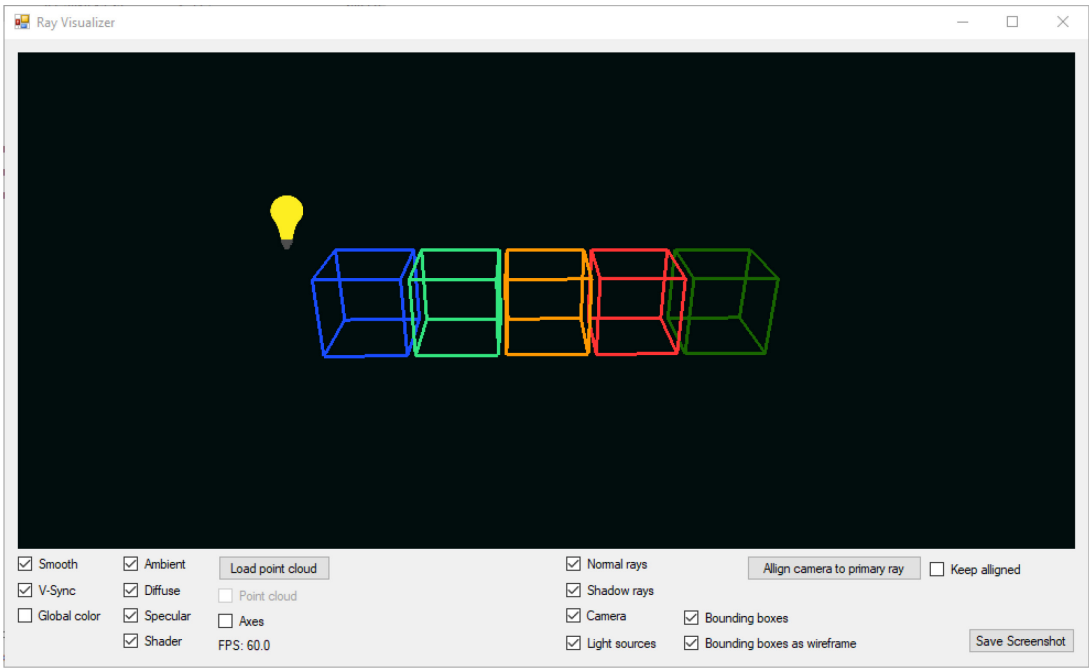


Obrázok 9 - RayVisualizer (kamera a svetlá)

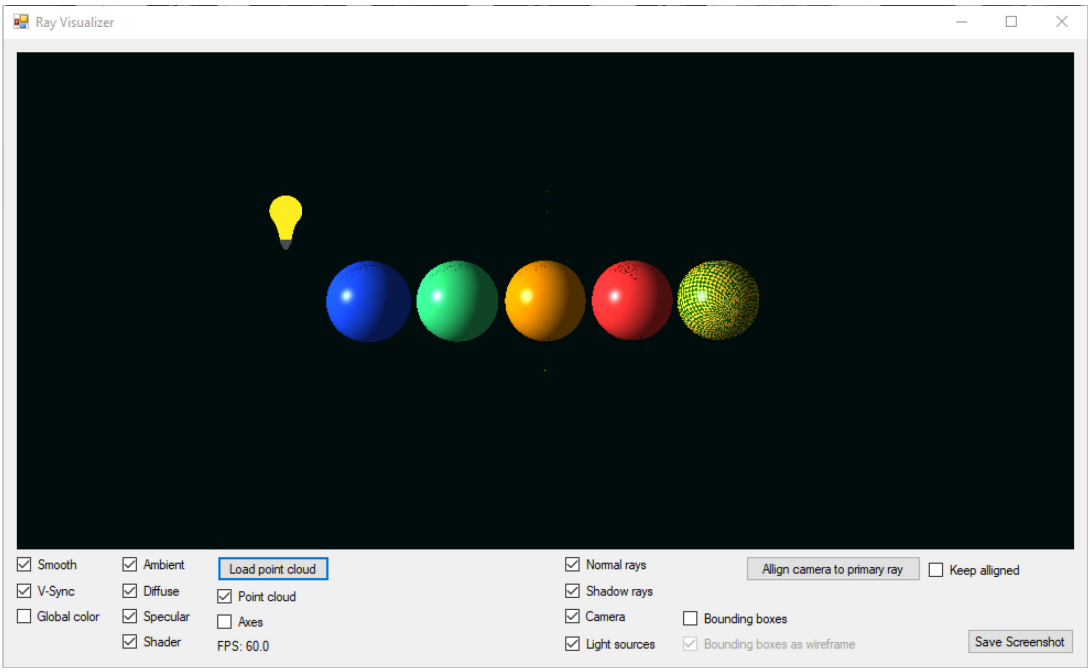
4.5.2 Vizualizácia scény

Na začiatku sa zobrazí vizualizácia scény pomocou bounding boxov (viď 2.2.1). Nakoľko objekt je vždy vo vnútri (alebo maximálne na hrane) svojho bounding boxu, skutočný priesečník lúču s objektom je taktiež vo vnútri (alebo na hrane) bounding boxu. Farba bounding boxov korešponduje farbe objektu, ku ktorému patria (viď Obrázok 10). Avšak pri viacfarebnej textúre je táto informácia nepresná.

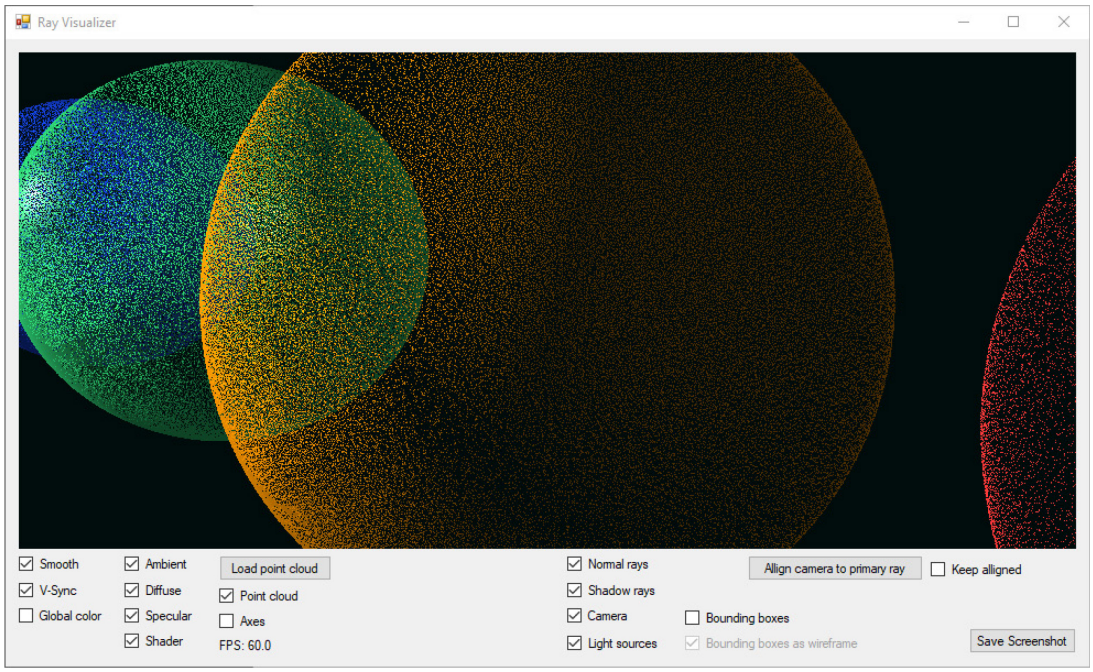
Ak bol zapnutý zber dát pre point cloud (checkbox *„point cloud“*), sprístupní sa tlačidlo **Load point cloud**. To zobrazí scénu pomocou point cloudu namiesto bounding boxov (viď Obrázok 11). Toto zobrazenie je oveľa vernejšie, ale vyžaduje už spomínaný zber dát, s ktorým je spojené priemerne 25% spomalenie renderovania (viď 6.10.1). Taktiež point cloud neobsahuje dáta o objektoch, ku ktorým sa nedostali lúče. Tieto objekty sú aj však z pohľadu *RayVisualizeru* irelevantné.



Obrázok 10 - Bounding boxy



Obrázok 11 - Point cloud (vzdialené)



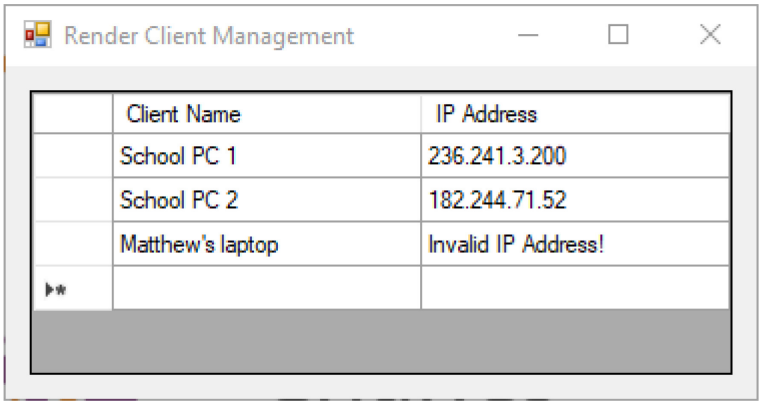
Obrázok 12 - Point cloud (priblížené)

4.6 RenderClient

4.6.1 Základné informácie

RenderClient je samostatná konzolová aplikácia, ktorá umožňuje hlavnému ray-traceru distribuované renderovanie na viacerých strojoch po sieti. Nie je s ním však možné použiť mapy a ani point cloud (viď 6.10.3).

4.6.2 Nadviazanie spojenia

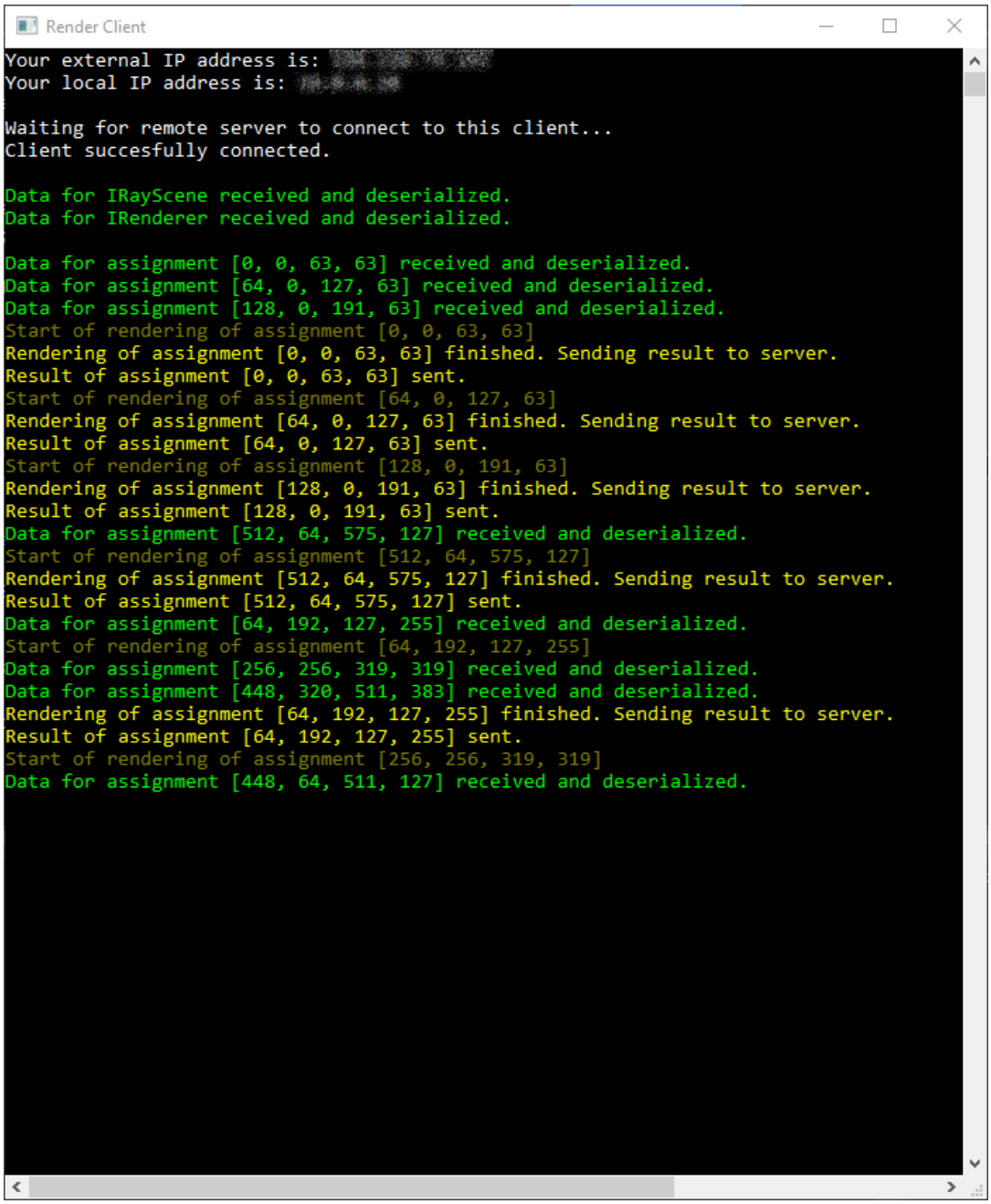


Obrázok 13 - Render Client Management

Na naviazanie spojenia stačí spustiť hlavný ray-tracer na jednom stroji a *RenderClient* na druhom (resp. na ľubovoľnom počte iných strojov). Následne sa v hlavnom ray-tracery otvorí **Render Client Management** pomocou tlačidla

Render Clients. Do neho sa vpíše názov klienta (čisto informatívny charakter; môže byť prázdne a viac klientov môže mať rovnaký názov) a jeho IP adresa (viď Obrázok 13).

Po začatí renderovania sa ray-tracer automaticky pokúsi nadviazať spojenie so vzdialeným *RenderClientom*. Všetko potrebné info sa zobrazí práve v konzolovom okne (viď Obrázok 14). Po skončení renderovania je možné stlačiť *Esc* a tým *RenderClient* ukončiť. Ak ostane otvorený, zostane pripravený na ďalšiu várku renderovania kedykoľvek ho o to hlavný ray-tracer požiada.



Obrázok 14 - Render Client

5 Programátorská dokumentácia

Táto dokumentácia sa zameriava len na danú knižnicu nástrojov (***RayVisualizer***, ***AdditionalViews*** a distribuované renderovanie po sieti pomocou ***RenderClient***). Nezameriava sa na existujúci ray-tracer *048rtmontecarlo-script*, ktorý je použitý na ukážku možností knižnice samotnej a praktické využitie pri výučbe predmetov na MFF UK.

Celá práca je napísaná v jazyku C#.

5.1 AdditionalViews

AdditionalViews je súprava máp a štatistických nástrojov pre ray-tracing. Pojmom mapa sa myslí bitmapa reprezentujúca aktuálnu scénu v inom štýle (s inou informáciou v pixeloch), ako je štandardný vyrenderovaný obrázok. Príkladom je napríklad hĺbková mapa farebne reprezentujúca hĺbku scény v danom pixeli.

Štatistické nástroje slúžia na získavanie presnejších údajom z jednotlivých máp, ako napr. priemerný počet poslaných primárnych lúčov na pixel.

Všetko potrebné k tento sekcii je obsiahnuté v súboroch *AdditionalViews.cs* a *AdditionalViewsForm.cs*.

5.1.1 Mapy vo všeobecnosti

Všetky mapy sú v triede *AdditionalViews* a sú odvodené od *Map<T>*. Tá následne implementuje interface (rozhranie) *IMap*.

Základný princíp fungovania je, že v metóde *Initialize* sa vytvoria nové mapy pre všetky typy (*RaysMap*, *DepthMap*, *NormalMap*, ...). Následne ray-tracer volá metódu *Register* (pomocou triedy *MainRayRegisterer*) na každý vrhnutý lúč s príslušnými parametrami. Táto metóda lúč zaregistruje – zapíše jeho hĺbku do hĺbkovej mapy, normálový vektor jeho priesečníku so scénou do normálovej mapy atď.

Najdôležitejšia časť *Map<T>* je dvojrozmerné pole *mapArray*. Jeho indexy odpovedajú pixelom hlavnej vyrenderovanej bitmapy. Toto pole uchováva dáta typu *T,* ktoré sa tam priradia práve z metódy *Register*. Kvôli použitiu viacerých lúčov na pixel je nutné každú mapu najprv spriemerovať. O to sa stará metóda *AverageMap*.

Tá na základe údajov z *primaryRaysMap*, ktorá uchováva počet primárnych lúčov, spriemeruje hodnoty v *mapArray* podľa metódy *DivideArray* (nutné kvôli typu *T*).

Nie všetky typy máp treba priemerovať (napr. *RaysMap*). Následne sa zavolá metóda *RenderMap*, ktorá z dát v *mapArray* vytvorí bitmapu *mapBitmap*. Tá sa použije v *AdvancedToolsForm*. Tvorba bitmapy je založená na metóde *GetAppropriateColor*. Na základe dát z *mapArray* sa rozhodne, akú farbu prideliť danému pixelu. Napríklad *DepthMap* používa logaritmickú škálu od bielej po čiernu farbu. Alebo *RaysMap* používajú lineárnu škálu farieb od modrej po červenú. Na to však treba vedieť maximálnu a minimálnu hodnotu v danej scéne. Tie sa nastavia pomocou *SetMinimumAndMaximum*, jednoduchým prechodom cez *mapArray*.

5.1.2 RaysMap

Dáta v *mapArray* sú typu *int*.

Jedná sa o najjednoduchší typ mapy, ktorý len uchováva počet lúčov vyslaných na daný pixel. Buď primárnych (*PrimaryRaysMap*) – vyslaných z kamery. Alebo všetkých (*AllRaysMap*) – do tohto počtu sa rátajú aj odrazené, zalomené, tieňové a všetky ostatné druhy lúčov. Rozhodnutie, či sa lúč zaráta len do *PrimaryRaysMap* záleží na hodnote premennej *level* v metóde *Register*, ktorá udáva rekurzívnu hĺbku odrazu lúču. Jedine primárne lúče majú *level* rovný hodnote 0.

Farba bitmapy je založená na farebnej škále od tmavo modrej (najmenej lúčov), cez zelenú a žltú, po červenú (najviac lúčov).

5.1.3 DepthMap

Dáta v *mapArray* sú typu *double* (môže byť aj *PositiveInfinity*).

Táto mapa udáva hĺbku scény v danom pixeli. Presnejšie povedané, vzdialenosť prvého priesečníku primárneho lúča so scénou. V prípade viacerých lúčov na pixel, je táto hodnota spriemerovaná. Treba upozorniť, že nie každý lúč pretne scénu. V takom prípade je hĺbka scény *PositiveInfinity*.

Farba bitmapy je založená na logaritmickej farebnej škále od čisto bielej (blízko) po čisto čiernu (ďaleko).

5.1.4 NormalMap

Dáta v *mapArray* sú typu *Vector3d*.

Normálová mapa uchováva informáciu o normálovom vektore z prvého priesečníku primárneho lúča (vyslaného do daného pixelu) so scénou. v prípade viacerých lúčov na pixel sa normálové vektory spriemerujú. Existujú 2 verzie – *AbsoluteNormalMap* a *RelativeNormalMap*, líšiace sa tým, či sa za normálový vektor považuje skutočný vektor na povrchu telesa alebo tento vektor relatívne k pohľadu kamery. Rozdiel medzi týmito 2 verziami normálových máp je naimplementovaný pomocou rozdielnej metódy *GetAppropriateColor*, vyberanej z delegáta *AppropriateColor*.

Farby výslednej bitmapy sú založené na konvencii:

X:	-1 až +1	Červená:	0 až 255
Y:	-1 až +1	Zelená:	0 až 255
Z:	0 až -1	Modrá:	128 až 255

kde X, Y a z sú súradnice spriemerovaného a normalizovaného normálového vektoru v danom pixeli (v danom mieste v *mapArray*).

5.1.5 Štatistiky

Dáta štatistík pochádzajú z 2 miest. Buď z triedy *Statistics* – celkový počet primárnych a všetkých vrhnutých lúčov. Alebo priamo z jednotlivých máp. Interface *IMap* má metódu *GetValueAtCoordinates*, ktorá vracia hodnotu v danom pixeli. Napríklad hĺbku alebo uhol normálového vektora.

5.1.6 ExportData

Pre možnosti dodatočnej externej analýzy (napríklad tvorba grafov v tabuľkových editoroch) obsahujú mapy jednoduchú metódu *ExportData* na exportovanie ich vnútorných dát. Táto metóda (pomocou *ExportMapData* pre asynchrónne spracovanie) uloží do súboru na disku spriemerované dáta vybranej mapy. Súbor samotný je vo formáte CSV [14], čiže jednoduchý zoznam hodnôt oddelený jednotným oddeľovačom. Jeden riadok tohto súboru odpovedá jednému riadku pixelov v mape. Práve hodnoty v jednotlivých pixeloch sú oddelené bodkočiarkou (**;**) pre zachovanie čo najlepšej kompatibility.

5.1.7 Rozšíriteľnosť

Mapy samotné sú v *AdditionalViewsForm* v podobe záložiek v *TabControl*. Pre pridanie novej mapy stačí pridať novú záložku a zopár prvkov do nej.

TabPage (záložka) s mapou obsahuje:

- **Tlačidlo Render** – Volá *RenderMapButton_Click*. Pomocou reflection sa vyberie správna mapa, na nej sa zavolá metóda *Render* a výsledná bitmapa sa zobrazí do *PixtureBoxu*.
- **Tlačidlo Reset** – Volá *ResetZoomAndPanButton_Click* na resetovanie priblíženia a posunu obrázku v *PictureBoxe*.
- **PictureBox** – Tento *PictureBox* zobrazuje výslednú mapu. Takisto umožňuje kliknutie na daný pixel (*MouseDown* a *MouseMove* volá príslušnú metódu **MapPictureBox_MouseDownAndMouseMove*) pre zobrazenie bližších informácií v postrannom *Labely*. v jednom *TabPage* sa očakáva len jeden *PictureBox*.
- **Jeden alebo viacero Labelov** – Tie zobrazujú buď všeobecné štatistiky z triedy *Statistics* (ako celkový počet primárnych lúčov a ich priemerný počet na pixel) alebo dáta získané z *GetValueAtCoordinates* príslušnej mapy po kliknutí na daný pixel *PixtureBoxu*.
- **Panel** – Na ňom sú položené všetky vyššie spomenuté *Controly*.
- **Tlačidlo Save Image** – Volá *SaveMapButton_Click*.
- **Tlačidlá na prezeranie histórie** – Dozadu (označené **<<**) a dopredu (označené **>>**), ktoré volajú *PreviousImageButton_Click* a *NextImageButton_Click* v uvedenom poradí.
- **Tlačidlo Export Data** – Musí zavolať *ExportDataButton_Click*. Toto slúži na zavolanie metódy *ExportData* príslušnej mapy a exportovanie dát.
- Pre správne fungovaniu reflection musí mať príslušný *TabPage* položku *Tag* rovnú názvu mapy, ktorá je triedou odvodenej od interfacu *IMap*.

5.2 Pohyb a priblíženie obrázku

Trieda *PanAndZoomSupport* (v rovnomennom súbore) pridáva statickému obrázku v *PictureBoxe* funkcionalitu pohybu obrázku do strán a jeho priblíženie. Je to podobné, ako práca s rôznymi obrázkovými editormi. Pre správne fungovanie je potrebné vytvoriť inštanciu tejto triedy cez konštruktor, v ktorom sa predá

- **PictureBox**, v ktorom má byť obrázok,
- **Image**, ktorý predstavuje samotný obrázok a

- callback **setWindowTitleSuffix**, ktorý sa zavolá so *stringom* reprezentujúcim aktuálnu úroveň priblíženia pri každej zmene (primárne použité na nastavenie tohto textu priamo do hornej časti okna – *Form.Text*).

Následne sa na túto inštanciu volajú funkcie:

- **SetNewImage** – vždy keď je k dispozícii nový obrázok, ktorý sa má zobraziť v *PictureBoxe*. Obrázok sa nastaví s rovnakým priblížením a na rovnakej pozícii (pozícia ľavého horného rohu bitmapy).
- **Reset** – presunie obrázok na pôvodné miesto a vráti priblíženie späť na hodnotu 100%.
- Z event handlerov pre daný *PictureBox* je nutné volať metódy:
    - MouseDownRegistration
        - v event handlery MouseDown (analogicky pre zvyšné)
    - MouseMoveRegistration
    - MouseUpRegistration
    - MouseWheelRegistration
    - KeyDownRegistration

5.2.1 História obrázkov

Okrem mechanizmov na priblíženie a pohyb obrázku sa trieda *PanAndZoomSupport* stará aj o históriu obrázkov. Práve v argumentoch metódy *SetNewImage* je *bool saveToHistory*, ktorý rozhoduje, či sa daný obrázok uloží do histórie. Toto je užitočné hlavne na to, aby sa do histórie neukladali zbytočne dočasne vyrenderované obrázky. Renderer počas renderovania automaticky obnovuje obrázok v hlavnom *PictureBoxe v* zadaných intervaloch tak, aby bolo vidieť priebeh renderovania. Tieto obrázky sú však nevhodné pre ukladania do histórie.

Naopak pri vyrenderovaní odlišného obrázku sa ten uloží do histórie. Tá sa prechádza pomocou *SetNextImageFromHistory* a *SetPreviousImageFromHistory*, ktoré sa volajú z event handlerov pre priradené tlačidlá. Pri prechode históriou sa nemení pozícia obrázku a ani jeho priblíženie. Vďaka tomuto je možné priblížiť sa na konkrétne miesto a porovnať ho s iným obrázkom z histórie. Toto je užitočné predovšetkým pri vyrenderovaní rovnakej scény s rozdielnymi parametrami.

Premenná *historyCapacity* udáva maximálny počet obrázkov uložených v histórii. Ten je predvolene 5. Nakoľko sa do histórie ukladajú aj mapy, by pri uložení viacerých obrázkov vo vysokom rozlíšení súčasne došlo k zaberaniu netriviálneho množstva pamäte. Pre ďalšie obmedzenia histórie viď podkapitolu 6.9 v záverečnej analýze.

5.3 RayVisualizer

RayVisualizer je okno, ktoré pomocou OpenGL [6] (zabalené v OpenTK [7]) zobrazuje reprezentáciu scény a šírenie lúča danou scénou.

Všetko potrebné k tento sekcii je obsiahnuté v súboroch *RayVisualizer.cs* a *RayVisualizerForm.cs*.

5.3.1 Vizualizácia scény

Za každých okolností sa zobrazí reprezentácia scény pomocou bounding boxov (viď 5.3.1.1). V prípade, že je povolené zbieranie dát pre point cloud (viď 5.3.1.2), je možné vyrenderovať ten.

5.3.1.1 Bounding box

Použitie metódy bounding boxov na zobrazenie každého objektu scény je jednoduché na implementáciu. Každý objekt sa nachádza vždy len vo vnútri svojho bounding boxu. A teda aj priesečník lúča s týmto objektom musí byť vo vnútri (prípadne na stene) daného bounding boxu. Z tohto dôvodu je lepšie zobraziť len hrany (wireframe) kvádrov reprezentujúcich bounding boxy.

Informácia o scéne je získaná z premennej typu *IRayScene*. Metóda *FillSceneObjects* vytvorí *List<SceneObject> sceneObjects*. Ten sa následne použije na vizualizáciu bounding boxov. Fungovanie *FillSceneObjects* je založená na prechádzaní stromu objektov v scéne pomocou rekurzívneho volania *EvaluateSceneNode*. Konkrétne prechodom scény a získaním všetkých telies typu *ISolid* spolu s transformačnou maticou (reprezentujúcou polohu/rotáciu/skosenie/... v scéne) a s farbou, ktorú má mať výsledný bounding box daného telesa. Tieto 3 údaje sú uložené v triede *SceneObject*.

Dáta potrebné pre bounding box sa získavajú priamo z *ISolid* metódou *GetBoundingBox*. Vďaka faktu, že bounding boxy sú rovnobežné so svetovými

osami scény, kváder ktorý ich reprezentuje je jednoznačne určený 2 protiľahlými bodmi. A práve to sa získa z metódy *GetBoundingBox*.

5.3.1.2 Point cloud

Na vytvorenie point cloudu je využitý existujúci systém ray-tracingu. Pri každom priesečníku lúču so scénou sa (okrem iného) tento bod zaregistruje aj do point cloudu.

Samotná implementácia point cloudu spočíva v registrácii priesečníkov lúča pri hlavnom ray-tracingu cez metódu *AddToPointCloud* triedy *PointCloud*. Táto trieda obsahuje *List<float>[]*. Každé vlákno má priradený jeden *List<float>*, preto ide o pole. Dáta sa ukladajú ako floatové hodnoty do jedného dlhého *List<float>* – pozícia XYZ (hodnoty pre 3 na seba kolmé osy), farba RGB a normálový vektor XYZ.

Zároveň je možné point cloud uložiť do formátu PLY [5]. Jedná sa o štandardnú ASCII verziu formátu PLY podľa normy.

Nakoľko sú dáta z point cloudu uložené ako polia *floatov*, stačí tieto polia spojiť a nabindovať ako buffer pre OpenGL. To dokáže renderovať priamo jednotlivé body. Na ne je použitý fragment shader [13] s Gouraudovým tieňovaním [11].

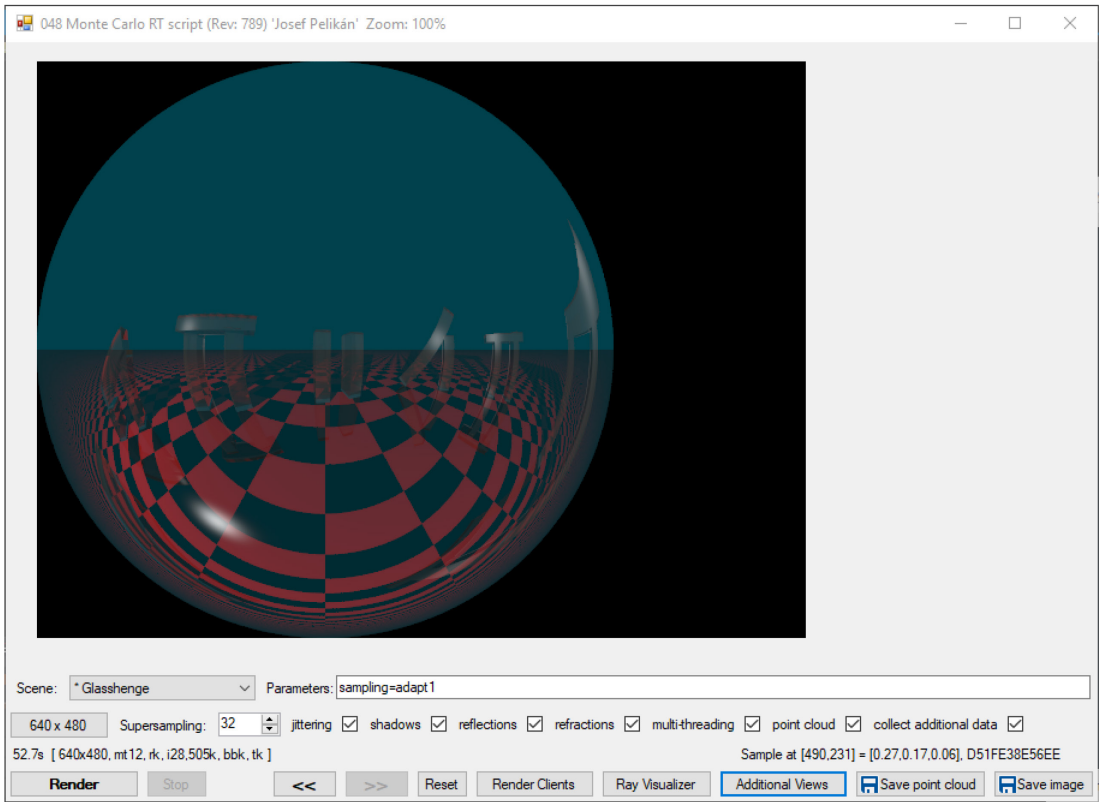
5.3.2 Princíp fungovania

Lúč sa vyberie kliknutím sa vyrenderovaný obrázok v samotnom ray-tracery. Trieda *RayVisualizer* dostáva informácie o novom normálnom (primárnom, odrazenom alebo zalomenom) lúči pomocou metódy *RegisterRay*. v prípade tieňového lúča je to *RegisterShadowRay*.

Následne sa trieda *RayVisualizerForm* postará o vyrenderovanie všetkého potrebného. Tam je najdôležitejšia metóda *RenderScene*. Tá postupne pre každý snímok volá metódy na vyrenderovanie bounding boxov/point cloudu, lúčov a reprezentáciu kamery a svetelných zdrojov. Kamera a svetlá sú v scéne reprezentované pomocou 2D objektov s billboardingom (2D objekt/obrázok, ktorý je natočený vždy smerom ku kamere, takže ho vidieť len spredu).

5.3.3 Reprezentácia lúčov

Lúče sú reprezentované začiatočnou a koncovou pozíciou (*Vector3d*). v *List<Vector3d> RayVisualizer.rays* (resp. *RayVisualizer.shadowRays*) sú pomyselne uložené dáta k jednotlivým lúčom za sebou. Každý párny index obsahuje začiatok daného lúča a každý nepárny zase koniec. Špeciálne sa zachádza s úplne prvým normálnym lúčom, keďže ten sa renderuje len niekedy (záleží na nastavení checkboxu „*Keep alligned*“). Taktiež sa používa na zistenie pozície kamery v scéne (začiatok prvého lúča), nakoľko kamera pozíciu ako takú implicitne nemá. Je to dané z dôvodu, že kamera môže byť rôzneho typu, ako napríklad parabolická (viď Obrázok 15), a určiť pri nej pozíciu nemusí byť jednoznačné.



Obrázok 15 - Parabolická kamera

5.4 RenderClient

RenderClient je samostatná konzolová aplikácia, ktorá umožňuje hlavnému ray-traceru distribuované renderovanie na viacerých strojoch po sieti. Tento *RenderClient* je prispôsobený pre ray-tracer *048rtmontecarlo-script*, ale sieťová architektúra ako aj prerozdeľovanie renderovacej práce je navrhnuté univerzálne. Stačí dodať scénu a renderer (ray-caster, ray-tracer, ...). Výstupom je bitmapa.

5.5 Workload distribution

Pre potreby *RenderClienta* bolo nutné vytvoriť nový systém prerozdeľovania renderovacej práce. Všetko potrebné je v súboroch *WorkloadDistribution.cs* a *RenderClientsForm.cs*. Tie obsahujú triedy *Assignment, Master* a *NetworkWorker*.

Odôvodnenie kompletného prepísania a porovnanie v podkapitole 6.6 v záverečnej analýze.

5.5.1 Assignment

Trieda *Assignment* je v jednoduchosti jedna jednotka renderovacej práce. Obsahuje informáciu a výreze a hustote (stride), pri akej sa má vyrenderovať. Pre viac informácií o stride viď podkapitolu 6.7.

Trieda *Assignment* obsahuje metódu *Render*, ktorá vracia vyrenderovaný výrez v podobe jednorozmerného poľa *floatov* (pixely po riadkoch; 3 *floaty* na pixel reprezentujúce RGB kanály).

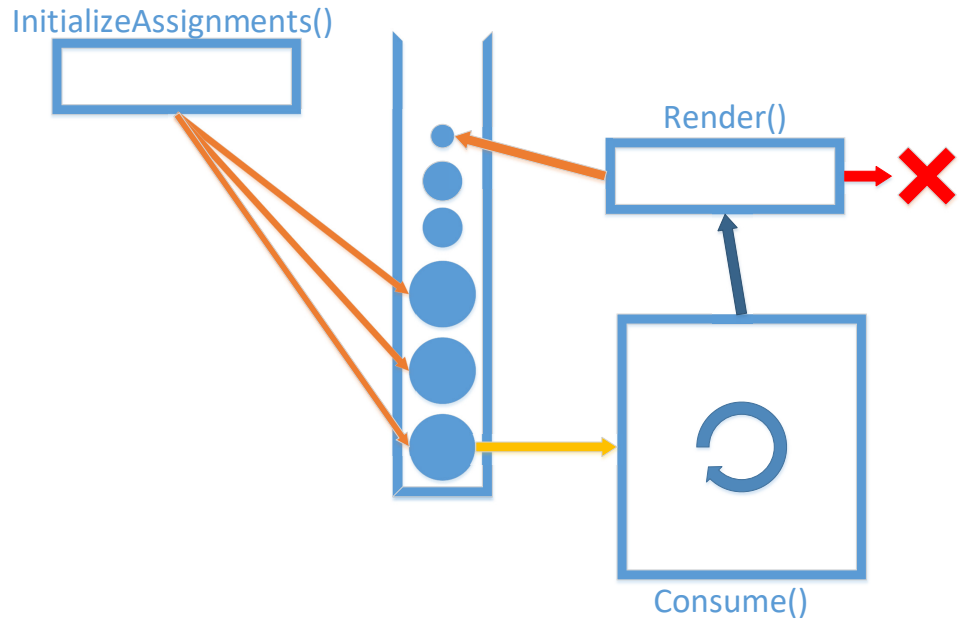
5.5.2 Master

Táto trieda sa stará o prerozdelenie práce medzi lokálne vlákna procesora a *RenderClientov*. Princíp fungovania je nasledujúci:

- Vytvorí sa fronta *Assignmentov* podľa veľkosti hlavnej bitmapy.
- Vytvoria sa inštancie triedy *NetworkWorker* – každý *NetworkWorker* naviaže spojenie s jedným *RenderClientom* po sieti (zoznam IP adries na naviazanie spojení je v *BindingList<Client> clients* pod triedou *RenderClientsForm*)
- Každý *NetworkWorker* si so vzdialeným *RenderClientom* vymení potrebné informácie (viď 5.5.3).
- Každému *NetworkWorkerovi* sa priradí toľko *Assignmentov*, koľko má jeho pridružený *RenderClient* k dispozícii vlákien na renderovanie. Počet *Assignmentov* je v skutočnosti navýšený o malú rezervu, aby sa zmiernil overhead sieťového prenosu.
- Lokálne vlákna neustále pracujú v metóde *Consume* (viď Obrázok 16) – z fronty *Assignmentov* sa odoberie nový *Assignment* a spracuje sa. Ak ešte nebol spracovaný úplne (*stride* je väčší ako 1), tak je vrátený do fronty (so zníženým *stride*).

- Medzitým 1 vlákno (v metóde *RenderedImageReceiver*) prechádza cez všetkých *NetworkWorkerov* a prijíma od pridružených *RenderClientov* vyrenderované obrázky – pri prijatí obrázka pridelí *NetworkWorkerovi* nový *Assignment* a ten ho pošle vzdialenému *RenderClientovi*. Kontrola nových dát v *NetworkWorkeroch* od vzdialených strojov funguje asynchrónne, takže sa toto vlákno nevyťažuje, ak nie sú k dispozícii žiadne nové dáta.
- Vyrenderované obrázky (v podobe jednorozmerného poľa *floatov*), či už od *NetworkWorkera* alebo lokálneho vlákna sa pošlú do metódy *BitmapMerger*, ktorá ich spojí do hlavnej bitmapy.
- Ak už neexistuje žiadny ďalší *Assignment* na pridelenie pre *NetworkWorkera*, *NetworkWorker* pošle *RenderClientovi* špeciálny *EndingAssignment*, ktorý značí, že *RenderClient* sa môže ukončiť a zbytočne nečakať na ďalšie *Assignmenty*.
- Renderovacia práca končí, keď sa vyrenderuje posledný *Assignment*.

V prípade prerušenia spojenia s *RenderClientom* sa nedokončený *Assignment* vráti späť do hlavnej fronty, kde sa ho ujme lokálne vlákno alebo iný *NetworkWorker*.



Obrázok 16 - Princíp fungovania triedy Master

5.5.3 Sieťové prepojenie

Sieťové prepojenie medzi *NetworkWorkerom* a *RenderClientom* funguje nasledovne:

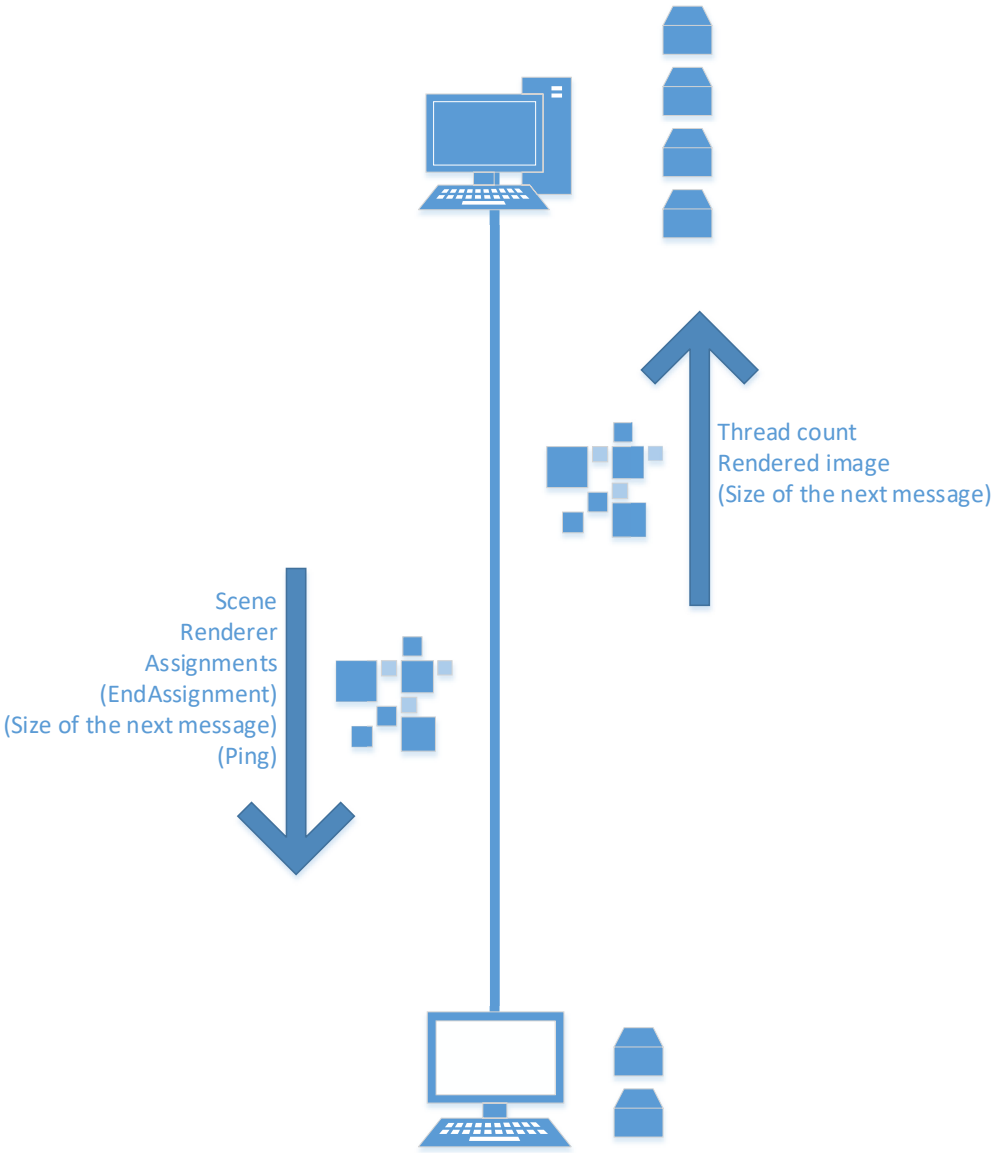
- Vytvorí sa štandardne TCP [15] spojenie.
- Metóda *ExchangeNecessaryInfo* zabezpečí predanie potrebných dát:
  - reprezentácia scény a rendereru samotného sa zoserializujú a pošlú po sieti,
  - prijme sa *int* reprezentujúci počet dostupných vlákien na vzdialenom stroji.
- Prenos *Assignmentov* je zabezpečený metódou *TryToGetNewAssignment*.
  - Rovnako ako v predchádzajúcom prípade sa dáta zoserializujú a pošlú (pomocou triedy *NetworkSupport*).
- Príjem vyrenderovaného obrázku je v podobe jednorozmerného poľa *floatov*:
  - prvé 2 *floaty* reprezentujú *intové* hodnoty udávajúce X-ovú a Y-ovú súradnicu ľavého horného rohu *Assignmentu* (pozícia výrezu v hlavnej bitmape),
  - zvyšok poľa reprezentuje pixely výrezu po riadkoch od ľavého horného rohu (kanály RGB – 3 *floaty* na 1 pixel).

  Náhľad na sieťovú komunikáciu poskytuje Obrázok 17.

Kvôli serializácii (pomocou *BinaryFormatter* [22]) je nutné, aby všetky serializované triedy (scéna, renderer, *Assignment*) a v nich použité triedy boli označené pomocou atribútu ***[Serializable]***.

5.5.4 RenderClient samotný

Jedná sa o samostatnú konzolovú aplikáciu. Pri spustení sa na danom stroji vytvorí *TcpListener*, ktorý čaká na pripojenie od hlavného rendereru (od nejakého *NetworkWorkeru*). Po pripojení čaká na prijatie *Assignmentov* a následne ich spracuje. Prerozdeľovanie renderovacej práce medzi lokálne vlákna funguje na rovnakom princípe ako trieda *Master* – stará sa o to trieda *ClientMaster*, ktorá je od triedy *Master* odvodená. Po prijatí špeciálneho *EndingAssignmentu* sa *RenderClient* prepne do rovnakého stavu, v akom bol po nadviazaní prvého spojenia, t.j. čaká na novú várku renderovacích *Assignmentov*.



Obrázok 17 - Sieťové spojenie s RenderClientom

6 Záverečná analýza

Táto kapitola slúži na porovnanie výkonu a opodstatnenie výberu hodnôt pre zásadné konštanty. Taktiež odôvodňuje rozhodnutie použiť niektoré kľúčové techniky a formáty namiesto iných.

6.1 Vizualizácia lúča v scéne

Zobrazenie lúča by bolo možné priamo vo vyrenderovanom obrázku. To však to značnej miery obmedzuje použiteľnosť. Hlavný obrázok sa renderuje pomocou ray-tracingu a ten je príliš pomalý. Nebolo by napríklad možné dostatočne plynule pohybovať kamerou a skúmať zobrazenie daného lúča.

Druhou možnosťou je použitie rasterizácie. Na to sa použije grafická knižnica OpenGL [6], resp. wrapper pre C# nazvaný OpenTK [7]. Rasterizácia však môže zobrazovať len trojuholníky (resp. plošné útvary). Pri scéne zadanej trojuholníkovou sieťou by to problém nebol. Scény v GrCis sú však zadané ako CSG, ktoré nie je možné priamo vykresliť pomocou OpenGL.

Na prevod z CSG na trojuholníkovú sieť existuje niekoľko algoritmov, ako napríklad Marching Cubes [16], ale všetky sú príliš náročné na implementáciu a nad úroveň tejto práce. Pre zobrazenie lúča však stačí jednoduchšia reprezentácia scény. Na to sa využijú 2 rozdielne techniky prebraté v nasledujúcich podkapitolách.

6.2 Bounding boxy

Táto technika veľmi zjednodušuje objekty (nezachováva žiadne informácie o objekte, len jeho vonkajšie hranice) a pri niektorých scénach v CSG môže kvôli použitiu binárnych operácii spôsobovať problémy.

Jediné, čo môžeme pomocou tejto techniky vidieť, je poloha a približná veľkosť objektov. Vo veľa scénach to však bohato stačí (viď Obrázok 10).

6.3 Point cloud

Toto zobrazenie je oproti bounding boxom omnoho detailnejšie a zachováva väčšinu informácií o objekte, avšak je výpočtovo náročnejšie a je potrebné vyrenderovať obrázok scény, aby sme získali dáta pre point cloud.

6.3.1 Dátové štruktúry

Point cloud sa vnútorne ukladá do *List<float>[]*. Každé procesorové vlákno má priradený jeden *List<float>*, preto ide o pole. Vďaka tomu môžu všetky renderovacie vlákna zapisovať dáta do point cloudu naraz. Použitie jediného Listu so zamykaním by bolo príliš pomalé. Dokonca ani použitie kontajnerov, ktoré sú očividne prispôsobené na prácu s viacerými vláknami, nie je rýchlejšie ako toto riešenie. Napríklad *ConcurrentBag* [19] by sa mohol zdať ako dobrá voľba, ale jeho metóda na pridanie nového prvku je extrémne pomalá v porovnaní s *List<float>*.

Dátové štruktúry pre 3D dáta ako *Octree* [17] alebo *k-d strom* [18] sú skvelé na prácu s nimi (hľadanie najbližších *n* bodov a podobne), ale toto nie je pri implementácii point cloudu v tejto práci potrebné. Jediné, čo je potrebné, je rýchle ukladanie a načítanie dát bez nutnosti akejkoľvek štruktúry alebo zoradenia. Pre potreby zobrazenia grafickou kartou je nutné mať hodnoty v jednorozmernom poli hneď za sebou, čiže žiadne stromy. A okrem iného nič nie je pamäťovo úspornejšie ako uloženie v jednorozmernom poli.

Dáta sa ukladajú ako floatové hodnoty – pozícia XYZ, farba RGB a normálový vektor XYZ. Hodnoty sú zoradené za sebou v jednom dlhom *List<float>*. Ide o menej prehľadné riešenie ako napríklad použitie tried alebo štruktúr na ukladanie informácie o každom bode, ale zase je najrýchlejšie a menej náročné na pamäť.

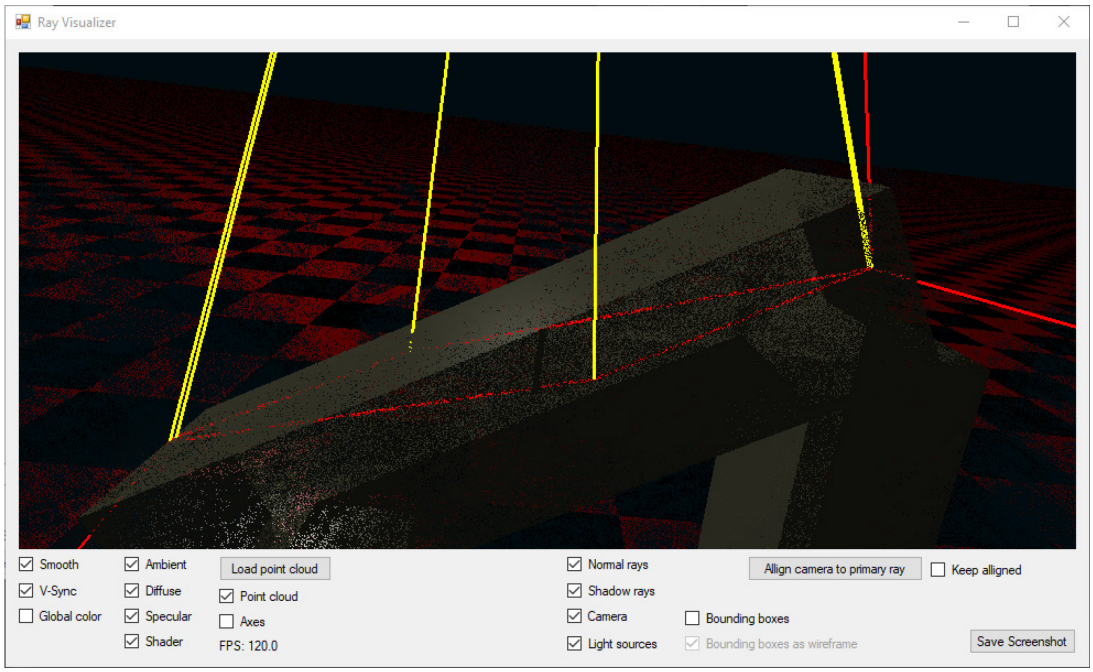
6.3.2 Formát PLY

Formát 3D dát PLY [5] bol na ukladanie point cloudu zvolený kvôli jeho jednoduchosti, otvorenosti a širokej podpore softwaru tretích strán. **MeshLab** [4] je voľný software, ktorý je príkladom programu vhodného na zobrazenie point cloudových dát vytvorených touto aplikáciou. Existuje binárna aj ASCII verzia PLY formátu. ASCII verzia bola zvolená kvôli jednoduchšiemu zachádzaniu. Taktiež je v nej možné ručne hľadať chyby, čo pri binárnej verzii nie je takmer vôbec možné. Jediná výhoda binárnej verzie oproti ASCII je menšia veľkosť súborov. Tá však nie je kritická pri uložení na disk.

6.3.3 Vykreslenie

Na vykreslenie point cloudu je použitý fragment shader s Gouraudovým tieňovaním [11]. Oproti Phongovmu tieňovaniu [12] je obecne Gouraudovo horšie, lebo interpoluje bez použitia normál. To však v prípade point cloudu je zanedbateľné, nakoľko každý bod má vlastnú farbu a žiadna interpolácia nenastáva. Z rovnakého dôvodu sa používa len tieňovanie vo fragment shadery a nie vo vertex shadery. Staršie grafické karty majú obecne viac jednotiek na počítanie fragment shaderu. Pri point cloude pojem „fragment“ a „vertex“ splývajú [6].

Zaujímavou vlastnosťou point cloudov je, že (pri dostatočnej hustote a vzdialenosti) sa objekty javia ako celistvé, ale pri priblížení je možné vidieť cez ne (viď Obrázok 11 a Obrázok 12). Presnejšie povedané, je možné vidieť pomedzi jednotlivé body point cloudu. To je v skutočnosti priaznivý efekt, ktorý umožňuje nahliadnutie do objektu a skúmanie zalamovania lúča vnútri v ňom (viď Obrázok 18).



Obrázok 18 - Zalamovanie lúča vo vnútri objektu

6.4 Hodnoty v mapách

6.4.1 Priemerovanie hodnôt

Všetky mapy počítajú výsledné hodnoty v pixeloch aritmetickým priemerom všetkých nazbieraných hodnôt pre daný pixel. To však pri hĺbkovej (DepthMap)

a obidvoch normálových mapách (NormalMapRelative a NormalMapAbsolute) spôsobuje nepresné meranie na hranách. Ak sa meria pixel, ktorý je na hrane jedného objektu a za ním je iný, výrazne vzdialený, tak sa pri vrhnutí viacerých lúčov na pixel môžu nazbierať hĺbky obidvoch objektov. Tieto hodnoty sa spriemerujú a teda výsledná hodnota (hĺbka) je niekde medzi nimi. To môže byť zavádzajúce. Avšak na odstránenie tohto problému nestačí brať vždy len jednu hodnotu na pixel (a teda nepriemerovať hodnoty). To by mohlo pri niektorých objektoch spôsobiť aliasing [10], čiže artefakty spôsobené prevodom spojitého signálu na diskrétny. Napríklad hĺbková mapa by sa tvárila, že je objekt plochý, aj keď je hrboľatý a podobne. Presne pre tento účel sa pri ray-tracingu posiela na pixel viacero lúčov. Rovnaký nežiaduci efekt nastáva pri spomínaných normálových mapách. Na odstránenie tohto efektu by bola potrebná netriviálne zložitá štatistická analýza celej mapy a označovanie objektov. To by však bolo výpočtovo a aj implementačne náročné a nad rozsah tejto práce.

6.4.2 Logaritmická škála

Väčšina máp používa na výslednú farbu pixelu lineárnu prechodovú škálu na prechod z hodnoty k farbe. Pri testovaní sa však ukázalo, že hĺbková mapa má príliš veľký rozsah hodnôt na lineárnu škálu. Preto je použitá logaritmická škála.

6.5 ExportData

Možnosť exportovať dáta máp sa využije pri rôznorodej analýze, napríklad pomocou grafov v tabuľkových editoroch. Ale nevylučuje sa ani import do iných programov a pokračovania spracovania máp tak, ako boli v pôvodnom programe.

Formát CSV [14] bol vybraný pre jeho jednoduchosť implementácie ako aj veľmi široké spektrum podpory programov tretích strán. Výhodná je taktiež čitateľnosť tohto formátu človekom, nakoľko ide o textový a nie binárny formát.

6.6 RenderImage

Prerozdeľovanie renderovacej práce medzi procesorové vlákna existovalo, samozrejme, aj v pôvodnom ray-tracery *048rtmontecarlo-script*. Je to jedna z kritických častí, na ktorých závisí rýchlosť renderovania. Hlavnou metódou prerozdeľovania práce je *RenderImage*. Kvôli potrebám distribuovaného

renderovania po sieti však pôvodný *RenderImage* nebol vhodný a musel byť kompletne prepísaný. To okrem prídavku väčšej flexibility (možnosť renderovania na lokálnom a aj vzdialených strojoch naraz) zlepšilo výkon renderovania vďaka lepšiemu prerozdeľovaniu medzi vlákna o 20 až 45% (viď Tabuľka 1).

Scéna	***Starý systém***	***Nový systém***	***Zlepšenie***
HedgehogInTheCage	*66,1 s*	*53,4 s*	23,8%
SpherePlane	*43,9 s*	*31,5 s*	39,5%
Circus	*51,7 s*	*42,7 s*	21,2%
SphereFlake	*22 054,3 s*	*15 268,9 s*	44,4%
Priemer	**----**	**----**	**32,3%**

Tabuľka 1 - Porovnanie starého a nového systému prerozdeľovania práce

Veľká odchýlka je len v rámci rôznych scén, nie pri renderovaní rovnakej scény opakovane. Extrémne rozdielne výsledky pri scéne *„SphereFlake“* od iných scén je dané veľkou komplexnosťou tejto scény a je to očakávaný výsledok.

6.6.1 Testovacie podmienky

Testy boli robené na Intel i7-3930k (6 jadier/12 vlákien; 4,48 GHz), rozlíšenie výstupu 1920x1080, 64-násobný supersampling, všetko (jittering, shadows, reflections, ...) zapnuté okrem zberu dát pre point cloud a triedu *AdditionalViews* (mapy a štatistiky). V každej bunke je priemer 5 spustení. Časy sú uvedené v sekundách. Sekundy aj percentá sú zaokrúhlené na 1 desatinné miesto.

6.7 Assignment

Trieda *Assignment* predstavuje jednu jednotku renderovacej práce. Ide o informáciu o:

- výreze (64x64 px) z hlavnej bitmapy, ktorý sa má vyrenderovať
- hustote (stride), pri ktorej sa má vyrenderovať.

Princíp stride je založený na algoritme Adam7 [20] (prekladanie/interlacing rastrových obrázkov – hlavne PNG). Napríklad pri stride == 8 sa renderuje len každý 8. pixel v každom 8. riadku. Hodnota tohto pixelu sa nakopíruje do okolitých prázdnych pixelov - do štvorca s ľavým horným rohom vo vyrenderovanom pixeli a dĺžkou strany rovnej hodnote stride. Stride nadobúda hodnoty 8 -> 4 -> 2 -> 1 s každým ďalším renderovaním daného *Assignmentu*. Vďaka tomuto je možné mať

adaptívne renderovanie, čiže vidieť ešte počas renderovania obrázok scény, aj keď v „nižšom“ rozlíšení, ktoré sa postupne zlepšuje.

6.7.1 Zvolená veľkosť Assignmentu

Výrez o veľkosti 64x64 px bol zvolený ako najlepší z pohľadu pomeru granularity a rýchlosti.

- Pri menšej veľkosti (32x32 px a menej), vlákna zbytočne často striedali *Assignmenty*, čo malo nepriaznivý dopad na rýchlosť.
- Pri väčšej veľkosti (nad 128x128 px) vznikalo príliš málo *Assignmentov*. To malo za následok (hlavne pri menších obrázkoch), že procesorové vlákna rýchlo prestali byť aktívne. Zostalo menej *Assignmentov* ako je počet vlákien a zároveň tie *Assignmenty* boli veľké, takže sa nerovnomerne využívali vlákna. Taktiež efekt adaptívneho renderovania nebol príliš dobrý, nakoľko sa vo výslednej bitmape zjavovali naraz príliš veľké časti.
- Pre jednoduchosť algoritmu a možnosť použiť modifikovanú verziu Adam7, je nutné, aby *Assignment* (resp. jeho priradený výrez bitmapy) bol štvorcový s dĺžkou strany, ktorá je mocninou dvojky.
  Najväčší krok pre stride (8 px) bol zvolený na podobnom princípe ako veľkosť výrezu.
- Pri počiatočnej hodnote stride väčšej ako 8, sa stalo, že vlákna zbytočne často striedali *Assignmenty*, čo malo dopad na rýchlosť. Takisto adaptívne renderovanie nemalo dostatočný účinok, keďže skutočne vyrenderované pixely boli príliš ďaleko od seba a až príliš sa stratili detaily scény.
- Pri menšej počiatočnej hodnote, trvalo príliš dlho, kým sa dokončilo prvé kolo renderovania a teda dlho nebolo vidieť dostatočne veľa zo scény.

Pri renderovaní *Assignmentu* na *RenderClientovi* sa renderuje celý *Assignment* naraz (t.j. so stride == 1), aby sa obmedzil overhead prenosu dát po sieti. S týmto treba počítať, keďže budú kde-tu vznikať úplne nevyrenderované časti (*Assignment* bežiaci na vzdialenom stroji), aj keď naokolo môžu už byť aspoň čiastočne vyrenderované *Assignmenty* (so stride napr. 8 alebo aj 4). Úloha distribuovaného renderovania je však renderovanie už finálneho obrázku a teda

výhoda vysokého výkonu prevyšuje nevýhodu nepoužiteľného adaptívneho renderovania.

6.8 Priblíženie obrázku v PictureBoxe

Je dôležité, aby v metóde *Paint* boli nastavené:

- *PixelOffsetMode.Half*,
- *InterpolationMode.NearestNeighbor* a
- *SmoothingMode.None*.

Vďaka tomu majú pixely výslednej bitmapy v *PictureBoxe* hodnoty skutočných pixelov bez akéhokoľvek priemerovania a vyhladzovania (viď Obrázok 7). Vyhladzovanie by bolo pre prehliadanie obrázku pri priblížení nežiaduci, nakoľko by sa tým stratila, resp. skryla, skutočná informácia. Presnejšie povedané - nformácia v pixeli na obrazovke by neodpovedala informácii v mape.

6.9 História obrázkov

História obrázkov má niekoľko obmedzení. Mapy sú vyrenderované, až keď sa prejde na konkrétnu záložku s nimi. Čiže pri prechode v histórii niektoré mapy nemusia byť k dispozícii. História je obmedzená na maximálne 5 obrázkov (na každú mapu zvlášť a hlavný vyrenderovaný obrázok). Taktiež treba brať do úvahy fakt, že do histórie sa ukladajú len bitmapy a nie dáta pre jednotlivé mapy. Je to kvôli tomu, že dáta na vytvorenie máp sú veľmi veľké a ich uloženie v histórii by zaberalo netriviálne veľké množstvo pamäte.

6.10 Zbieranie dodatočných dát

V základnom nastavení renderovania je možné povoliť samostatne zber dát pre point cloud (ak ho chceme zobraziť v *RayVisualizery* alebo ho uložiť na disk) a dát pre *AdditionalViews* (mapy a štatistiky). Toto však logicky spomaľuje samotné renderovanie. v nasledujúcich podkapitolách sú výsledky meraní skutočného zaťaženia zberom týchto dát.

Merania boli uskutočnená za rovnakých podmienok aké sú popísané v kapitole 6.6.1. Pri meraní spomalenia zberu dát pre point cloud bol zber dát pre *AdditionalViews* vypnutý, a obrátene. Zbery týchto 2 rôznych setov dát sú vzájomne

disjunktné a nijako sa neovplyvňujú. Z toho vyplýva, že celkové spomalenie pri obidvoch zberoch zapnutých je súčtom spomalenia jednotlivých zberov samostatne.

6.10.1 Zbieranie dát pre point cloud

Zapína sa checkboxom *„point cloud“*. Toto nastavenie je predvolene **vypnuté**, nakoľko spomalenie renderovania je netriviálne (viď Tabuľka 2).

Scéna	***Zakázané***	***Povolené***	***Spomalenie***
HedgehogInTheCage	*53,4 s*	*68,7 s*	22,3%
SpherePlane	*31,5 s*	*47,5 s*	33,6%
Circus	*42,7 s*	*56,3 s*	24,2%
Priemer	**----**	**----**	**26,7%**

Tabuľka 2 - Spomalenie pri zbere dát pre point cloud

6.10.2 Zbieranie dát pre AdditionalViews

Zapína sa checkboxom *„collect additional data“*. Vďaka minimálnemu vplyvu na výkon (viď Tabuľka 3) je toto nastavenie predvolene **zapnuté**.

Scéna	***Zakázané***	***Povolené***	***Spomalenie***
HedgehogInTheCage	*53,4 s*	*56,0 s*	4,7%
SpherePlane	*31,5 s*	*32,3 s*	2,4%
Circus	*42,7 s*	*44,1 s*	3,2%
Priemer	**----**	**----**	**3,4%**

Tabuľka 3 - Spomalenie pri zbere dát pre AdditionalViews

6.10.3 Obmedzenia

Kvôli faktu, že distribuované renderovanie po sieti je predovšetkým prítomné pre čo najväčšiu rýchlosť renderovania už finálneho obrázku, sa po sieti neprenášajú dáta pre mapy, štatistiky ani point cloud.

7 Záver

Táto kapitola je poslednou samostatnou kapitolou a stručne zhŕňa prínos práce ako aj možnosť budúceho rozšírenia.

7.1 Prínos práce

Knižnica GrCis obsahuje sadu nástrojov, ktoré sa používajú pri výučbe počítačovej grafiky na MFF UK. Okrem iného obsahuje aj program na vykresľovanie (renderovanie) obrázkov 3D scén pomocou techniky sledovania lúča (ray-tracing), ktorá sa radí medzi historicky významné techniky. Avšak v tejto knižnici chýbali nástroje na vizualizáciu rozšírených informácií o renderovaní a predovšetkým možnosť vidieť, ako daný lúč prechádza scénou.

V rámci tejto bakalárskej práce vznikol doplnok *RayVisualizer* pre ray-tracer z knižnice GrCis, ktorý umožňuje interaktívne skúmať šírenie, zalamovanie a odrážanie sa lúča scénou. To môže výrazne pomôcť pri výučbe počítačovej grafiky, či už priamo na prednáškach alebo v rámci samoštúdia.

Ďalším rozšírením ray-traceru pridaného v rámci tejto práce je kolekcia máp, čiže obrázkov podobných tomu hlavnému vyrenderovanému, ale nesúcich v pixeloch rozdielne informácie. Príkladom je napríklad hĺbková mapa, ktorá pomocou čierno-bielej farebnej škály zobrazuje hĺbku scény v danom pixeli. Veľmi užitočnou je aj mapa počtu vrhnutých lúčov na pixel, nakoľko ten môže byť vďaka adaptívnemu supersamplingu rozdielny v rôznych častiach obrázku.

V rámci implementovania týchto doplnkov vznikla aj možnosť vytvoriť point cloud reprezentujúci 3D scénu a ten potom použiť v *RayVisualizer* alebo si ho uložiť na disk.

RenderClient je zase malá konzolová aplikácia, ktorá umožňuje distribuované renderovanie po sieti. To však vyžadovalo prepísanie hlavnej metódy na prerozdeľovanie renderovacej práce medzi vlákna a ako vedľajší efekt tohto sa renderovanie zrýchlilo v priemere o vyše 30%.

Všetky tieto nástroje nájdu uplatnenie aj pri ladení vlastných 3D scén alebo iných doplnkov. Práve *RayVisualizer* pomohol identifikovať minimálne 2 netriviálne chyby v pôvodnom ray-tracery (zalamovanie lúča v objektoch a nesprávna kalkulácia priesečníku s torusom).

7.2 Možnosti rozšírenia

Nie je nezvyčajné, že sa študentom predmetov počítačovej grafiky zadáva na domáce úlohy rozšírenie funkcionality knižnice GrCis. Preto celé toto dielo a predovšetkým triedy *AdditionalViews* a *AdditionalViewsForm* sú navrhnuté s maximálnou možnosťou rozšíriteľnosti. To sa predovšetkým týka pridávania nových máp. Viac v programátorskej dokumentácii 5.1.7.

Ray-tracer ako aj rozšírenia, ktoré sú predmetom tohto diela, sú príkladom programu, ktorý sa dá rozširovať a vylepšovať donekonečna. Príkladom je renderovanie scén zadaných ako sieť trojuholníkov, optimalizácia vyprodukovaných point cloudov atď.

Konkrétnejším príkladom môže byť práve optimalizácia point cloudu. Momentálne sa používajú 3 *floaty* na pozíciu, 3 *floaty* na normálový vektor a 3 *floaty* na farebné zložky RGB. To by sa pravdepodobne dalo zredukovať na normálu reprezentovanú 2 uhlami (na to stačia dva 16b *integery*) a 3B na farby. Zníženie kvality by malo byť minimálne a zároveň zníženie pamäťových nárokom dosť podstatné.

8 Referencie

1. https://cgg.mff.cuni.cz/~pepca/grcis/ - knižnica GrCis.

2. https://www.rtvtk.org/ - Ray Tracing Visualization Toolkit.

3. https://github.com/daseyb/pathgraph - Path Graph.

4. http://www.meshlab.net - MeshLab.

5. http://paulbourke.net/dataformats/ply/ - formát 3D dát PLY.

6. https://www.opengl.org/ - OpenGL.

7. https://opentk.net/ - OpenTK.

8. https://docs.microsoft.com/en-us/windows/desktop/directx - DirectX.

9. Foley, James D.. *12.7 Constructive Solid Geometry*. Computer Graphics: Principles and Practice, Addison-Wesley Professional, 1996. strany 557–558. ISBN 978-0201848403.

10. Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory To Implementation, 3rd Edition.* Morgan Kaufmann, 2016. ISBN 978-0128006450.

11. https://graphics.fandom.com/wiki/Gouraud_shading - Gouraudovo tieňovanie.

12. Bui Tuong Phong. *Illumination for Computer Generated Pictures*, Communications of the ACM, 1975. Vol 18 (6), strany 311–317.

13. https://www.khronos.org/opengl/wiki/Shader - shadere.

14. http://edoceo.com/utilitas/csv-file-format – formát CSV.

15. Vinton G. Cerf; Robert E. Kahn. *A Protocol for Packet Network Intercommunication*. IEEE Transactions on Communications, 1974. 22 (5): strany 637–648. doi:10.1109/tcom.1974.1092259

16. Lorensen, W. E. and Cline, H. E.. *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*. Communications of the ACM, 1987. strany 163–169. ISBN 0-89791-227-6.

17. Meagher, Donald. *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*. Rensselaer Polytechnic Institute (Technical Report IPL-TR-80-111).

18. Bentley, J. L.. *Multidimensional binary search trees used for associative searching*. Communications of the ACM, 1975. 18 (9): strany 509–517. doi:10.1145/361002.361007.

19. https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent.concurrentbag-1 - trieda Concurrent Bag v .NET.
20. https://en.wikipedia.org/wiki/Adam7_algorithm – algoritmus Adam7.
21. Lafortune, E. *Mathematical Models and Monte Carlo Algorithms for Physically Based Rendering*. (dizertačná práca), 1996.
22. https://docs.microsoft.com/en-us/dotnet/api/system.runtime.serialization.formatters.binary.binaryformatter - trieda BinaryFormatter v .NET.

9 Zoznam použitých skratiek

- CSG – Constructive Solid Geometry
- px – pixel; obrazový bod
- PLY – Polygon File Format (formát súborov s 3D dátami)
- CSV – Comma Separated Values (formát súborov s tabuľkovými dátami)
- PNG – Portable Network Graphics (formát 2D obrázkov)
- RGB – Red, Green, Blue
- TCP – Transmission Control Protocol
- ASCII – American Standard Code for Information Interchange

10 Zoznam tabuliek

11 Zoznam obrázkov

12 Prílohy

12.1 Príloha 1 – Obsah priloženého CD-ROM

Koreňový priečinok na priloženom CD nosiči obsahuje túto prácu v *.rtf* a *.pdf* formáte. Tu sa tiež nachádza priečinok **Bakalárska práca** s celým zdrojovým kódom. Ten obsahuje aj pôvodný ray-tracer 048rtmontecarlo-script a potrebné časti z knižnice GrCis. Pod priečinkom **048rtmontecarlo-script** sa nachádza súbor **048rtmontecarlo-script.sln** obsahujúci solution pre Visual Studio.

Ďalej pod priečinkom **RenderClient** nachádza solution pre túto časť, nakoľko ide o samostatnú časť nezávislú na solution 048rtmontecarlo-script.

Ak by sa z dôvodu premiestňovania a preskupovania priečinkov nezobrazili v ponuke scén v 048rtmontecarlo-script iné scény než „Test scene“, je nutné nastaviť Working directory (a prípadne aj Command line arguments) v nastaveniach solution vo Visual Studiu tak, aby spolu ukazovali na úložisko scén (priečinok **rtscenes**).