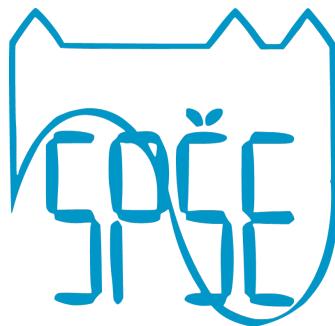


**STREDNÁ PRIEMYSELNÁ ŠKOLA  
ELEKTROTECHNICKÁ  
ZOCHOVA 9, BRATISLAVA**



*Image - I - \2*

**Golang Ray-Tracer**  
projekt z praktickej časti odbornej zložky maturitnej skúšky

**Meno kandidáta:** Matúš Benček

**Trieda:** 4.D

**Vedúci práce:** Mgr. Jakub Krcho

**Šk. rok:** 2024/2025

**STREDNÁ PRIEMYSELNÁ ŠKOLA ELEKTROTECHNICKÁ  
ZOCHOVA 9, BRATISLAVA**

**Meno kandidáta: Matúš Benček**

**Trieda: 4.D**

**Vedúci práce: Mgr. Jakub Krcho**

**Rozsah: xx**

Úloha: Navrhnite program, ktorý je schopný renderovať 3D scénu pomocou techniky ray tracing v programovacom jazyku Go. Program bude sledovať svetelné lúče, ktoré prechádzajú cez 3D objekty, pričom bude počítať ich interakcie s povrchmi, ako sú odrazy, tiene a lomy. Hlavným cieľom je dosiahnuť realistické zobrazenie scény na základe presných fyzikálnych výpočtov svetla a jeho správania. Program musí byť optimalizovaný tak, aby dokázal spracovať aj komplexnejšie scény v rozumnom čase.

## **Čestné prehlásenie**

Čestne prehlasujem, že problematiku týkajúcu sa náplne zadaného projektu v rámci praktickej časti odbornej zložky maturitnej skúšky formou obhajoby som spracoval sám za pomocí uvedenej použitej literatúry. Inú, ako uvedenú literatúru, som nepoužil.

V Bratislave dňa: 27. 3. 2025

Matúš Benček

## **Pod'akovanie**

Checel by som sa pod'akovovať svojmu vedúcemu práce nie len za pomoc a vedení pri tvorbe maturitného projektu ale aj za celé štyri roky, ktorý vo mňa veril a naučil ma mnoho do budúcnosti.

## **Obsah**

|  |    |
|--|----|
| Úvod.....  | 1  |
| 1.0 TEORETICKÁ ČASŤ.....   | 2  |
| 1.1 Ray Tracing.....   | 2  |
| 1.2 Voxel Rendering.....   | 2  |
| 1.3 Ray Marching.....  | 2  |
| 1.4 Podpora Shaderov.....  | 3  |
| 1.5 Použité technológie.....   | 3  |
| 2.0 PRAKTICKÁ ČASŤ.....  | 4  |
| 2.1 Architektúra Projektu GO-Draaw.....                              | 4  |
| 2. Backend.....  | 4  |
| 2.2 Backend Implementácia Web Servera.....                           | 4  |
| 2.2.1 Koncové body.....  | 4  |
| 2.3 Dokumentácia Frontend Komponentov Ray Tracingu.....              | 9  |
| 2.3.1 Color Picker.....  | 9  |
| 2.3.2 Texture Color Picker.....                                      | 10 |
| 2.3.3 Shader Menu.....   | 11 |
| 2.3.4 Render Options.....  | 18 |
| Horná Lišta.....   | 18 |
| Menu Pozície Kamery.....   | 19 |
| Menu Ukážky Rendera.....   | 20 |
| 2.3.5 Volume Picker.....   | 25 |
| 3.0 Princíp fungovania BVH.....                                      | 27 |
| 3.2 Surface Area Heuristic (SAH).....                                | 28 |
| 3.3 Reprezentácia Trojuholníkov a Materiálové Vlastnosti.....        | 29 |
| 3.3.1 Geometrická Reprezentácia.....                                 | 30 |
| 3.3.2 Materiálové Vlastnosti.....                                    | 30 |
| 3.4 Nová Implementácia BVHLean.....                                  | 33 |
| 3.5 BVH a jej implementácia.....                                     | 35 |
| 3.5.1 Evolúcia BVH štruktúry.....                                    | 36 |
| 3.5.2 Optimalizovaná BVHLean.....                                    | 37 |
| 3.5.3 Optimalizácia BVHLean - porovnanie 2 bounding boxov naraz..... | 38 |
| 3.5.4 Experimentálna array-based implementácia.....                  | 40 |
| 4.0 Podpora Načítavania 3D Geometrie.....                            | 41 |
| 4.1 Načítavanie .OBJ Súborov.....                                    | 41 |
| 5.0 RayTracing Vývoj Funkcionality.....                              | 43 |
| TraceRay.....  | 43 |
| TraceRayV2.....  | 45 |
| TraceRayV3.....  | 46 |
| TraceRayV3Advance.....   | 48 |
| TraceRayV3AdvanceTexture.....  | 51 |

|   |           |
|---|-----------|
| TraceRayV4AdvanceTexture.....                                 | 53        |
| <b>6.0 Systém Benchmarkovania a Výkonnostnej Analýzy.....</b> | <b>63</b> |
| 6.1 Zhrnutie Štatistik.....                                   | 63        |
| 6.2 Technologické Rozdiely Verzií.....                        | 65        |
| 6.3 Záver Testov.....   | 66        |
| 6.3.1 Median Graph.....                                       | 67        |
| 6.3.2 Mean Graph.....   | 67        |
| 6.3.3 STD Graph.....  | 68        |
| 6.3.4 Min Frame Time.....                                     | 69        |
| 6.3.5 Max Frame Time.....                                     | 69        |
| 6.3.6 Bottom 10 % Frame Time.....                             | 70        |
| 6.3.7 Top 10 % Frame Time.....                                | 70        |
| 6.4 Úvod do Benchmarkingu.....                                | 71        |
| 6.4.1 Testované Verzie Rendereru:.....                        | 71        |
| 6.4.2 Príprava Testovania.....                                | 72        |
| 6.4.3 Špecifická Implementácie.....                           | 73        |
| 6.5 Metriky Výkonu.....                                       | 73        |
| 6.5.1 Sledované Ukazovatele.....                              | 73        |
| 6.5.2 Výstup a Analýzy.....                                   | 74        |
| 6.5.3 Postprocessing dát.....                                 | 75        |
| 6.6 Implementácia Benchmarku v Go.....                        | 75        |
| 6.7 Kľúčové Výhody Benchmark Systému.....                     | 76        |
| <b>7.0 Rendrovacie Pomocné Funkcie.....</b>                   | <b>77</b> |
| 7.1 FresnelSchlick Funkcia.....                               | 77        |
| 7.1.1 Účel.....   | 77        |
| 7.1.2 Parametre.....  | 77        |
| 7.1.3 Ako Funguje.....  | 78        |
| 7.1.4 Praktické Efekty.....                                   | 78        |
| 7.2 GGX Distribučná Funkcia.....                              | 79        |
| 7.2.1 Účel.....   | 79        |
| 7.2.2 Parametre.....  | 79        |
| 7.2.3 Ako Funguje.....  | 80        |
| <b>8.0 Voxel Rendering.....</b>                               | <b>80</b> |
| 8.1 Kľúčové Vlastnosti:.....                                  | 81        |
| 8.2 Interaktívne Editačné Funkcie.....                        | 82        |
| 8.3 Optimalizácia Renderingu Voxelov.....                     | 83        |
| <b>9.0 Implementácia Objemového Rendering-u.....</b>          | <b>86</b> |
| 9.1 Kľúčové Funkcie:.....                                     | 88        |
| 9.2 Optimalizácie Výkonu.....                                 | 88        |
| 9.3 Fyzikálne Modely Objemu.....                              | 89        |
| <b>10.0 Raymarching – Základný Popis.....</b>                 | <b>90</b> |

|   |            |
|---|------------|
| 10.1 Výhody a Nevýhody.....                       | 90         |
| 10.1. Aktuálny Stav.....                          | 90         |
| 10.4 Signed Distance Fields (SDF).....            | 91         |
| 10.5 Druhá Verzia Ray Marching-u.....             | 92         |
| 10.6 Potencionálne Vylepšenia.....                | 93         |
| <b>11.0 Podpora Post-Processing Shaderov.....</b> | <b>95</b>  |
| 11.1 Úvod do Post-Processingu.....                | 95         |
| 11.2 Jazyk Shaderov Kage.....                     | 95         |
| 11.3 Podporované Post-Processing Efekty.....      | 96         |
| 11.3.1 Základné Efekty.....                       | 96         |
| 11.3.2 Komplexné Vizuálne Efekty.....             | 97         |
| 11.4 Výhody Implementácie.....                    | 98         |
| 11.5 Záver.....                                   | 98         |
| <b>Záver.....</b>                                 | <b>98</b>  |
| <b>ZOZNAM BIBLIOGRAFICKÝCH ODKAZOV.....</b>       | <b>100</b> |
| Online Knihy o Ray Tracingu.....                  | 100        |
| Technické Videá a Prezentácie.....                | 100        |

## ZOZNAM OBRÁZKOV

| Číslo<br>Obrázku | Názov   | Číslo Strany |
|------------------|---|--------------|
| 1                | <a href="#"><u>Image - Frontend</u></a>           | 14           |
| 2                | <a href="#"><u>Image - Color Picker</u></a>       | 16           |
| 3                | <a href="#"><u>Image - Render Bez Shadrov</u></a> | 17           |
| 4                | <a href="#"><u>Image - Kontrast Shader</u></a>    | 17           |
| 5                | <a href="#"><u>Image - Tint Shader</u></a>        | 18           |

|           |  |           |
|-----------|--|-----------|
| <b>6</b>  | <a href="#"><u>Image - Bloom Shader</u></a>                | <b>18</b> |
| <b>7</b>  | <a href="#"><u>Image - Bloom V2 Shader</u></a>             | <b>19</b> |
| <b>8</b>  | <a href="#"><u>Image - Sharpness Shader</u></a>            | <b>19</b> |
| <b>9</b>  | <a href="#"><u>Image - Color Mapping Shader</u></a>        | <b>20</b> |
| <b>10</b> | <a href="#"><u>Image - Chromatin Aberration Shader</u></a> | <b>20</b> |
| <b>11</b> | <a href="#"><u>Image - Edge Detection Shader</u></a>       | <b>21</b> |
| <b>12</b> | <a href="#"><u>Image - Lighten Shader</u></a>              | <b>21</b> |
| <b>13</b> | <a href="#"><u>Image - Vignette Shader</u></a>             | <b>22</b> |
| <b>14</b> | <a href="#"><u>Image - Horná Lišta</u></a>                 | <b>22</b> |
| <b>15</b> | <a href="#"><u>Image - Pozícia Kamery</u></a>              | <b>23</b> |
| <b>16</b> | <a href="#"><u>Image - Ukážka Rendera</u></a>              | <b>23</b> |
| <b>17</b> | <a href="#"><u>Image</u></a>                               | <b>24</b> |
| <b>18</b> | <a href="#"><u>Image - Vypriemerovany Render</u></a>       | <b>24</b> |
| <b>19</b> | <a href="#"><u>Image - Ray Marching V1</u></a>             | <b>26</b> |
| <b>20</b> | <a href="#"><u>Image - Ray Marching V2</u></a>             | <b>27</b> |
| <b>21</b> | <a href="#"><u>Image - Ray Marching V2</u></a>             | <b>27</b> |
| <b>22</b> | <a href="#"><u>Image - Render Options GUI</u></a>          | <b>28</b> |
| <b>23</b> | <a href="#"><u>Image - Volume Options GUI</u></a>          | <b>29</b> |
| <b>24</b> | <a href="#"><u>Image - BVH</u></a>                         | <b>30</b> |
| <b>25</b> | <a href="#"><u>Image - BVH</u></a>                         | <b>31</b> |
| <b>26</b> | <a href="#"><u>Image - Triangle Simple Struct</u></a>      | <b>34</b> |

|           |  |           |
|-----------|--|-----------|
| <b>27</b> | <a href="#"><u>Image - TriangleBbox Struct</u></a>       | <b>34</b> |
| <b>28</b> | <a href="#"><u>Image - Texture Struct</u></a>            | <b>35</b> |
| <b>29</b> | <a href="#"><u>Image - BVH Porovnanie</u></a>            | <b>37</b> |
| <b>30</b> | <a href="#"><u>Image - BBoxPair Funkcia</u></a>          | <b>38</b> |
| <b>31</b> | <a href="#"><u>Image - BBox Porovanie</u></a>            | <b>39</b> |
| <b>32</b> | <a href="#"><u>Image - V1 Benchmark</u></a>              | <b>42</b> |
| <b>33</b> | <a href="#"><u>Image - V1 Render</u></a>                 | <b>42</b> |
| <b>34</b> | <a href="#"><u>Image - V2 Benchmark Table</u></a>        | <b>43</b> |
| <b>35</b> | <a href="#"><u>Image - V2 Benchmark Flame Graph</u></a>  | <b>43</b> |
| <b>36</b> | <a href="#"><u>Image - V2M Benchmark</u></a>             | <b>44</b> |
| <b>37</b> | <a href="#"><u>Image - V2M Render</u></a>                | <b>44</b> |
| <b>38</b> | <a href="#"><u>Image - V2Lin Benchmark Table</u></a>     | <b>45</b> |
| <b>39</b> | <a href="#"><u>Image - V2Lin Flame Graph</u></a>         | <b>45</b> |
| <b>40</b> | <a href="#"><u>Image - V2Lin Render</u></a>              | <b>46</b> |
| <b>41</b> | <a href="#"><u>Image - V2Log Benchmark</u></a>           | <b>46</b> |
| <b>42</b> | <a href="#"><u>Image - V2Log Render</u></a>              | <b>47</b> |
| <b>43</b> | <a href="#"><u>Image - V2Lin Texture Table</u></a>       | <b>48</b> |
| <b>44</b> | <a href="#"><u>Image - V2Lin Texture Flame Graph</u></a> | <b>48</b> |
| <b>45</b> | <a href="#"><u>Image - V2Lin Texture Render</u></a>      | <b>48</b> |
| <b>46</b> | <a href="#"><u>Image - V2Log Texture Flame Graph</u></a> | <b>49</b> |
| <b>47</b> | <a href="#"><u>Image - V2Log Texture Table</u></a>       | <b>49</b> |

|           |  |           |
|-----------|--|-----------|
| <b>48</b> | <a href="#"><u>Image - V2Log Texture Render</u></a>    | <b>49</b> |
| <b>49</b> | <a href="#"><u>Image - V4Lin Table</u></a>             | <b>51</b> |
| <b>50</b> | <a href="#"><u>Image - V4Lin Flame Graph</u></a>       | <b>51</b> |
| <b>51</b> | <a href="#"><u>Image - V4Lin Render</u></a>            | <b>52</b> |
| <b>52</b> | <a href="#"><u>Image - V4Log Table</u></a>             | <b>52</b> |
| <b>53</b> | <a href="#"><u>Image - V4Log Flame Graph</u></a>       | <b>52</b> |
| <b>54</b> | <a href="#"><u>Image - V4Log Render</u></a>            | <b>53</b> |
| <b>55</b> | <a href="#"><u>Image - V4Lin Optim Flame Graph</u></a> | <b>53</b> |
| <b>56</b> | <a href="#"><u>Image - V4Lin Optim Table</u></a>       | <b>54</b> |
| <b>57</b> | <a href="#"><u>Image - V4Lin Optim Render</u></a>      | <b>54</b> |
| <b>58</b> | <a href="#"><u>Image - V4Log Optim Table</u></a>       | <b>55</b> |
| <b>59</b> | <a href="#"><u>Image - V4Log Optim Flame Graph</u></a> | <b>55</b> |
| <b>60</b> | <a href="#"><u>Image - V4Log Optim Render</u></a>      | <b>55</b> |
| <b>61</b> | <a href="#"><u>Image - V4V2 Table</u></a>              | <b>56</b> |
| <b>62</b> | <a href="#"><u>Image - V4V2 Flame Graph</u></a>        | <b>56</b> |
| <b>63</b> | <a href="#"><u>Image - V4V2 Render</u></a>             | <b>56</b> |
| <b>64</b> | <a href="#"><u>Image - Normals</u></a>                 | <b>57</b> |
| <b>65</b> | <a href="#"><u>Image</u></a>                           | <b>58</b> |
| <b>66</b> | <a href="#"><u>Image</u></a>                           | <b>58</b> |
| <b>67</b> | <a href="#"><u>Image</u></a>                           | <b>58</b> |
| <b>68</b> | <a href="#"><u>Image</u></a>                           | <b>59</b> |

|           |  |           |
|-----------|--|-----------|
| <b>69</b> | <a href="#"><u>Image - Štatistiky</u></a>                | <b>59</b> |
| <b>70</b> | <a href="#"><u>Image - Median</u></a>                    | <b>62</b> |
| <b>71</b> | <a href="#"><u>Image - Mean</u></a>                      | <b>62</b> |
| <b>72</b> | <a href="#"><u>Image - Štandardná Diviácia</u></a>       | <b>63</b> |
| <b>73</b> | <a href="#"><u>Image - Minimálny čas</u></a>             | <b>63</b> |
| <b>74</b> | <a href="#"><u>Image - Maximálny čas</u></a>             | <b>64</b> |
| <b>75</b> | <a href="#"><u>Image</u></a>                             | <b>64</b> |
| <b>76</b> | <a href="#"><u>Image</u></a>                             | <b>65</b> |
| <b>77</b> | <a href="#"><u>Image - vypnutie GC počas testu</u></a>   | <b>66</b> |
| <b>78</b> | <a href="#"><u>Image - Implementácia Testu</u></a>       | <b>69</b> |
| <b>79</b> | <a href="#"><u>Image - Fresnel efekt</u></a>             | <b>71</b> |
| <b>80</b> | <a href="#"><u>Image - Voxel Rendering</u></a>           | <b>73</b> |
| <b>81</b> | <a href="#"><u>Image</u></a>                             | <b>74</b> |
| <b>82</b> | <a href="#"><u>Image</u></a>                             | <b>74</b> |
| <b>83</b> | <a href="#"><u>Image - Voxel reprezentácia</u></a>       | <b>75</b> |
| <b>84</b> | <a href="#"><u>Image - Optimalizácia Voxelov</u></a>     | <b>76</b> |
| <b>85</b> | <a href="#"><u>Image - Volume Rendering</u></a>          | <b>77</b> |
| <b>86</b> | <a href="#"><u>Image</u></a>                             | <b>78</b> |
| <b>87</b> | <a href="#"><u>Image - SDF Funkcia</u></a>               | <b>80</b> |
| <b>88</b> | <a href="#"><u>Image - SDF Funkcie</u></a>               | <b>82</b> |
| <b>89</b> | <a href="#"><u>Image - SDF funkcie pre iné Tvary</u></a> | <b>83</b> |

|           |   |           |
|-----------|---|-----------|
| <b>90</b> | <a href="#"><b>Image - Príklad Kage Shadera</b></a> | <b>85</b> |
|           |   |           |

*Table 1.*

## ZOZNAM SKRATIEK

**Skratka**            **Popis**

---

BVH                      Bounding volume hierarchy

SDF                      signed distance field

BBOX                    Bounding Box

---

|      |                                 |
|------|---------------------------------|
| SPD  | Speed                           |
| Go   | Golang                          |
| RGBA | Červená / Zelená / Modrá / Alpa |

## Úvod

V súčasnej dobe počítačová grafika zohráva kľúčovú úlohu v mnohých oblastiach, od herného priemyslu až po vedecké vizualizácie. Jednou z najvýznamnejších technológií v tejto oblasti je Ray-Tracing, ktorý umožňuje vytvárať fotorealistické zobrazenia 3D scén simuláciou fyzikálnych vlastností svetla. Táto maturitná práca sa zameriava na implementáciu vlastného 3D engine-u, ktorý využíva práve túto pokročilú technológiu renderovania.

Hlavným cieľom práce je vytvoriť flexibilný a výkonný 3D engine, ktorý bude schopný nielen základného renderovania 3D scén pomocou Ray-Tracingu, ale poskytne aj možnosť využívať rôzne shadre pre pokročilé vizuálne efekty. Významnou súčasťou projektu je implementácia podpory pre renderovanie volumetrických materiálov prostredníctvom technológie Voxel, čo ďalej rozširuje možnosti vizualizácie komplexných objektov a efektov.

Pre implementáciu bol zvolený programovací jazyk Golang, ktorý sa vyznačuje niekoľkými kľúčovými výhodami. Prvou je jeho efektívna podpora multiprocesingu prostredníctvom Go rutín, čo je esenciálne pre optimalizáciu výkonu pri ray-tracingu. Druhou výhodou je jeho výkonnosť, ktorá sa približuje tradičným systémovým jazykom ako C a C++. Pre implementáciu shaderových programov bude využitý jazyk Kage, ktorý bol vyvinutý pre Ebiten 2D engine. Kage poskytuje intuitívnu syntax inšpirovanú jazykom Go, čo umožňuje efektívny vývoj shaderov.

Aplikácia poskytne užívateľom možnosť interaktívne upravovať vlastnosti 3D geometrie, vrátane farieb a rôznych aspektov materiálov. Dôraz je kladený na optimalizáciu výkonu, aby bolo možné renderovať scény v realistickom čase.

## **1.0 TEORETICKÁ ČASŤ**

### **1.1 Ray Tracing**

Ray tracing je pokročilá technológia renderovania 3D scén, ktorá simuluje fyzikálne vlastnosti svetla pre dosiahnutie fotorealistických zobrazení. Využíva sa na vytváranie detailných a presných odrazov, lomov a tieňov, čo prispieva k celkovému realizmu renderovaného obrazu. Táto technológia je kľúčová pre dosiahnutie vysokého stupňa vizuálnej kvality v 3D grafike. Ray tracing sleduje cestu jednotlivých lúčov svetla od zdroja svetla cez scénu až po oko pozorovateľa (alebo kameru). Pri každom strete lúča s objektom sa vypočítajú odrazy, lomy a tiene na základe vlastností materiálu objektu a uhla dopadu lúča. Tento proces sa opakuje pre množstvo lúčov, čím sa vytvorí detailný a realistický obraz.

### **1.2 Voxel Rendering**

Voxel rendering je technológia, ktorá umožňuje renderovanie volumetrických materiálov. Na rozdiel od tradičného renderovania, ktoré pracuje s povrchmi objektov, voxel rendering pracuje s objemovými dátami, čo umožňuje vizualizáciu komplexných efektov ako dym, hmla alebo oheň. Táto technológia rozširuje možnosti vizualizácie a pridáva do 3D scén ďalšiu úroveň detailu a realizmu. Voxely sú 3D pixely, ktoré reprezentujú objemové prvky v priestore. Voxel rendering rozdeľuje 3D priestor na mriežku voxelov a každému voxelu priradí informáciu o materiáli, farbe a hustote. Renderovanie sa potom vykonáva na základe týchto objemových dát, čo umožňuje vytváranie efektov, ktoré sú ľahko dosiahnuteľné tradičnými metódami.

### **1.3 Ray Marching**

Ray marching je technika renderovania, ktorá sa používa na vizualizáciu implicitných povrchov definovaných matematickými funkiami. Táto technika je efektívna pre renderovanie fraktálov, oblakov a iných organických tvarov, ktoré je ľahké modelovať tradičnými metódami. Ray marching umožňuje vytváranie zložitých a detailných geometrických štruktúr s relatívne nízkymi výpočtovými nárokmi. Namiesto sledovania lúčov svetla, ray marching postupuje pozdĺž lúča v malých krokoch a testuje, či sa lúč pretína s povrhom objektu. Ak sa pretne, vypočíta sa farba a osvetlenie daného bodu. Táto technika je vhodná pre renderovanie objektov, ktoré nemajú explicitne definované povrhy, ale sú popísané matematickými funkiami.

## 1.4 Podpora Shaderov

Shadery sú programy, ktoré sa spúšťajú na grafickom procesore a umožňujú pokročilé vizuálne efekty a úpravy renderovaného obrazu. V kontexte 3D engine-u umožňujú shadery implementáciu rôznych post-processing efektov, ako sú úpravy farieb, kontrastu, ostrosti a ďalšie, čo prispieva k finálnemu vizuálnemu štýlu a kvalite renderovaného obrazu. Shadery umožňujú programovateľné renderovanie, čo znamená, že vývojári môžu prispôsobiť grafický výstup podľa svojich potrieb. Používajú sa na implementáciu rôznych efektov, ako sú textúrovanie, osvetlenie, tiene, odrazy, lomy a ďalšie.

## 1.5 Použité technológie

Backend:

- Golang: Programovací jazyk použitý pre backendovú časť aplikácie. Golang, tiež známy ako Go, je kompilovaný, staticky typovaný programovací jazyk vyvinutý spoločnosťou Google. Je známy svojou jednoduchosťou, efektivitou a výkonom, čo ho robí vhodným pre vývoj backendových aplikácií, ktoré vyžadujú vysokú konkurentnosť a škálovateľnosť.
- Echo Framework: Webový framework v Golangu, ktorý uľahčuje vývoj webových aplikácií. Echo je ľahký a vysoko výkonný HTTP router a webový framework pre Golang. Poskytuje sadu nástrojov a funkcií, ktoré uľahčujú vývoj RESTful API a webových aplikácií, ako je smerovanie, spracovanie požiadaviek a odpovedí, middleware a ďalšie.

Frontend:

- Vue.js: JavaScriptový framework použitý pre vývoj frontendovej časti aplikácie. Vue.js je progresívny JavaScriptový framework pre budovanie používateľských rozhraní. Je navrhnutý tak, aby bol postupne adoptovateľný, čo znamená, že ho možno ľahko integrovať do existujúcich projektov. Vue.js sa zameriava na vrstvu pohľadu a uľahčuje vývoj interaktívnych a dynamických webových aplikácií.

Shader programy:

- Kage: Jazyk vyvinutý pre Ebiten 2D engine, použitý na implementáciu shaderových programov. Kage je špecifický jazyk pre písanie shaderov pre Ebiten, čo je jednoduchý a prenosný 2D grafický engine v Golangu. Používa sa na vytváranie vlastných vizuálnych efektov a úpravu renderovaného obrazu v 2D grafike.

## 2.0 PRAKTICKÁ ČASŤ

### 2.1 Architektúra Projektu GO-Draaw

Projekt je rozdelený na dve hlavné časti:

- Frontend: Vue.js
- Backend: Golang with Echo Framework a RayTracer

#### 2. Backend

- Vyvinutý v **Go (Golang)** s použitím **Echo framework**
- Pozostáva z dvoch hlavných komponentov:
  - **Ray-tracing engine**: Jadro výpočtového systému pre renderovanie
  - **Webový server**: Zabezpečuje komunikáciu s frontendovou časťou
- Backend beží asynchronne vo vlastných go-rutinách, čo minimalizuje potrebu zložitého manažmentu stavu a používania mutexov s pozitím unsefe, čím sa dosahuje vyšší výkon a lepšia odozva systému.
- Táto architektúra umožňuje efektívne oddelenie prezentačnej vrstvy od výpočtovej, pričom zachováva vysokú mieru interaktivity pre používateľa a zároveň poskytuje výkonný rendering komplexných 3D scén.

#### 2.2 Backend Implementácia Web Servera

##### 2.2.1 Koncové body

- **POST /submitColor** : Odoslať farebné údaje z Colour Picker-u  
Color struct {  
    R           float64 `json:"r"`  
    G           float64 `json:"g"`  
    B           float64 `json:"b"`  
    A           float64 `json:"a"`  
    Reflection float64 `json:"reflection"`  
    Roughness  float64 `json:"roughness"`

```

    DirectToScatter float64 `json:"directToScatter"`
    Metallic      float64 `json:"metallic"`
    RenderVolume   bool   `json:"renderVolume"`
    RenderVoxels   bool   `json:"renderVoxels"`
}


```

- **POST /submitVoxel** : Odoslat voxel údaje

```

type Volume struct {
    Density          float64 `json:"density"`
    Transmittance    float64 `json:"transmittance"`
    Randomnes        float64 `json:"randomness"`
    SmokeColorR      float64 `json:"smokeColorR"`
    SmokeColorG      float64 `json:"smokeColorG"`
    SmokeColorB      float64 `json:"smokeColorB"`
    SmokeColorA      float64 `json:"smokeColorA"`
    VoxelColorR      float64 `json:"voxelColorR"`
    VoxelColorG      float64 `json:"voxelColorG"`
    VoxelColorB      float64 `json:"voxelColorB"`
    VoxelColorA      float64 `json:"voxelColorA"`
    RandomnessVoxel float64 `json:"randomnessVoxel"`
    RenderVolume     bool   `json:"renderVolume"`
    RenderVoxel      bool   `json:"renderVoxel"`
    OverWriteVoxel   bool   `json:"overWriteVoxel"`
    VoxelModification string `json:"voxelModification"`
    UseRandomnessForPaint bool  `json:"useRandomnessForPaint"`
    ConvertVoxelsToSmoke bool  `json:"convertVoxelsToSmoke"`
}

```

- **POST /submitTextures** : Odoslat textúrové údaje

```

type TextureRequest struct {
    Textures      map[string]interface{} `json:"textures"`
    Normals       map[string]interface{} `json:"normals"`
    DirectToScatter float64           `json:"directToScatter"`
    Reflection    float64           `json:"reflection"`
    Roughness     float64           `json:"roughness"`
}

```

```

Metallic      float64      `json:"metallic"`
Index         int          `json:"index"`
Specular      float64      `json:"specular"`
ColorR        float64      `json:"colorR"`
ColorG        float64      `json:"colorG"`
ColorB        float64      `json:"colorB"`
ColorA        float64      `json:"colorA"`
}

}

```

- **POST /submitRenderOptions** : Odoslat konfiguráciu renderingu

```

type RenderOptions struct {
    Depth        int      `json:"depth"`
    Scatter      int      `json:"scatter"`
    Gamma        float64 `json:"gamma"`
    SnapLight    string   `json:"snapLight"`
    RayMarching  string   `json:"rayMarching"`
    Performance  string   `json:"performance"`
    Mode         string   `json:"mode"`
    Resolution   string   `json:"resolution"`
    Version      string   `json:"version"`
    FOV          float64 `json:"fov"`
    LightIntensity float64 `json:"lightIntensity"`
    R            float64 `json:"r"`
    G            float64 `json:"g"`
    B            float64 `json:"b"`
}

```

- **POST /submitShader** : Odoslat konfiguráciu shadera

```

type ShaderParam struct {
    Type     string      `json:"type"`
    Parameters map[string]interface{} `json:"params"`
}

```

- **GET /getCameraPosition** : Získat aktuálnu pozíciu kamery

```

type Position struct {
    X    float64 `json:"x"`
}

```

- ```

Y      float64 `json:"y"`
Z      float64 `json:"z"`
CameraX float64 `json:"cameraX"`
CameraY float64 `json:"cameraY"`

}

●   POST /moveToPosition : Presunúť kameru na určenú pozíciu
type Position struct {
    X      float64 `json:"x"`
    Y      float64 `json:"y"`
    Z      float64 `json:"z"`
    CameraX float64 `json:"cameraX"`
    CameraY float64 `json:"cameraY"`
}

●   POST /moveToPosition : Presunúť kameru na určenú pozíciu
type Position struct {
    X      float64 `json:"x"`
    Y      float64 `json:"y"`
    Z      float64 `json:"z"`
    CameraX float64 `json:"cameraX"`
    CameraY float64 `json:"cameraY"`
}

●   GET /getCurrentImage : Získa aktuálny vyrenderovaný obrázok.
●   GET /getSpheres : Získa aktuálne vlastnosti objektov pre SDF rendering, ako
sú pozícia a farba.

```

API odošle na frontend pole objektov: []Sphere{}

```

type Sphere struct {
    CenterX      float64 `json:"centerX"`
    CenterY      float64 `json:"centerY"`
    CenterZ      float64 `json:"centerZ"`
    Radius       float64 `json:"radius"`
    ColorR       float64 `json:"colorR"`
    ColorG       float64 `json:"colorG"`
    ColorB       float64 `json:"colorB"`
    ColorA       float64 `json:"colorA"`
    IndexOfOtherSphere float64 `json:"indexOfOtherSphere"`
}

```

```

SdfType      float64 `json:"sdfType"`
Amount       float64 `json:"amount"`

}

```

- **GET /getTypes** : Pošle mapu objektov na frontend s ID pre jednotlivé typy objektov

```

types := map[string]int{
    "distance":        int(distance),
    "union":          int(union),
    "smoothUnion":     int(smoothUnion),
    "intersection":   int(intersection),
    "smoothIntersection": int(smoothIntersection),
    "subtraction":    int(subtraction),
    "smoothSubtraction": int(smoothSubtraction),
    "addition":       int(addition),
    "smoothAddition": int(smoothAddition),
    "smoothUnionNoColorMix": int(smoothUnionNoColorMix),
}

}

```

- **POST /updateSphere** : API slúži na odoslanie modifikovaného SDF objektu na backend

```

type SphereUpdate struct {
    Amount      float32 `json:"amount"`
    CenterX    float32 `json:"centerX"`
    CenterY    float32 `json:"centerY"`
    CenterZ    float32 `json:"centerZ"`
    ColorA     uint8  `json:"colorA"`
    ColorB     uint8  `json:"colorB"`
    ColorG     uint8  `json:"colorG"`
    ColorR     uint8  `json:"colorR"`
    Index      int     `json:"index"`
    IndexOfOtherSphere int    `json:"indexOfOtherSphere"`
    Radius     float32 `json:"radius"`
    SdfType    int     `json:"sdfType"`

}

```

- **POST /moveCamera** : API slúži na vygenerovanie pozícii, cez ktoré sa má kamera v 3D scéne pohybovať

```

type Positions struct {
    Positions []Position `json:"positions"`
    TimeDuration float64 `json:"timeDuration"`
}

```

```

type Position struct {
    X      float64 `json:"x"`
    Y      float64 `json:"y"`
    Z      float64 `json:"z"`
    CameraX float64 `json:"cameraX"`
    CameraY float64 `json:"cameraY"`
}

```

## 2.3 Dokumentácia Frontend Komponentov Ray Tracingu

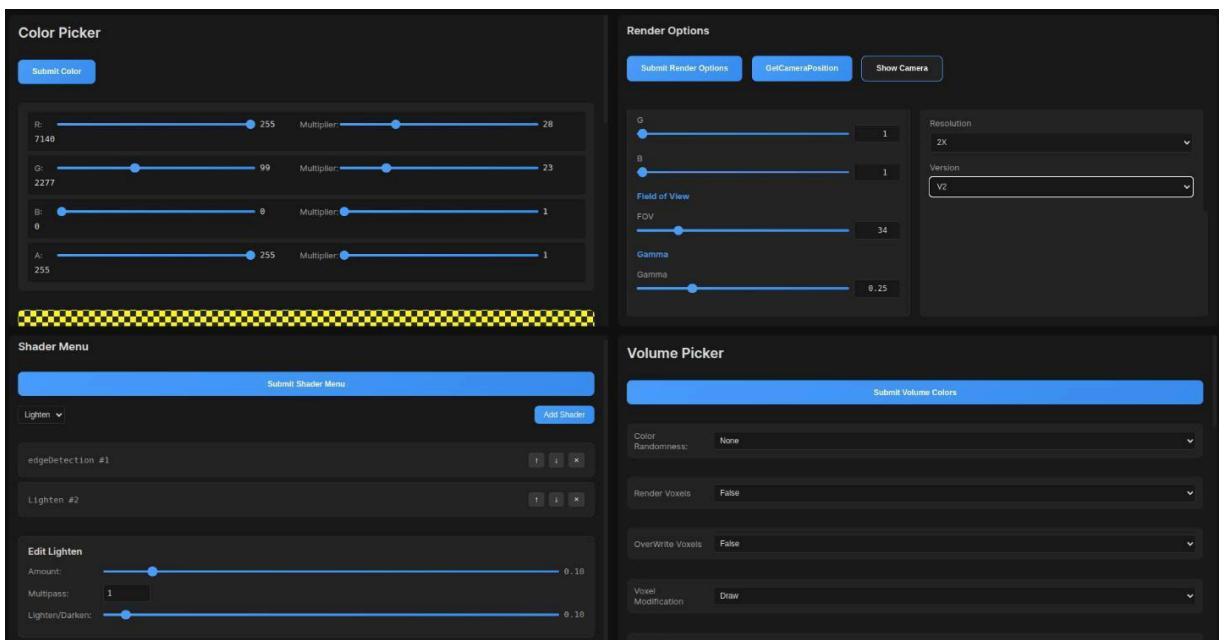


Image - 2 - \2

### 2.3.1 Color Picker

#### Farebné Kanály (plus Multiplayer)

- Červená (R): 0-256

- Zelená (G): 0-256
- Modrá (B): 0-256
- Alfa (A): 0-256

### Funkcie

- Náhľad Farby: Zobrazuje presne vybranú farbu
- Aplikácia Farby na Trojuholník: Umožňuje nastaviť farbu pre kliknutý trojuholník

### 2.3.2 Texture Color Picker

#### Výber Textúry

- Rozsah: 1-128 textúr
- Náhľad Textúry: Rozlíšenie  $128 \times 128 \times 4$  float

#### Interakcia s Textúrou

- Tlačidlo Nahraj Textúru: Nahratie textúry
- Schopnosť zobraziť a upravovať textúru na základe vybranej farby z Color Pickeru

#### Normal Mapa

- Rozlíšenie:  $128 \times 128 \times 3$
- Tlačidlo na zmenu normalizácie normálovej mapy medzi rozsahmi 0/1 a -1/1

#### Funkcie Normal Mapy

- Tlačidlo Nahraj Normal Mapu: Umožňuje nahrať normal mapy
- Tlačidlo "Žiadna Normal Mapa": Otvára online generátor normal máp (<https://cpetry.github.io/NormalMap-Online/>)

#### Zobrazenie Materiálových Vlastností

- Odraz
- Priamy na Rozptyl
- Drsnosť
- Kovový Lesk
- Špecular

## Úprava Textúry

- Posuvníky pre materiálové vlastnosti (rozsah 0-1)
- Násobiteľe Kanálov:
  - Červený Kanál
  - Zelený Kanál
  - Modrý Kanál
  - Alfa Kanál

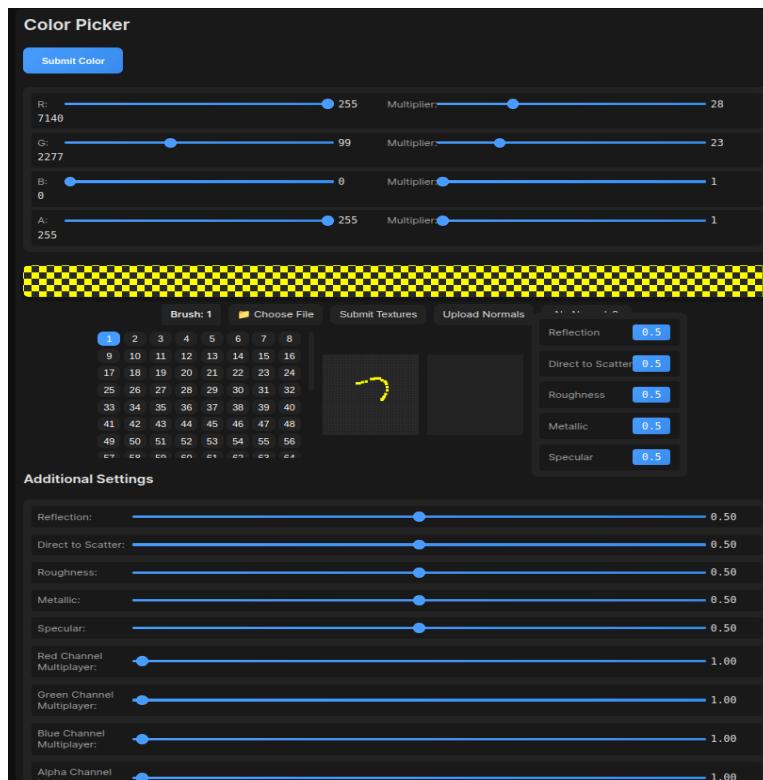


Image - 3 - \2

### 2.3.3 Shader Menu

## Účel

Vytváranie refázcov post-processingových shaderov (napr. pôvodný obrázok → kontrast → tint → finálny obrázok)

## Správa Shaderov

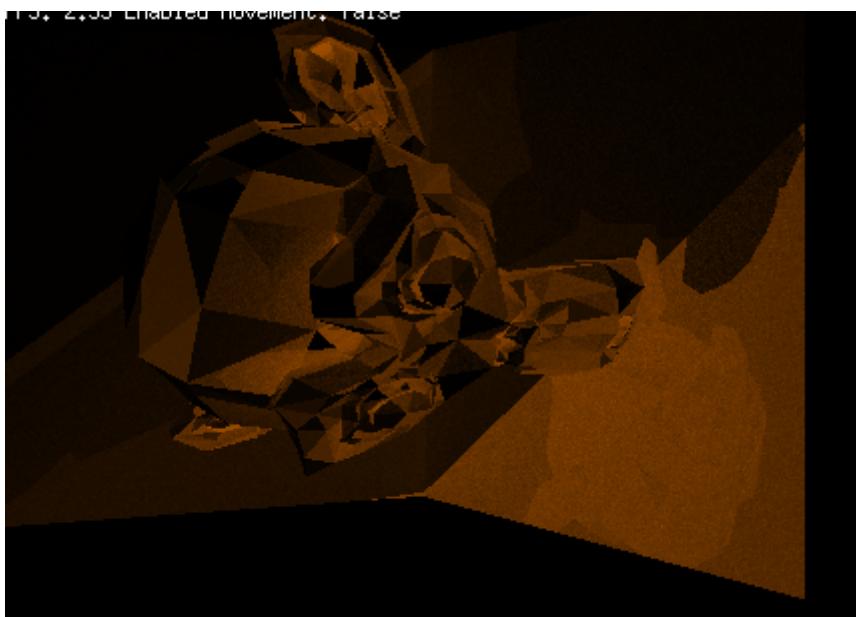
- Výber Shaderu
- Tlačidlo Pridať Shader
- Tlačidlo Odoslať Shader Menu

## Parametre Shaderov

### Spoločné Parametre

- amount : Podiel upraveného obrázku, ktorý sa pridá do renderingu
- multipass : Počet po sebe nasledujúcich aplikácií shaderu

### Render Bez Shadrov

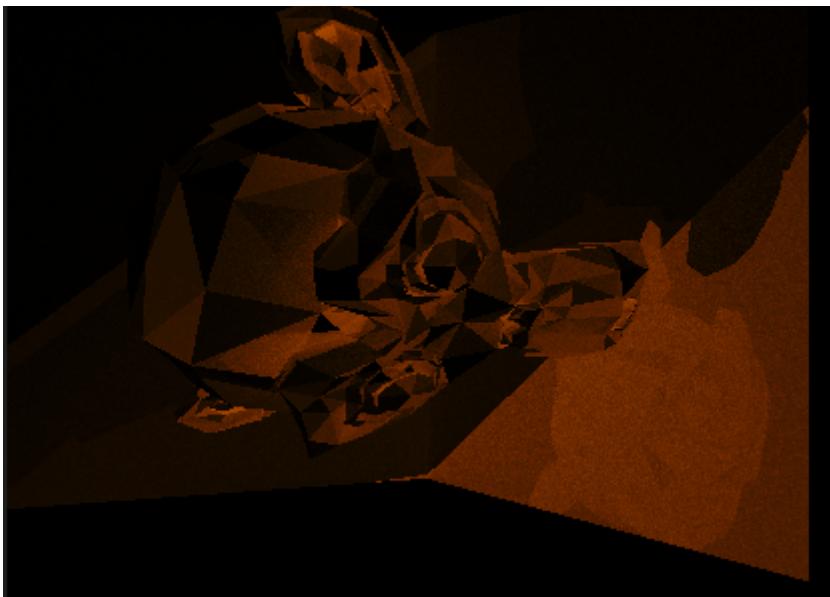


*Image - 4 - \2*

### Podporované Shadery

#### Kontrast

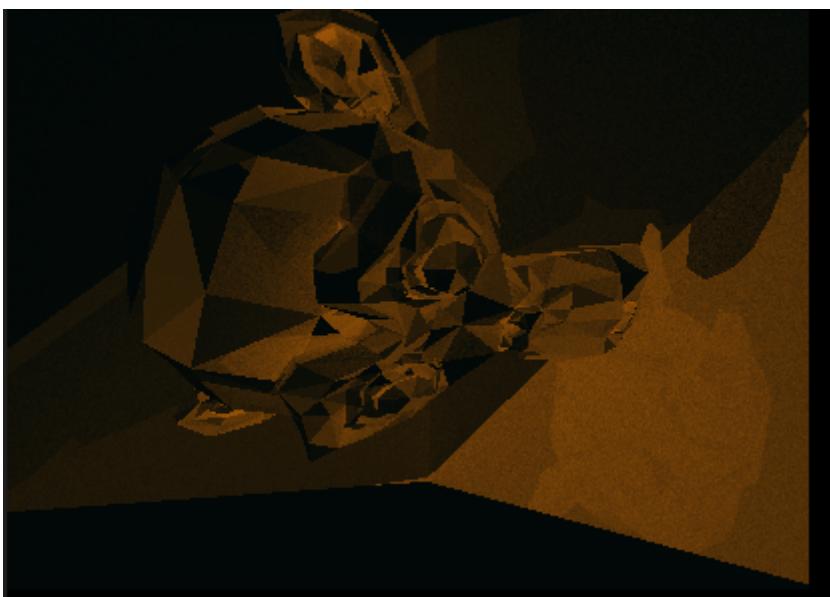
- Množstvo
- Multipass
- Sila kontrastu



*Image - 5 - \2*

### **Tint**

- Množstvo
- Multipass
- Tint farba
- Sila tint shaderu Tint

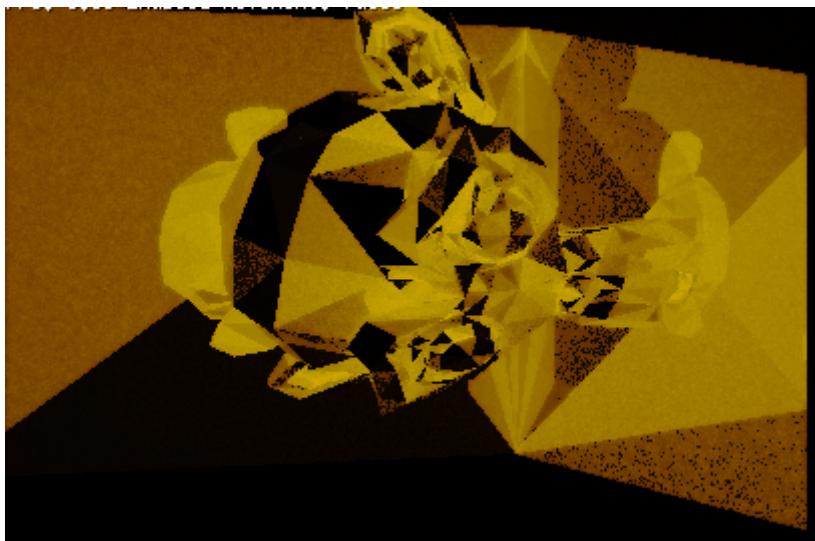


*Image - 6 - \2*

### **Bloom**

- Množstvo
- Multipass

- Prahová hodnota
- Intenzita



*Image - 7 - \2*

### **BloomV2**

Podobné Bloomu s miernym variantom BloomV2



*Image - 8 - \2*

### **Ostrost'**

- Množstvo
- Multipass
- Sila filtra

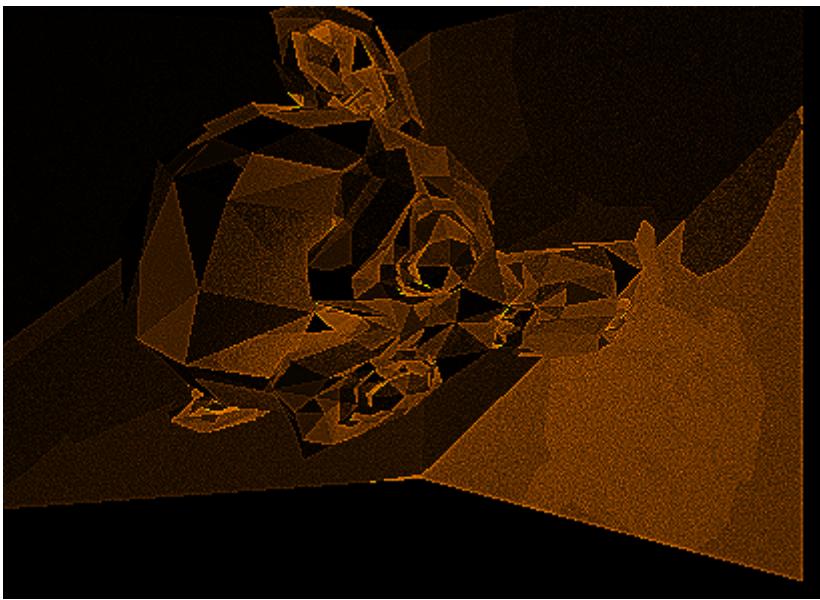


Image - 9 - \2

### Mapovanie Farieb

- Množstvo
- Multipass
- Farebné kanály (R/G/B)
- Definuje distribúciu farieb (napr. 2 úrovne: 0% alebo 100%)

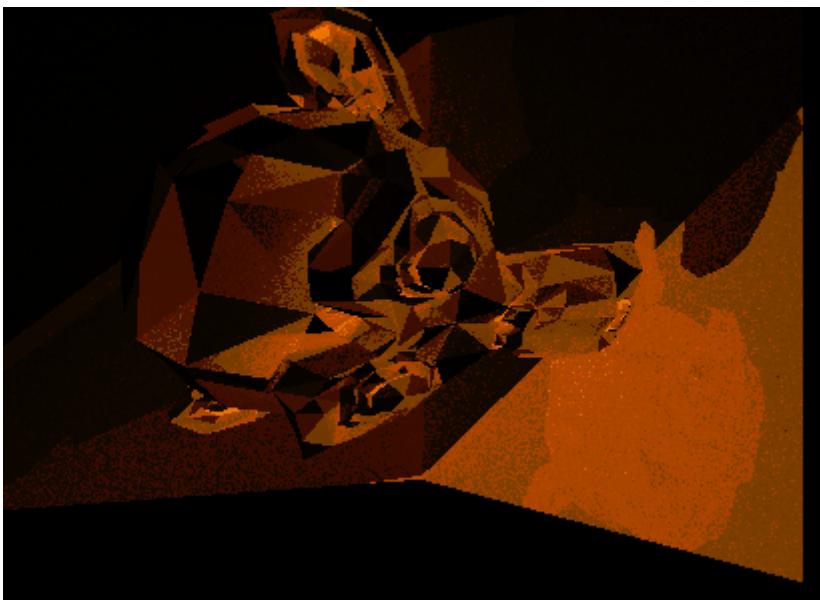
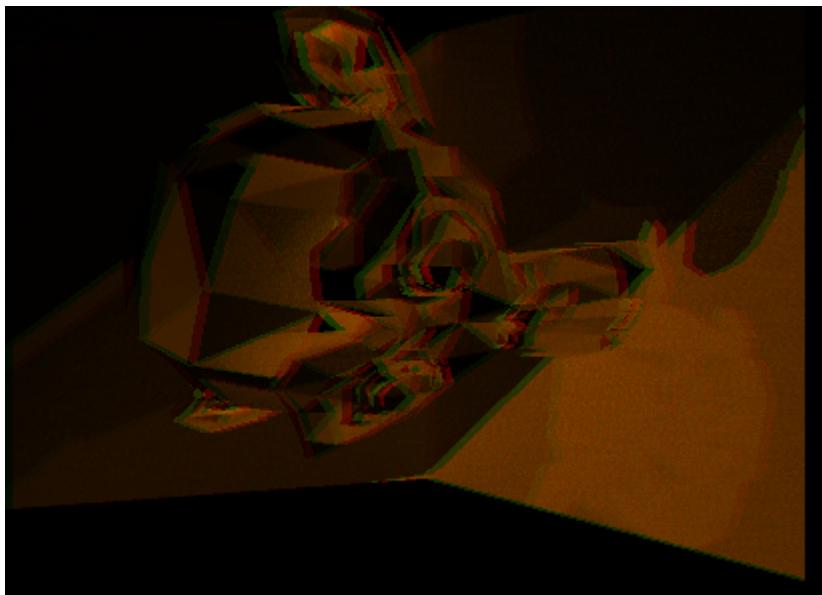


Image - 10 - \2

### Chromatická Aberácia

- Množstvo
- Multipass
- Sila filtra

- Posun farebného kanála (Červená vľavo, Modrá vpravo) image

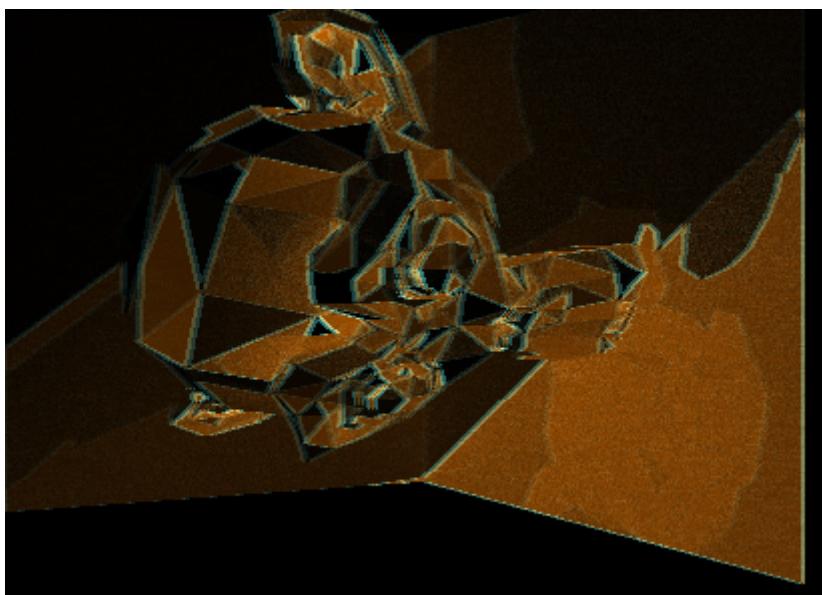


*Image - 11 - \2*

### **Detekcia Hrán**

Používa Sobelov filter

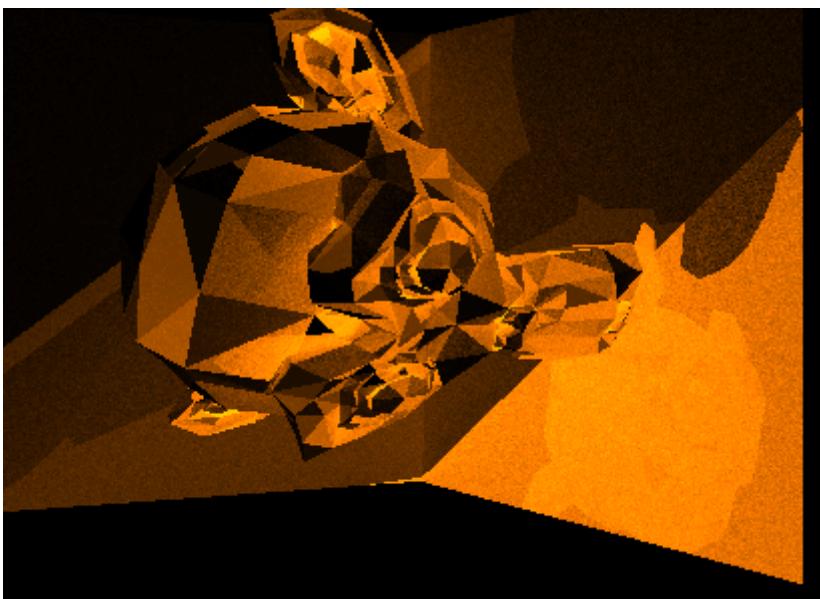
- Množstvo
- Multipass
- Sila zvýraznenia hrán
- Nastaviteľná farba hrán (R/G/B) image



*Image - 12 - \2*

### **Zosvetlenie**

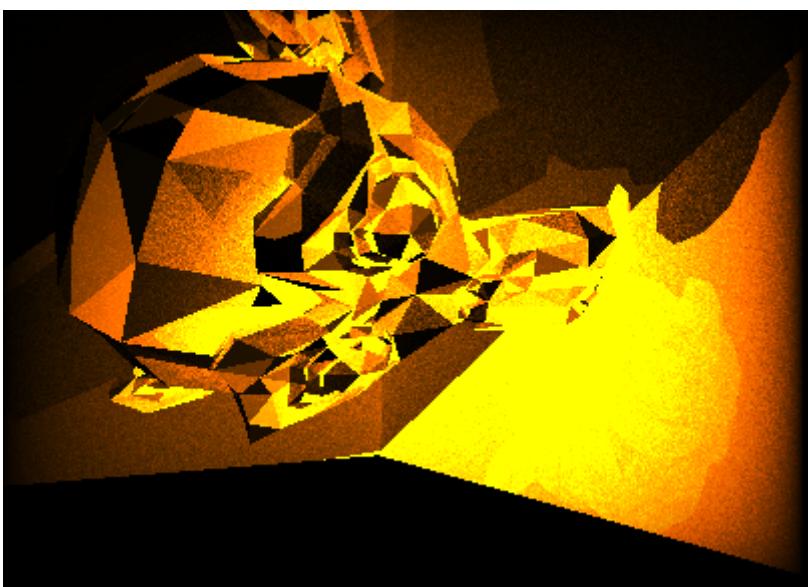
- Množstvo
- Multipass
- Sila filtra image



*Image - 13 - \2*

### Vignette

- Množstvo
- Multipass
- Base
- Glow
- Radius image

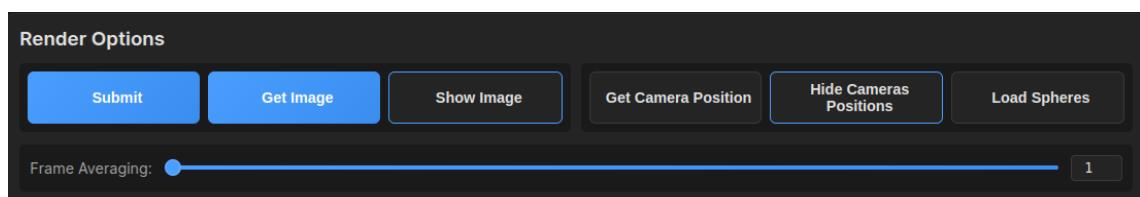


*Image - 14 - \2*

## 2.3.4 Render Options

### Horná Lišta

- Odoslať Render Možnosti
- Tlačidlo Získať Pozíciu Kamery
- Skryť/Zobraziť Pozíciu Kamery
- Tlačidlo Získať Vyrendrovaný Obrázok
- Tlačidlo Ukázať Vyrendrovaný Obrázok
- Tlačidlo Načítať SDF Objekty
- Slider na určenie snímkov, z ktorých sa vytvorí obrázok



*Image - 15 - \2*

### Menu Pozície Kamery

- V danom menu je možné vidieť získané pozície a presunúť kameru na danú pozíciu alebo vytvoriť animáciu medzi viacerými pozíciami

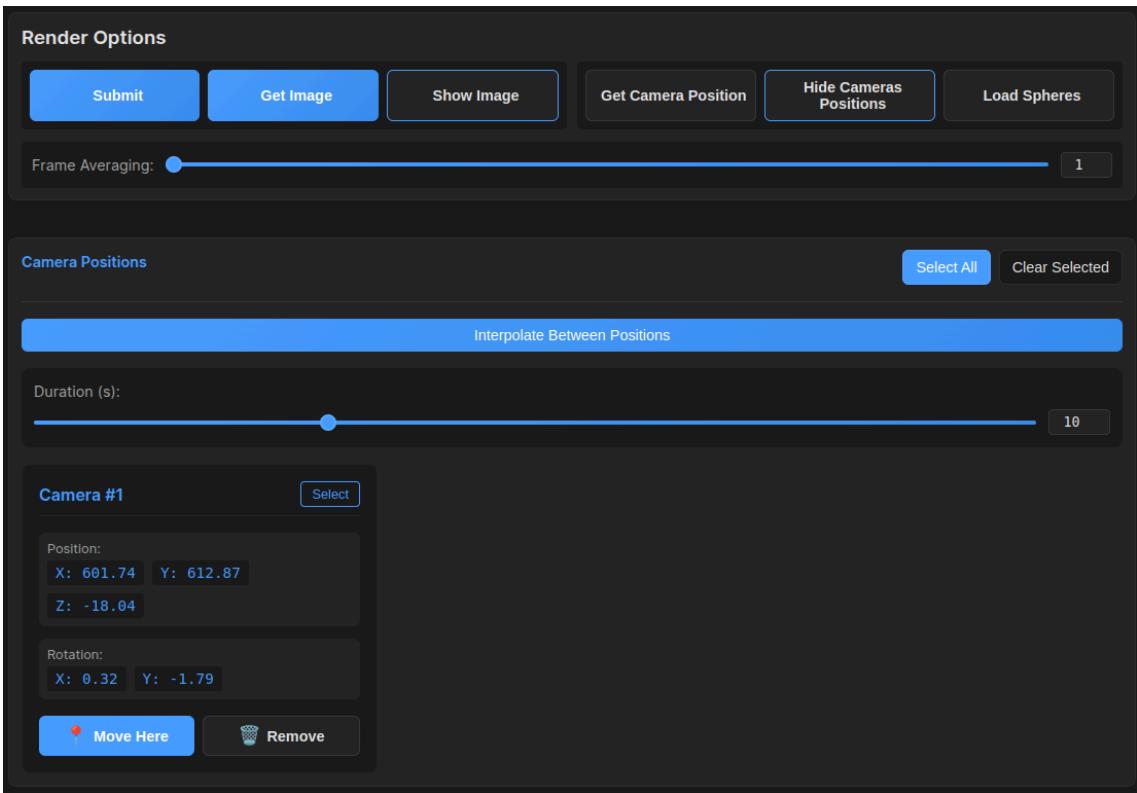
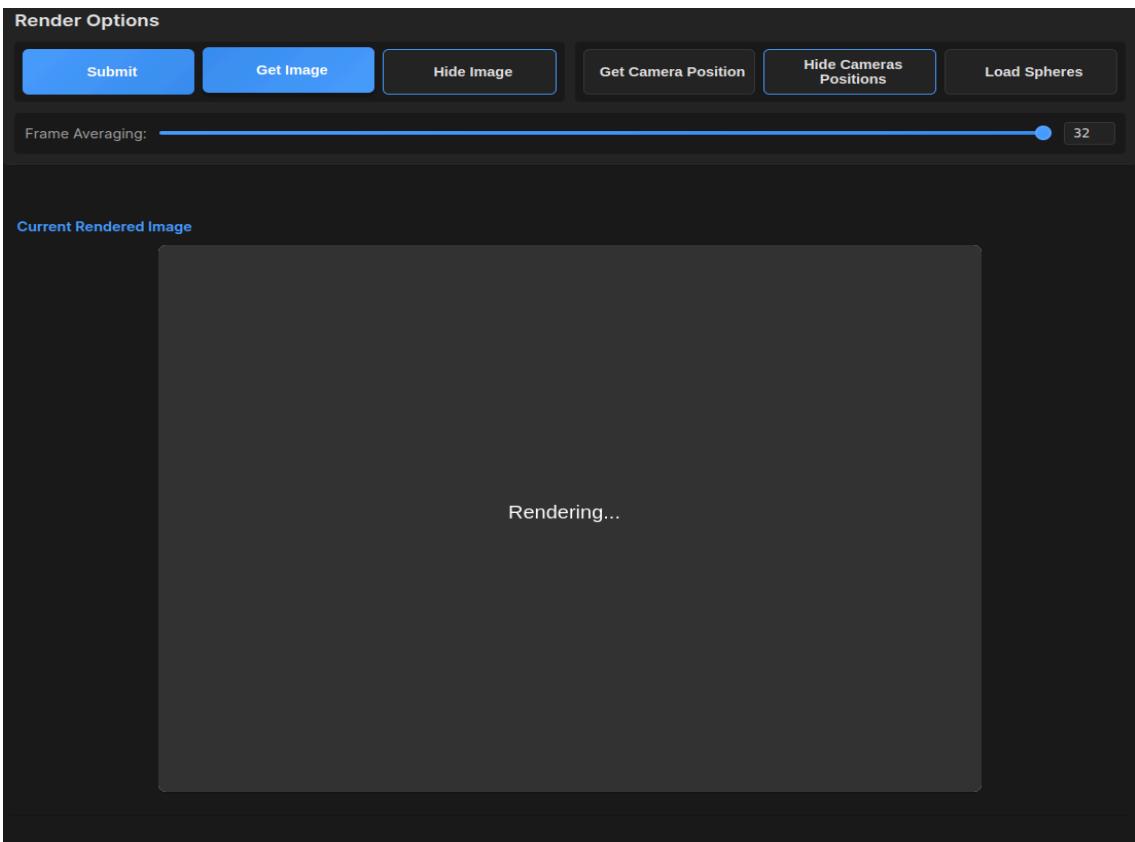


Image - 16 - \2

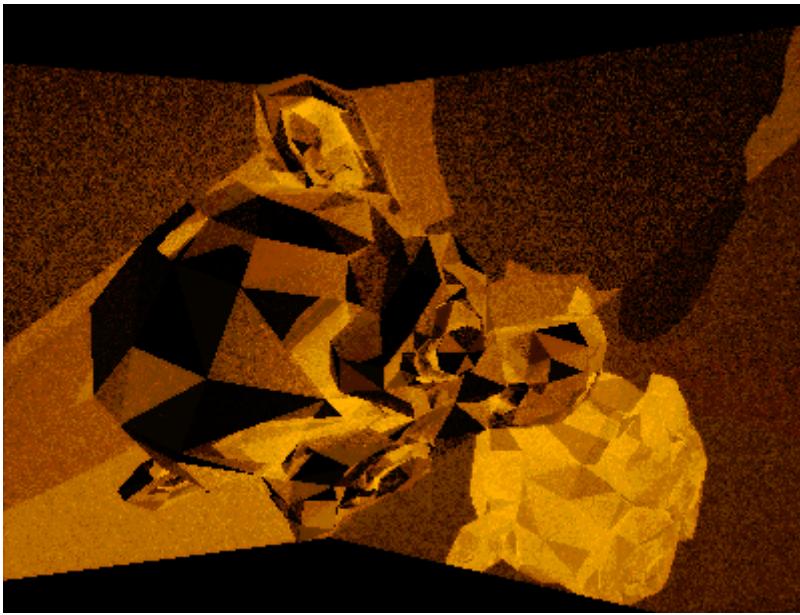
### Menu Ukážky Render-a

- Menu slúžiace na zobrazenie vyrendrovaného obrázku



*Image - 17 - \2*

- Jeden obrázok, z ktorého je vykonaný render, je viac šumový



*Image - 18 - \2*

- 32 obrázkov, ktoré sú spriemernené do jedného obrázku

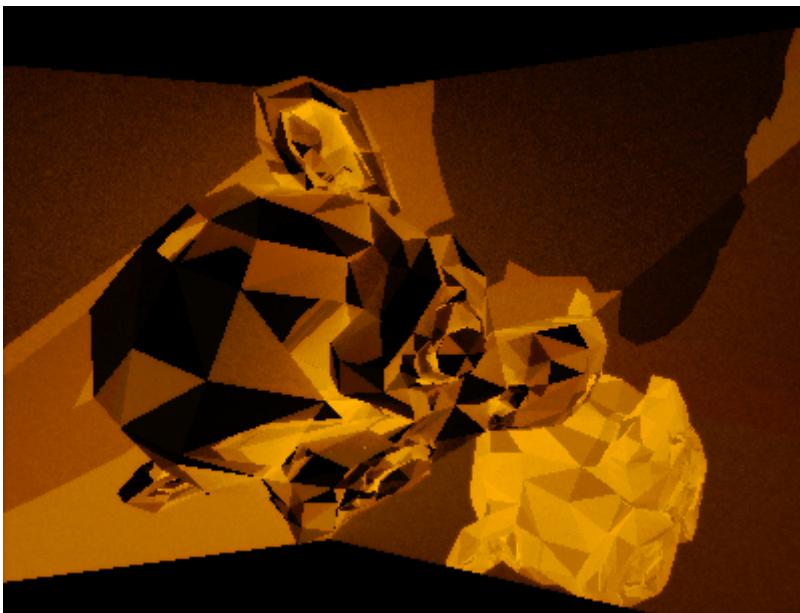


Image - 19 - \2

### Hlavné Parametre Renderingu

- **Hĺbka:** Počet odrazov na renderovanie
- **Rozptyl:** Počet lúčov rozptýlených z povrchu (zvyšuje detail)
- 

### Parametre Osvetlenia

- Intenzita Svetla
- Farba Svetla (R/G/B)
- Zorné Pole
- Gama: Kontrast a jas medzi tmavými a svetlými tónmi

### Nastavenia Renderingu

- Pripnúť Svetlo ku Kamere
- Raymarching
- Performance Mód
  - Odobratie wg.Wait
  - Potenciálne menej plynulý rendering
  - Maximalizácia využitia hardvéru
- Rozlíšenie
  - Natívne (aktuálne neimplementované)
  - 2X
  - 4X
  - 8X

## Verzia RayMarchingu

- V1 - Využíva BVH pre efektívnejšie rendrovanie

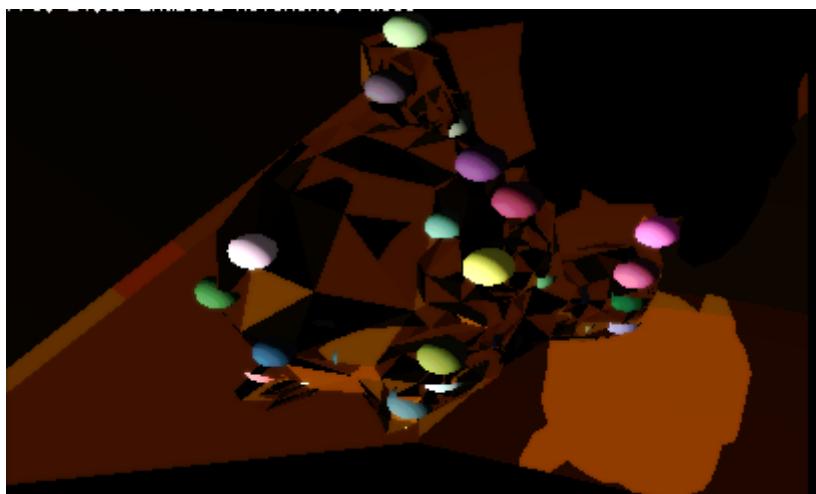


Image - 20 - \2

- V2 - Umožňuje meniť radius alebo SDF funkciu

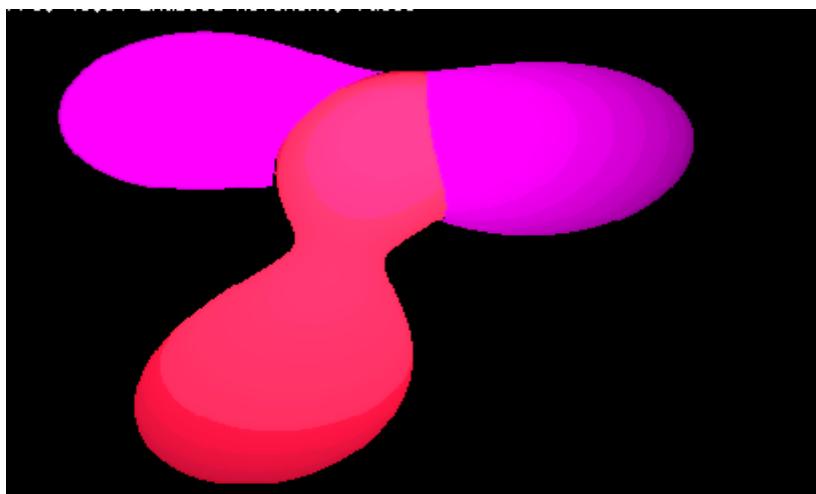
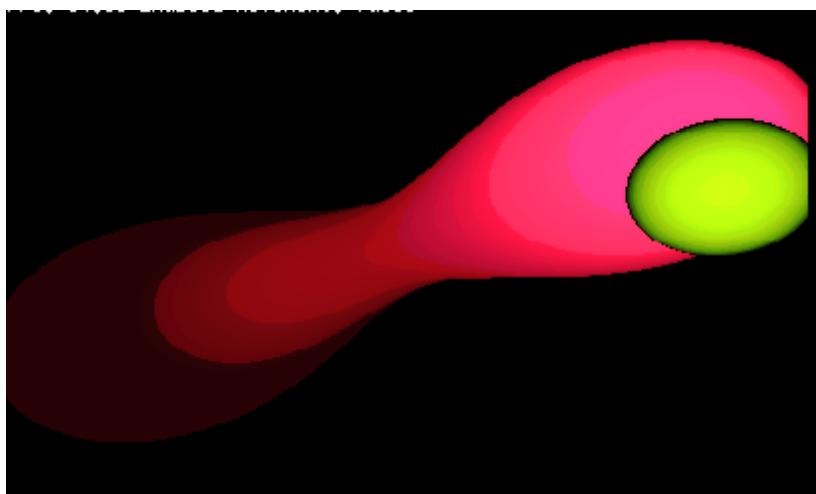
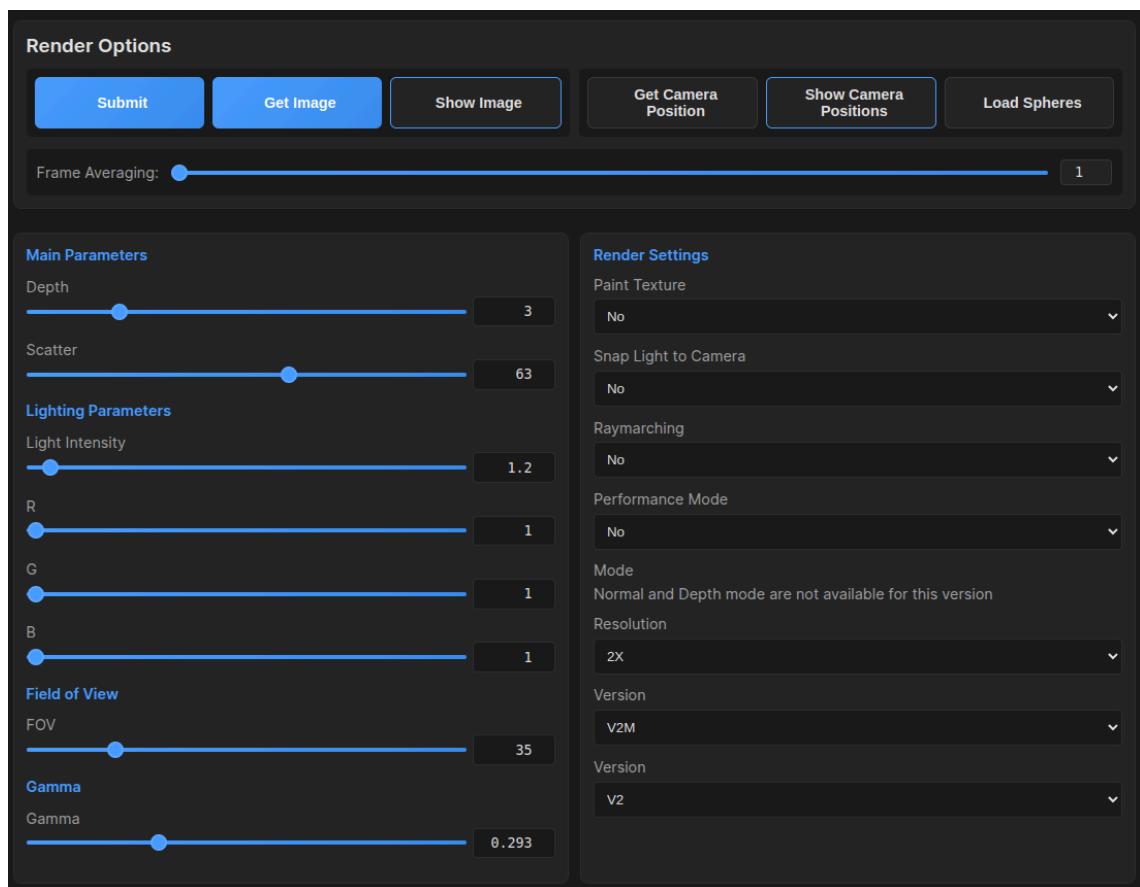


Image - 21 - \2



## Módy Renderingu

- Klasický: Štandardné renderovanie
- Normál: Renderovanie normálových povrchov (V2Log, V2Lin, V2LogTexture, V2LinTexture, V4Log, V4Lin, V4LinOptim, V4LogOptim, V4LinOptim-V2, V4LogOptim-V2, V4Optim-V2)
- Vzdialenosť: Momentálne nesprávne implementované



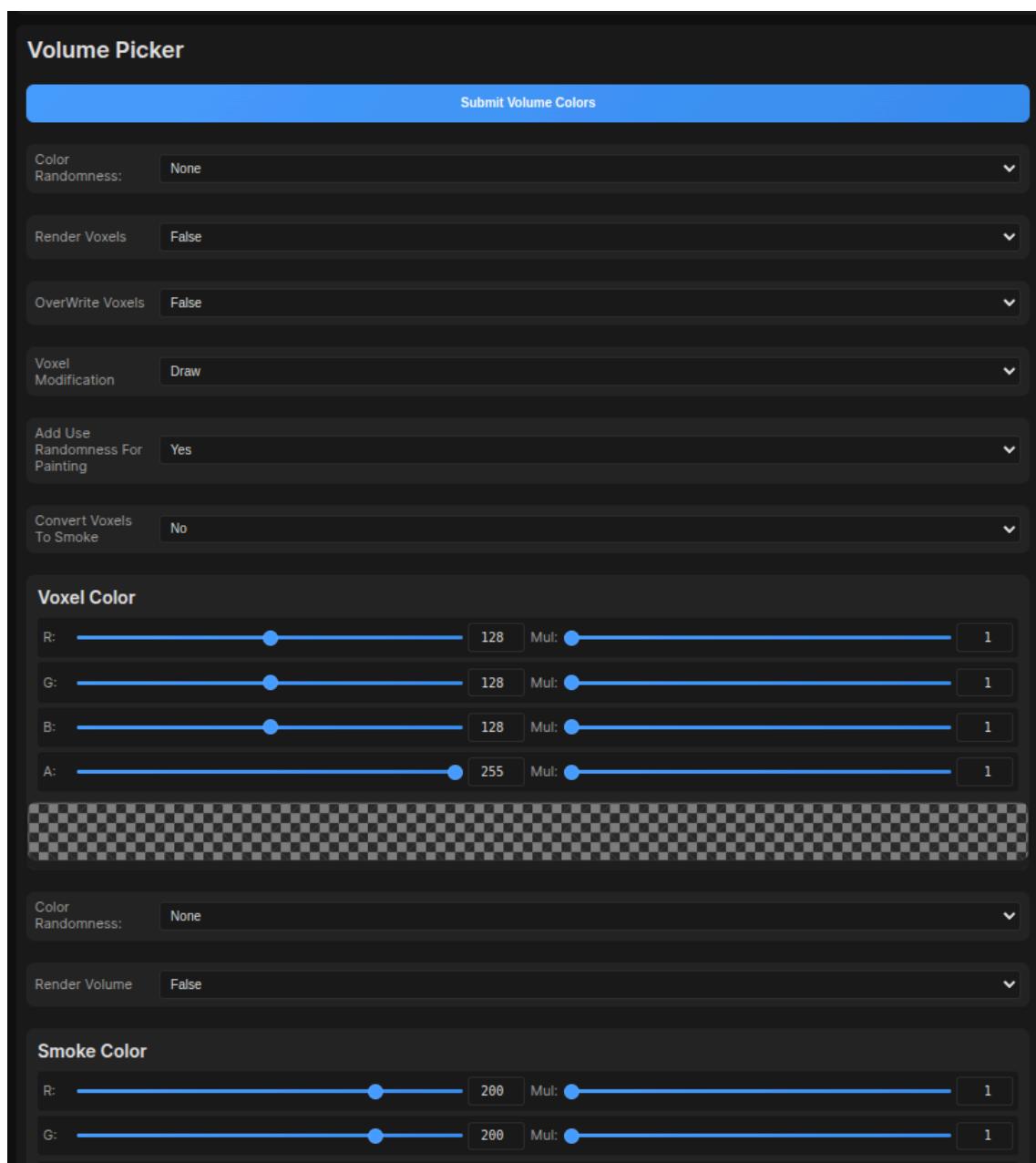
## 2.3.5 Volume Picker

## Správa Farby Objemu

- Odoslat' Farby Objemu
- Náhodnosť Farby
- Prepínač Renderingu Voxelov
- Prepísat' Voxely
- Pridať Náhodnosť do Maľovania
- Konvertovať Voxely na Dym (rendering objemu ako dym, sklo)

## Vlastnosti Objemu

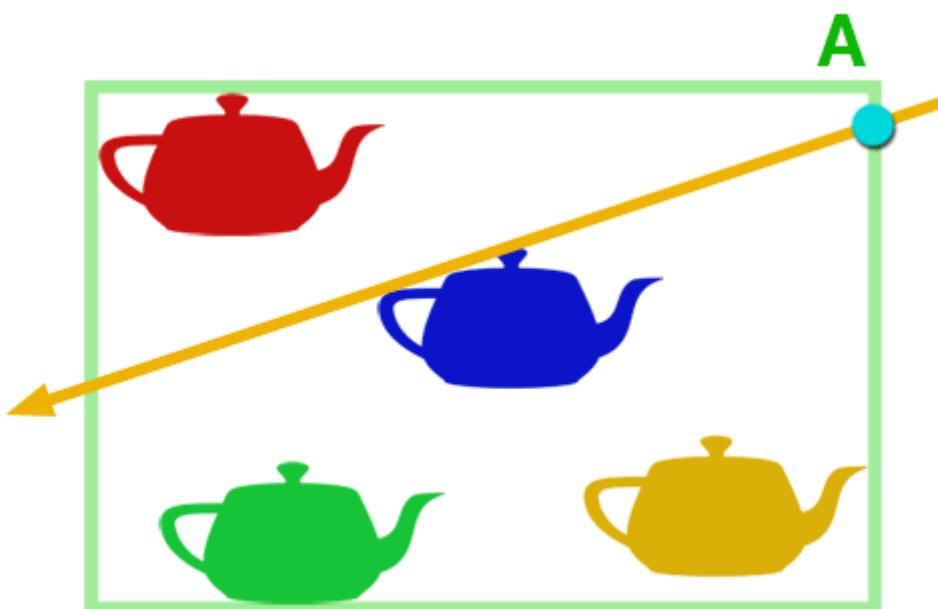
- Výber Farby Voxelov s Náhl'adom
- Výber Farby Dymu
- Hustota
- Priehl'adnosť (priehl'adnosť objemu)



### 3.0 Princíp fungovania BVH

Pri ray-tracingu je kľúčovou operáciou hľadanie priesecníkov medzi lúčom vyslaným z kamery a objektmi v scéne. Bez optimalizačnej štruktúry by bolo potrebné testovať každý lúč s každým objektom v scéne, čo by viedlo k časovej zložitosti  $O(n)$  pre každý lúč, kde  $n$  je počet objektov v scéne. BVH rieši tento problém vytvorením hierarchickej štruktúry obalujúcich objemov (najčastejšie osovo zarovnaných boxov - AABB), ktorá umožňuje rýchlo eliminovať veľké časti scény, ktoré lúč nemôže zasiahnuť.

Ked' lúč prechádza scénou, najprv sa testuje prienik s head Node BVH. Ak lúč nezasiahne obalujúci objem uzla, môžeme okamžite preskočiť všetky objekty v tomto podstrome. Ak prienik existuje, algoritmus rekurzívne pokračuje do potomkov uzla, až kým nedosiahne listové uzly obsahujúce konkrétné objekty scény.



### 3.2 Surface Area Heuristic (SAH)

Pre optimálny výkon BVH je kľúčové, ako sa scéna rozdelí na podpriestory. Tu prichádza do hry Surface Area Heuristic (SAH). Táto heuristika optimalizuje rozdelenie objektov medzi children každej Node na základe plochy ich objemov. Cieľom je minimalizovať očakávaný čas potrebný na prechádzanie stromom a testovanie prienikov.

SAH pracuje na princípe, že pravdepodobnosť, že lúč zasiahne daný objem, je približne úmerná jeho povrchu. Pri delení uzla sa teda snažíme minimalizovať funkciu:

$$C = Ct + (SA(L)/SA(P)) * NL * Ci + (SA(R)/SA(P)) * NR * Ci$$

kde:

**Ct** je cena prechodu cez Node

**Ci** je cena testovania prieniku s objektom

**SA(X)** je plocha povrchu objemu

- NL a NR sú počty objektov v ľavom a pravom potomkov
- L , R , P označujú ľavého potomka, pravého potomka a parent Node

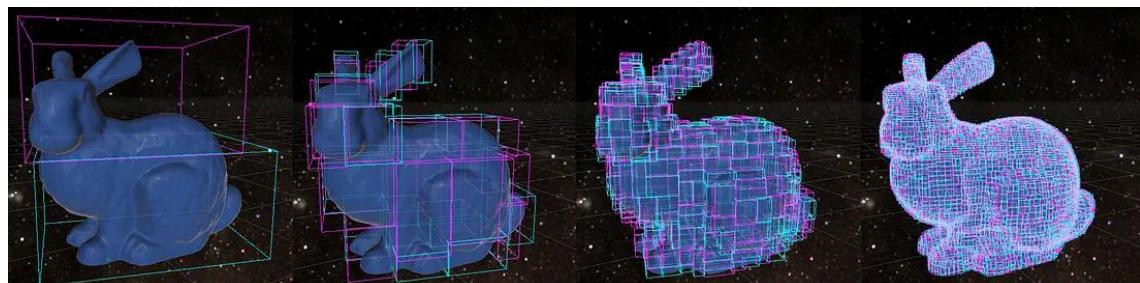


Image - 26 - 12

### 3.3 Reprezentácia Trojuholníkov a Materiálové Vlastnosti

Základným stavebným prvkom 3D scény v implementovanom ray-traceri je trojuholník, ktorý je reprezentovaný štruktúrou TriangleSimple. Táto štruktúra kombinuje geometrické vlastnosti trojuholníka s jeho materiálovými charakteristikami, čo umožňuje realistické zobrazenie rôznych povrchov a materiálov.

### 3.3.1 Geometrická Reprezentácia

```
type TriangleSimple struct {  
  
    v1, v2, v3 Vector           // Vrcholy trojuholníka  
  
    Normal        Vector       // Normálový vektor  
  
    // ... materiálové vlastnosti  
  
}
```

Geometria trojuholníka je definovaná troma 3D vektormi ( $v_1, v_2, v_3$ ), ktoré predstavujú jeho vrcholy v priestore. Pre optimalizáciu výkonu je súčasťou štruktúry aj predpočítaný normálový vektor (Normal). Tento prístup významne urýchľuje proces renderovania, keďže normál Vector je kľúčová pri výpočtoch osvetlenia a nie je potrebné ju opakovane počítať pri každom prieniku lúča s trojuholníkom.

### 3.3.2 Materiálové Vlastnosti

Materiálové vlastnosti trojuholníka sú reprezentované niekoľkými kľúčovými parametrami, ktoré určujú jeho vizuálne charakteristiky:

#### a) Farba (color ColorFloat32)

```
type ColorFloat32 struct {
```

```
R, G, B, A float32
```

```
}
```

Farba povrchu je reprezentovaná pomocou vlastnej štruktúry ColorFloat32, ktorá využíva pre každý farebný kanál (červený, zelený, modrý) a alfa kanál hodnoty typu float32. Toto riešenie prináša niekoľko klúčových výhod oproti tradičnej RGBA reprezentácii (uint8):

#### 1. **Vysoký Dynamický Rozsah (HDR):**

- Na rozdiel od štandardnej RGBA reprezentácie, kde je každý kanál limitovaný rozsahom 0-255 (uint8), float32 umožňuje reprezentovať hodnoty výrazne presahujúce hodnotu 1.0. Toto je esenciálne pre realistické zobrazenie emisívnych materiálov, ktoré môžu vyžarovať svetlo s intenzitou mnohonásobne vyššou než 1.0.

#### 2. **Emisívne Materiály:**

- ColorFloat32 umožňuje definovať materiály, ktoré aktívne emitujú svetlo do scény. Hodnoty vyššie ako 1.0 reprezentujú materiály, ktoré pridávajú energiu do scény.
- Toto je klúčové pre implementáciu svetelných zdrojov priamo ako súčasti geometrie scény.

#### 3. **Presnosť Výpočtov:**

- Float32 poskytuje vyššiu presnosť pri výpočtoch s farbami.
- Eliminuje sa problém kvantizácie, ktorý je typický pre uint8 reprezentáciu.
- Umožňuje jemnejšie prechody a gradienty v renderovanom obraze.

#### 4. **Fyzikálna Korektnosť:**

- Reprezentácia pomocou float32 lepšie zodpovedá fyzikálnej realite, kde intenzita svetla nie je zhora obmedzená
- Umožňuje presnejšiu simuláciu svetelných interakcií v scéne

Táto implementácia je kľúčová pre dosiahnutie fotorealistického renderovania, keďže umožňuje pracovať s realistickými

svetelnými podmienkami a materiálmi, ktoré by nebolo možné reprezentovať v štandardnom 8-bitovom farebnom priestore.

Zároveň poskytuje základ pre implementáciu pokročilých renderovacích techník ako HDR rendering a tone mapping.

## 5. Direct-to-Scatter Ratio (directToScatter float32)

Tento parameter, definovaný v rozsahu [0, 1], určuje pomer medzi priamym odrazom svetla a difúznym rozptylom:

- Hodnota blízka 0: Väčšina svetla je rozptylená náhodným smerom (matný povrch)
- Hodnota blízka 1: Prevláda priamy odraz svetla (lesklý povrch) Tento parameter je kľúčový pre realistické zobrazenie

## 6. Reflection Coefficient (reflection float32) Koeficient odrazu, definovaný v rozsahu [0, 1], určuje, ako silno povrch odráža

okolité prostredie:

- 0: Žiadne odrazy okolitého prostredia
- 1: Dokonalé zrkadlové odrazy Tento parameter ovplyvňuje pomer medzi vlastnou farbou objektu a farbou odrazenou z

okolia, čo umožňuje simulať materiály od úplne matných až po zrkadlové povrhy.

7. **Specular Intensity (specular float32)** Parameter v rozsahu [0, 1] určuje intenzitu spekulárneho odrazu:

- 0: Žiadny spekulárny odraz
- 1: Maximálny spekulárny odraz Tento

### 3.4 Nová Implementácia BVHLean

V novej implementácii BVHLean je štruktúra trojuholníka významne zjednodušená:

#### Pôvodná Štruktúra TriangleSimple

```
type TriangleSimple struct {
    // size=88 (0x58)
    v1, v2, v3 Vector
    // color color.RGBA
    color ColorFloat32
    Normal Vector
    reflection float32
    directToScatter float32
    specular float32
    Roughness float32
    Metallic float32
    id uint8
}
```

*Image - 27 - \2*

#### Nová Štruktúra TriangleBBOX

```
type TriangleBBOX struct {
    // size=52 (0x34)
    V1orBBoxMin, V2orBBoxMax, V3 Vector
    normal Vector
    id int32
}
```

*Image - 28 - \2*

Kľúčové zmeny:

- Veľkosť štruktúry sa zmenšila z 88 na 52 bajtov
- Veľkosť štruktúry sa zmenšila z 88 na 52 bajtov

- Vlastnosti trojuholníka sú teraz definované samostatne
- Zjednotenie bounding boxu a trojuholníka

## Nová Štruktúra Textúry

```
type Texture struct {
    texture [128][128]ColorFloat32
    normals [128][128]Vector

    // Materiálové vlastnosti
    reflection      float32
    directToScatter float32
    specular        float32
    Roughness       float32
    Metallic        float32
}
```

*Image - 29 - \2*

Táto nová implementácia umožnila zrýchlenie BVH o:

- 18 % na procesore Ryzen 9 5950X
- Systém s 72 GB RAM

Táto optimalizácia zjednodušuje štruktúru dát a umožňuje efektívnejšiu prácu s pamäťou počas ray-tracingu.

### 3.5 BVH a jej implementácia

V procese optimalizácie ray-tracingu bola implementácia efektívnej akceleračnej štruktúry klíčovým faktorom pre zlepšenie výkonu. Evolúcia riešenia prešla niekoľkými fázami:

**Vývojová cestaprístup** - Pôvodná implementácia testovala prienik lúča s každým trojuholníkom v scéne, čo viedlo k lineárnej časovej zložitosti  $O(n)$  a výrazne limitovalo výkon pri rastúcom počte trojuholníkov.

1. **Bounding Box optimalizácia** - Ako prvý krok optimalizácie boli implementované ohraničujúce boxy (Bounding Boxes) pre skupiny trojuholníkov, čo umožnilo rýchlejšie vylúčenie objektov mimo lúča. Toto zlepšenie však stále nebolo dostatočné pre komplexné scény.
2. **BVH implementácia** - Finálnym riešením bola implementácia Bounding Volume Hierarchy (BVH), ktorá hierarchicky organizuje priestor a umožňuje efektívne prechádzanie len relevantných častí scény, čím znížuje časovú zložitosť na približne  $O(\log n)$

### 3.5.1 Evolúcia BVH štruktúry

#### Pôvodná BVH implementácia

```
type BVHNode struct { // veľkosť=136  
    (0x88) bajtov Left, Right *BVHNode
```

BoundingBox [2]Vector

Triangles

TriangleSimple

active

bool

}

Prvá verzia BVH používala štandardnú stromovú štruktúru s ukazovateľmi na ľavý a pravý podstrom. Táto implementácia však trpela rastúcou veľkosťou uzlov kvôli pridávaniu materiálových vlastností a normálových vektorov pre trojuholníky.

### 3.5.2 Optimalizovaná BVHLean

```
type BVHLeanNode struct { // veľkosť=72  
(0x48) bajtov Left, Right *BVHLeanNode
```

TriangleBBOX TriangleBBOX

|        |      |
|--------|------|
| active | bool |
|--------|------|

}

```
type TriangleBBOX struct { // veľkosť=52  
(0x34) bajtov V1orBBoxMin,  
V2orBBoxMax, V3 Vector
```

|        |        |
|--------|--------|
| normal | Vector |
|--------|--------|

|    |       |
|----|-------|
| id | int32 |
|----|-------|

}

Table 2.

- Classic BVHNode : 407.888788ms
- BVHLean : 341.148485ms

## BVH Implementation Performance Comparison

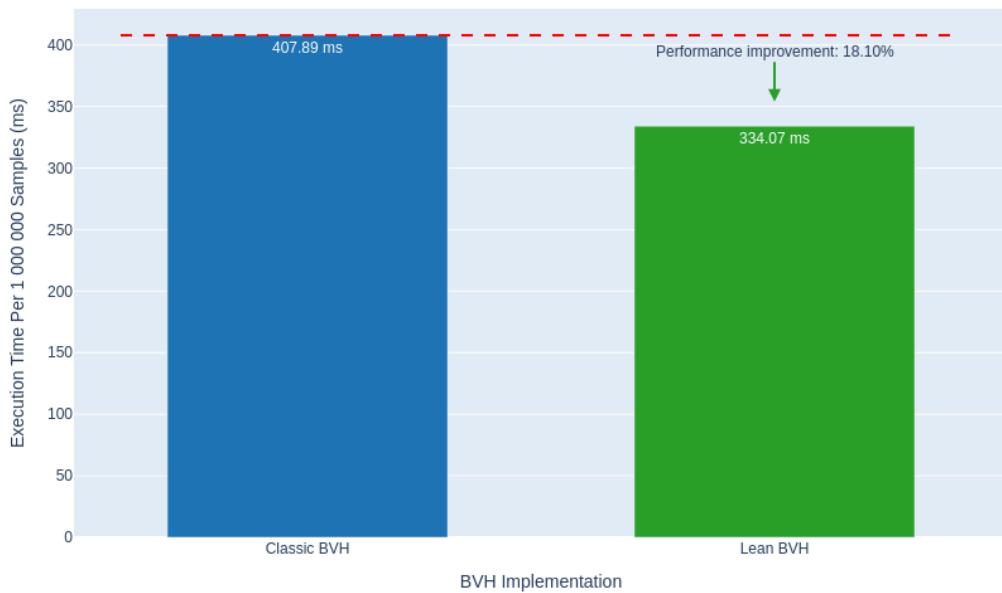


Image - 30 - 12

Pre verziu V4 bola vytvorená optimalizovaná implementácia BVHLean, ktorá:

- Zmenšila veľkosť uzla takmer na polovicu (zo 136 bajtov na 72 bajtov) Zlúčila ohraničujúci box a trojuholník do jednej štruktúry pre lepšiu lokalitu dát
- Odstránila priame ukladanie materiálových vlastností v BVHNode a nahradila ich systémom ID odkazov na textúry

### 3.5.3 Optimalizácia BVHLean - porovnanie 2 bounding boxov naraz

Kedže vždy musím pozerať intersekcii s oboma dvoma bounding boxami, je omnoho efektívnejšie porovnať intersekcii v jednej funkcií, takže sa dá vyhnúť počiatočnej inverse direction

```

func BoundingBoxCollisionPair(box1Min, box1Max, box2Min, box2Max Vector, ray Ray) (bool, bool, float32, float32) {
    // Precompute the inverse direction (once for both boxes)
    invDirX := 1.0 / ray.direction.x
    invDirY := 1.0 / ray.direction.y
    invDirZ := 1.0 / ray.direction.z
    // Box 1 intersection
    tx1_1 := (box1Min.x - ray.origin.x) * invDirX
    tx2_1 := (box1Max.x - ray.origin.x) * invDirX
    tmin_1 := min(tx1_1, tx2_1)
    tmax_1 := max(tx1_1, tx2_1)
    ty1_1 := (box1Min.y - ray.origin.y) * invDirY
    ty2_1 := (box1Max.y - ray.origin.y) * invDirY
    tmin_1 = max(tmin_1, min(ty1_1, ty2_1))
    tmax_1 = min(tmax_1, max(ty1_1, ty2_1))
    tz1_1 := (box1Min.z - ray.origin.z) * invDirZ
    tz2_1 := (box1Max.z - ray.origin.z) * invDirZ
    tmin_1 = max(tmin_1, min(tz1_1, tz2_1))
    tmax_1 = min(tmax_1, max(tz1_1, tz2_1))
    // Box 2 intersection
    tx1_2 := (box2Min.x - ray.origin.x) * invDirX
    tx2_2 := (box2Max.x - ray.origin.x) * invDirX
    tmin_2 := min(tx1_2, tx2_2)
    tmax_2 := max(tx1_2, tx2_2)
    ty1_2 := (box2Min.y - ray.origin.y) * invDirY
    ty2_2 := (box2Max.y - ray.origin.y) * invDirY
    tmin_2 = max(tmin_2, min(ty1_2, ty2_2))
    tmax_2 = min(tmax_2, max(ty1_2, ty2_2))
    tz1_2 := (box2Min.z - ray.origin.z) * invDirZ
    tz2_2 := (box2Max.z - ray.origin.z) * invDirZ
    tmin_2 = max(tmin_2, min(tz1_2, tz2_2))
    tmax_2 = min(tmax_2, max(tz1_2, tz2_2))
    // Check intersections
    hit1 := tmax_1 >= max(0.0, tmin_1)
    hit2 := tmax_2 >= max(0.0, tmin_2)
    // Return hit status and distances
    return hit1, hit2, tmin_1, tmin_2
}

```

*Image - 31 - 12*

- Toto vylepšenie je výkonnejšie zhruba o 25.86%
- BoundingBoxCollisionVector: 291.248548ms
- BoundingBoxCollisionPair: 215.934921ms

### Bounding Box Collision Functions Performance Comparison

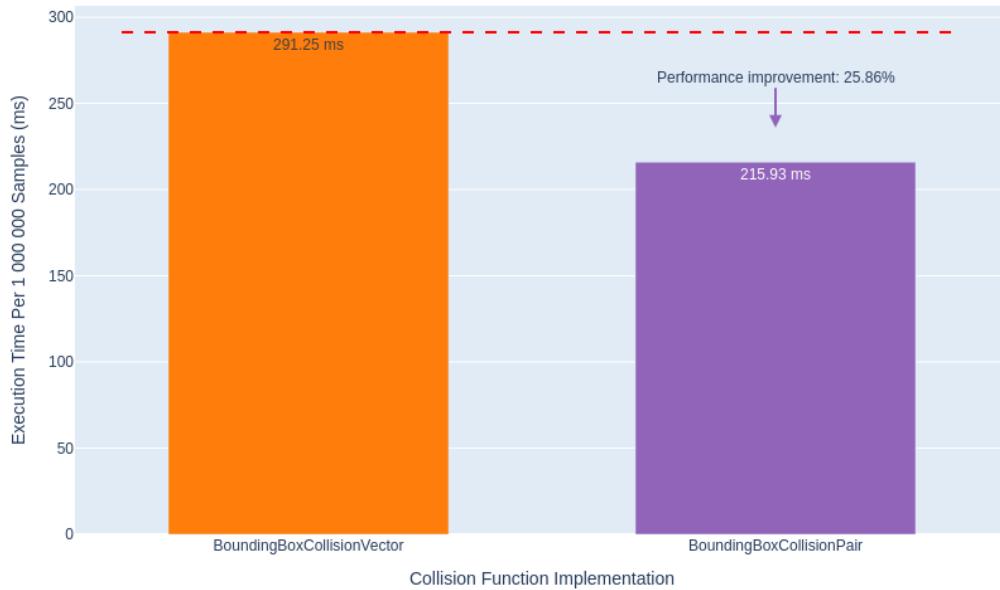


Image - 32 - 12

#### 3.5.4 Experimentálna array-based implementácia

```
type BVHArray struct { // veľkosť=65538508 (0x3e809cc) bajtov
```

```
    triangles [NumNodes]TriangleBBOX
```

```
    textures [128]Texture
```

```
}
```

V rámci ďalšej optimalizácie bola experimentálne vytvorená array-based reprezentácia BVH, kde:

- Uzly sú uložené v súvislom poli miesto rozptýlených alokácií
- Vzťahy medzi uzlami sú implicitné (ľavý potomok má index  $2n$ , pravý  $2n+1$ )  
Zlepšuje sa lokalita referencií a efektivita cache pamäte procesora

Testovanie preukázalo 21% zlepšenie výkonu oproti klasickej implementácii (218,831 ns/op vs. 278,146 ns/op) vďaka lepšiemu cache využitiu pri sekvenčnom prístupe k dátam.

## Výkonnostné výsledky

Implementácia Array-based BVH poskytla merateľné zlepšenie výkonu:

Klasická implementácia: 278,146 ns/op

Array-based implementácia: 218,831 ns/op

Zlepšenie: ~21%

Testovanie dostupné na: <https://github.com/DarkBenky/testBinaryTree>

Array-based implementácia zostala v experimentálnej fáze z dôvodu časových obmedzení projektu, ale predstavuje sľubný smer pre ďalší vývoj.

## 4.0 Podpora Načítavania 3D Geometrie

### 4.1 Načítavanie .OBJ Súborov

Implementovaný ray-tracer poskytuje robustnú podporu pre načítavanie 3D geometrie prostredníctvom štandardného .obj formátu, čo výrazne zvyšuje flexibilitu a použiteľnosť aplikácie.

#### Kľúčové vlastnosti implementácie

- Načítavanie priestorových vrcholov (vertices)
- Extrakcia normálových vektorov
- Podpora textúrovacích koordinát
- Konverzia polygónov na trojuholníkovú siet'

### 2. Podpora Materiálov

- Parsing .mtl súborov
- Načítavanie základných materiálových vlastností:

### 3. Optimalizačné Techniky

- Predpočítavanie normálových vektorov

- Efektívna konverzia na interný formát TriangleSimple
- Podpora pre zložitejšie geometrické útvary

## Proces Načítavania .OBJ Súborov

Proces načítavania .obj súborov zahŕňa niekoľko kľúčových krokov:

1. **Parsovanie priestorových súradníc vertices**
2. **Identifikácia a konverzia polygónov na trojuholníky**
3. **Priradenie materiálových vlastností jednotlivým geometrickým prvkom**

## 5.0 RayTracing Vývoj Funkcionality

Pôvodná funkcia, ktorá poskytuje základnú ray tracing funkcionalitu:

### TraceRay

#### Web Name : V1

- Podpora pre zložitejšie geometrické útvary
- Používa BVH štruktúru pre testy priesecníkov
- Vykonáva základný výpočet rozptýleného svetla pomocou cosine-weighted hemisphere sampling
- Počíta priame odrazy a zrkadlové body pomocou jednoduchého svetelného modelu
- Používa rekurzívny prístup pre hĺbkové odrazy
- Kombinuje priame svetlo, rozptýlené svetlo a odrazy lineárne



Image - 34 - 33 - 12

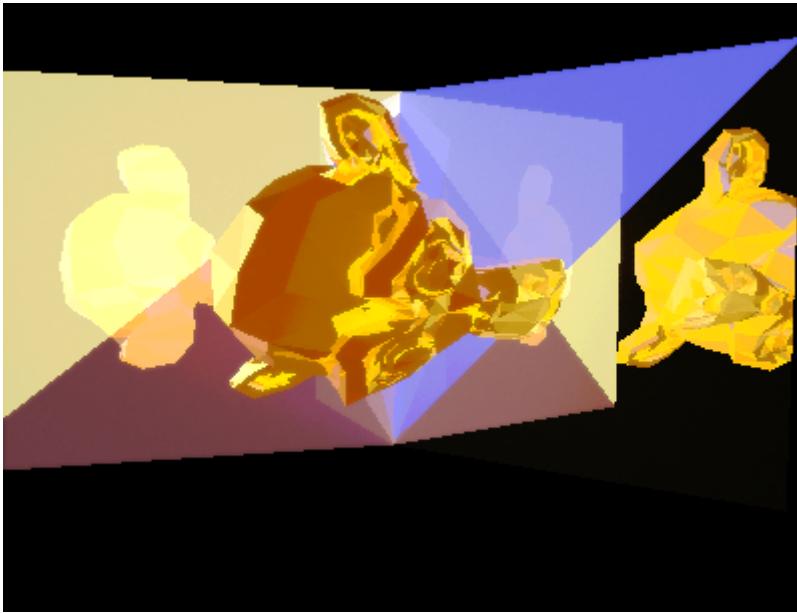


Image - 35 - 12

- V1 Profile

## TraceRayV2

### Web Name : V2

- Logickejšie organizuje kód, oddeluje priame osvetlenie, nepriame osvetlenie a odrazy
- Pridáva perturbáciu smerov odrazov založenú na drsnosti
- Používa hemisphere sampling so zlepšenou logikou rozptylu
- Lepšie spracováva medzi difúznym a priamim odrazením svetlom



Image - 36 - 12

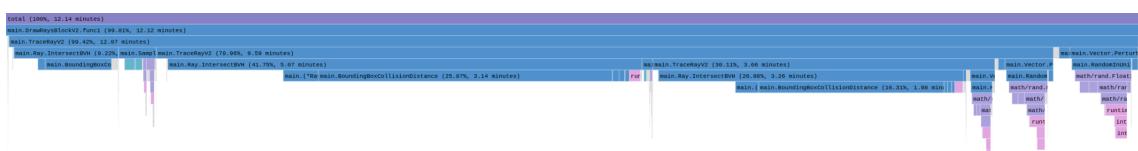


Image - 37 - 12

- V2 Profile

## TraceRayV3

## Web Name : V2M

PBR (Physically Based Rendering) prístup, ktorý:

- Implementuje Fresnel-Schlick aproximáciu pre výpočet odrazov
- Používa GGX distribúciu pre microfacet-based zrkadlové body
- Lepšie simuluje materiálové vlastnosti ako kovový lesk a drsnosť
- Používa presnejšiu energetickú konzerváciu pre kombinovanie komponentov
- Vracia jednu farebnú hodnotu
- 

## V2M

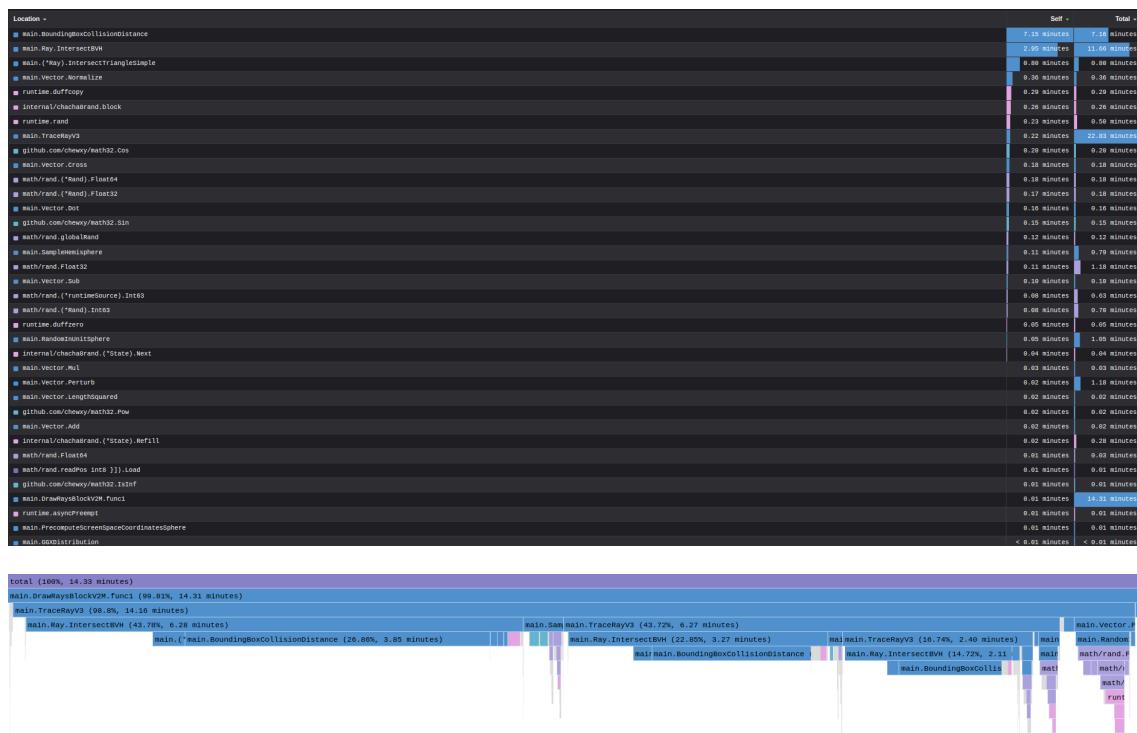
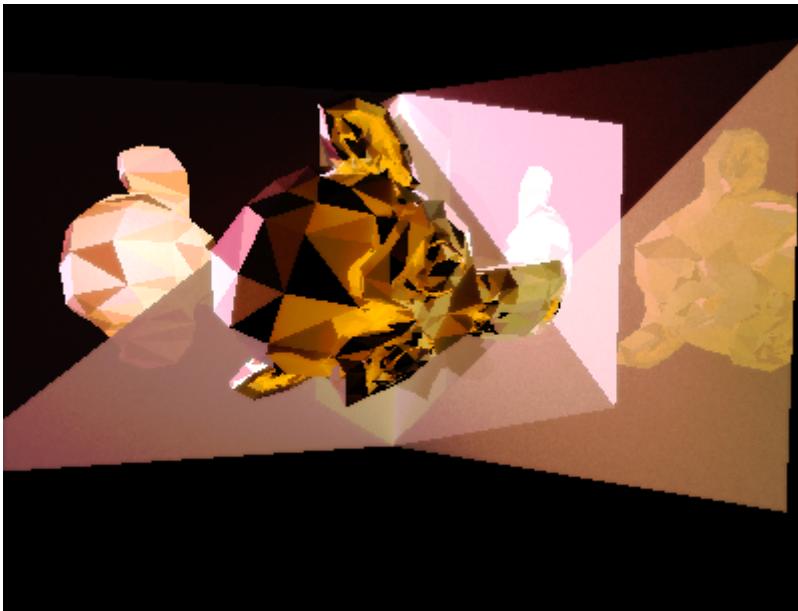


Image - 39 - 38 - |2



*Image - 40 - \2*

- [V2M Profile](#)

### **TraceRayV3Advance**

#### **Web Name : V2Liner / V2Log**

Rozšírenie TraceRayV3, ktoré:

Vracia dodatočné dátá: farbu,  
vzdialenosť a normálový vektor ktorý  
umožňuje pokročilejšie post-processing  
techniky Inak používa rovnaký PBR  
prístup ako TraceRayV3

#### **V2Lin**

| Location -                          | Self -       | Total -       |
|-------------------------------------|--------------|---------------|
| main.BoundingBoxCollisionDistance   | 7.12 minutes | 7.12 minutes  |
| main.Ray_IntersectBVH               | 3.49 minutes | 11.79 minutes |
| main.(~Ray).IntersectTriangleSimple | 0.62 minutes | 0.62 minutes  |
| main.Vector.Normalize               | 0.38 minutes | 0.38 minutes  |
| runtime.duffcopy                    | 0.29 minutes | 0.29 minutes  |
| internal/chacharand.block           | 0.27 minutes | 0.27 minutes  |
| runtime.rand                        | 0.21 minutes | 0.49 minutes  |
| github.com/chewy/math32.Cos         | 0.20 minutes | 0.20 minutes  |
| main.Vector.Cross                   | 0.18 minutes | 0.18 minutes  |
| math/rand.(~Rand).Float32           | 0.17 minutes | 0.17 minutes  |
| math/rand.(~Rand).Float64           | 0.17 minutes | 0.17 minutes  |
| main.Vector.Dot                     | 0.16 minutes | 0.16 minutes  |
| github.com/chewy/math32.Sin         | 0.15 minutes | 0.15 minutes  |
| main.TraceRay3dAdvance              | 0.13 minutes | 14.25 minutes |
| main.SampleHemisphere               | 0.12 minutes | 0.78 minutes  |
| math/rand.globalrand                | 0.11 minutes | 0.11 minutes  |
| math/rand.Float32                   | 0.11 minutes | 0.14 minutes  |
| main.Vector.Sub                     | 0.10 minutes | 0.10 minutes  |
| main.TraceRay3                      | 0.08 minutes | 0.71 minutes  |
| math/rand.(~Rand).Int63             | 0.08 minutes | 0.68 minutes  |
| math/rand.(~runtimeSource).Int63    | 0.07 minutes | 0.08 minutes  |
| main.RandomInUnitSphere             | 0.06 minutes | 1.03 minutes  |
| runtime.duffzero                    | 0.05 minutes | 0.05 minutes  |
| internal/chacharand.(~State).Next   | 0.04 minutes | 0.04 minutes  |
| github.com/chewy/math32.Pow         | 0.04 minutes | 0.06 minutes  |
| main.Vector.Perturb                 | 0.03 minutes | 1.16 minutes  |
| github.com/chewy/math32.max         | 0.03 minutes | 0.63 minutes  |
| github.com/chewy/math32.archExp     | 0.03 minutes | 0.03 minutes  |
| main.Vector.Mul                     | 0.03 minutes | 0.03 minutes  |
| main.DrawWaypointAdvance.func1      | 0.03 minutes | 14.39 minutes |
| github.com/chewy/math32.archLog     | 0.03 minutes | 0.03 minutes  |
| runtime.asyncPreempt                | 0.02 minutes | 0.02 minutes  |
| github.com/chewy/math32.isinf       | 0.02 minutes | 0.02 minutes  |
| github.com/chewy/math32.modf        | 0.02 minutes | 0.02 minutes  |
| internal/chacharand.(~State).Refill | 0.02 minutes | 0.28 minutes  |
| math/rand.readBox.int64.load        | 0.02 minutes | 0.02 minutes  |

Image - 41 - \2

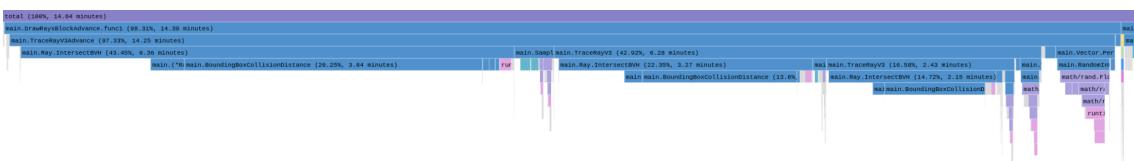


Image - 42 - \2

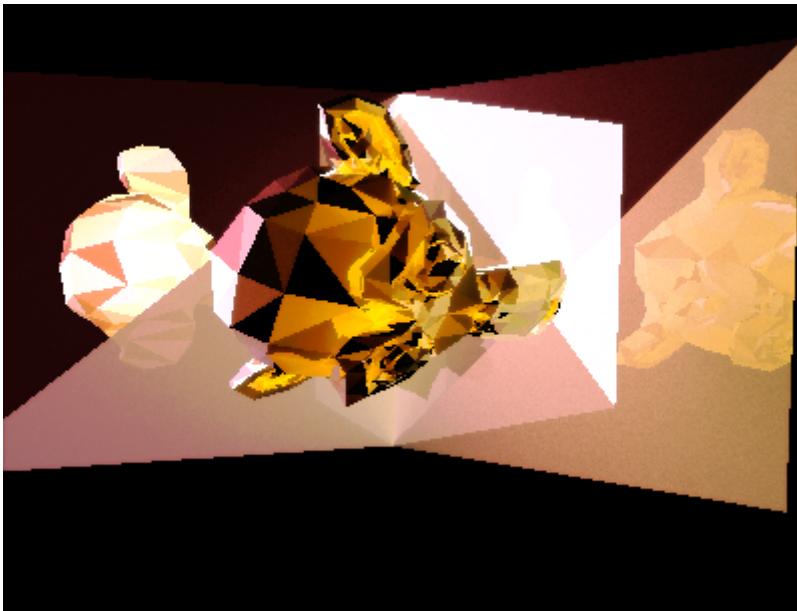


Image - 43 - \2

- V2Lin Profile

## V2Log

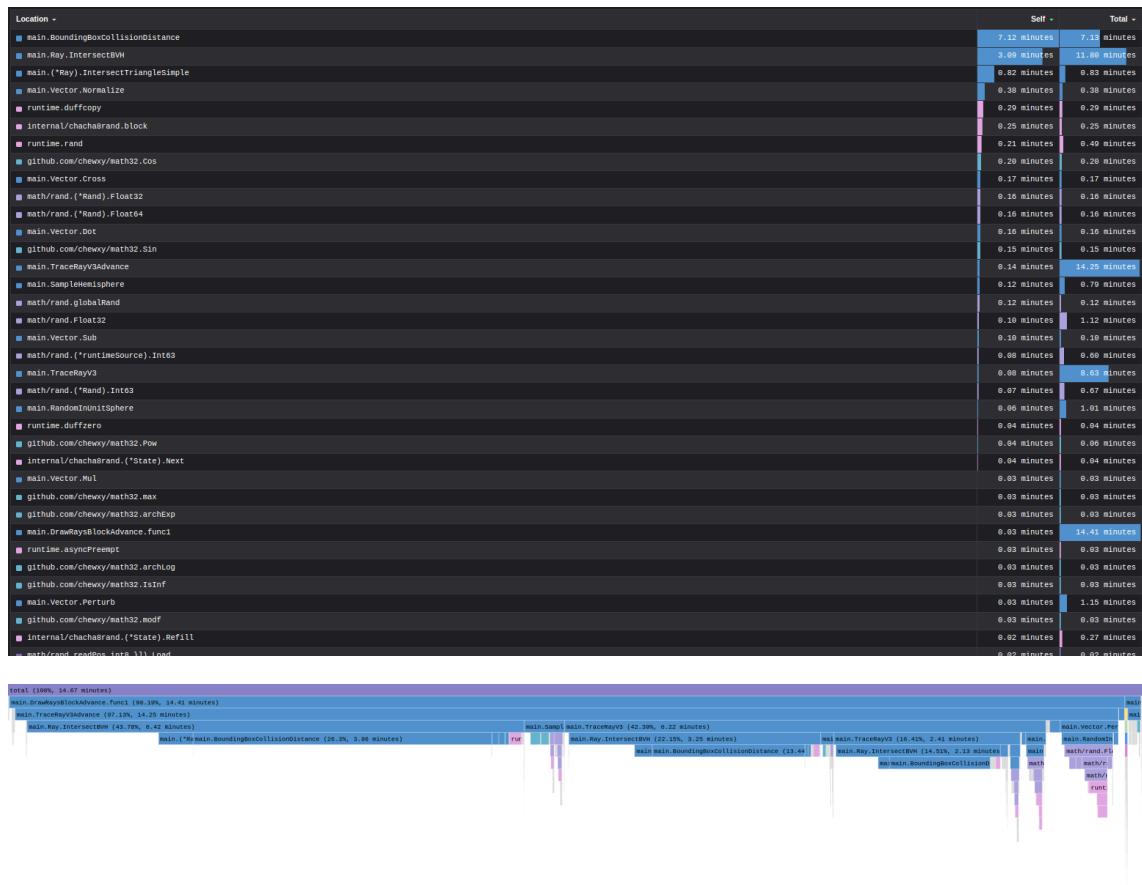


Image - 45 - 44 - \2

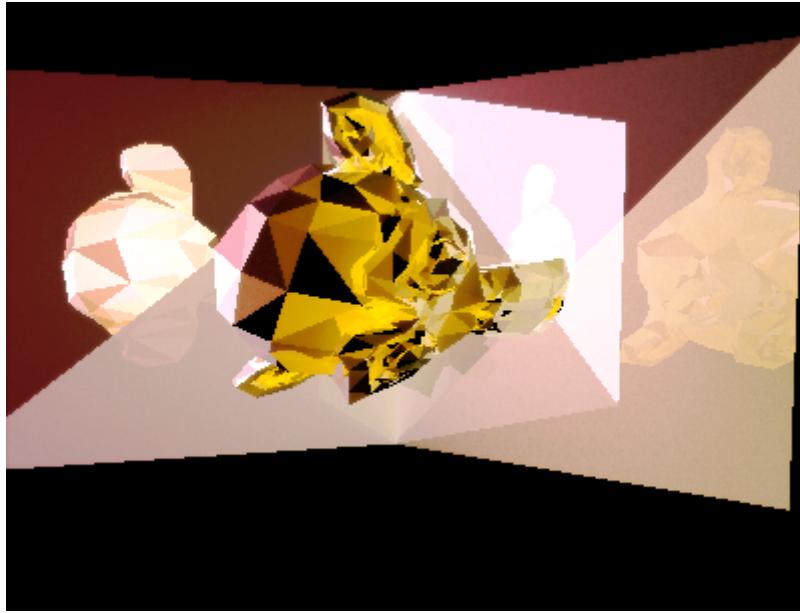


Image - 46 - \2

- V2Log Profile

## TraceRayV3AdvanceTexture

### Web Name : V2LinearTexture / V2LogTexture

Verzia s podporou textúr, ktorá:

- Integruje materiálové vlastnosti z textúr
- Používa textureMap parameter pre prístup k dátam textúr
- Vracia informácie o farbe a normále pre post-processing
- Používa špecializovaný BVH traversal funkciu pre podporu textúr
- Aplikuje dátá textúr na materiálové parametre ako drsnosť a kovový lesk

### V2LinTexture

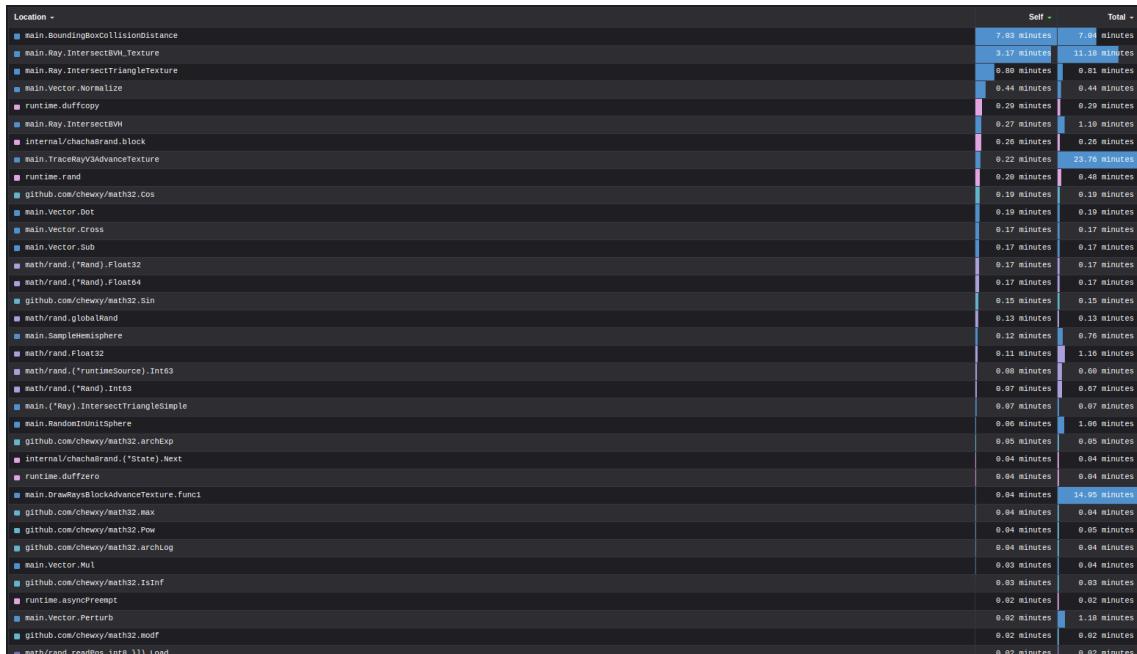


Image - 47 - \2

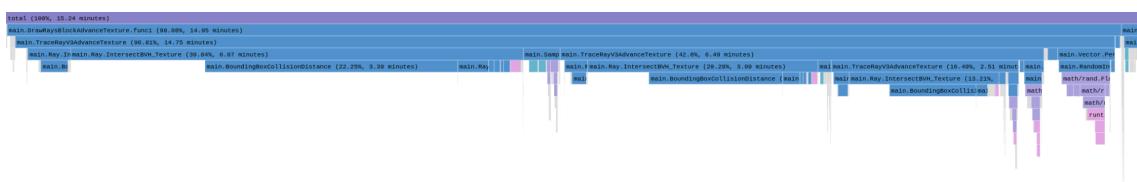


Image - 48 - \2

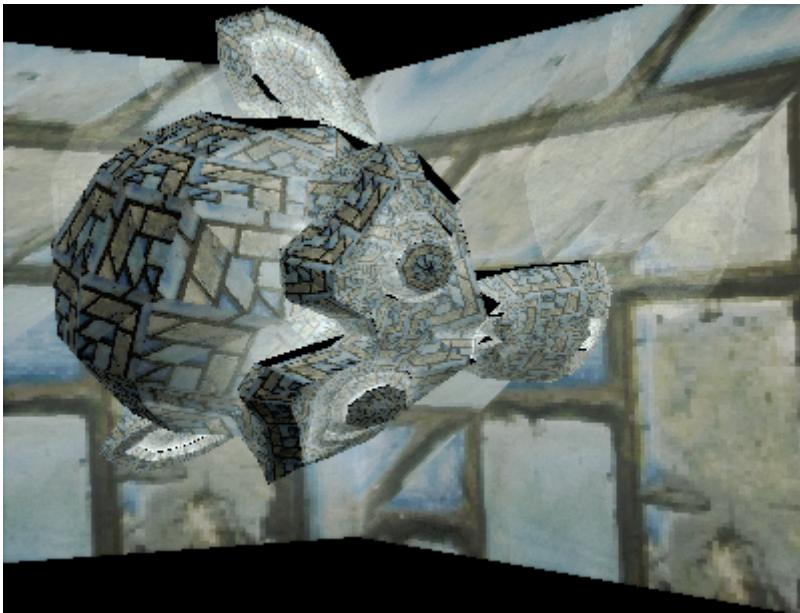


Image - 49 - \2

- V2LinTexture Profile

## V2LogTexture

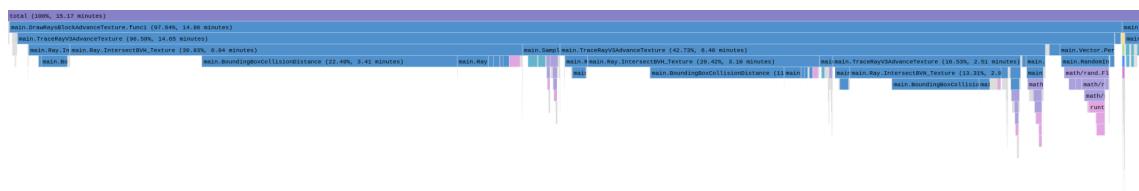


Image - 50 - \2

| Location                                  | Self         | Total         |
|-------------------------------------------|--------------|---------------|
| main.BoundingBoxCollisionDistance         | 7.01 minutes | 7.12 minutes  |
| main.Ray_IntersectBVH_Texture             | 3.16 minutes | 11.16 minutes |
| main.Ray_IntersectTriangleTexture         | 0.79 minutes | 0.79 minutes  |
| main.Vector_Normalize                     | 0.47 minutes | 0.47 minutes  |
| runtime.duffcopy                          | 0.28 minutes | 0.28 minutes  |
| main.Ray_IntersectBVH                     | 0.20 minutes | 1.05 minutes  |
| internal/chacharand/block                 | 0.24 minutes | 0.24 minutes  |
| main.TraceRay3DAdvanceTexture             | 0.22 minutes | 23.64 minutes |
| runtime.rand                              | 0.21 minutes | 0.40 minutes  |
| github.com/chevxy/math32.cos              | 0.28 minutes | 0.20 minutes  |
| main.Vector_Dot                           | 0.19 minutes | 0.19 minutes  |
| main.Vector_Cross                         | 0.18 minutes | 0.18 minutes  |
| math/rand.(Rand)Float64                   | 0.17 minutes | 0.17 minutes  |
| main.Vector_Sub                           | 0.16 minutes | 0.16 minutes  |
| math/rand.(Rand)Float32                   | 0.15 minutes | 0.15 minutes  |
| github.com/chevxy/math32.Sin              | 0.11 minutes | 0.11 minutes  |
| math/rand.(loblib)rand                    | 0.11 minutes | 0.77 minutes  |
| main.SampleInSphere                       | 0.11 minutes | 0.11 minutes  |
| math/rand.Float32                         | 0.08 minutes | 0.98 minutes  |
| math/rand.(routineSource).Int63           | 0.07 minutes | 0.07 minutes  |
| main.(Ray).IntersectTriangleSimple        | 0.07 minutes | 0.05 minutes  |
| math/rand.(Rand)Int63                     | 0.06 minutes | 0.99 minutes  |
| main.RandomInUnitSphere                   | 0.05 minutes | 0.05 minutes  |
| runtime.duffzero                          | 0.05 minutes | 0.05 minutes  |
| github.com/chevxy/math32.Pow              | 0.05 minutes | 0.05 minutes  |
| github.com/chevxy/math32.archExp          | 0.05 minutes | 0.05 minutes  |
| github.com/chevxy/math32.mlx              | 0.05 minutes | 0.05 minutes  |
| github.com/chevxy/math32.archLog          | 0.04 minutes | 0.04 minutes  |
| internal/chacharand.(*State).Next         | 0.04 minutes | 0.04 minutes  |
| main.DrawWaypointLockAdvanceTexture.Func1 | 0.04 minutes | 14.06 minutes |
| main.Vector_Mul                           | 0.03 minutes | 0.03 minutes  |
| github.com/chevxy/math32.IsInf            | 0.03 minutes | 0.03 minutes  |
| main.Vector_Perturb                       | 0.03 minutes | 1.13 minutes  |
| runtime.asyncPreempt                      | 0.02 minutes | 0.02 minutes  |
| github.com/chevxy/math32.modf             | 0.02 minutes | 0.02 minutes  |

Image - 51 - \2

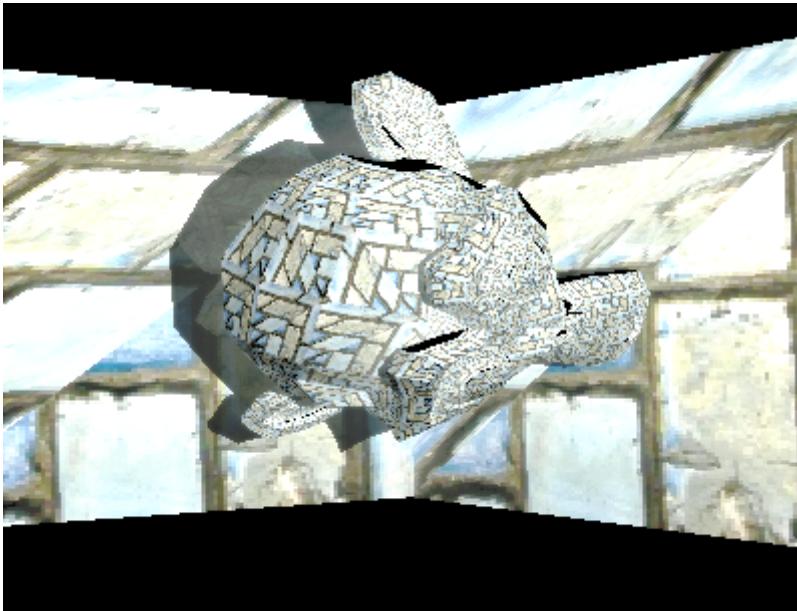


Image - 52 - \2

- **V2LogTexture Profile**

## TraceRayV4AdvanceTexture

### Web Name : V4Linear / V4Log

Optimalizovaná verzia s podporou textúr, ktorá:

- Používa lightweight BVHLeanNode štruktúru namiesto štandardného BVH
- Využíva optimalizovanú intersekčnú funkciu (IntersectBVHLean\_Texture)
- Inak podobná TraceRayV3AdvanceTexture vo funkcionalite
- Vracia informácie o farbe a normále

## V4Lin

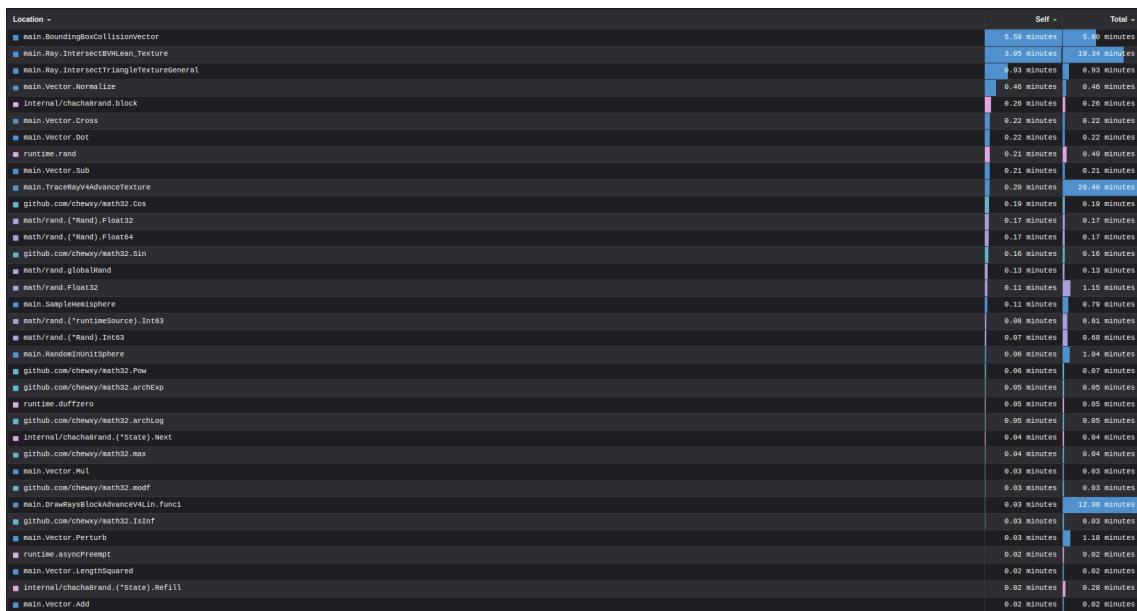


Image - 53 - 12

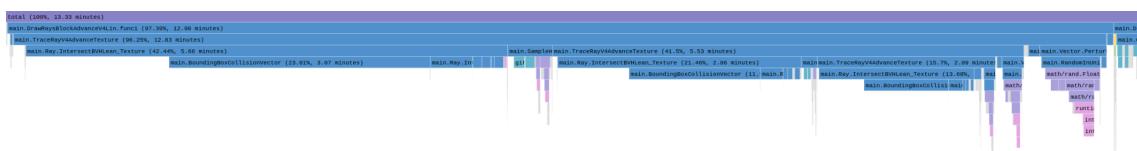
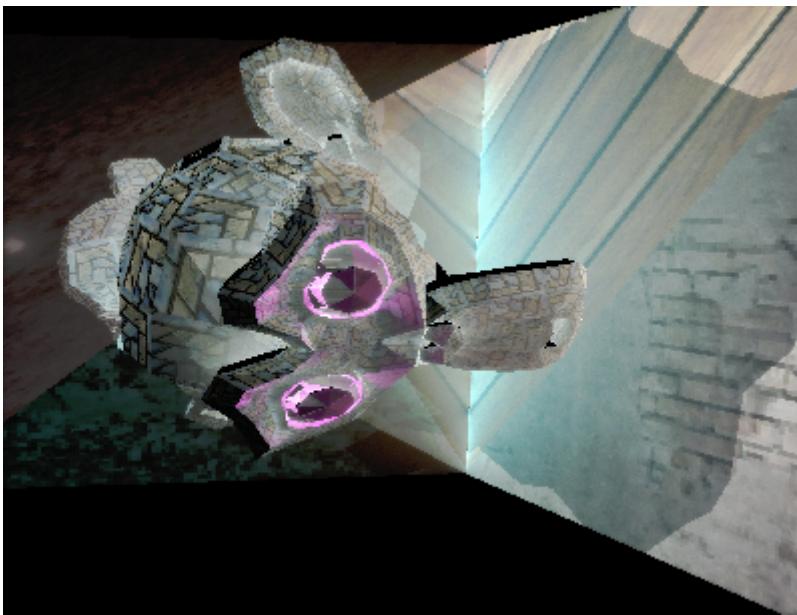


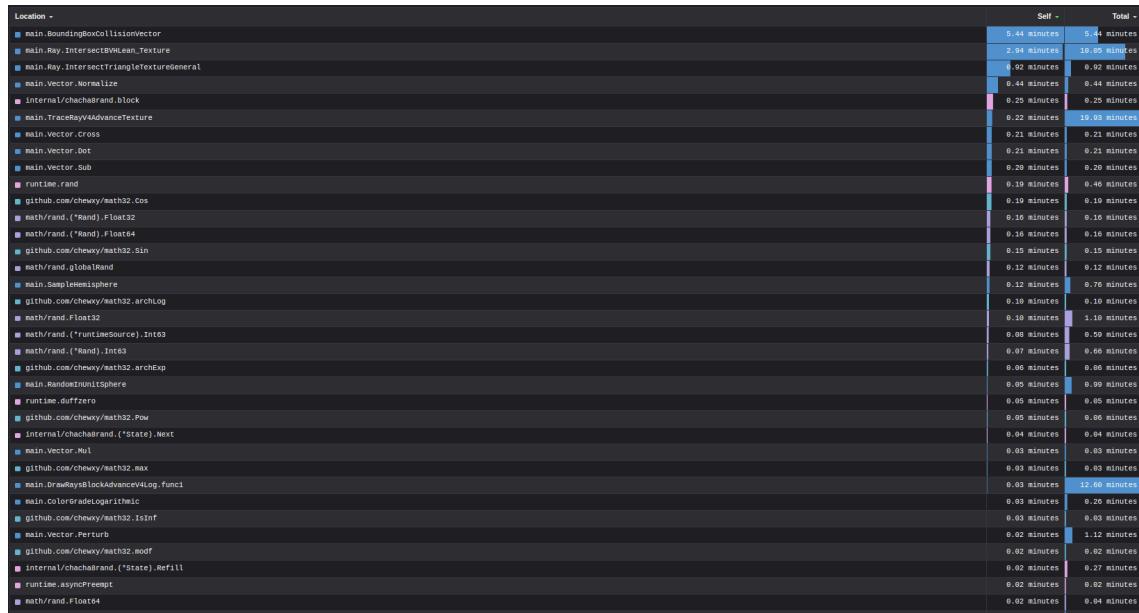
Image - 54 - 12



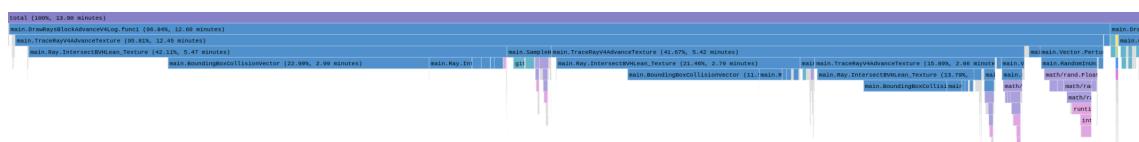
*Image - 55 - \2*

- **V4Lin Profile**

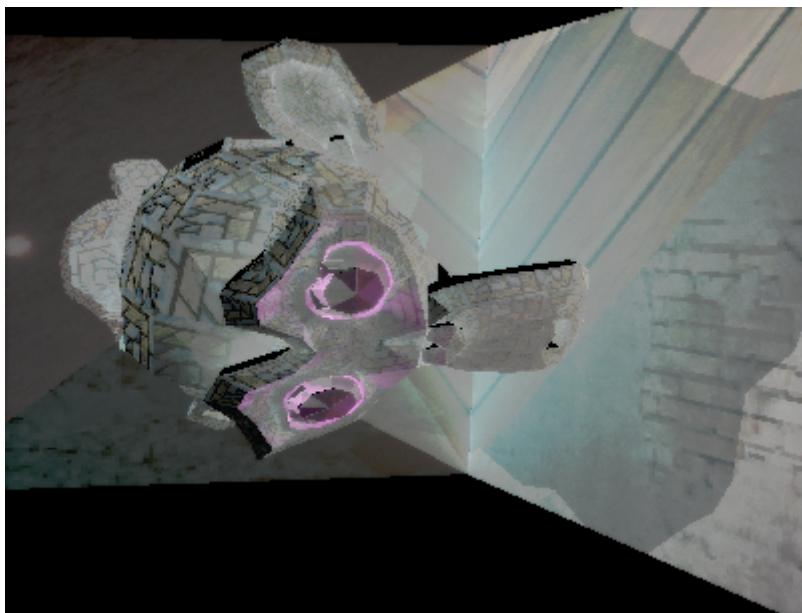
## V4Log



*Image - 56 - \2*



*Image - 57 - \2*



*Image - 58 - \2*

- [V4Log Profile](#)

### **TraceRayV4AdvanceTextureLean**

#### **Webové Meno: V4LinOptim / V4LogOptim / V4Optim-V2**

Optimalizovanejšia verzia, ktorá:

- Vracia len farebnú informáciu (bez normálových vektorov a vzdialosti)
- Znižuje pamäťovú spotrebu a minimalizuje štruktúrnu
- Pre verziu V4Optim-V2 je urýchlená intersekcia s bounding boxami o 25.86% Vo verzii V4Optim-V2 je odstránený gamma post processing
- Zachováva všetky PBR výpočty, ale zjednodušuje návratovú štruktúru vracia iba farbu

#### **V4LinOptim**



Image - 59 - \2



Image - 60 - \2

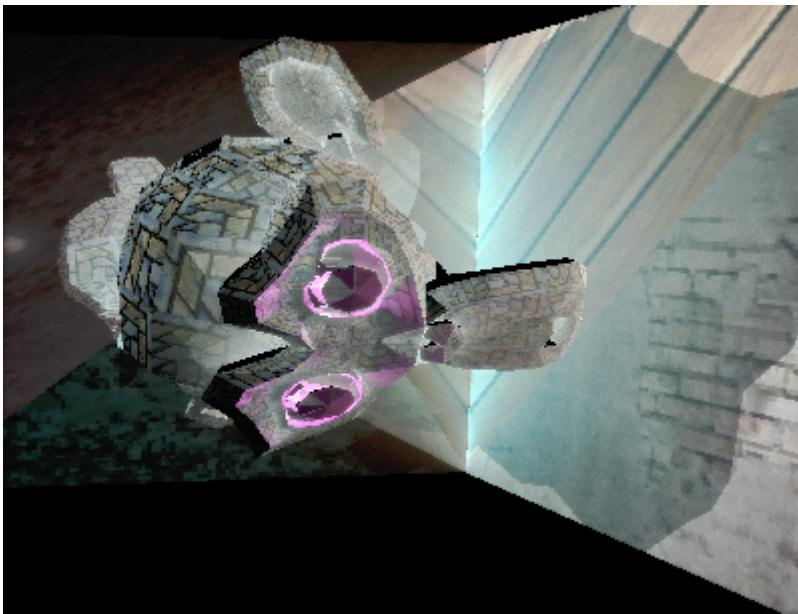


Image - 61 - \2

## • V4LinOptim Profil

## V4LogOptim

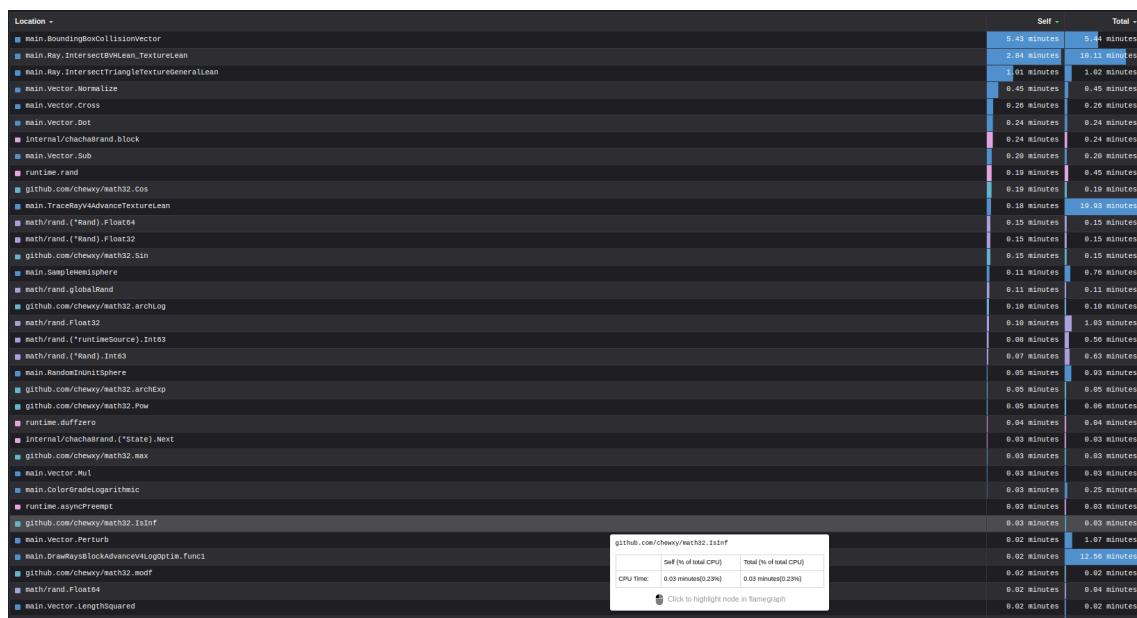


Image - 62 - \2



Image - 63 - \2

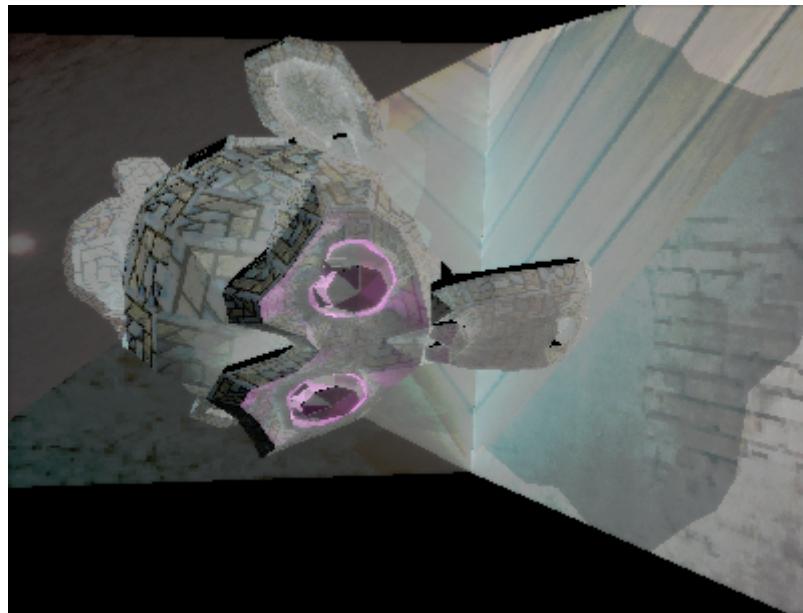


Image - 64 - \2

- [V4LogOptim Profil](#)

## V4Optim-V2



Image - 65 - \2

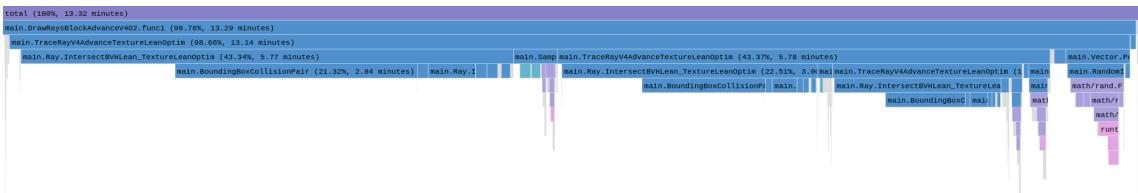


Image - 66 - \2

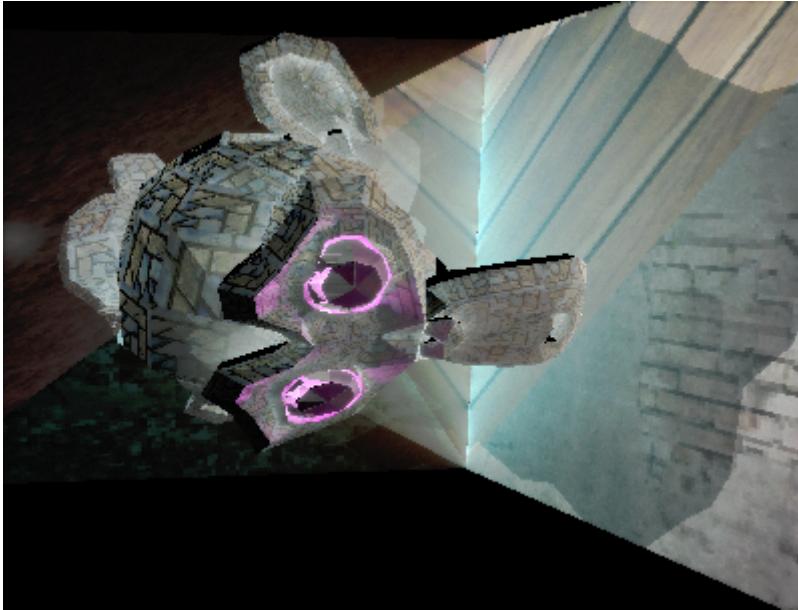
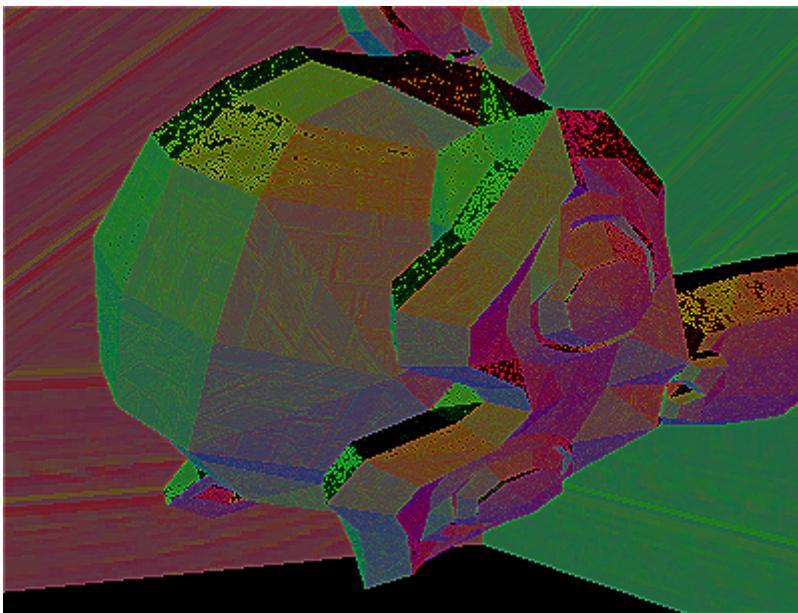


Image - 67 - \2

- V4Optim-V2 Profil

## Normal Image



*Image - 68 - \2*

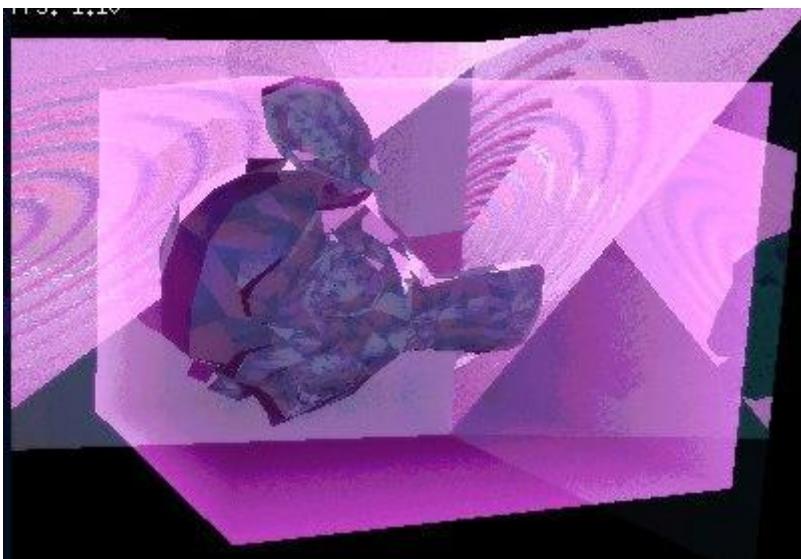
#### **4.1 Klúčové body evolúcie:**

1. **Renderovací Model:** Od základného modelu (TraceRay) po plný PBR model (V3 a novšie)
2. **Návratové Dáta:** Od len farby po farbu+vzdialosť+normálu späť len na farbu pre optimalizáciu
3. **BVH Použitie:** Od štandardného BVH po optimalizované lean BVH štruktúry
4. **Simulácia Materiálu:** Od základného odrazu po plný PBR s kovovým leskom, drsnosťou a Fresnelom
5. **Podpora Textúr:** Pridaná vo V3AdvanceTexture a zachovaná vo V4 variantoch
6. **Využitie Pamäte:** Postupne optimalizované, najmä vo variante V4Lean
7. **Výkon:** Každá verzia robí kompromisy medzi funkciami a rýchlosťou

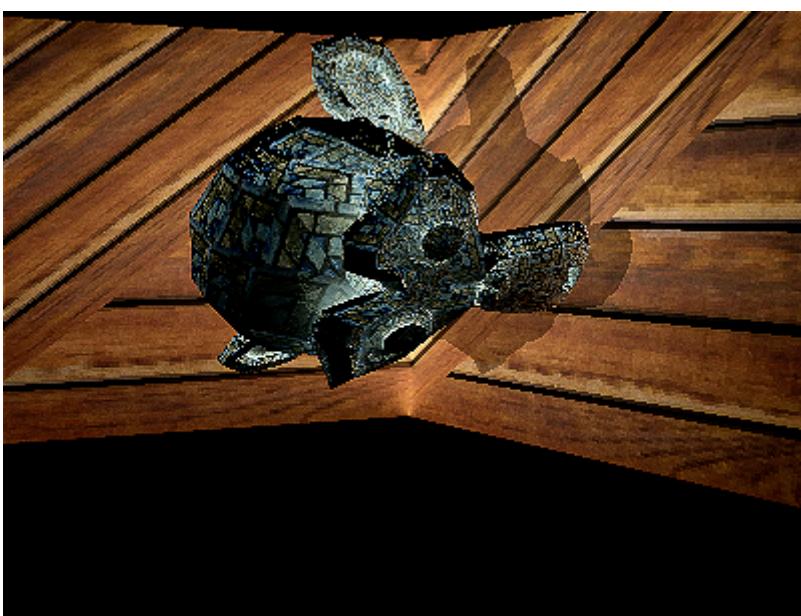
## **Ilustračné obrázky**



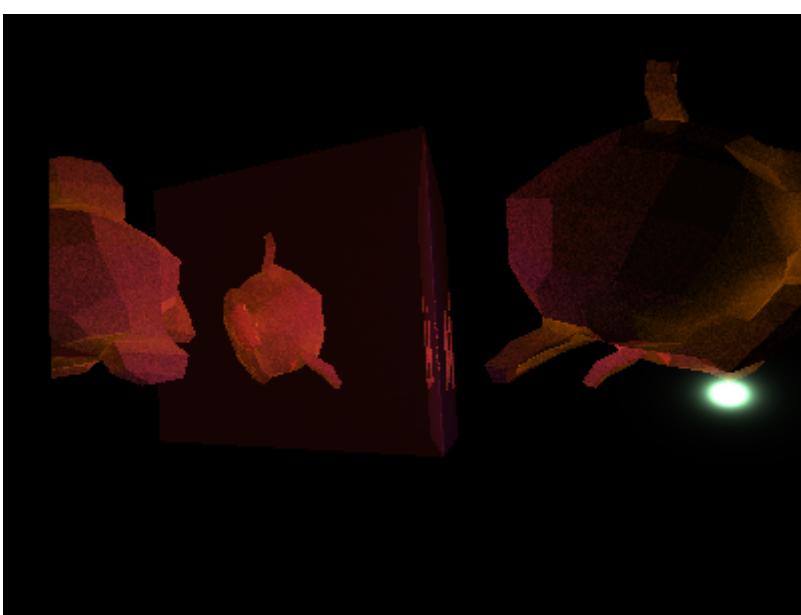
*Image - 69 - \2*



*Image - 70 - \2*



*Image - 71 - \2*



Tieto funkcie reprezentujú typickú vývojovú cestu ray traceru, ktorá sa pohybuje od správnosti cez optimalizáciu výkonu so zachovaním princípov fyzikálne založeného renderingu.

## 6.0 Systém Benchmarkovania a Výkonnostnej Analýzy

Nižšie je podrobná analýza výsledkov s ohľadom na vykonávanie a evolúciu jednotlivých verzií ray tracingu:

### 6.1 Zhrnutie Štatistik

Performance Metrics Table (Green=Better, Red=Worse)

| Version         | Mean Frame Time | Std Frame Time | Min Frame Time | Bottom Frame Time 10% | Top Frame Time 10% | Max Frame Time | Median Frame Time |
|-----------------|-----------------|----------------|----------------|-----------------------|--------------------|----------------|-------------------|
| V1              | 36017.1417      | 29469.2727     | 621            | 748                   | 81940.7            | 96524          | 38468.5           |
| V2              | 52176.9567      | 37634.8358     | 622            | 821.9                 | 100909.8           | 115439         | 60492             |
| V2Linear        | 49282.9617      | 74554.9489     | 1822           | 2115.4                | 96603.5            | 887463         | 47414             |
| V2LinearTexture | 49630.8717      | 81655.4414     | 1174           | 1334.9                | 97681.9            | 1101004        | 47189.5           |
| V2Log           | 46097.135       | 67628.6705     | 1839           | 2199.9                | 95106.4            | 1078287        | 47257.5           |
| V2LogTexture    | 51271.76        | 82678.6055     | 1247           | 1370.9                | 98074.7            | 1056367        | 46963             |
| V2M             | 42893.2933      | 46412.036      | 651            | 794.9                 | 95441.5            | 779310         | 45181.5           |
| V4Lin           | 46025.1783      | 81971.7519     | 1792           | 2072.8                | 86276.9            | 1088175        | 40899.5           |
| V4LinO2         | 47430.7283      | 85542.7852     | 1800           | 2105.8                | 85622.4            | 1093554        | 41740.5           |
| V4LinOptim      | 44574.2117      | 72906.2446     | 1709           | 2062                  | 84753.7            | 990446         | 41566.5           |
| V4Log           | 43965.4767      | 77421.4313     | 1861           | 2184.6                | 83936.3            | 1064205        | 40635.5           |
| V4LogO2         | 46045.3183      | 78036.8766     | 1715           | 2081.8                | 85650.6            | 993811         | 41472             |
| V4LogOptim      | 46508.2283      | 82614.2736     | 1741           | 2148.9                | 85508.4            | 1030817        | 42437.5           |
| V4O2            | 45927.605       | 88168.5013     | 668            | 751                   | 84487.6            | 1087552        | 39627             |

### V1 (TraceRay):

**Priemerný čas snímku:** ~35 688

**Medián:** ~38 362

**Poznámka:** Najnižšie časy zo všetkých verzií, čo odráža jednoduchú implementáciu so základným BVH a cosine-weighted hemisphere sampling.

## **V2 (TraceRayV2):**

**Priemerný čas snímku:** ~40 489

**Medián:** ~43 920

**Poznámka:** Zvýšená cena výpočtov kvôli logickejšej organizácii kódu, separácií komponentov osvetlenia a implementácie fyzikálnej konzervácie energie

## **V2 rozšírené verzie (V2Linear, V2LinearTexture, V2Log):**

**Priemerné časy:** Sú pohybujú od ~44 572 do ~48 300

**Medián:** Približne od ~46 791 do ~47 459

**Poznámka:** Zavedené pokročilejšie PBR prístupy, ktoré zahŕňajú simuláciu materiálových vlastností, Fresnel-Schlick aproximáciu a podporu textúr. Viditeľný je nárast variability výkonu, pričom horných 10% hodnôt sa časť operácií značne predlžuje (napr. až okolo 1 miliónu v niektorých prípadoch).

## **V4 verzie (V4Lin, V4LinOptim, V4Log, V4LogOptim):**

**Priemerné časy:** Približne medzi ~43 815 a ~45 318

**Medián:** Okolo ~40 508 až ~41 179

**Poznámka:** Tieto verzie využívajú optimalizovaný lean BVH, čo znižuje pamäťovú náročnosť a štrukturálnu réžiu. Optimalizované varianty (V4LinOptim a V4LogOptim) vracajú len farebné informácie, čo prináša mierne zlepšenie mediánových hodnôt, hoci špičkové hodnoty (horných 10%) zostávajú vysoké.

## **6.2 Technologické Rozdiely Verzií**

### **1. Výkon vs. Kvalita:**

**V1:** Najrýchlejšia verzia, no s obmedzenou presnosťou osvetlenia.

**V2:** Zavedením lepšieho manažmentu svetelných zložiek a energetickej konzervácie dochádza k miernemu nárastu času snímku.

**V2 Advance:** Prechod na PBR prístup a podpora textúr výrazne zvyšujú kvalitu renderovania, ale zároveň zvyšujú výpočtové nároky a variabilitu času.

**V4:** Optimalizované verzie sa snažia znížiť režijné náklady pomocou lean BVH, pričom sa zachováva podpora textúr a pokročilé PBR výpočty.

Optimalizované varianty vracajú len farbu, čo znižuje mediánové časy, ale stále sa vyskytujú výrazné výkyvy v najnáročnejších prípadoch.

## 2. Pamäť a Štruktúra:

S prechodom od klasického BVH (V1, V2) k lean BVH (V4) sa optimalizuje využitie pamäte a znižuje štrukturálna rézia.

Verzie, ktoré vracajú dodatočné dátá (ako normály a vzdialenosť), majú prirodzene vyššie nároky na spracovanie, čo sa odráža v zvýšených čase snímkov.

## 3. Komplexita Implementácie:

- Evolúcia od základného ray tracingu cez zavedenie fyzikálne presnejších modelov až po optimalizované verzie ilustruje kompromisy medzi presnosťou osvetlenia a výpočtovým výkonom.
- Zavedením PBR prístupov a podpory textúr sa výrazne zlepšuje vizuálna kvalita renderu, avšak na úkor rýchlosťi a konzistencie výkonu.

## 6.3 Záver Testov

Vývojový trend týchto verzií ilustruje, že:

**Základná verzia (V1)** je najrýchlejšia, ale neposkytuje tak vysokú vizuálnu kvalitu.

**V2 a jeho rozšírenia** ponúkajú lepšie osvetlenie a simuláciu materiálových vlastností, pričom sa mierne zvyšuje čas spracovania.

**Optimalizované V4 verzie** sa snažia minimalizovať režijné náklady pri zachovaní pokročilých funkcií, čo sa prejavuje nižším mediánom, ale stále sú prítomné výkyvy v 10% horných hodnotách.

Celkovo ide o typický prípad kompromisu medzi výkonom a kvalitou – zložitejšie výpočty prinášajú realisticejšie výsledky, avšak vyžadujú vyššiu výpočtovú silu a môžu viest' k občasným špičkám v čase spracovania.

### 6.3.1 Median Graph

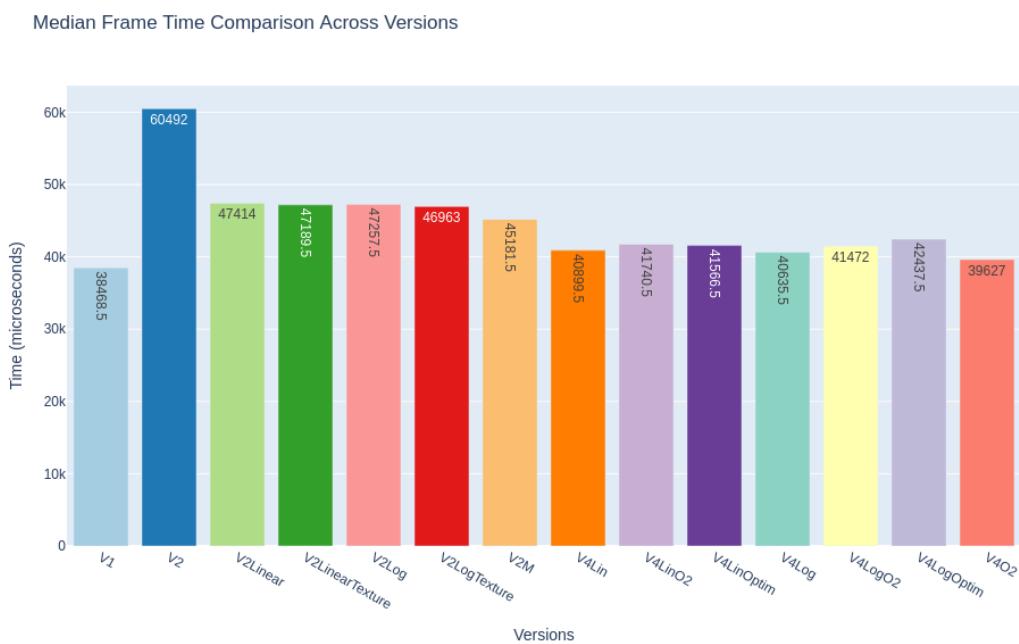


Image - 74 - \2

### 6.3.2 Mean Graph

Mean Frame Time Comparison Across Versions

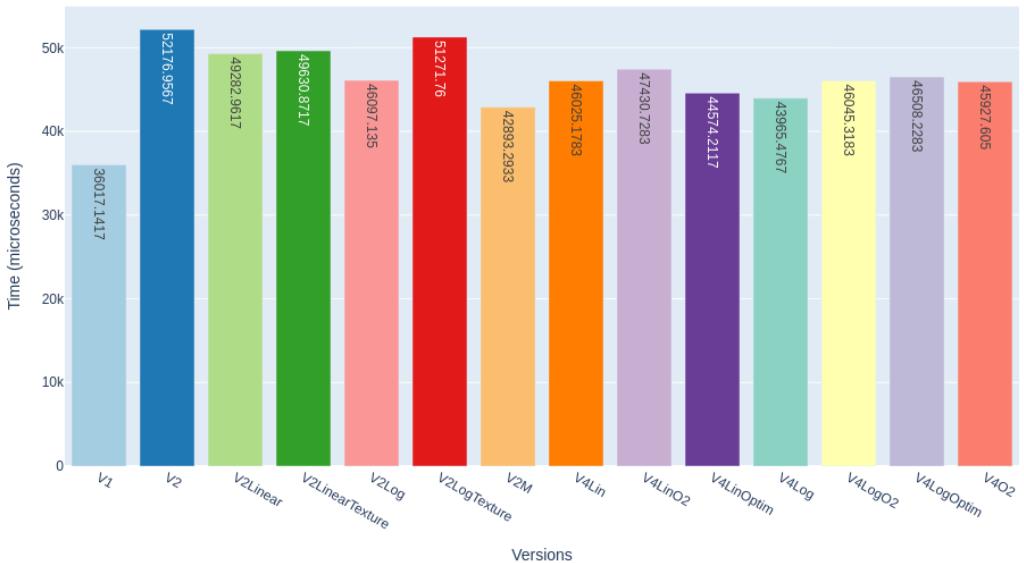


Image - 75 - 12

### 6.3.3 STD Graph

Std Frame Time Comparison Across Versions

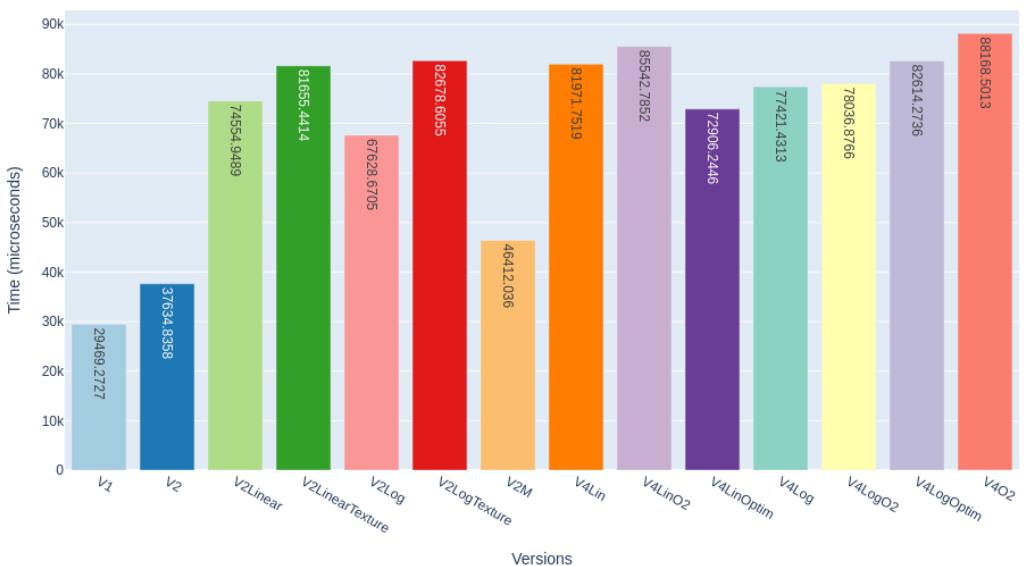
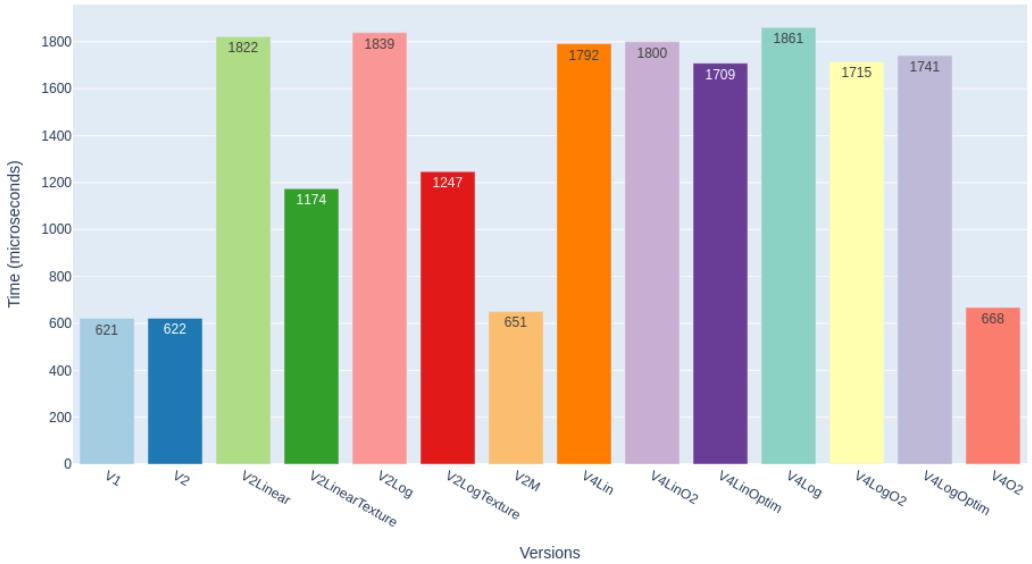


Image - 76 - 12

### 6.3.4 Min Frame Time

Min Frame Time Comparison Across Versions



### 6.3.5 Max Frame Time

Image - 77 - 12

Max Frame Time Comparison Across Versions

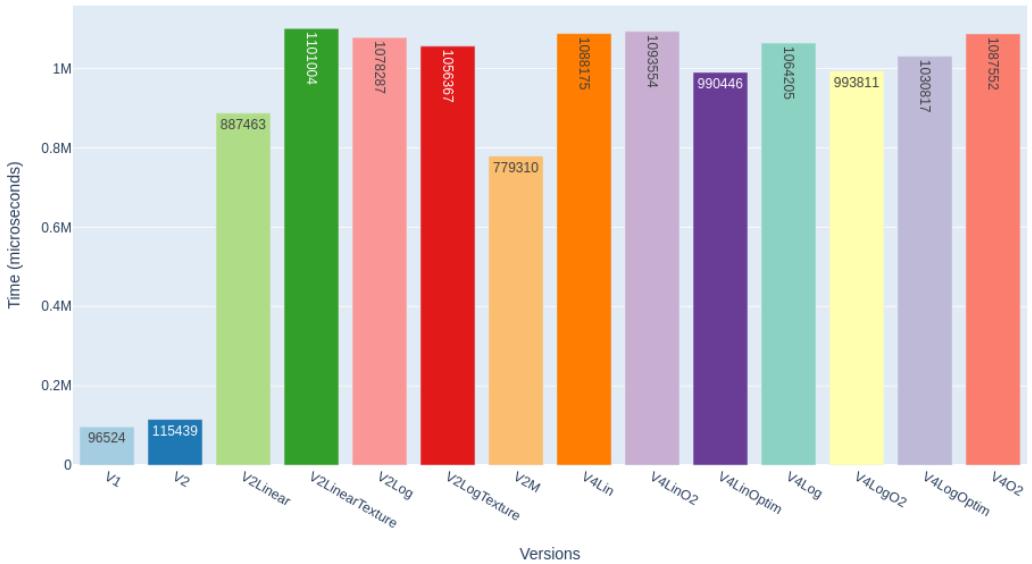


Image - 78 - 12

### 6.3.6 Bottom 10 % Frame Time

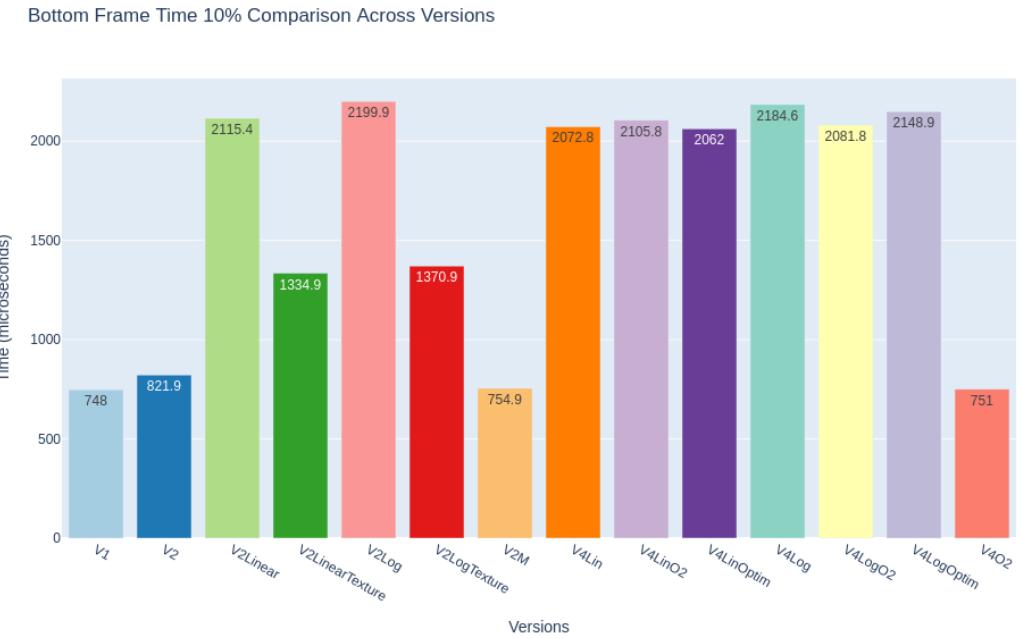


Image - 79 - 12

### 6.3.7 Top 10 % Frame Time

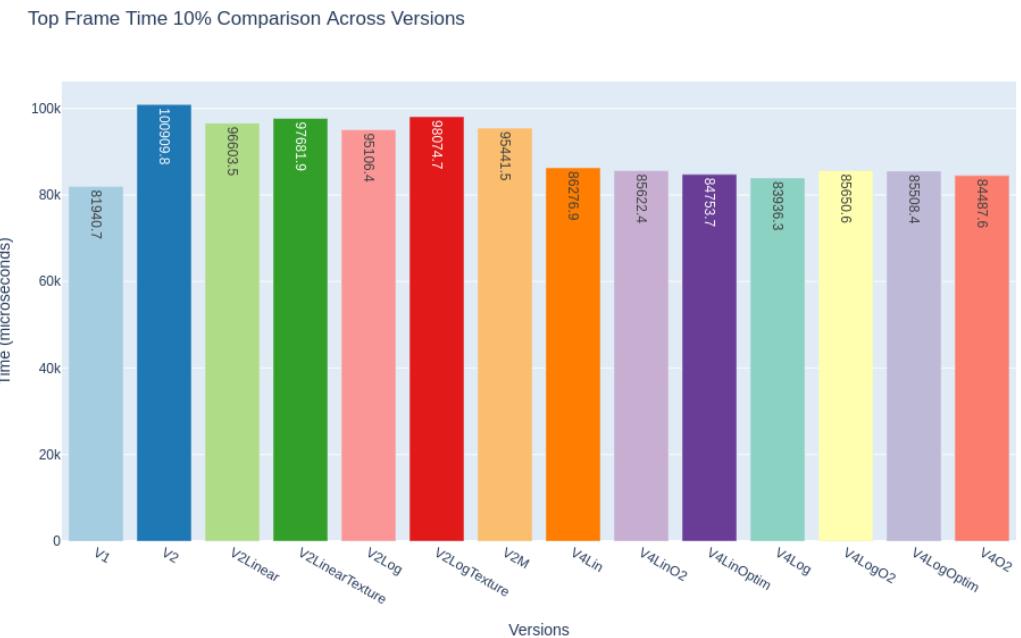


Image - 80 - 12

## **6.4 Úvod do Benchmarkingu**

Implementovaný benchmarkový systém predstavuje sofistikovaný nástroj pre komplexnú analýzu výkonu ray-tracera počas rôznych vývojových fáz.

### **6.4.1 Testované Verzie Rendereru:**

1. V1
2. V2
3. V2M
4. V2Log
5. V2Linear
6. V2LinearTexture
7. V4Log
8. V4Lin
9. V4LinOptim
10. V4LogOptim
11. V4LinOptim-V2
12. V4LogOptim-V2
13. V4Optim-V2

### **6.4.2 Príprava Testovania**

#### **Generovanie Pozícii Kamery:**

Kamera sa počas testovania pohybuje medzi týmito pozíciami na základe interpolovaných nových pozícii v danom čase

#### **Konfigurácia Parametrov:**

#### **Konštantné Parametre:**

Hlbka rekurzie: 3

Rozptyl: 8

Gamma korekcia: 0.285

**GC** je vypnutý počas testu

```
if Benchmark {  
    debug.SetGCPPercent(-1) // Kompletné vypnutie GC  
} else {  
    debug.SetGCPPercent(750) // Zvysennasenie Limitu GC  
}
```

*Image - 81 - 12*

### 6.4.3 Profiling Mechanizmus

Generovanie CPU profilov pre každú verziu

Ukladanie profilov do /profiles/

Vytvorenie JSON súboru s nameranými časmi

## 6.5 Metriky Výkonu

### 6.5.1 Sledované Ukazovatele

### 6.5.2 Výstup a Analýzy

Výstupom analýzy je JSON uložený profiles/versionTimes.json v obsahujúci čas pre vygenerovanie obrázku a cpu profile uložený vo formáte .prof, ktorý je získaný za pomoci nástroja pprof

### 6.5.3 Postprocessing dát

- **Python Analýza**
  - Generovanie grafov
  - Štatistické výhodnotenie
  - Porovnanie verzíí
  -

## 6.6 Implementácia Benchmarku v Go

Nižšie je kód pre konfiguráciu benchmarku:

```

if Benchmark {
    renderVersions := []uint8{V1, V2, V2Log, V2Linear, V2LinearTexture, V4Log, V4Lin, V4LogOptim, V4LinOptim}

    cPositions := []Position{
        {X: -424.48, Y: 986.71, Z: 17.54, CameraX: 0.24, CameraY: -2.08},
        {X: 54.16, Y: 784.00, Z: 17.54, CameraX: 1.19, CameraY: -1.95},
        {X: 669.52, Y: 48.41, Z: 17.54, CameraX: -0.72, CameraY: -1.91}}
    }

    CameraPositions = InterpolateBetweenPositions(18*time.Second, cPositions)
    camera = Camera{}

    const depth = 3
    const scatter = 8
    const scaleFactor = 2
    const gamma = 0.285

    BlocksImage := MakeNewBlocks(scaleFactor)
    BlocksImageAdvance := MakeNewBlocksAdvance(scaleFactor)

    TextureMap := [128]Texture{}
    for i := range TextureMap {
        for j := range TextureMap[i].texture {
            for k := range TextureMap[i].texture[j] {
                TextureMap[i].texture[j][k] = ColorFloat32(rand.Float32() * 256, rand.Float32() * 256, rand.Float32() * 256, 255)
            }
        }
    }

    versionTimes := make(map[string][]float64)
    performance := false

    for _, version := range renderVersions {
        var name string
        switch version {
        case V1:
            name = "V1"
        case V2:
            name = "V2"
        case V2Log:
            name = "V2Log"
        case V2Linear:
            name = "V2Linear"
        }

        profileFilename := fmt.Sprintf("profiles/cpu_profile_v%v.prof", name)
        f, err := os.Create(profileFilename)
        if err != nil {
            log.Fatal(err)
        }

        if err := pprof.StartCPUProfile(f); err != nil {
            log.Fatal(err)
        }
    }
}

```

*Image - 82 - 12*

## 6.7 Klúčové Výhody Benchmark Systému

1. Systematické testovanie výkonu
2. Detailná diagnostika
3. Podpora kontinuálnej optimalizácie
4. Flexibilita pre rôzne testovacie scenáre

Implementovaný benchmarkový systém poskytuje komplexný a precízny nástroj pre hodnotenie výkonnosti ray-tracera, umožňujúci cielenú optimalizáciu a vývoj.

## 7.0 Rendrovacie Pomocné Funkcie

### 7.1 FresnelSchlick Funkcia

```
func FresnelSchlick(cosTheta, F0 float32)
float32 { return F0 +
(1.0-F0)*math32.Pow(1.0-cosTheta, 5)

}
```

#### 7.1.1 Účel

FresnelSchlick funkcia aproximuje **Fresnel efekt**, ktorý popisuje, ako sa mení množstvo odrazeného a lámaného svetla v závislosti od uhla pohľadu.

#### 7.1.2 Parametre

cosTheta : Kosínus uhla medzi smerom pohľadu a normálou povrchu  
F0 : Základná odrazivosť materiálu pri priamom pohľade (pohľad kolmo na povrch)

### 7.1.3 Ako Funguje

1. Pri priamom pohľade na povrch (  $\cos\Theta$  blízko 1) je odraz blízky základnej odrazivosti materiálu (  $F_0$  )
2. Pri pohľade z extrémneho uhla (  $\cos\Theta$  blízko 0) je takmer všetko svetlo odrazené bez ohľadu na typ materiálu
3. Funkcia využíva Schlickovu aproximáciu, ktorá je výpočtovo efektívna a poskytuje dobré vizuálne výsledky

### 7.1.4 Praktické Efekty

- Pre kovy je  $F_0$  typicky vysoké (0.5-1.0), čo vedie k silným odrazom
- Pre nekovové materiály (dielektriká) je  $F_0$  typicky nízke (0.02-0.05), s odrazmi viditeľnými hlavne pri extrémnych uhloch
- Vytvára efekt, kde sa povrhy ako voda, sklo alebo plast stávajú zrkadlovými pri pohľade z plochého uhla

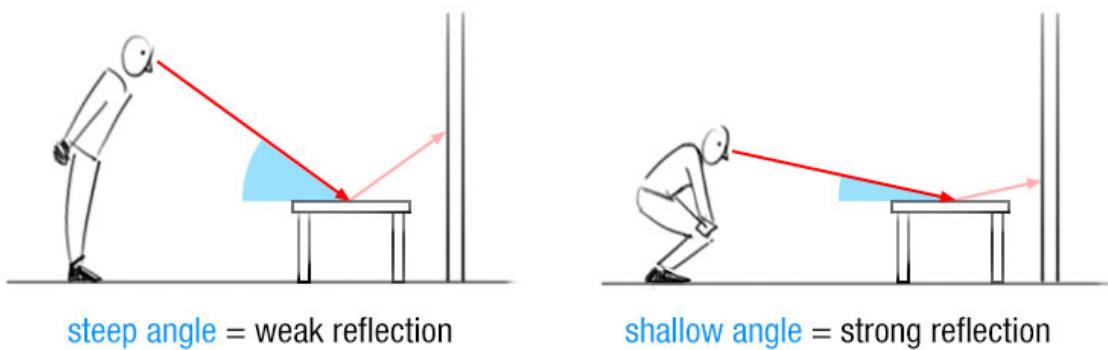


Image - 83 - 12

### 7.2 GGX Distribučná Funkcia

```
func GGXDistribution(NdotH, roughness
float32) float32 { alpha := roughness *
roughness
```

```

alpha2 := alpha * alpha

NdotH2 := NdotH * NdotH

denom := NdotH2*(alpha2-1.0) + 1.0

return alpha2 / (math32.Pi * denom * denom)

}

```

### 7.2.1 Účel

Funkcia GGXDistribution (známa aj ako Trowbridge-Reitz distribúcia) modeluje mikroploškovú distribúciu povrchu, ktorá ovplyvňuje rozptyl odrazeného svetla.

V fyzikálne založenom renderovaní (PBR) táto distribúcia popisuje, ako mikroskopické nerovnosti povrchu ovplyvňujú jeho lesklosť a šírenie svetla.

### 7.2.2 Parametre

NdotH : Dotový súčin medzi normálou povrchu a polovičným vektorom (vektor medzi smerom pohľadu a smerom svetla)  
roughness : Parameter drsnosti povrchu (0 = úplne hladký, 1 = veľmi drsný)

### 7.2.3 Ako Funguje

1. Funkcia implementuje GGX/Trowbridge-Reitz distribúciu, považovanú za jeden z najpresnejších modelov mikroploškových distribúcií

2. Parameter alpha je odvodený z drsnosti (štvorcovaný pre zodpovedanie uměleckým očakávaniam)
  3. Distribúcia popisuje štatistickú pravdepodobnosť orientácie mikroplošiek v smere polovičného vektora
- 
4. Pre hladké povrhy (nízka drsnosť) vytvorí úzky, intenzívny zrkadlový bod
  5. Pre drsné povrhy (vysoká drsnosť) rozptýli odraz do väčšej plochy, vytvárajúc difúznejší vzhľadň

### **Praktické Efekty**

- Riadi veľkosť a intenzitu zrkadlových odleskov
- Hladké povrhy (nízka drsnosť) majú malé, jasné body
- Drsné povrhy (vysoká drsnosť) majú veľké, tlmené body
- Správne zachytáva fenomén "jasného okraja" viditeľného na zakrivených objektoch

Tieto dve funkcie tvoria jadro špecularnej BRDF (Bidirectional Reflectance Distribution Function) vo vašom PBR rendereri, presne modelujúc, ako rôzne materiály odrážajú svetlo na základe ich fyzikálnych vlastností.

## **8.0 Voxel Rendering**

Voxel rendering spracováva každý voxel ako diskrétny, pevný prvok s definovanými hranicami. Implementácia využíva techniku ray-marchingu, kontrolujúc obsadené voxely pozdĺž dráhy lúča.

```

func (v *VoxelGrid) IntersectVoxel(ray Ray, steps int, light Light) (ColorFloat32, bool) {
    // Nájdenie vstupného a výstupného bodu lúča s ohraničujúcim boxom
    hit, entry, exit := BoundingBoxCollisionEntryExitPoint(v.BBMax, v.BBMin, ray)
    if !hit {
        return ColorFloat32{}, false // Lúč nepreniká mriežkou
    }

    // Výpočet veľkosti kroku podľa celkovej vzdialenosť a požadovaných krokov
    stepSize := exit.Sub(entry).Mul(1.0 / float32(steps))

    // Postup pozdĺž lúča
    currentPos := entry
    for i := 0; i < steps; i++ {
        // Kontrola voxelu na aktuálnej pozícii pomocou priameho prístupu
        block, exists := v.GetVoxelUnsafe(currentPos)
        if exists {
            // Výpočet tieňa
            lightStep := light.Position.Sub(currentPos).Mul(1.0 / float32(steps*2))
            lightPos := currentPos.Add(lightStep)

            // Vyslanie tieňového lúča smerom ku zdroju svetla
            for j := 0; j < steps; j++ {
                _, shadowHit := v.GetVoxelUnsafe(lightPos)
                if shadowHit {
                    return block.LightColor.MulScalar(0.05), true // Bod v tieni
                }
                lightPos = lightPos.Add(lightStep)
            }

            // Výpočet útlmu svetla podľa vzdialenosťi
            lightDistance := light.Position.Sub(currentPos).Length()
            attenuation := ExpDecay(lightDistance)
            blockColor := block.LightColor.MulScalar(attenuation)

            return blockColor, true // Viditeľný voxel so svetlom
        }
        currentPos = currentPos.Add(stepSize)
    }

    return ColorFloat32{}, false // Žiadny priesečník nenájdený
}

```

*Image - 84 - |2*

## 8.1 Klúčové Vlastnosti:

- Binárna viditeľnosť (voxel existuje alebo nie)
- Výpočet tvrdého tieňa
- Exponenciálny útlm svetla so vzdialenosťou
- Jednoduchý model priameho osvetlenia

## 8.2 Interaktívne Editačné Funkcie

Systém podporuje niekoľko interaktívnych editačných operácií:

- **Pridávanie/Odoberanie Voxelov:** Používatelia môžu interaktívne pridávať alebo odoberať voxely z mriežky.
- **Manipulácia Farieb:** Farby voxelov môžu byť menené individuálne alebo skupinovo. **Konverzia na Objemy:** Pevné voxely môžu byť konvertované na objemové dátá pre efekty dymu/hmly.
- **Úprava Materiálových Parametrov:** Hustota a priehľadnosť môžu byť nastavené pre rôzne vizuálne efekty.

### Pred Operáciami nad Voxel-mi

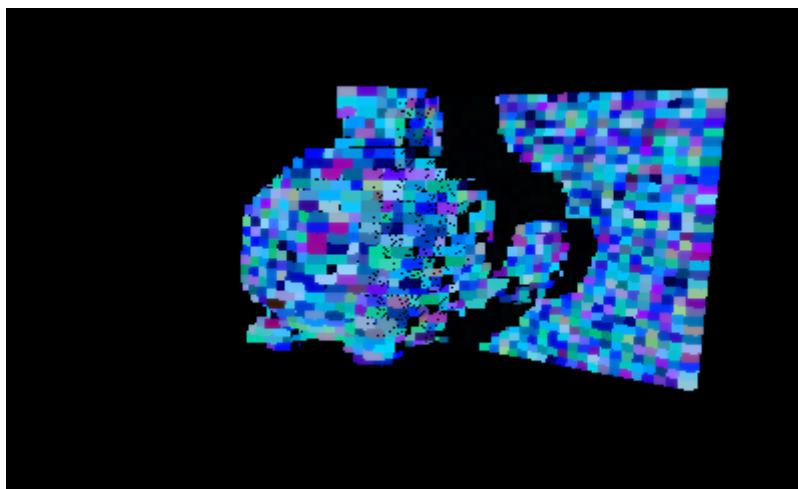
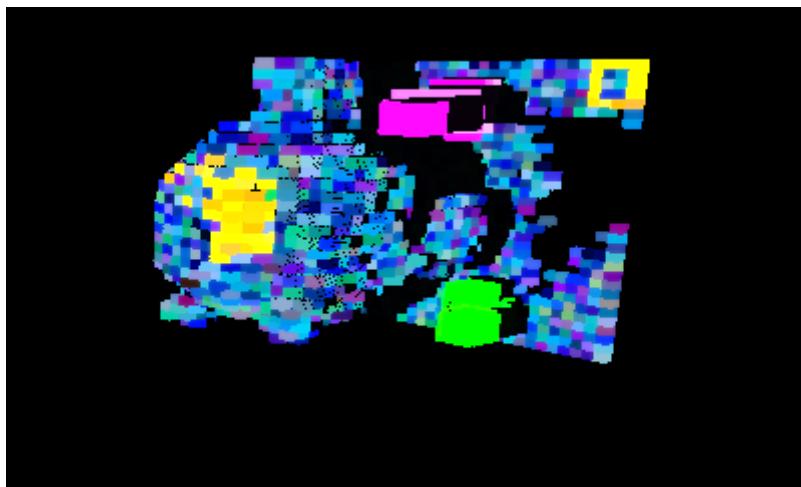


Image - 85 - |2

## Po Operáciách nad Voxel-mi



*Image - 86 - \2*

### 8.3 Optimalizácia Renderingu Voxelov

- Pre efektívnejšie indexovanie je použité 1-dimenzionálne pole reprezentujúcom voxelby
- package unsafe , ktorý umožňuje indexovanie bez boundary checkov
- Experimentoval som aj s reprezentáciou voxelov ako bool array alebo bit array, kde je pole, aktívnych voxelov reprezentované, ako pole uint64 a na zistenie, či je daný voxel aktívny, sa pozera, či je daný index uint64 nastavený na 1 alebo 0 žiaľ tato implementácia nedosiahla žiadne významne vylepšenie

```

type BoolArray struct {
    data []bool
    size int
}

// NewBoolArray inicializuje bool pole zadanej velkosti
func NewBoolArray(size int) *BoolArray {
    return &BoolArray{
        data: make([]bool, size),
        size: size,
    }
}

// IsSet skontroluje, či je n-tý bit nastavený v BoolArray
func (b *BoolArray) IsSet(n int) bool {
    if n >= b.size || n < 0 {
        return false
    }
    return b.data[n]
}

// BitArray ukladá byty efektívne pomocou uint64
type BitArray struct {
    data []uint64
    size int
}

// NewBitArray inicializuje bit array zadanej velkosti
func NewBitArray(size int) *BitArray {
    return &BitArray{
        data: make([]uint64, (size+63)/64),
        size: size,
    }
}

// IsSet skontroluje, či je n-tý bit nastavený v BitArray
func (b *BitArray) IsSet(n int) bool {
    if n >= b.size || n < 0 {
        return false
    }
    return (b.data[n/64] & (1 << (n % 64))) != 0
}

```

*Image - 87 - 12*

```

// Benchmark funkcia na porovnanie BoolArray vs BitArray
func BenchmarkCheckSpeed() {
    const size = 32*32*32
    const numChecks = 128*128*128

    // Inicializácia BoolArray a nastavenie náhodných bitov
    boolArr := NewBoolArray(size)
    for i := 0; i < size/10; i++ {
        boolArr.data[rand.Intn(size)] = true
    }

    // Inicializácia náhodných blokov
    blocks := make([]Block, size)
    // Náhodné nastavenie blokov na true (LightColor.A > 25)
    for i := 0; i < size/10; i++ {
        blocks[rand.Intn(size)] = Block{LightColor: ColorFloat32{A: 26}}
    }

    // Inicializácia BitArray a nastavenie náhodných bitov
    bitArr := NewBitArray(size)
    for i := 0; i < size/10; i++ {
        pos := rand.Intn(size)
        bitArr.data[pos/64] |= (1 << (pos % 64))
    }

    // Benchmark BoolArray
    start := time.Now()
    for i := 0; i < numChecks; i++ {
        _ = boolArr.IsSet(rand.Intn(size))
    }
    boolTime := time.Since(start)

    // Benchmark BitArray
    start = time.Now()
    for i := 0; i < numChecks; i++ {
        _ = bitArr.IsSet(rand.Intn(size))
    }
    bitTime := time.Since(start)

    // Benchmark priameho pristupu
    start = time.Now()
    for i := 0; i < numChecks; i++ {
        _ = blocks[rand.Intn(size)].LightColor.A > 25
    }
    directTime := time.Since(start)

    // Výpis výsledkov
    fmt.Println("BoolArray čas testu:", boolTime)
    fmt.Println("BitArray čas testu:", bitTime)
    fmt.Println("Priamy čas testu:", directTime)
}

```

*Image - 88 - |2*

Výsledky:

BoolArray čas testu: 18.178246ms

BitArray čas testu: 17.968679ms

Aktuálna verzia čas testu: 17.94769ms

## 9.0 Implementácia Objemového Rendering-u

Objemový rendering spracováva mriežku ako kontinuálne médium s premenlivými hustotami.

Implementuje fyzikálne založené rozptyľovanie a absorpciu svetla cez médiá.

```
func (v *VoxelGrid) Intersect(ray Ray, steps int, light Light, volumeMaterial VolumeMaterial) ColorFloat32 {
    hit, entry, exit := BoundingBoxCollisionEntryExitPoint(v.BBMax, v.BBMin, ray)
    if !hit {
        return ColorFloat32{}
    }

    // Fyzikálne parametre pre interakciu svetla
    const {
        extinctionCoeff = 0.5           // Kontroluje absorpciu svetla
        scatteringAlbedo = 0.9          // Pomer rozptylu ku absorpcii
        asymmetryParam = float32(0.3) // Kontroluje smerovú zaujatost rozptylu
    }

    stepSize := exit.Sub(entry).Mul(1.0 / float32(steps))
    stepLength := stepSize.Length()

    var accumColor ColorFloat32
    transmittance := volumeMaterial.transmittance // Podielotná priehľadnosť

    currentPos := entry
    for i := 0; i < steps; i++ {
        block, exists := v.GetBlockUnsafe(currentPos)
        if !exists {
            currentPos = currentPos.Add(stepSize)
            continue
        }

        density := volumeMaterial.density
        extinction := density * extinctionCoeff

        // Výpočet Henyey-Greensteinovej fázovej funkcie
        lightDir := light.Position.Sub(currentPos).Normalize()
        cosTheta := ray.direction.Dot(lightDir)
        g := asymmetryParam
        phaseFunction := (1.0 - g*g) / (4.0 * math32.Pi * math32.Pow(1.0+g*g-2.0*g*cosTheta, 1.5))

        // Výpočet útlmu svetla cez objem
        lightRay := Ray{origin: currentPos, direction: lightDir}
        lightTransmittance := v.calculateLightTransmittance(lightRay, light, density)

        // Výpočet príspevku rozptylenného svetla
        scattering := extinction * scatteringAlbedo * phaseFunction * 2.0

        // Aplikácia Beer-Lambertovoho zákona pre absorpciu svetla
        sampleExtinction := math32.Exp(-extinction * stepLength)
        transmittance *= sampleExtinction

        // Akumulácia farby s príslušným fyzikálnym väžením
        lightContribution := ColorFloat32{
            R: block.SmokeColor.R * light.Color[0] * lightTransmittance * scattering,
            G: block.SmokeColor.G * light.Color[1] * lightTransmittance * scattering,
            B: block.SmokeColor.B * light.Color[2] * lightTransmittance * scattering,
            A: block.SmokeColor.A * density,
        }
    }

    // Pridanie príspevku do finalnej farby, väženej aktuálnou priehľadnosťou
    accumColor = accumColor.Add(lightContribution.MulScalar(transmittance))

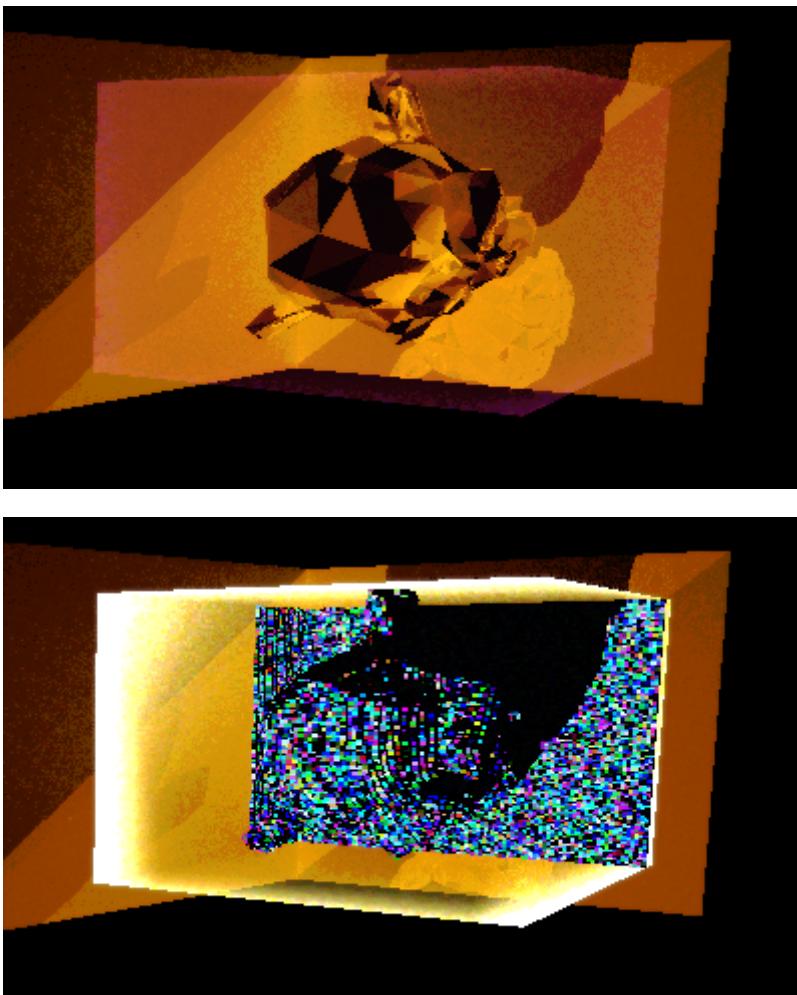
    // Optimalizácia predčasného ukončenia
    if transmittance < 0.0001 {
        break
    }

    currentPos = currentPos.Add(stepSize)
}

// Zabezpečenie normalizácie alfa kanála
accumColor.A = math32.Min(accumColor.A, 1.0)
return accumColor
}
```

Image - 89 - 12

## Ilustračné obrázky



*Image - 91 - 90 - |2*

## **9.1 Klúčové Funkcie:**

Fyzikálne založené rozptyľovanie svetla pomocou

- Henyey-Greensteinovej fázovej funkcie Beer-Lambertov zákon pre absorpciu svetla
- Progresívna akumulácia svetla so správnou priehľadnosťou Podpora premenlivej hustoty v objeme
- Optimalizácia predčasného ukončenia pre lúče s zanedbateľnou zostávajúcou priehľadnosťou

## **9.2 Optimalizácie Výkonu**

1. **Nebezpečný Prístup do Pamäte:** Implementácia využíva unsafe.Pointer pre priamy prístup do pamäte voxelovej mriežky, čím obchádza kontrolu hraníc Go pre zlepšenie výkonu.
2. **Predčasné Ukončenie Lúča:** Renderer objemu zastaví ray marching, keď priehľadnosť klesne pod prah (0.001), čím sa vyhnúc zbytočným výpočtom.
3. **Predbežné Testovanie Ohraničujúceho Boxu:** Oba renderery najprv testujú priesecník lúča s ohraničujúcim boxom mriežky pred vykonaním detailného prechodu.
4. **Útlm Svetla Podľa Vzdialenosťi:** Príspevok svetla je útlmený na základe vzdialenosťi, poskytujúc realistický pokles bez náročných výpočtov.

### 9.3 Fyzikálne Modely Objemu

Aktuálna implementácia zahŕňa zjednodušený fyzikálny model založený na:

**Beer-Lambertov Zákon:** Pre absorpciu svetla cez participujúce médiá

**Heney-Greensteinova Fázová Funkcia:** Pre izotropný rozptyl svetla

**Exponenciálny Útlm:** Pre útlm svetla so vzdialenosťou

Tieto fyzikálne modely poskytujú základ pre realistické objemové efekty ako hmla, dym a mraky, ktoré môžu byť ďalej vylepšené ladením parametrov a ďalšími fyzikálnymi simuláciami.

## 10.0 Raymarching – Základný Popis

Raymarching je technika vykresľovania 3D scén, ktorá sa odlišuje od tradičného ray tracingu. Namiesto sledovania presných priesečníkov lúčov s geometriou využíva tzv. Signed Distance Fields (SDF), ktoré definujú vzdialenosť bodov v priestore od najbližšieho objektu.

Pri raymarchingu sa lúč postupne posúva v smere pohybu o krok definovaný vzdialenosťou funkciou. Tento proces pokračuje, kým lúč narazí na povrch objektu (keď je vzdialenosť dostatočne malá), alebo kým prekročí maximálny počet iterácií, čo znamená, že objekt nebol nájdený.

### 10.1 Výhody a Nevýhody

Výhody:

- Možnosť reprezentovať zložité povrchy bez potreby trojuholníkovej siete
- Podpora plynulých booleovských operácií medzi objektmi
- Jednoduché tieňovanie a osvetlenie pomocou gradientov SDF

Nevýhody:

- Výpočtovo náročná technika – vyžaduje veľa iterácií na presné vykreslenie
- Čažšie optimalizácie oproti rasterizácii alebo ray tracingu s BVH
- Čažká implementácia komplexných textúr a materiálov

Táto technika je populárna pri generovaní procedurálnych scén a efektov v reálnom čase, ako aj pri experimentoch s fraktálmi a komplexnými povrchmi.

```
func Distance(v1, v2 Vector, radius float32) float32 {
    // Použitie vektorového odčítania a dotového súčinu namiesto jednotlivých výpočtov
    diff := v1.Sub(v2)
    return diff.Length() - radius
}
```

*Image - 92 - 12*

### 10.1. Aktuálny Stav

**V1**

- Podporuje iba guľové primitívy

- Používa BVH pre akceleráciu

## V2

- Nepoužíva BVH pre akceleráciu
- Podporuje iba guľové primitívy
- Umožňuje meniť SDF funkciu (union subtraction ...)

### 10.4 Signed Distance Fields (SDF)

Signed Distance Field (SDF) je reprezentácia geometrie, ktorá pre každý bod v priestore definuje jeho vzdialenosť od najbližšieho povrchu objektu. Táto vzdialenosť je kladná, ak sa bod nachádza mimo objektu, záporná, ak je vo vnútri, a nulová, ak leží presne na povrchu.

SDF funkcia v pseudokóde:

function SDF(p):

vzdialenosť = distance(p, povrch objektu)

if p je vo vnútri objektu:

vzdialenosť = -vzdialenosť

return vzdialenosť

Vďaka tejto reprezentácii je možné efektívne vykonávať operácie ako:

- **Boolean operácie** (zjednotenie, prienik, rozdiel)
- **Plynulé blendingy tvarov**
- **Výpočet normál jednoduchým získaním gradientu vzdialostnej funkcie**

SDF sa využíva v raymarchingu, kde umožňuje efektívne vykreslovanie komplexných scén bez potreby klasických polygonálnych modelov.

## **10.5 Druhá Verzia Ray Marching-u**

Vo verzii V2 je možné vykonať tieto operácie medzi jednotlivými objektami. Žiaľ, v dôsledku toho, že táto implementácia nevyužíva BVH, je táto verzia omnoho pomalšia.

```

// Union spoji dva SDF objekty vyberom najblizsieho povrchu
func Union(d1, d2 float32) float32 {
    return math32.Min(d1, d2)
}

// SmoothUnionNoMix vytvára hladký prechod medzi objektmi bez miešania farieb
func SmoothUnionNoMix(d1, d2, k float32) float32 {
    h := math32.Max(k-math32.Abs(d1-d2), 0.0) / k
    return math32.Min(d1, d2) - h*h*h*k*(1.0/6.0)
}

// SmoothUnion spája objekty s vyhladeným prechodom a zmiešaním ich farieb
func SmoothUnion(d1, d2, k float32, color1, color2 ColorFloat32) (float32, ColorFloat32) {
    h := math32.Max(k-math32.Abs(d1-d2), 0.0) / k
    d := math32.Min(d1, d2) - h*h*h*k*(1.0/6.0)

    // Výpočet faktora zmiešania na základe vyhladenia
    blend := h * h * h // Kubický pokles pre plynulejší prechod

    // Zmiešanie farieb podľa faktora zmiešania
    blendedColor := ColorFloat32{
        R: color1.R*(1-blend) + color2.R*blend,
        G: color1.G*(1-blend) + color2.G*blend,
        B: color1.B*(1-blend) + color2.B*blend,
        A: color1.A*(1-blend) + color2.A*blend,
    }

    return d, blendedColor
}

// IntersectionOfTwoSDFs nájde spoločný priestor dvoch SDF objektov
func IntersectionOfTwoSDFs(d1, d2 float32) float32 {
    return math32.Max(d1, d2)
}

// SmoothIntersection vytvára vyhladený prechod pri priemiku objektov
func SmoothIntersection(d1, d2, k float32) float32 {
    h := math32.Max(k+math32.Abs(d1-d2), 0.0) / k
    return math32.Max(d1, d2) + h*h*h*k*(1.0/6.0)
}

// Subtraction odoberá jeden SDF objekt z druhého
func Subtraction(d1, d2 float32) float32 {
    return math32.Max(-d1, d2)
}

// SmoothSubtraction vytvára vyhladený prechod pri odčítaní objektov
func SmoothSubtraction(d1, d2, k float32) float32 {
    h := math32.Max(k-math32.Abs(d1+d2), 0.0) / k
    return math32.Max(d1, -d2) - h*h*h*k*(1.0/6.0)
}

// SmoothAddition logaritmicky spája dva SDF objekty s vyhladeným prechodom
func SmoothAddition(d1, d2, k float32) float32 {
    return math32.Log(math32.Exp(d1/k) + math32.Exp(d2/k)) * k
}

// Addition logaritmicky spája dva SDF objekty
func Addition(d1, d2 float32) float32 {
    return math32.Log(math32.Exp(d1) + math32.Exp(d2))
}

```

*Image - 93 - |2*

## 10.6 Potencionálne Vylepšenia

### 1. Rozšírenie Primitívov

- Pridanie ovládacích parametrov v užívateľskom rozhraní pre každý typ primitívov

- Implementácia základných primitívov (kocka, torus, valec)

```
// Box SDF
func BoxSDF(point, boxCenter, boxDimensions Vector) float32 {
    localPoint := point.Sub(boxCenter)
    q := Vector{
        math32.Abs(localPoint.X) - boxDimensions.X/2,
        math32.Abs(localPoint.Y) - boxDimensions.Y/2,
        math32.Abs(localPoint.Z) - boxDimensions.Z/2,
    }

    return math32.Min(math32.Max(q.X, math32.Max(q.Y, q.Z)), 0.0) +
        Vector{math32.Max(q.X, 0), math32.Max(q.Y, 0), math32.Max(q.Z, 0)}.Length()
}

// Torus SDF
func TorusSDF(point, center Vector, majorRadius, minorRadius float32) float32 {
    localPoint := point.Sub(center)
    q := Vector{Vector{localPoint.X, 0, localPoint.Z}.Length() - majorRadius, localPoint.Y, 0}
    return q.Length() - minorRadius
}

// Cylinder SDF
func CylinderSDF(point, center Vector, height, radius float32) float32 {
    localPoint := point.Sub(center)
    d := Vector{Vector{localPoint.X, 0, localPoint.Z}.Length() - radius, math32.Abs(localPoint.Y) - height}
    return math32.Min(math32.Max(d.X, d.Y), 0) +
        Vector{math32.Max(d.X, 0), math32.Max(d.Y, 0), 0}.Length()
}
```

*Image - 94 - |2*

## 2. Optimalizácia Výkonu

Rozšírenie BVH akceleračnej štruktúry pre všetky SDF primitívy

Implementácia priestorovej partície špecifickej pre raymarching

## 3. Užívateľské Rozhranie

- Vytvorenie dedikovaného ovládacieho panelu pre raymarching

Tento rozšírený ray-marchingový systém umožní vytváranie komplexných tvarov prostredníctvom konštruktívnej solid geometrie, čo používateľom umožní budovať

zložité modely, ktoré by bolo ťažké dosiahnuť s tradičnou trojuholníkovou geometriou.

## 11.0 Podpora Post-Processing Shaderov

### 11.1 Úvod do Post-Processingu

Post-processing shadre sú kľúčovým nástrojom na vizuálne vylepšenie výstupného obrazu v ray-traceri. Umožňujú sofistikované úpravy renderovaného obrazu po jeho primárnom vygenerovaní.

### 11.2 Jazyk Shaderov Kage

**Pôvod a Charakteristika:** Kage je originálny shading jazyk vyvinutý pre herný engine Ebitengine. Má syntax podobnú jazyku Go, čo uľahčuje jeho použitie pre vývojárov oboznámených s týmto jazykom. Kage je navrhnutý s dôrazom na prenositeľnosť, pričom umožňuje kompliaciu shaderov na rôznych platformách bez potreby meniť ich kód.

**Podporované Typy a Funkcie:** Kage podporuje základné typy ako `bool`, `int`, `float`, vektory (`vec2`, `vec3`, `vec4`) a matice (`mat2`, `mat3`, `mat4`). Umožňuje využitie vstavaných funkcií na matematické operácie, manipuláciu s vektormi a textúrami.

## Príklad Syntaxe Kage Shaderu:

```
package main

// Edge detection strength
var Strength float
var AlphaR float
var AlphaG float
var AlphaB float
var Alpha float

// Convert RGB to grayscale intensity
func luminance(c vec3) float {
    return (c.r + c.g + c.b) / 3.0
}

func Fragment(position vec4, texCoord vec2, color vec4) vec4 {
    // Define pixel offset based on texture size
    offset := vec2(Strength, Strength)

    // Sample neighboring pixels
    topLeft := imageSrc0At(texCoord + vec2(-offset.x, -offset.y)).rgb
    top := imageSrc0At(texCoord + vec2(0.0, -offset.y)).rgb
    topRight := imageSrc0At(texCoord + vec2(offset.x, -offset.y)).rgb
    left := imageSrc0At(texCoord + vec2(-offset.x, 0.0)).rgb
    right := imageSrc0At(texCoord + vec2(offset.x, 0.0)).rgb
    bottomLeft := imageSrc0At(texCoord + vec2(-offset.x, offset.y)).rgb
    bottom := imageSrc0At(texCoord + vec2(0.0, offset.y)).rgb
    bottomRight := imageSrc0At(texCoord + vec2(offset.x, offset.y)).rgb

    middle := imageSrc0At(texCoord) * Alpha

    tl := luminance(topLeft)
    t := luminance(top)
    tr := luminance(topRight)
    l := luminance(left)
    r := luminance(right)
    bl := luminance(bottomLeft)
    b := luminance(bottom)
    br := luminance(bottomRight)

    // Sobel kernel
    gx := (-1.0 * tl) + (-2.0 * l) + (-1.0 * bl) + (1.0 * tr) + (2.0 * r) + (1.0 * br)
    gy := (-1.0 * tl) + (-2.0 * t) + (-1.0 * tr) + (1.0 * bl) + (2.0 * b) + (1.0 * br)

    // Compute gradient magnitude
    edge := sqrt((gx * gx) + (gy * gy)) * Alpha

    // Output edge as grayscale
    return vec4(middle.r + edge*AlphaR, middle.g + edge*AlphaB, middle.b + edge*AlphaB, middle.a * Alpha)
}
```

*Image - 95 - |2*

## 11.3 Podporované Post-Processing Efekty

### 11.3.1 Základné Efekty

- **Tint** (Farebný nádych):

Aplikácia farebného odtieňa na celý obraz. Tento efekt umožňuje upraviť celkový vizuálny tón renderovaného obrazu, čím sa dosahuje želaný **estetický vzhľad**.

- **Contrast** (Kontrast):

Zvyšovanie alebo znižovanie kontrastu medzi svetlými a tmavými časťami obrazu. Tento efekt umožňuje lepšie zvýraznenie detailov v scéne a zlepšuje vizuálny dojem.

- **Bloom** (Svetelný efekt):

Efekt, ktorý simuluje rozmazanie intenzívneho svetla na okrajoch svetelných zdrojov, čím sa vytvára dojem jasu a rozžiarenia. Tento efekt je často využívaný na zlepšenie realistikosti a vizuálnej dynamiky svetelných scén.

### 11.3.2 Komplexné Vizuálne Efekty

- **Bloom V2:**

Vylepšená verzia tradičného bloom efektu s lepším rozmazaním a prirodzenejšími svetelnými prechodmi. Pomáha dosiahnuť jemnejší a realistickejší vizuálny efekt svetelných zdrojov.

- **Sharpness** (Ostrost):

Zvýraznenie okrajov objektov v obraze, čo vedie k ostrejšiemu zobrazeniu detailov. Tento efekt môže byť užitočný pri zvýraznení textúr alebo kontúr v scéne.

- **Color Mapping** (Farebné mapovanie):

Tento efekt limituje počet možných hodnôt farebných kanálov v obraze, čím vytvára efekt zjednodušenia farebného spektra. Používa sa na dosiahnutie vizuálnych štýlov, kde je potrebné orezanie farebných hodnôt (napríklad v prípade retro alebo pixel art štýlov). Tento proces môže byť použitý aj na korekciu farebného gamutu obrazu pre špecifické farebné odtiene alebo estetické efekty.

- **Chromatic Aberration** (Chromatická aberácia):

Efekt, ktorý simuluje skreslenie svetelných lúčov pri prechode cez šošovku, spôsobujúce rozdelenie farieb na okrajoch objektov. Tento efekt pridáva vizuálny realizmus, často používaný v simulačných a filmových aplikáciách.

- **Edge Detection** (Detekcia hrán):

Tento efekt identifikuje hranice objektov v obraze, čím vytvára vizuálny efekt zvýraznenia kontúr. Môže byť užitočný na analyzovanie geometrie v obraze alebo na aplikovanie špecifických efektov na základné tvary.

- **Lighten** (Zosvetlenie):

Aplikácia svetelného efektu na zjasnenie tmavších oblastí obrazu. Tento efekt je vhodný na dosiahnutie jemného vylepšenia vizuálnej svetlosti v scénach s nízkym osvetlením.

## 11.4 Výhody Implementácie

Implementácia post-processing shaderov, najmä s využitím grafickej karty, ponúka zásadné výhody v oblasti výkonu. Grafické karty sú optimalizované na paralelný výpočtový proces, čo im umožňuje vykonávať komplexné výpočty oveľa rýchlejšie než CPU. Vďaka tejto schopnosti sú post-processing efekty schopné bežať efektívne aj v reálnom čase, čo výrazne zlepšuje vizuálny dojem bez výrazného spomalenia výkonu.

## 11.5 Záver

Implementácia post-processing shaderov predstavuje sofistikovaný prístup k vizuálnemu vylepšeniu obrazu generovaného raytracerom. Vďaka bohatému spektru efektov a minimálnej výpočtovej náročnosti je tento prístup veľmi efektívny.

Post-processing techniky nielenže zlepšujú estetiku renderovaných scén, ale tiež umožňujú flexibilitu pri tvorbe komplexných vizuálnych štýlov, ktoré by bolo ľahšie dosiahnuť pomocou tradičných metód.

## Záver

Vytvorením vlastného 3D ray-tracingového engine-u sme úspešne spojili teoretické poznatky z oblasti počítačovej grafiky s ich praktickou implementáciou. Hlavným cieľom bolo vybudovať vizualizačný systém, ktorý by nebol iba funkčný, ale zároveň aj

dostatočne flexibilný pre ďalší rozvoj. Počas vývoja sme prešli od jednoduchého zobrazovania trojuholníkov, cez optimalizáciu výkonu až po integráciu pokročilých efektov a podpory objemového (voxel) renderingu.

Veľkým prínosom bolo prehĺbenie poznatkov o fungovaní ray-tracingu na nízkej úrovni — od pochopenia samotného princípu sledovania lúčov v priestore, výpočtu ich prienikov so scénou až po simuláciu fyzikálnych vlastností svetla. Implementácia efektívnej štruktúry BVH pre urýchlenie výpočtov nám umožnila zvládať aj zložitejšie scény bez výrazného dopadu na výkon.

Súčasťou projektu bolo aj vytvorenie podpory pre volume rendering, vďaka čomu sa engine stal schopným vizualizovať nielen povrchové modely, ale aj objemové dátá, čo môže byť využiteľné napríklad v simuláciách alebo technickej vizualizácii. K tomu sme pridali aj systém post-processingu, vrátane podpory pre vlastné shaderové efekty, ktoré ešte viac rozšírili vizuálne možnosti engine-u a otvorili dvere k ďalšiemu experimentovaniu.

Za zmienku stojí aj jednoduché frontendové rozhranie, ktoré umožňuje základnú úpravu scény bez nutnosti priamej manipulácie s kódom. Tento krok bol dôležitý najmä z pohľadu rozšíriteľnosti projektu a pohodlia pri testovaní rôznych nastavení.

Počas vývoja sme museli riešiť aj rôzne optimalizačné problémy — od optimalizácie dátových štruktúr až po správu pamäte a využívanie možností paraleлизácie. Projekt bol teda výbornou príležitosťou na získanie praktických skúseností nielen s počítačovou grafikou, ale aj so samotným programovacím jazykom Go, s ktorým som pred začiatkom projektu nemal výraznejšie skúsenosti.

Výsledný engine predstavuje solídný základ pre budúce smerovanie. V budúcnosti by bolo možné integrovať pokročilejšie algoritmy na realistickejšie osvetlenie a globálnu ilumináciu, pridať podporu pre real-time ray-tracing alebo rozšíriť systém o animácii a dynamické objekty.

Projekt zároveň ukázal, že aj s relatívne obmedzeným časom a bez použitia hotových grafických knižníc je možné navrhnúť vlastný ray-tracingový systém, ktorý kombinuje výkon, flexibilitu a rozšíriteľnosť.

Počas práce som si nielen osvojil množstvo technických poznatkov, ale tiež získal cenné skúsenosti s navrhovaním komplexnejších systémov a hľadaním riešení na rôzne

technické problémy. Tento projekt pre mňa predstavoval nielen výzvu, ale aj obohacujúcu skúsenosť, ktorá ma posunula bližšie k profesionálnej úrovni v oblasti počítačovej grafiky a optimalizácií.

## ZOZNAM BIBLIOGRAFICKÝCH ODKAZOV

### Online Knihy o Ray Tracingu

[RayTracingInOneWeekend](#)

[Ray Tracing: The Next Week](#)

[Ray Tracing: The Rest of Your Life](#)

### Technické Videá a Prezentácie

[Why is recursion bad?](#)

[Why you should avoid Linked List](#)

[How Big Budget AAA Games Render Bloom](#)

[Andrew Kelley Practical Data Oriented Design \(DoD\)](#)