STREDNÁ PRIEMYSELNÁ ŠKOLA ELEKTROTECHNICKÁ
ZOCHOVA 9, BRATISLAVA

Golang Ray-Tracer

projekt z praktickej časti odbornej zložky maturitnej skúšky

Meno kandidáta: Matus Bencek **Trieda: 4.D**

Vedúci práce: Mgr. Jakub Krcho **Šk. rok: 2023/2024**

STREDNÁ PRIEMYSELNÁ ŠKOLA ELEKTROTECHNICKÁ

ZOCHOVA 9, BRATISLAVA

Meno kandidáta: Matus Bencek **Trieda: 4.D**

Vedúci práce: Mgr. Jakub Krcho

Rozsah: 11 stran

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: Matus Bencek

Obsah

1.1 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.

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 frameworku**
- Pozostáva z dvoch hlavných komponentov:
    - **Ray-tracing engine**: Jadro výpočtového systému pre renderovanie
    - **Webový server**: Zabezpečuje komunikáciu s frontendovou časťou
- Backend beží asynchrónne vo vlastných go-rutinách, čo minimalizuje potrebu zložitého manažmentu stavu a používania mutexov s pozitim usefe, 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

```
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:"metalic"`
    RenderVolume    bool    `json:"renderVolume"`
    RenderVoxels    bool    `json:"renderVoxels"`
}
```

- `POST /submitVoxel`: Odoslať 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`: Odoslať 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` : Odoslať 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` : Odoslať konfiguráciu shadera

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

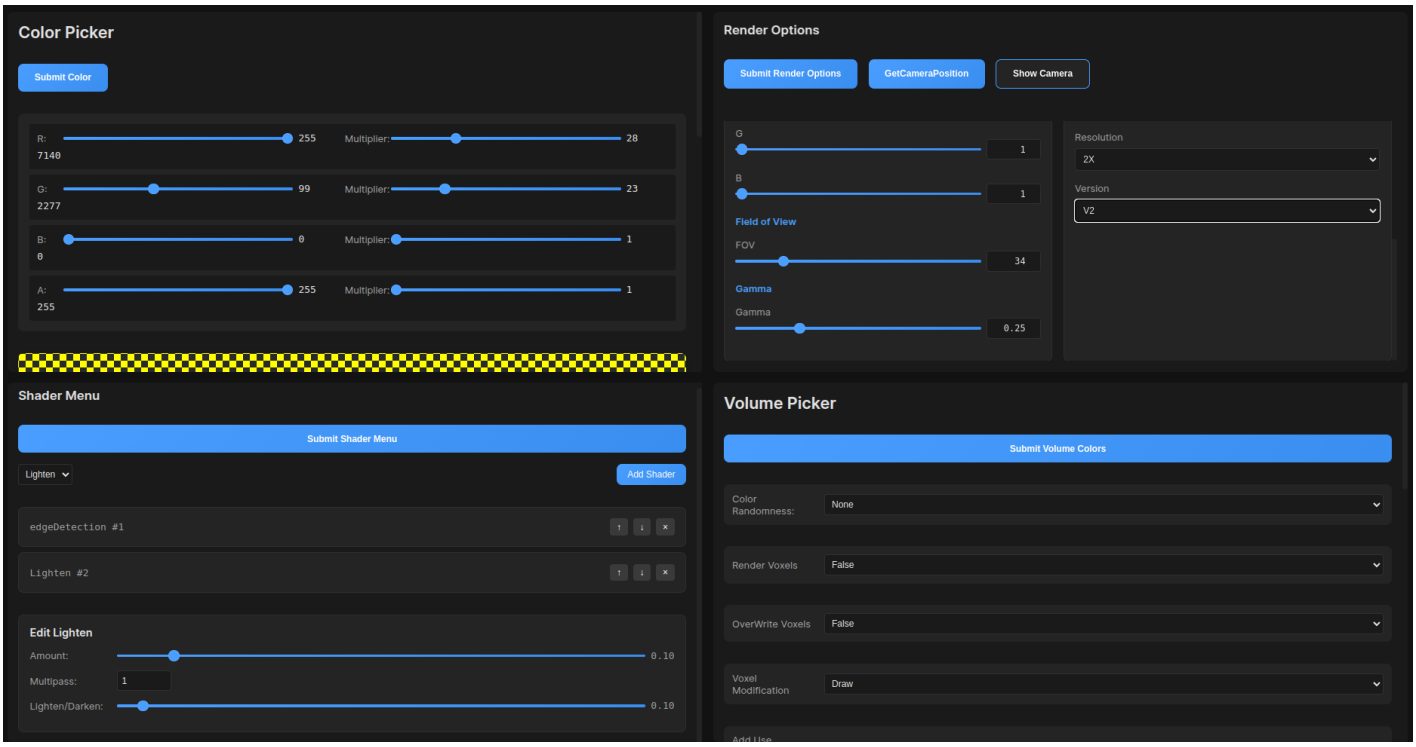
- `GET /getCameraPosition` : Získať aktuálnu pozíciu kamery

```
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"`
}
```

2.3 Dokumentácia Frontend Komponentov Ray Tracingu



2.3.1 Color Picker

Farebné Kanály (Multiplayer Aktivovaný)

- 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 × 128 × 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 × 128 × 3
- Rozsah Normalizácie: -1 až 1
- Konverzia na Backende: Normalizovaná na vektor

Funkcie Normal Mapy

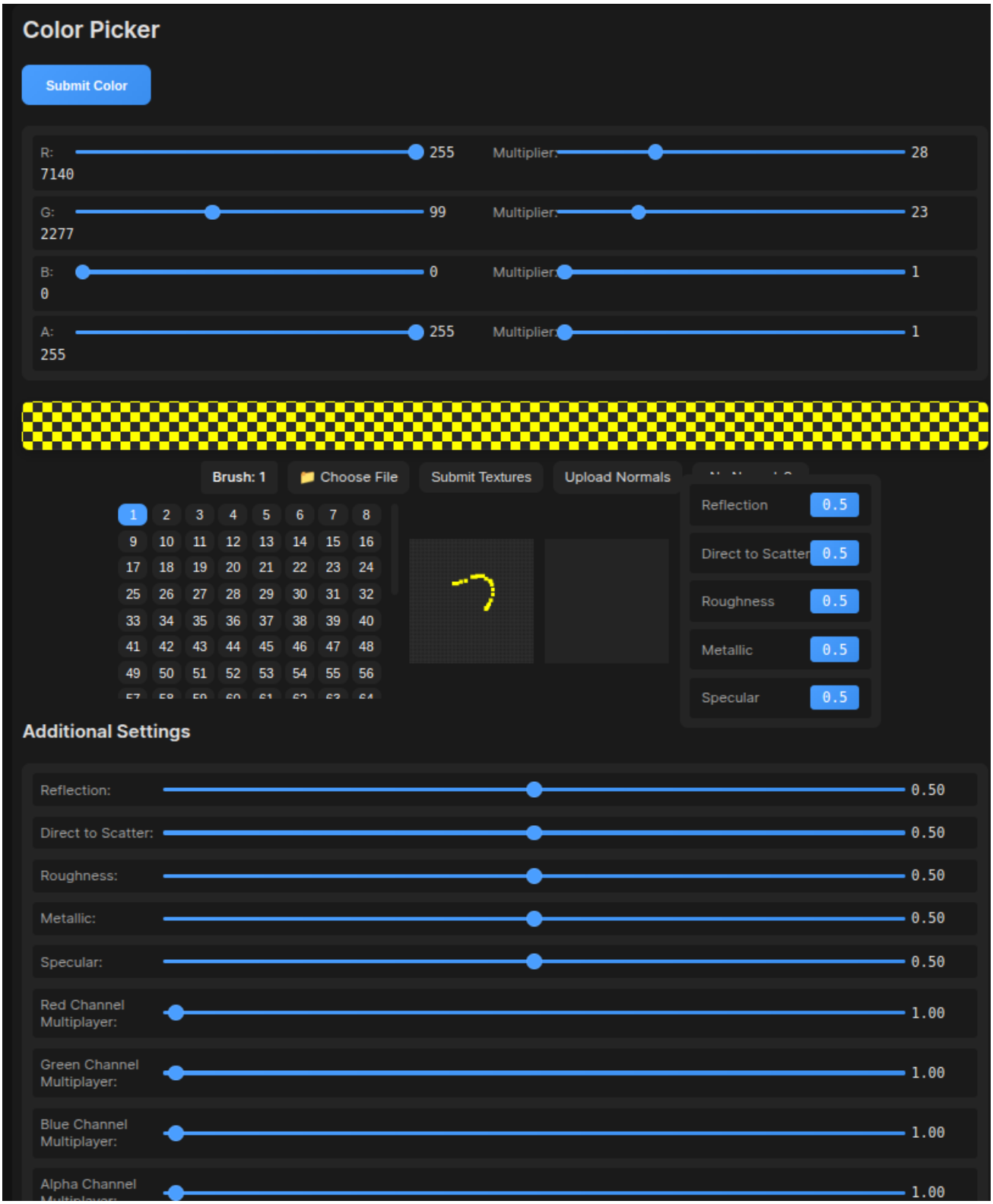
- Tlačidlo Nahraj Normal Mapu: Umožňuje nahrať normal mapy
- Tlačidlo "Žiadna Normal Mapa": Otvára online generátor normal máp (https://cpetry.github.io/NormalMap-Online/)

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



2.3.3 Shader Menu

Účel

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

Správa Shaderov

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

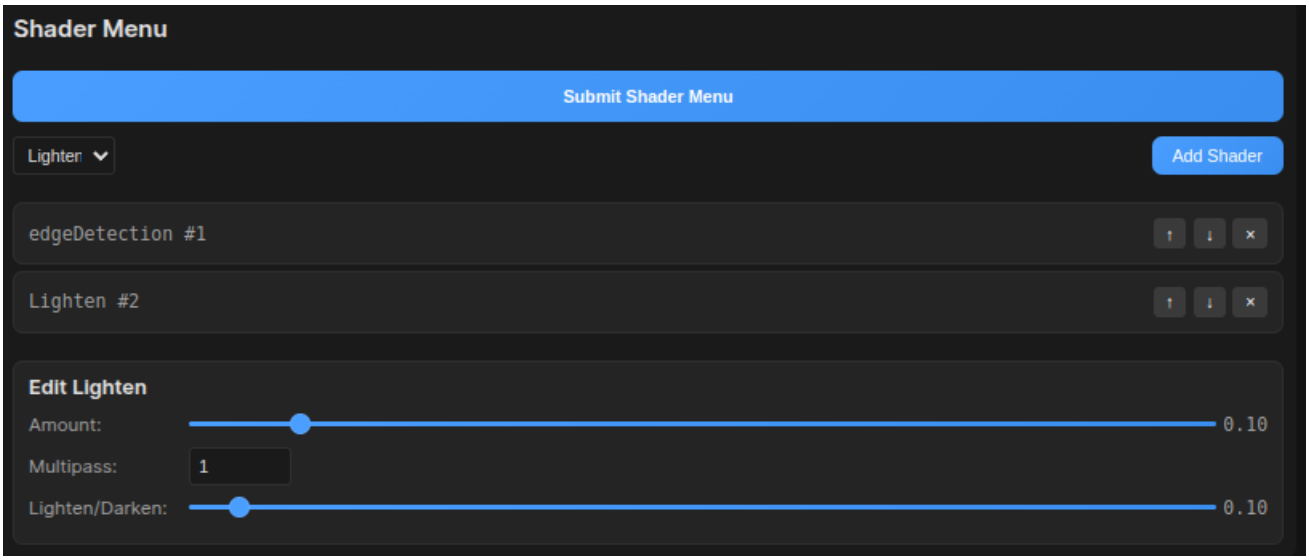
Parametre Shaderov

Spoločné Parametre

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

Podporované Shadery

1. **Kontrast**
    - Množstvo
    - Multipass
    - Sila kontrastu
2. **Tint**
    - Množstvo
    - Multipass
    - Tint farba
    - Sila tint shaderu
3. **Bloom**
    - Množstvo
    - Multipass
    - Prahová hodnota
    - Intenzita
4. **BloomV2**
    - Podobné Bloomu s miernym variantom
5. **Ostrosť**
    - Množstvo
    - Multipass
    - Sila filtra
6. **Mapovanie Farieb**
    - Množstvo
    - Multipass
    - Farebné kanály (R/G/B)
    - Definuje distribúciu farieb (napr. 2 úrovne: 0% alebo 100%)
7. **Chromatická Aberácia**
    - Množstvo
    - Multipass
    - Sila filtra
    - Posun farebného kanála (Červená vľavo, Modrá vpravo)
8. **Detekcia Hrán**
    - Používa Sobelov filter
    - Množstvo
    - Multipass
    - Sila zvýraznenia hrán
    - Nastaviteľná farba hrán (R/G/B)
9. **Zosvetlenie**
    - Množstvo
    - Multipass
    - Sila filtra



2.3.4 Render Options

Správa Kamery

- Odoslať Render Možnosti
- Tlačidlo Získať Pozíciu Kamery
- Skryť/Zobraziť Pozíciu Kamery
- Presunúť Kameru na Špecifickú Pozíciu

Hlavné Parametre Renderingu

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

Parametre Osvetlenia

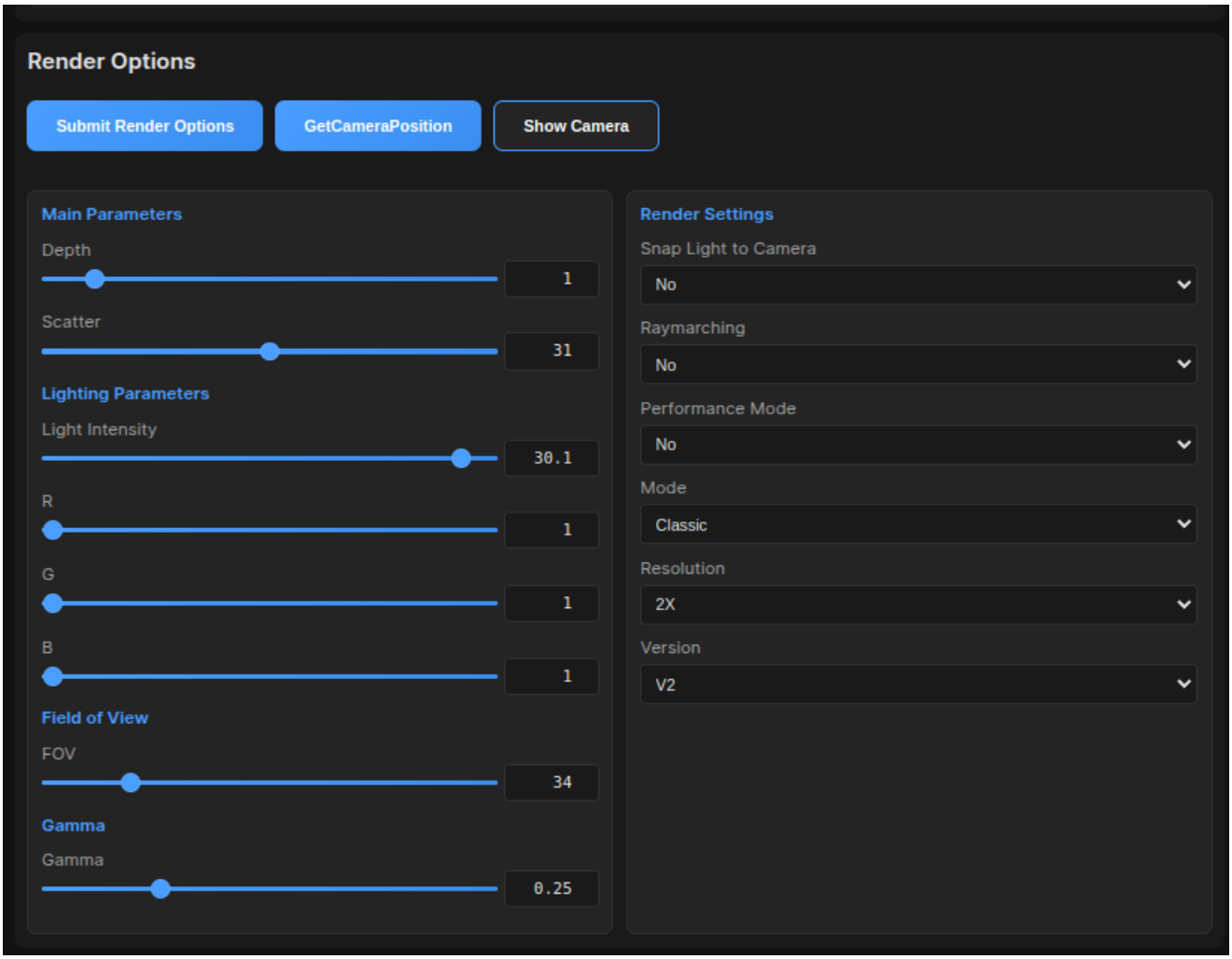
- Intenzita Svetla
- Farba Svetla (R/G/B)
- Zorné Pole
- Gama: Kontrast a jas medzi tmavými a svetlými tónmi

Nastavenia Renderingu

- Pripnúť Svetlo ku Kamere
- Raymarching (momentálne neimplementované)
- Performance Mód
  - Odobratie `wg.Wait`
  - Potenciálne menej plynulý rendering
  - Maximalizácia využitia hardvéru

Módy Renderingu

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



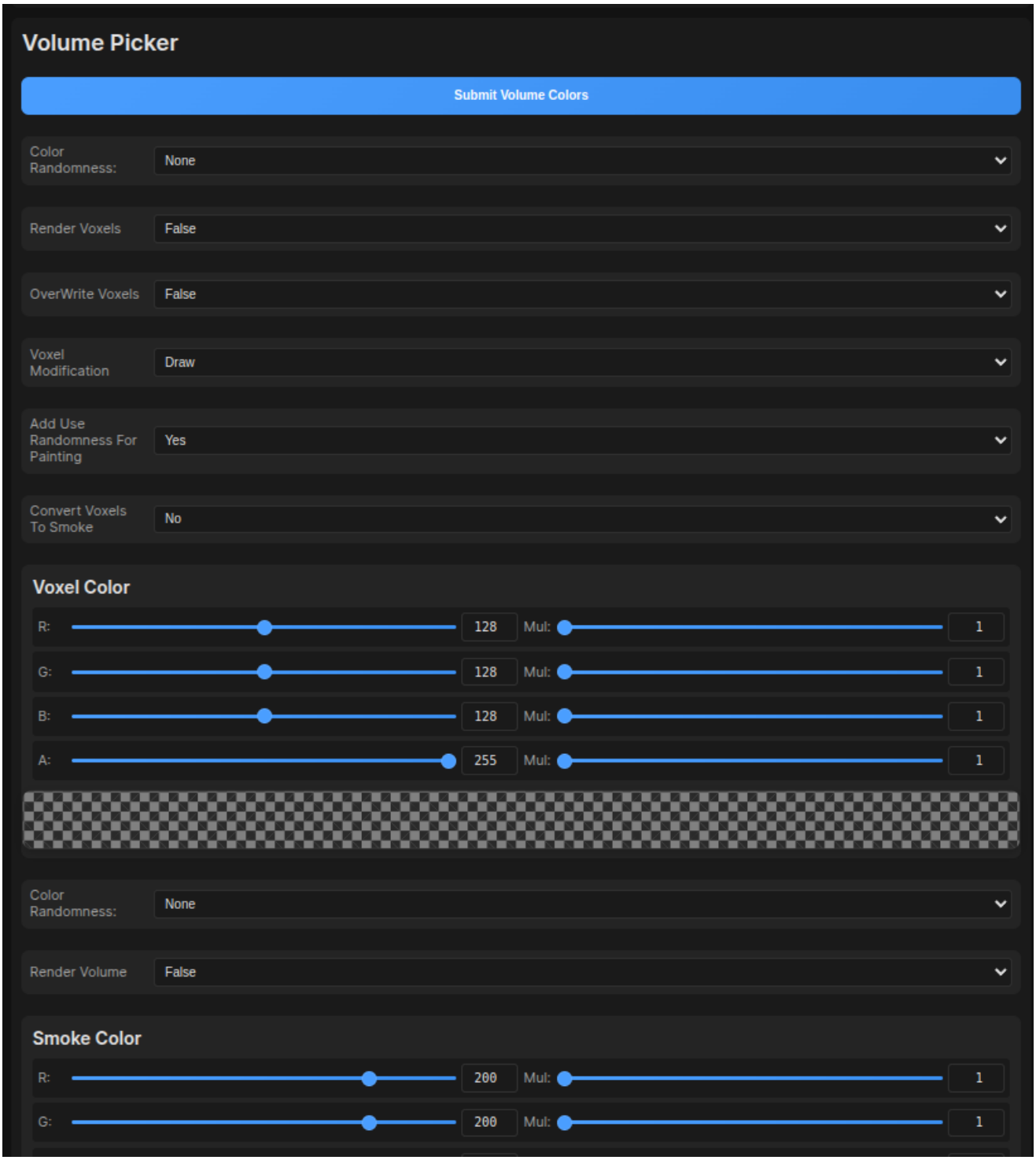
2.3.5 Volume Picker

Správa Farby Objemu

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

Vlastnosti Objemu

- Výber Farby Voxelov s Náhľadom
- Výber Farby Dymu
- Hustota
- Priehľadnosť (priehľadnosť objemu)



3.1 Princíp fungovania BVH

Pri ray-tracingu je kľúčovou operáciou hľadanie priesečníkov 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 obaľujúcich objemov (najčastejšie osovo zarovnaných boxov - AABB), ktorá umožňuje rýchlo eliminovať veľké časti scény, ktoré lúč nemôže zasiahnuť.

Keď lúč prechádza scénou, najprv sa testuje prienik s head Node BVH. Ak lúč nezasiahne obaľujúci 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 konkrétne 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 potomkovi
- `L`, `R`, `P` označujú ľavého potomka, pravého potomka a parent Node

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 tromi 3D vektormi (v1, v2, v3), 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 kľúčových 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
    - Umožňuje presnejšie zachytenie a reprezentáciu svetelných efektov v scéne
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 kľúčové 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
    - Podporuje fyzikálne korektné miešanie farieb a svetelných príspevkov

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 rozptýlená náhodným smerom (matný povrch)
- Hodnota blízka 1: Prevláda priamy odraz svetla (lesklý povrch) Tento parameter je kľúčový pre realistické zobrazenie rôznych typov materiálov, od matných až po vysoko lesklé povrchy.

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 simulovať materiály

od úplne matných až po zrkadlové povrchy.

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

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

3.3.3 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
}
```

Nová Štruktúra TriangleBBOX

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

Kľúčové zmeny:

- Veľkosť štruktúry sa zmenšila z 88 na 52 bajtov
- Zjednotenie bounding boxu a trojuholníka
- Vlastnosti trojuholníka sú teraz definované samostatne

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
}
```

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.

3.3 BVH a jej implementácia

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

Vývojová cesta

1. **Naivný prístup** - 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.
2. **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.
3. **BVH implementácia** - Finálnym riešením bola implementácia Bounding Volume Hierarchy (BVH), ktorá hierarchicky organizuje priestor a umožňuje efektívne prechádzanie len relevantných častí scény, čím znižuje časovú zložitosť na približne O(log n).

Evolúcia BVH štruktúry

Pôvodná BVH implementácia

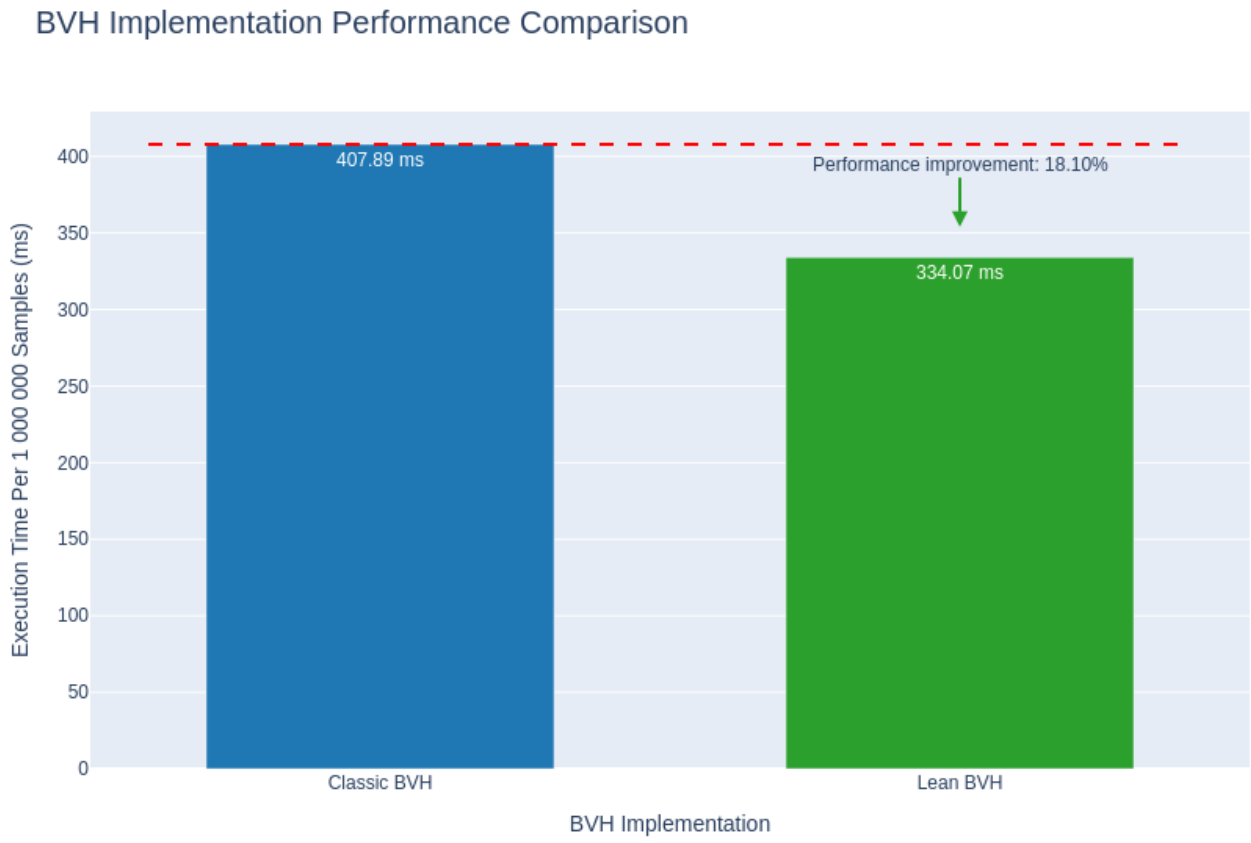
```
type BVHNode struct { // veľkosť=136 (0x88) bajtov
    Left, Right *BVHNode
    BoundingBox [2]Vector
    Triangles   TriangleSimple
    active      bool
}
```

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.

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
}
```

- Classic BVHNode : 407.888788ms
- BVHLean : 341.148485ms



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 uzle a nahradila ich systémom ID odkazov na textúry

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.

3.4 Podpora Načítavania 3D Geometrie

3.4.1 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

1. Podpora Geometrie

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

2. Podpora Materiálov

- **Parsing .mtl súborov**
- **Načítavanie základných materiálových vlastností:**
  - Difúzna farba
  - Odrazivosť
  - Spekulárne vlastnosti
  - Priehľadnosť

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. **Extrahovanie normálových vektorov**
3. **Identifikácia a konverzia polygónov na trojuholníky**
4. **Priradenie materiálových vlastností jednotlivým geometrickým prvkom**

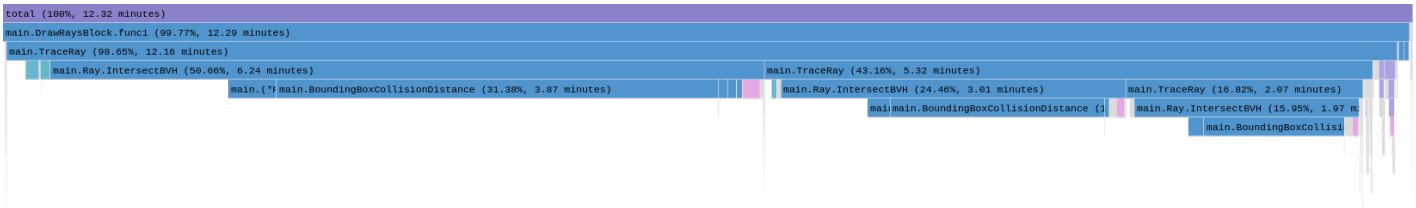
4.0 RayTracing Vývoj Funkcionality

Pôvodná funkcia, ktorá poskytuje základnú ray tracing funkcionalitu:

TraceRay

Web Name : V1

- Používa BVH štruktúru pre testy priesečníkov
- 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
- Vykonáva výpočet tieňov pomocou shadow rays
- Kombinuje priame svetlo, rozptýlené svetlo a odrazy lineárne



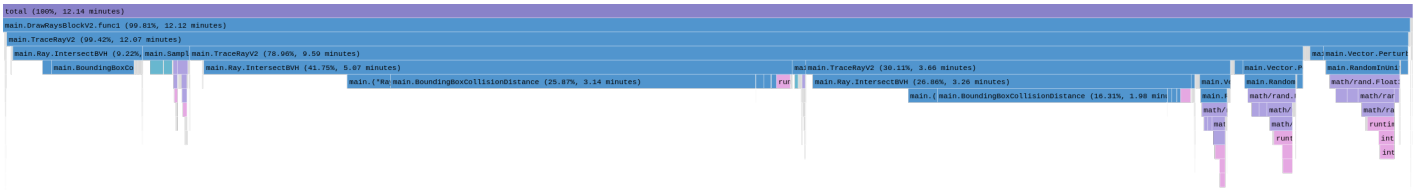
Location	Self	Total
main.BoundingBoxCollisionDistance	6.96 minutes	6.98 minutes
main.Ray.IntersectBVH	2.78 minutes	11.22 minutes
main.(*Ray).IntersectTriangleSimple	0.75 minutes	0.75 minutes
runtime.duffcopy	0.28 minutes	0.28 minutes
main.TraceRay	0.27 minutes	19.55 minutes
github.com/chewxy/math32.Cos	0.19 minutes	0.19 minutes
main.Vector.Cross	0.14 minutes	0.15 minutes
main.Vector.Dot	0.14 minutes	0.14 minutes
github.com/chewxy/math32.Sin	0.13 minutes	0.13 minutes
main.Vector.Sub	0.10 minutes	0.10 minutes
main.Vector.Mul	0.08 minutes	0.08 minutes
main.Vector.Normalize	0.07 minutes	0.07 minutes
internal/chacha8rand.block	0.06 minutes	0.06 minutes
runtime.rand	0.05 minutes	0.12 minutes
runtime.duffzero	0.05 minutes	0.05 minutes
math/rand.(*Rand).Float64	0.04 minutes	0.04 minutes
math/rand.(*Rand).Float32	0.03 minutes	0.03 minutes
math/rand.globalRand	0.02 minutes	0.02 minutes
runtime.asyncPreempt	0.02 minutes	0.02 minutes
math/rand.(*runtimeSource).Int63	0.02 minutes	0.15 minutes
main.DrawRaysBlock.func1	0.01 minutes	12.29 minutes
math/rand.Float64	0.01 minutes	0.03 minutes
math/rand.(*Rand).Int63	0.01 minutes	0.16 minutes
main.PrecomputeScreenSpaceCoordinatesSphere	0.01 minutes	0.01 minutes
internal/chacha8rand.(*State).Next	0.01 minutes	0.01 minutes
main.Vector.Add	0.01 minutes	0.01 minutes
math/rand.Float32	0.01 minutes	0.15 minutes
github.com/chewxy/math32.IsInf	< 0.01 minutes	< 0.01 minutes
github.com/chewxy/math32.IsNaN	< 0.01 minutes	< 0.01 minutes
internal/chacha8rand.(*State).Refill	< 0.01 minutes	0.07 minutes
main.clampUint8	< 0.01 minutes	< 0.01 minutes
runtime.memmove	< 0.01 minutes	< 0.01 minutes
math.Sqrt	< 0.01 minutes	< 0.01 minutes
github.com/chewxy/math32.max	< 0.01 minutes	< 0.01 minutes
math/rand.readPos int8 }]).Load	< 0.01 minutes	< 0.01 minutes
runtime.futex	< 0.01 minutes	< 0.01 minutes

- V1 Profile

TraceRayV2

Web Name : V2

- Logickejšie organizuje kód, oddeľuje priame osvetlenie, nepriame osvetlenie a odrazy
- Pridáva perturbáciu smerov odrazov založenú na drsnosti
- Implementuje fyzikálnejšiu energetickú konzerváciu
- Používa hemisphere sampling so zlepšenou logikou rozptylu
- Kombinuje komponenty pomocou fyzikálnejšieho prístupu
- Lepšie spracováva energetickú rovnováhu medzi difúznym a zrkadlovým svetlom



Location	Self	Total
main.BoundingBoxCollisionDistance	5.83 minutes	5.83 minutes
main.Ray.IntersectBVH	2.31 minutes	9.45 minutes
main.(*Ray).IntersectTriangleSimple	0.70 minutes	0.70 minutes
main.Vector.Normalize	0.36 minutes	0.36 minutes
internal/chacha8rand.block	0.30 minutes	0.30 minutes
runtime.rand	0.25 minutes	0.56 minutes
math/rand.(*Rand).Float64	0.23 minutes	0.23 minutes
runtime.duffcopy	0.23 minutes	0.23 minutes
math/rand.(*Rand).Float32	0.22 minutes	0.22 minutes
main.TraceRayV2	0.20 minutes	25.32 minutes
github.com/chewxy/math32.Cos	0.15 minutes	0.15 minutes
main.Vector.Dot	0.15 minutes	0.15 minutes
main.Vector.Cross	0.13 minutes	0.13 minutes
math/rand.Float32	0.11 minutes	1.36 minutes
math/rand.globalRand	0.10 minutes	0.10 minutes
github.com/chewxy/math32.Sin	0.10 minutes	0.10 minutes
main.Vector.Sub	0.09 minutes	0.09 minutes
main.SampleHemisphere	0.09 minutes	0.55 minutes
math/rand.(*runtimeSource).Int63	0.08 minutes	0.69 minutes
math/rand.(*Rand).Int63	0.07 minutes	0.76 minutes
main.RandomInUnitSphere	0.07 minutes	1.31 minutes
main.Vector.Mul	0.06 minutes	0.06 minutes
internal/chacha8rand.(*State).Next	0.04 minutes	0.04 minutes
main.Vector.Perturb	0.04 minutes	1.51 minutes
runtime.duffzero	0.03 minutes	0.03 minutes
internal/chacha8rand.(*State).Refill	0.02 minutes	0.32 minutes
main.Vector.LengthSquared	0.02 minutes	0.02 minutes
main.Vector.Add	0.02 minutes	0.02 minutes
main.DrawRaysBlockV2.func1	0.01 minutes	12.12 minutes
github.com/chewxy/math32.max	0.01 minutes	0.01 minutes
math/rand.readPos int8 }]).Load	0.01 minutes	0.01 minutes
main.ColorFloat32.MulScalar	0.01 minutes	0.01 minutes
main.ColorFloat32.Add	< 0.01 minutes	< 0.01 minutes
main.PrecomputeScreenSpaceCoordinatesSphere	< 0.01 minutes	< 0.01 minutes
math/rand.Float64	< 0.01 minutes	0.02 minutes

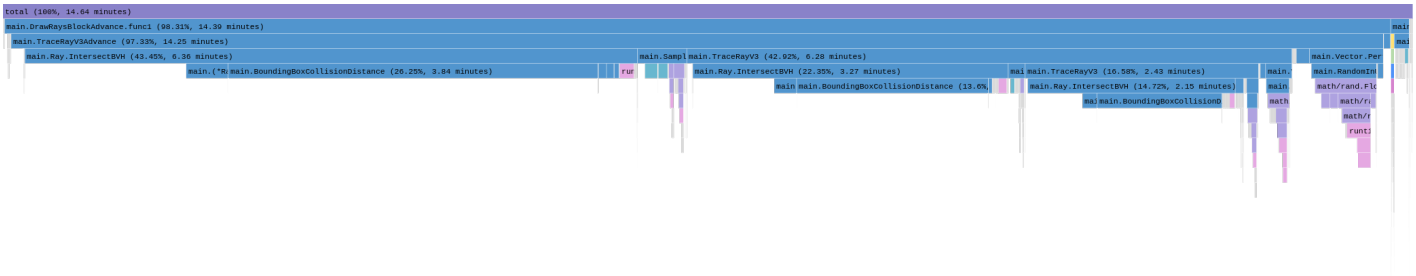
- V2 Profile

TraceRayV3

Web Name : Not Implemented

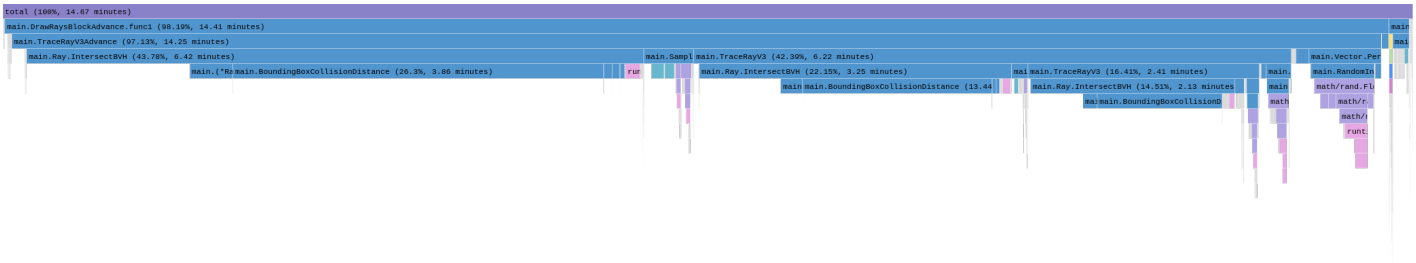
PBR (Physically Based Rendering) prístup, ktorý:

- Implementuje Fresnel-Schlick aproximáciu pre výpočet odrazov
- Používa GGX distribúciu pre microfacet-based zrkadlové body
- Počíta dôležité dot produkty (NdotL, NdotV, NdotH) pre PBR výpočty
- Lepšie simuluje materiálové vlastnosti ako kovový lesk a drsnosť
- Používa presnejšiu energetickú konzerváciu pre kombinovanie komponentov
- Vracia jednu farebnú hodnotu



Location	Self	Total
main.BoundingBoxCollisionDistance	7.12 minutes	7.13 minutes
main.Ray.IntersectBVH	3.09 minutes	11.79 minutes
main.(*Ray).IntersectTriangleSimple	0.82 minutes	0.82 minutes
main.Vector.Normalize	0.38 minutes	0.38 minutes
runtime.duffcopy	0.29 minutes	0.29 minutes
internal/chacha8rand.block	0.27 minutes	0.27 minutes
runtime.rand	0.21 minutes	0.49 minutes
github.com/chewxy/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/chewxy/math32.Sin	0.15 minutes	0.15 minutes
main.TraceRayV3Advance	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	1.14 minutes
main.Vector.Sub	0.10 minutes	0.10 minutes
main.TraceRayV3	0.08 minutes	8.71 minutes
math/rand.(*Rand).Int63	0.08 minutes	0.68 minutes
math/rand.(*runtimeSource).Int63	0.07 minutes	0.60 minutes
main.RandomInUnitSphere	0.06 minutes	1.03 minutes
runtime.duffzero	0.05 minutes	0.05 minutes
internal/chacha8rand.(*State).Next	0.04 minutes	0.04 minutes
github.com/chewxy/math32.Pow	0.04 minutes	0.06 minutes
main.Vector.Perturb	0.03 minutes	1.16 minutes
github.com/chewxy/math32.max	0.03 minutes	0.03 minutes
github.com/chewxy/math32.archExp	0.03 minutes	0.03 minutes
main.Vector.Mul	0.03 minutes	0.03 minutes
main.DrawRaysBlockAdvance.func1	0.03 minutes	14.39 minutes
github.com/chewxy/math32.archLog	0.03 minutes	0.03 minutes
runtime.asyncPreempt	0.02 minutes	0.02 minutes
github.com/chewxy/math32.IsInf	0.02 minutes	0.02 minutes
github.com/chewxy/math32.modf	0.02 minutes	0.02 minutes
internal/chacha8rand.(*State).Refill	0.02 minutes	0.28 minutes
math/rand.readPos.int8_11).Load	0.02 minutes	0.02 minutes

- V2Lin Profile



Location	Self	Total
main.BoundingBoxCollisionDistance	7.12 minutes	7.13 minutes
main.Ray.IntersectBVH	3.09 minutes	11.80 minutes
main.(*Ray).IntersectTriangleSimple	0.82 minutes	0.83 minutes
main.Vector.Normalize	0.38 minutes	0.38 minutes
runtime.duffcopy	0.29 minutes	0.29 minutes
internal/chacha8rand.block	0.25 minutes	0.25 minutes
runtime.rand	0.21 minutes	0.49 minutes
github.com/chewxy/math32.Cos	0.20 minutes	0.20 minutes
main.Vector.Cross	0.17 minutes	0.17 minutes
math/rand.(*Rand).Float32	0.16 minutes	0.16 minutes
math/rand.(*Rand).Float64	0.16 minutes	0.16 minutes
main.Vector.Dot	0.16 minutes	0.16 minutes
github.com/chewxy/math32.Sin	0.15 minutes	0.15 minutes
main.TraceRayV3Advance	0.14 minutes	14.25 minutes
main.SampleHemisphere	0.12 minutes	0.79 minutes
math/rand.globalRand	0.12 minutes	0.12 minutes
math/rand.Float32	0.10 minutes	1.12 minutes
main.Vector.Sub	0.10 minutes	0.10 minutes
math/rand.(*runtimeSource).Int63	0.08 minutes	0.60 minutes
main.TraceRayV3	0.08 minutes	8.63 minutes
math/rand.(*Rand).Int63	0.07 minutes	0.67 minutes
main.RandomInUnitSphere	0.06 minutes	1.01 minutes
runtime.duffzero	0.04 minutes	0.04 minutes
github.com/chewxy/math32.Pow	0.04 minutes	0.06 minutes
internal/chacha8rand.(*State).Next	0.04 minutes	0.04 minutes
main.Vector.Mul	0.03 minutes	0.03 minutes
github.com/chewxy/math32.max	0.03 minutes	0.03 minutes
github.com/chewxy/math32.archExp	0.03 minutes	0.03 minutes
main.DrawRaysBlockAdvance.func1	0.03 minutes	14.41 minutes
runtime.asyncPreempt	0.03 minutes	0.03 minutes
github.com/chewxy/math32.archLog	0.03 minutes	0.03 minutes
github.com/chewxy/math32.IsInf	0.03 minutes	0.03 minutes
main.Vector.Perturb	0.03 minutes	1.15 minutes
github.com/chewxy/math32.modf	0.03 minutes	0.03 minutes
internal/chacha8rand.(*State).Refill	0.02 minutes	0.27 minutes
math/rand.readPos.int8_11).Load	0.02 minutes	0.02 minutes

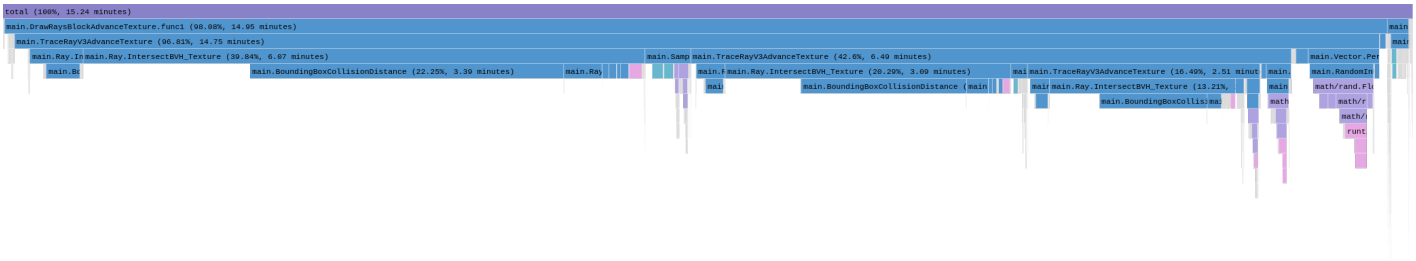
- V2Log Profile

TraceRayV3Advance

Web Name : V2Liner / V2Log

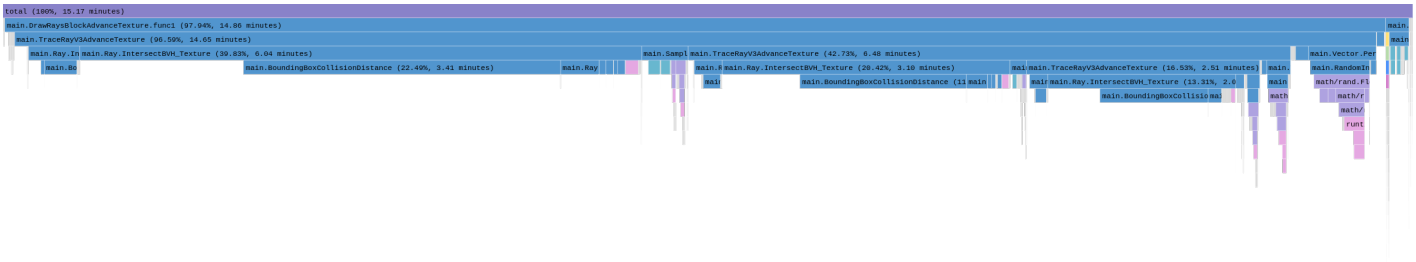
Rozšírenie TraceRayV3, ktoré:

- Vracia dodatočné dáta: farbu, vzdialenosť a normálový vektor
- Umožňuje pokročilejšie post-processing techniky
- Inak používa rovnaký PBR prístup ako TraceRayV3
- Podporuje ukladanie dát pre deferred shading techniky



Location	Self	Total
main.BoundingBoxCollisionDistance	7.03 minutes	7.04 minutes
main.Ray.IntersectBVH_Texture	3.17 minutes	11.18 minutes
main.Ray.IntersectTriangleTexture	0.80 minutes	0.81 minutes
main.Vector.Normalize	0.44 minutes	0.44 minutes
runtime.duffcopy	0.29 minutes	0.29 minutes
main.Ray.IntersectBVH	0.27 minutes	1.10 minutes
internal/chacha8rand.block	0.26 minutes	0.26 minutes
main.TraceRayV3AdvanceTexture	0.22 minutes	23.76 minutes
runtime.rand	0.20 minutes	0.48 minutes
github.com/chewxy/math32.Cos	0.19 minutes	0.19 minutes
main.Vector.Dot	0.19 minutes	0.19 minutes
main.Vector.Cross	0.17 minutes	0.17 minutes
main.Vector.Sub	0.17 minutes	0.17 minutes
math/rand.(*Rand).Float32	0.17 minutes	0.17 minutes
math/rand.(*Rand).Float64	0.17 minutes	0.17 minutes
github.com/chewxy/math32.Sin	0.15 minutes	0.15 minutes
math/rand.globalRand	0.13 minutes	0.13 minutes
main.SampleHemisphere	0.12 minutes	0.76 minutes
math/rand.Float32	0.11 minutes	1.16 minutes
math/rand.(*runtimeSource).Int63	0.08 minutes	0.60 minutes
math/rand.(*Rand).Int63	0.07 minutes	0.67 minutes
main.(*Ray).IntersectTriangleSimple	0.07 minutes	0.07 minutes
main.RandomInUnitSphere	0.06 minutes	1.06 minutes
github.com/chewxy/math32.archExp	0.05 minutes	0.05 minutes
internal/chacha8rand.(*State).Next	0.04 minutes	0.04 minutes
runtime.duffzero	0.04 minutes	0.04 minutes
main.DrawRaysBlockAdvanceTexture.func1	0.04 minutes	14.95 minutes
github.com/chewxy/math32.max	0.04 minutes	0.04 minutes
github.com/chewxy/math32.Pow	0.04 minutes	0.05 minutes
github.com/chewxy/math32.archLog	0.04 minutes	0.04 minutes
main.Vector.Mul	0.03 minutes	0.04 minutes
github.com/chewxy/math32.IsInf	0.03 minutes	0.03 minutes
runtime.asyncPreempt	0.02 minutes	0.02 minutes
main.Vector.Perturb	0.02 minutes	1.18 minutes
github.com/chewxy/math32.modf	0.02 minutes	0.02 minutes
math/rand.readPos[int8_11].Load	0.02 minutes	0.02 minutes

- V2LinTexture Profile



Location	Self	Total
main.BoundingBoxCollisionDistance	7.01 minutes	7.01 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	0.26 minutes	1.05 minutes
internal/chacha8rand.block	0.24 minutes	0.24 minutes
main.TraceRayV3AdvanceTexture	0.22 minutes	23.64 minutes
runtime.rand	0.21 minutes	0.46 minutes
github.com/chewxy/math32.Cos	0.20 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	0.16 minutes	0.16 minutes
github.com/chewxy/math32.Sin	0.15 minutes	0.15 minutes
math/rand.globalRand	0.11 minutes	0.11 minutes
main.SampleHemisphere	0.11 minutes	0.77 minutes
math/rand.Float32	0.11 minutes	1.09 minutes
math/rand.(*runtimeSource).Int63	0.08 minutes	0.58 minutes
main.(*Ray).IntersectTriangleSimple	0.07 minutes	0.07 minutes
math/rand.(*Rand).Int63	0.07 minutes	0.65 minutes
main.RandomInUnitSphere	0.06 minutes	0.99 minutes
runtime.duffzero	0.05 minutes	0.05 minutes
github.com/chewxy/math32.Pow	0.05 minutes	0.06 minutes
github.com/chewxy/math32.archExp	0.05 minutes	0.05 minutes
github.com/chewxy/math32.max	0.05 minutes	0.05 minutes
github.com/chewxy/math32.archLog	0.04 minutes	0.04 minutes
internal/chacha8rand.(*State).Next	0.04 minutes	0.04 minutes
main.DrawRaysBlockAdvanceTexture.func1	0.04 minutes	14.86 minutes
main.Vector.Mul	0.03 minutes	0.03 minutes
github.com/chewxy/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/chewxy/math32.modf	0.02 minutes	0.02 minutes

- V2LinTexture Profile

TraceRayV3AdvanceTexture

Web Name : V2LinearTexture / V2LogTexture

Verzia s podporou textúr, ktorá:

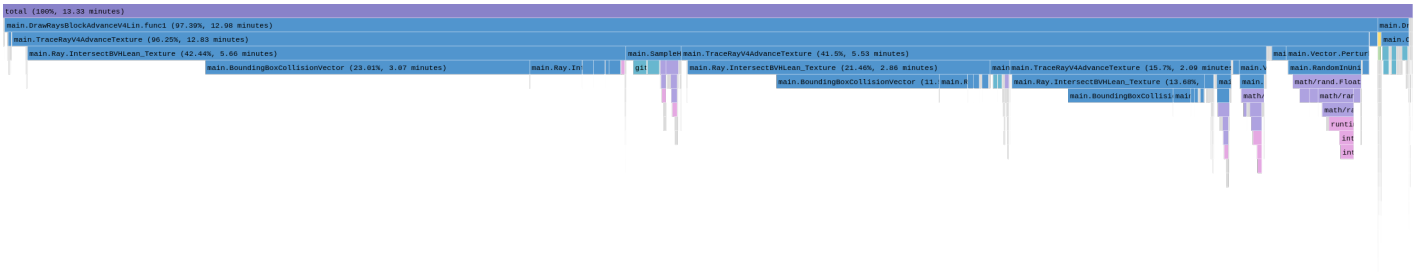
- Integruje materiálové vlastnosti z textúr
- Používa textureMap parameter pre prístup k dátam textúr
- Vracia informácie o farbe a normále pre post-processing
- Používa špecializovaný BVH traversal (`IntersectBVH_Texture`) pre podporu textúr
- Aplikuje dáta textúr na materiálové parametre ako drsnosť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

TraceRayV4AdvanceTexture

Web Name : V4Linear / V4Log

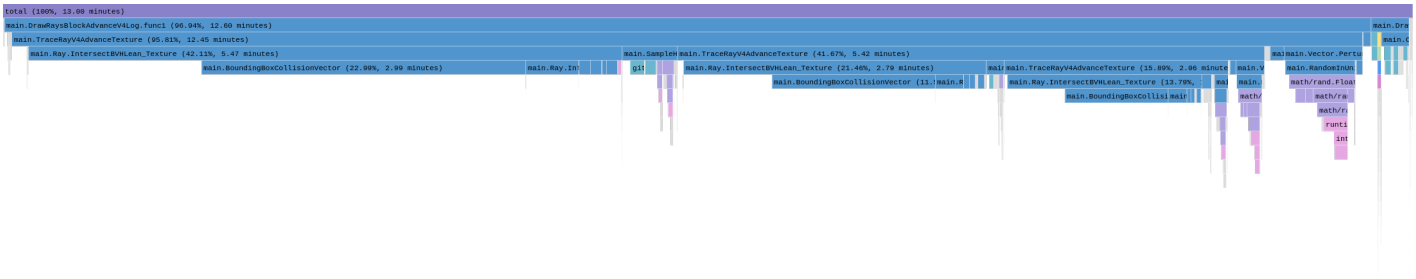
Optimalizovaná verzia s podporou textúr, ktorá:

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



Location	Self	Total
main.BoundingBoxCollisionVector	5.59 minutes	5.60 minutes
main.Ray.IntersectBVHLean_Texture	3.05 minutes	10.34 minutes
main.Ray.IntersectTriangleTextureGeneral	0.93 minutes	0.93 minutes
main.Vector.Normalize	0.46 minutes	0.46 minutes
internal/chacha8rand.block	0.26 minutes	0.26 minutes
main.Vector.Cross	0.22 minutes	0.22 minutes
main.Vector.Dot	0.22 minutes	0.22 minutes
runtime.rand	0.21 minutes	0.49 minutes
main.Vector.Sub	0.21 minutes	0.21 minutes
main.TraceRayV4AdvanceTexture	0.20 minutes	20.46 minutes
github.com/chewxy/math32.Cos	0.19 minutes	0.19 minutes
math/rand.(*Rand).Float32	0.17 minutes	0.17 minutes
math/rand.(*Rand).Float64	0.17 minutes	0.17 minutes
github.com/chewxy/math32.Sin	0.16 minutes	0.16 minutes
math/rand.globalRand	0.13 minutes	0.13 minutes
math/rand.Float32	0.11 minutes	1.15 minutes
main.SampleHemisphere	0.11 minutes	0.79 minutes
math/rand.(*runtimeSource).Int63	0.08 minutes	0.61 minutes
math/rand.(*Rand).Int63	0.07 minutes	0.68 minutes
main.RandomInUnitSphere	0.06 minutes	1.04 minutes
github.com/chewxy/math32.Pow	0.06 minutes	0.07 minutes
github.com/chewxy/math32.archExp	0.05 minutes	0.05 minutes
runtime.duffzero	0.05 minutes	0.05 minutes
github.com/chewxy/math32.archLog	0.05 minutes	0.05 minutes
internal/chacha8rand.(*State).Next	0.04 minutes	0.04 minutes
github.com/chewxy/math32.max	0.04 minutes	0.04 minutes
main.Vector.Mul	0.03 minutes	0.03 minutes
github.com/chewxy/math32.modf	0.03 minutes	0.03 minutes
main.DrawRaysBlockAdvanceV4Lin.func1	0.03 minutes	12.98 minutes
github.com/chewxy/math32.IsInf	0.03 minutes	0.03 minutes
main.Vector.Perturb	0.03 minutes	1.18 minutes
runtime.asyncPreempt	0.02 minutes	0.02 minutes
main.Vector.LengthSquared	0.02 minutes	0.02 minutes
internal/chacha8rand.(*State).Refill	0.02 minutes	0.28 minutes
main.Vector.Add	0.02 minutes	0.02 minutes

- V4Lin Profile



Location	Self	Total
main.BoundingBoxCollisionVector	5.44 minutes	5.44 minutes
main.Ray.IntersectBVHLean_Texture	2.94 minutes	10.05 minutes
main.Ray.IntersectTriangleTextureGeneral	0.92 minutes	0.92 minutes
main.Vector.Normalize	0.44 minutes	0.44 minutes
internal/chacha8rand.block	0.25 minutes	0.25 minutes
main.TraceRayV4AdvanceTexture	0.22 minutes	19.93 minutes
main.Vector.Cross	0.21 minutes	0.21 minutes
main.Vector.Dot	0.21 minutes	0.21 minutes
main.Vector.Sub	0.20 minutes	0.20 minutes
runtime.rand	0.19 minutes	0.46 minutes
github.com/chewxy/math32.Cos	0.19 minutes	0.19 minutes
math/rand.(*Rand).Float32	0.16 minutes	0.16 minutes
math/rand.(*Rand).Float64	0.16 minutes	0.16 minutes
github.com/chewxy/math32.Sin	0.15 minutes	0.15 minutes
math/rand.globalRand	0.12 minutes	0.12 minutes
main.SampleHemisphere	0.12 minutes	0.76 minutes
github.com/chewxy/math32.archLog	0.10 minutes	0.10 minutes
math/rand.Float32	0.10 minutes	1.10 minutes
math/rand.(*runtimeSource).Int63	0.08 minutes	0.59 minutes
math/rand.(*Rand).Int63	0.07 minutes	0.66 minutes
github.com/chewxy/math32.archExp	0.06 minutes	0.06 minutes
main.RandomInUnitSphere	0.05 minutes	0.99 minutes
runtime.duffzero	0.05 minutes	0.05 minutes
github.com/chewxy/math32.Pow	0.05 minutes	0.06 minutes
internal/chacha8rand.(*State).Next	0.04 minutes	0.04 minutes
main.Vector.Mul	0.03 minutes	0.03 minutes
github.com/chewxy/math32.max	0.03 minutes	0.03 minutes
main.DrawRaysBlockAdvanceV4Log.func1	0.03 minutes	12.60 minutes
main.ColorGradeLogarithmic	0.03 minutes	0.26 minutes
github.com/chewxy/math32.IsInf	0.03 minutes	0.03 minutes
main.Vector.Perturb	0.02 minutes	1.12 minutes
github.com/chewxy/math32.modf	0.02 minutes	0.02 minutes
internal/chacha8rand.(*State).Refill	0.02 minutes	0.27 minutes
runtime.asyncPreempt	0.02 minutes	0.02 minutes
math/rand.Float64	0.02 minutes	0.04 minutes

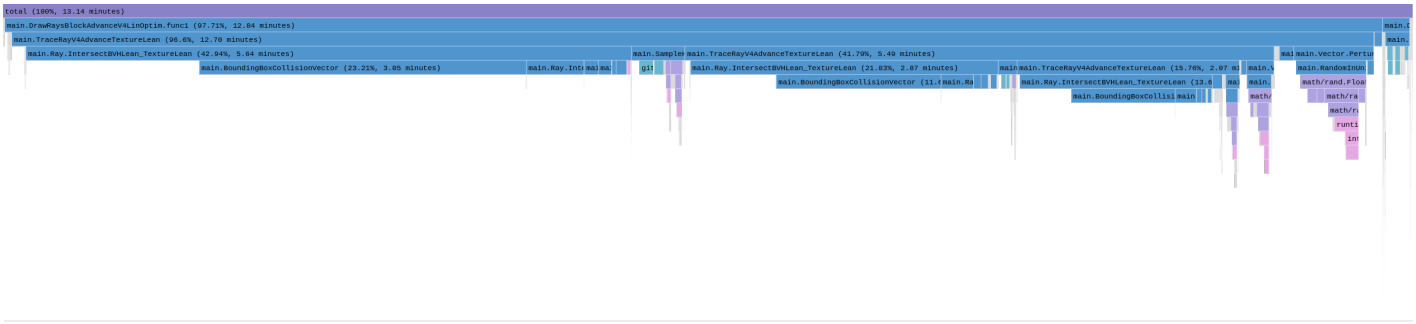
- V4Log Profile

TraceRayV4AdvanceTextureLean

Web Name : V4LinOptim / V4LogOptim

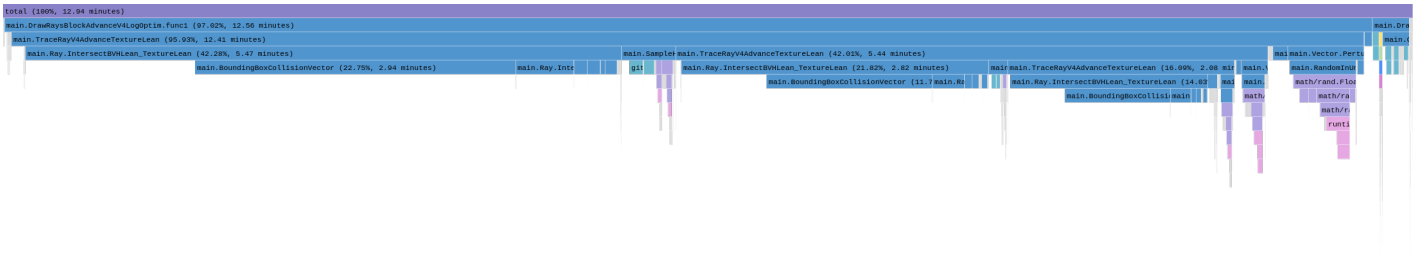
Optimalizovanejšia verzia, ktorá:

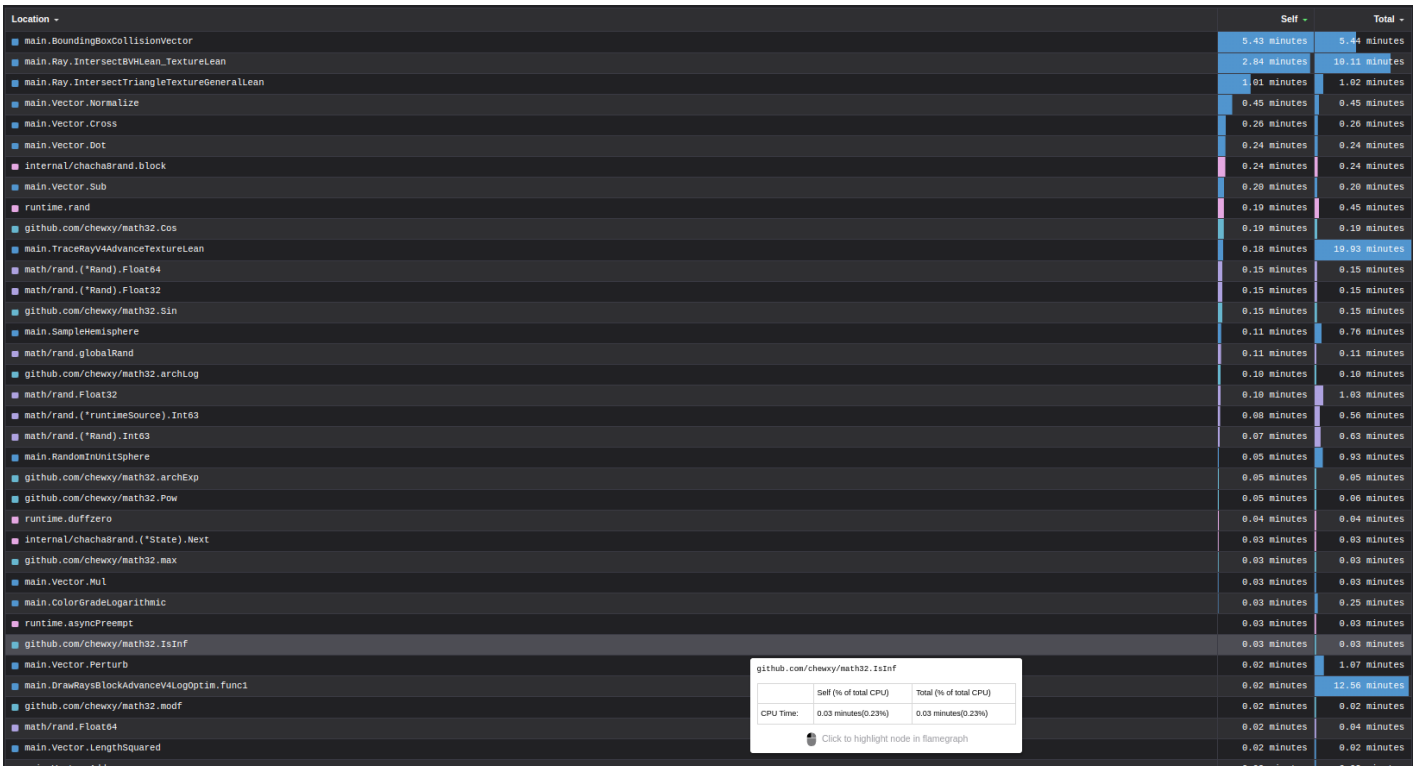
- Vracia len farebnú informáciu (bez normálových vektorov a vzdialenosti)
- Používa minimálny `IntersectBVHLean_TextureLean` intersekčný postup
- Znižuje pamäťovú spotrebu a minimalizuje štruktúrnu réžiu
- Zachováva všetky PBR výpočty, ale zjednodušuje návratovú štruktúru
- Špecificky navrhnutá pre čistý farebný rendering bez ďalších dát



Location	Self	Total
main.BoundingBoxCollisionVector	5.55 minutes	5.56 minutes
main.Ray.IntersectBVHLean_TextureLean	2.90 minutes	10.31 minutes
main.Ray.IntersectTriangleTextureGeneralLean	1.03 minutes	1.04 minutes
main.Vector.Normalize	0.45 minutes	0.45 minutes
main.Vector.Cross	0.27 minutes	0.27 minutes
main.Vector.Dot	0.25 minutes	0.25 minutes
internal/chacha8rand.block	0.24 minutes	0.24 minutes
runtime.rand	0.21 minutes	0.47 minutes
main.Vector.Sub	0.20 minutes	0.20 minutes
github.com/chewxy/math32.Cos	0.20 minutes	0.20 minutes
main.TraceRayV4AdvanceTextureLean	0.18 minutes	20.26 minutes
math/rand.(*Rand).Float64	0.17 minutes	0.17 minutes
math/rand.(*Rand).Float32	0.16 minutes	0.16 minutes
github.com/chewxy/math32.Sin	0.14 minutes	0.14 minutes
math/rand.globalRand	0.12 minutes	0.12 minutes
main.SampleHemisphere	0.12 minutes	0.77 minutes
math/rand.Float32	0.10 minutes	1.10 minutes
math/rand.(*runtimeSource).Int63	0.08 minutes	0.58 minutes
math/rand.(*Rand).Int63	0.07 minutes	0.65 minutes
main.RandomInUnitSphere	0.06 minutes	1.00 minutes
github.com/chewxy/math32.archExp	0.05 minutes	0.05 minutes
runtime.duffzero	0.05 minutes	0.05 minutes
github.com/chewxy/math32.archLog	0.05 minutes	0.05 minutes
github.com/chewxy/math32.Pow	0.04 minutes	0.06 minutes
github.com/chewxy/math32.max	0.04 minutes	0.04 minutes
internal/chacha8rand.(*State).Next	0.04 minutes	0.04 minutes
main.Vector.Mul	0.03 minutes	0.03 minutes
github.com/chewxy/math32.IsInf	0.03 minutes	0.03 minutes
main.Vector.Perturb	0.03 minutes	1.13 minutes
main.DrawRaysBlockAdvanceV4LinOptim.func1	0.02 minutes	12.84 minutes
github.com/chewxy/math32.modf	0.02 minutes	0.02 minutes
math/rand.Float64	0.02 minutes	0.04 minutes
main.Vector.LengthSquared	0.02 minutes	0.02 minutes
runtime.asyncPreempt	0.02 minutes	0.02 minutes
internal/chacha8rand.(*State).Refill	0.02 minutes	0.26 minutes

- V4LinOptim Profile





- V4LogOptim Profile

4.1 Kľúčové body evolúcie:

1. **Renderovací Model**: Od základného modelu (TraceRay) po plný PBR model (V3 a novšie)
2. **Návratové Dáta**: Od len farby po farbu+vzdialenosť+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

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

4.1.1 Systém Benchmarkovania a Výkonnostnej Analýzy

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

Zhrnutie Štatistík

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	35687.6933	29134.8735	669	761.8	81486.2	90386	38361.5
V2	40488.55	33250.3741	619	767	92560.7	105778	43920
V2Linear	44571.5783	54838.4226	1880	2195.9	95526.4	1102380	47353
V2LinearTexture	48300.4983	71178.14	1144	1329.2	98659.9	1072749	47459.5
V2Log	42980.4517	33730.8425	1957	2215.3	94611.8	109871	46791.5
V4Lin	45298.8867	76989.723	1734	2042	83079.6	838942	40508
V4LinOptim	45318.155	81374.7091	1731	1995.8	84782.4	1093434	40618
V4Log	43815.0217	79293.0855	1713	2057.7	84458	1080198	40202.5
V4LogOptim	43891.7667	75062.6395	1675	2010.9	84803.2	1094455	41179

- **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, separácii komponentov osvetlenia a implementácii fyzikálnej konzervácie energie.

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

  - **Priemerné časy:** Sa 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.

Technologické Rozdiely a Vývojová Trajektória

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 rozšírenia:** 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 réžia.
   - Verzie, ktoré vracajú dodatočné dáta (ako normály a vzdialenosti), 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 rýchlosti a konzistencie výkonu.

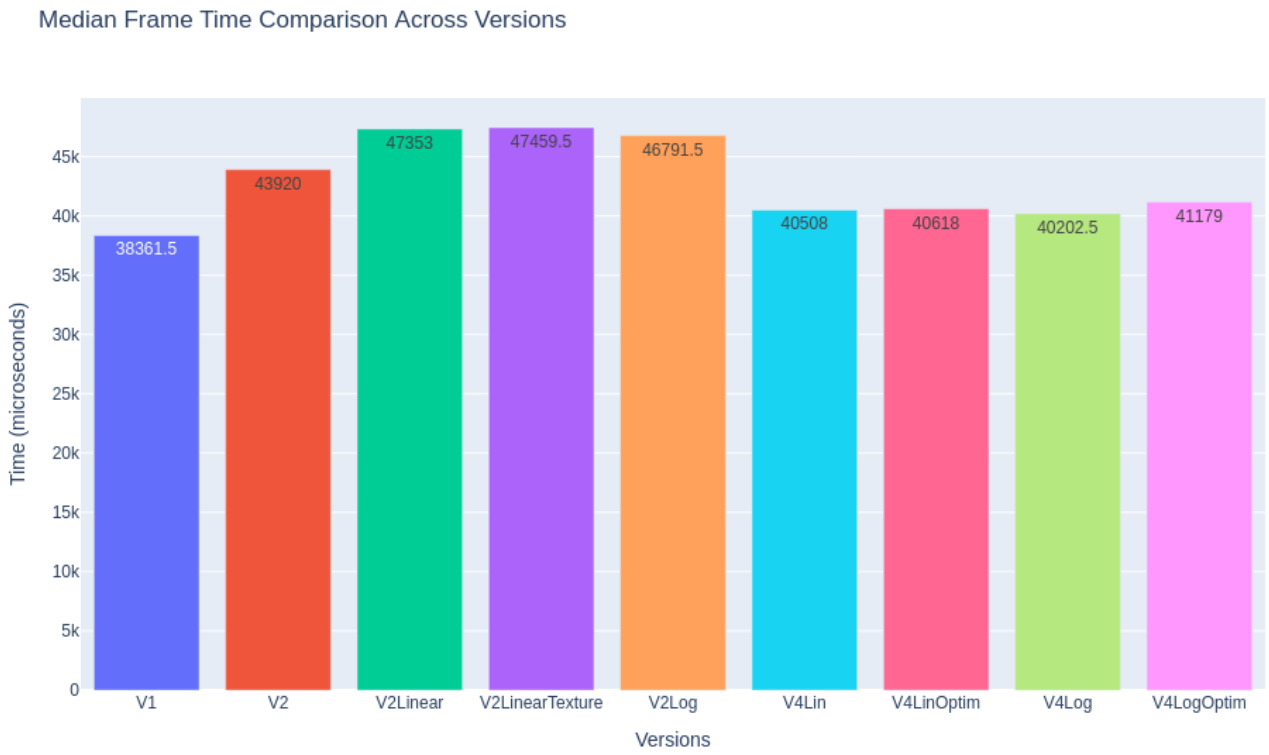
Záver

Vývojový trend týchto verzií ilustruje, že:

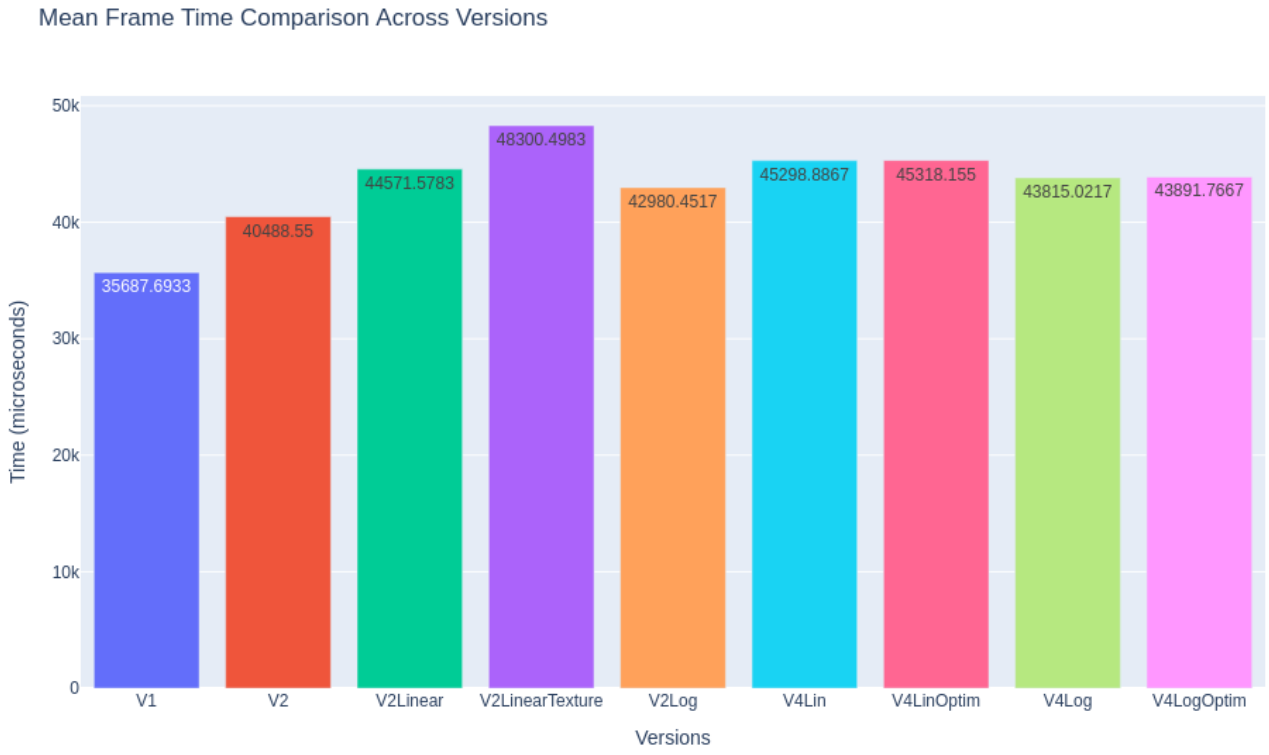
- **Základná verzia (V1)** je najrýchlejšia, ale neposkytuje tak vysokú vizuálnu kvalitu.
- **V2 a jeho rozšírenia** ponúkajú lepšie osvetlenie a simuláciu materiálových vlastností, pričom sa mierne zvyšuje čas spracovania.
- **Optimalizované V4 verzie** sa snažia minimalizovať režijné náklady pri zachovaní 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 realistickejšie výsledky, avšak vyžadujú vyššiu výpočtovú silu a môžu viesť k občasným špičkám v čase spracovania.

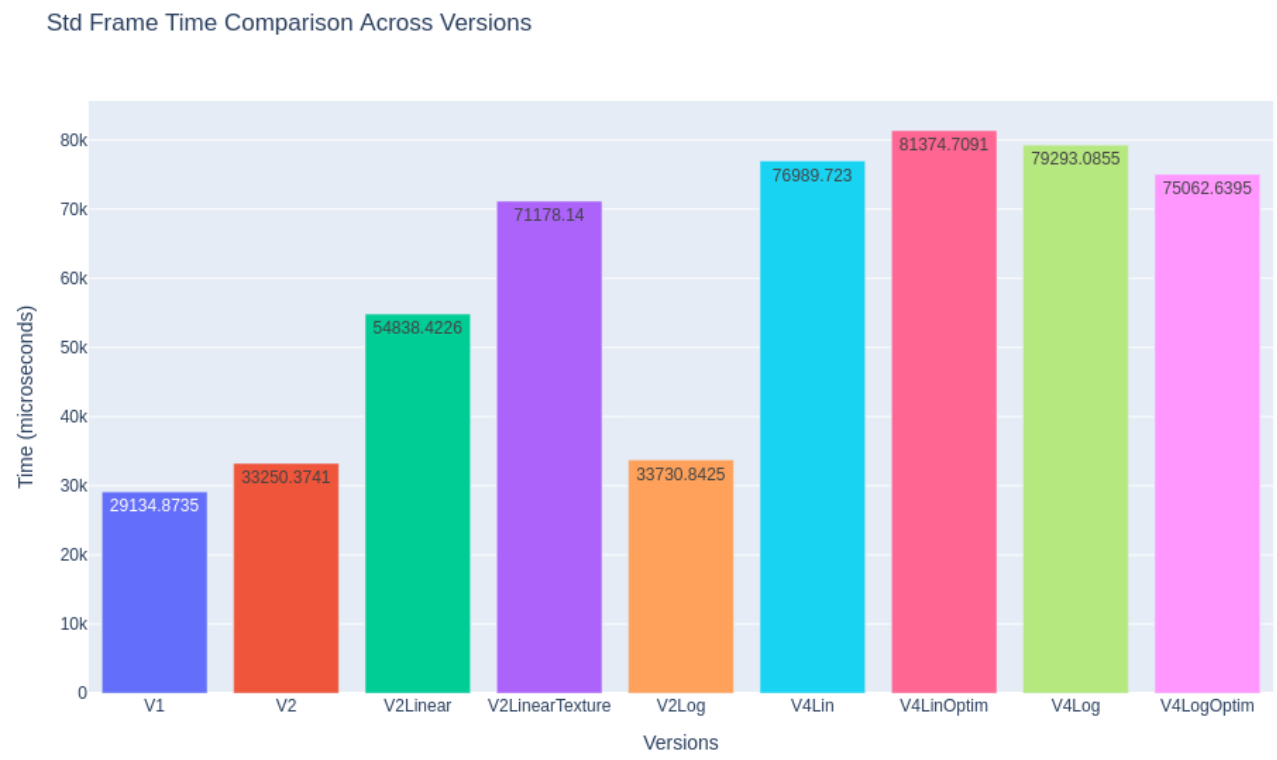
Median Graph



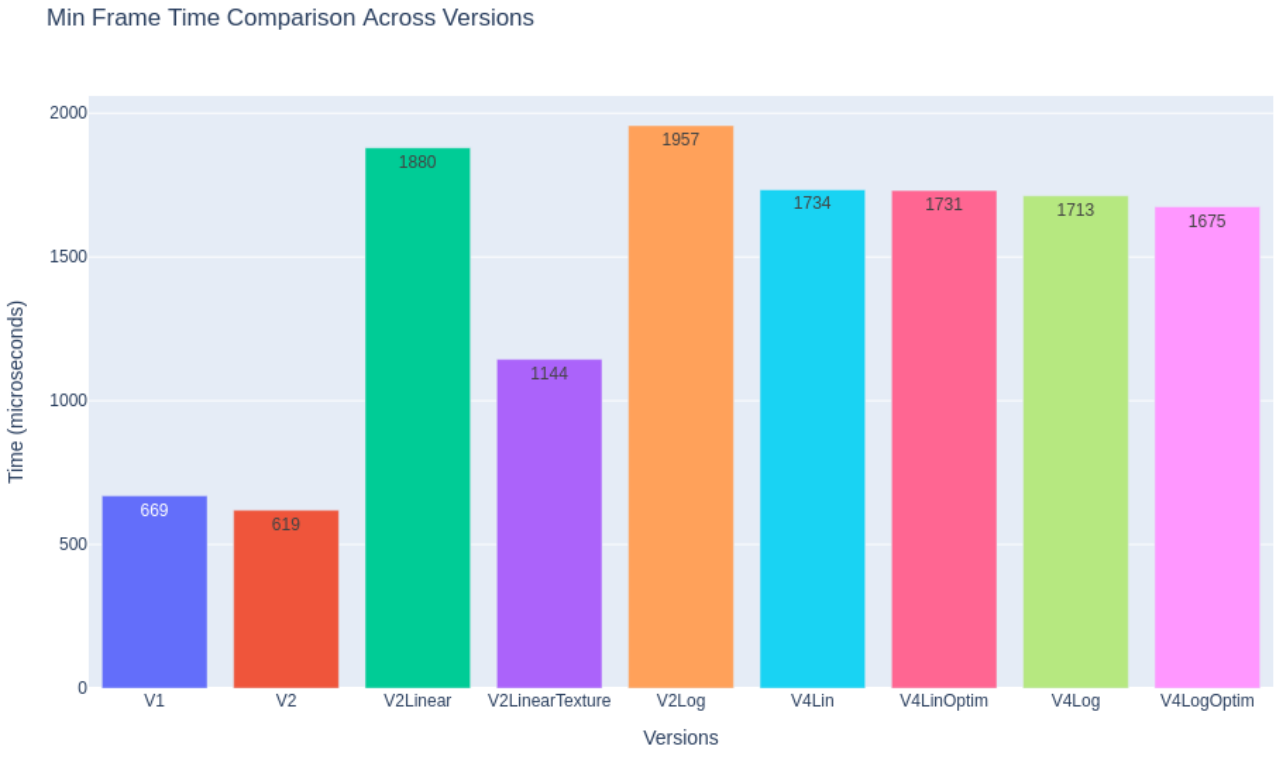
Mean Graph



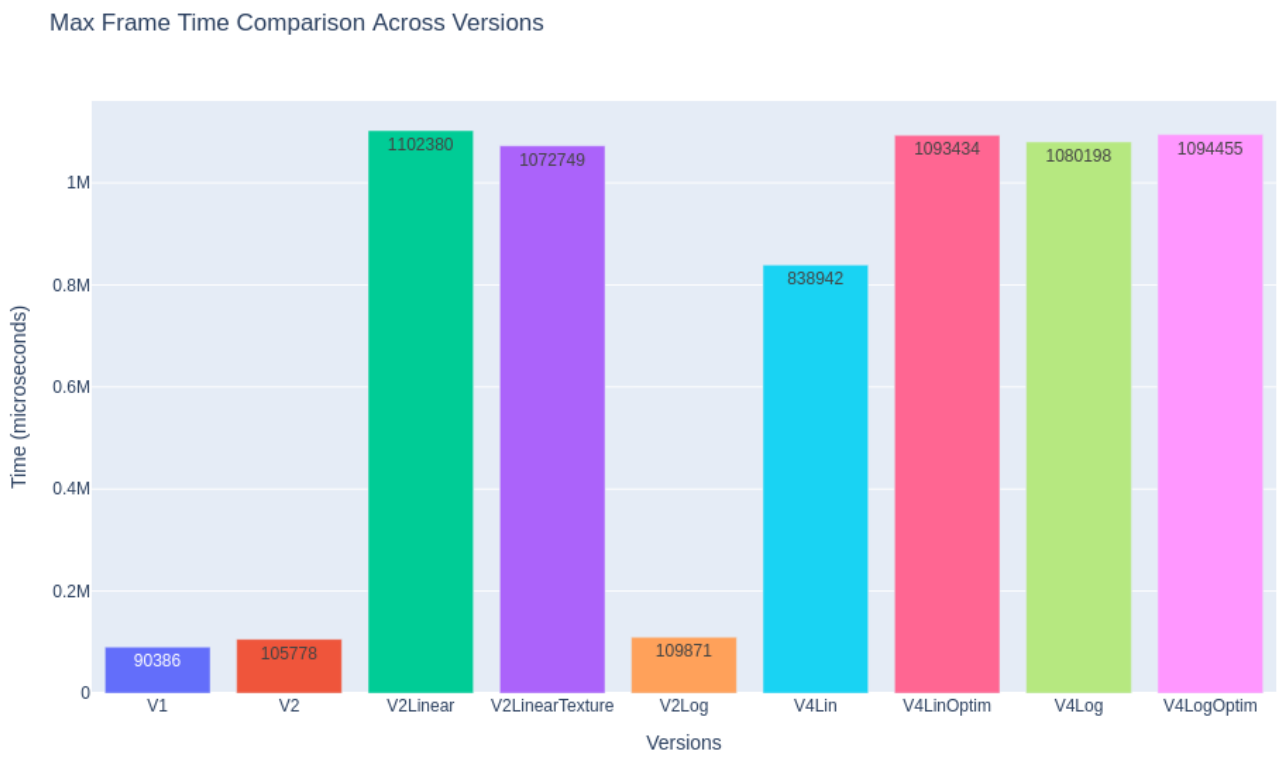
STD Graph



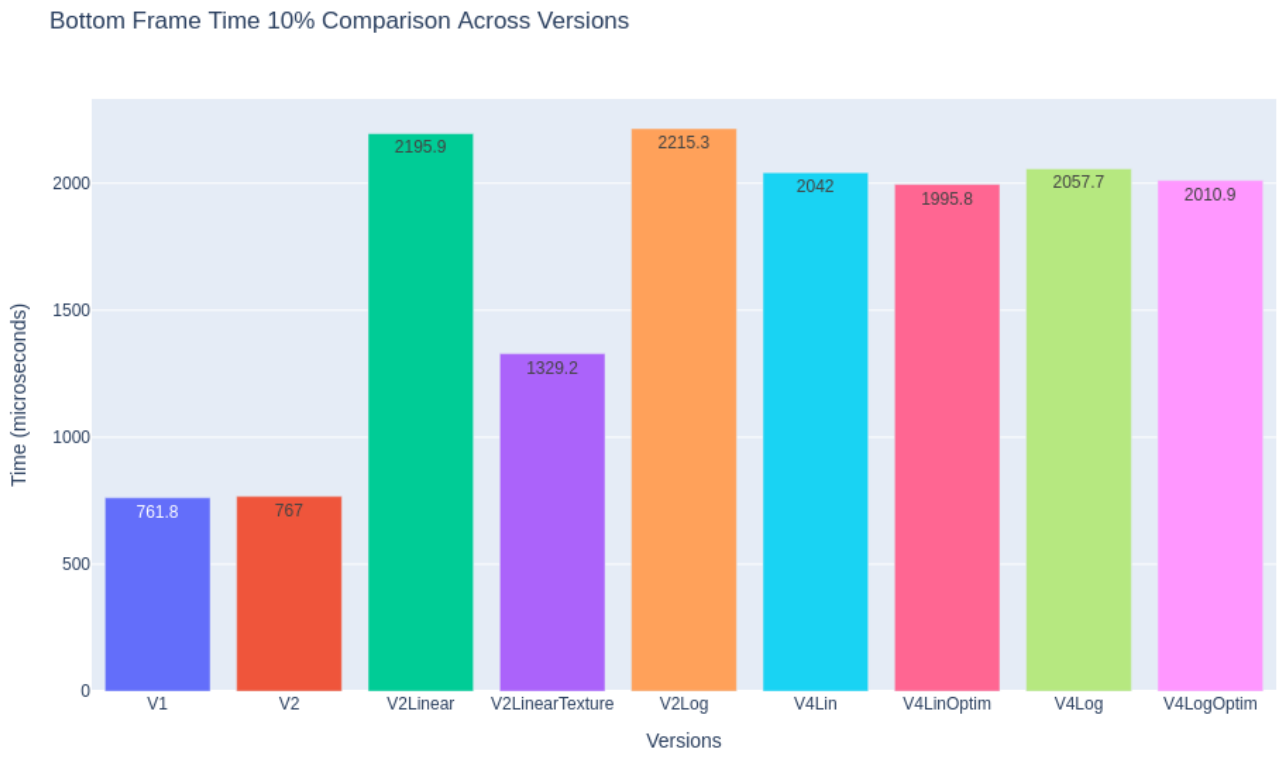
Min Frame Time



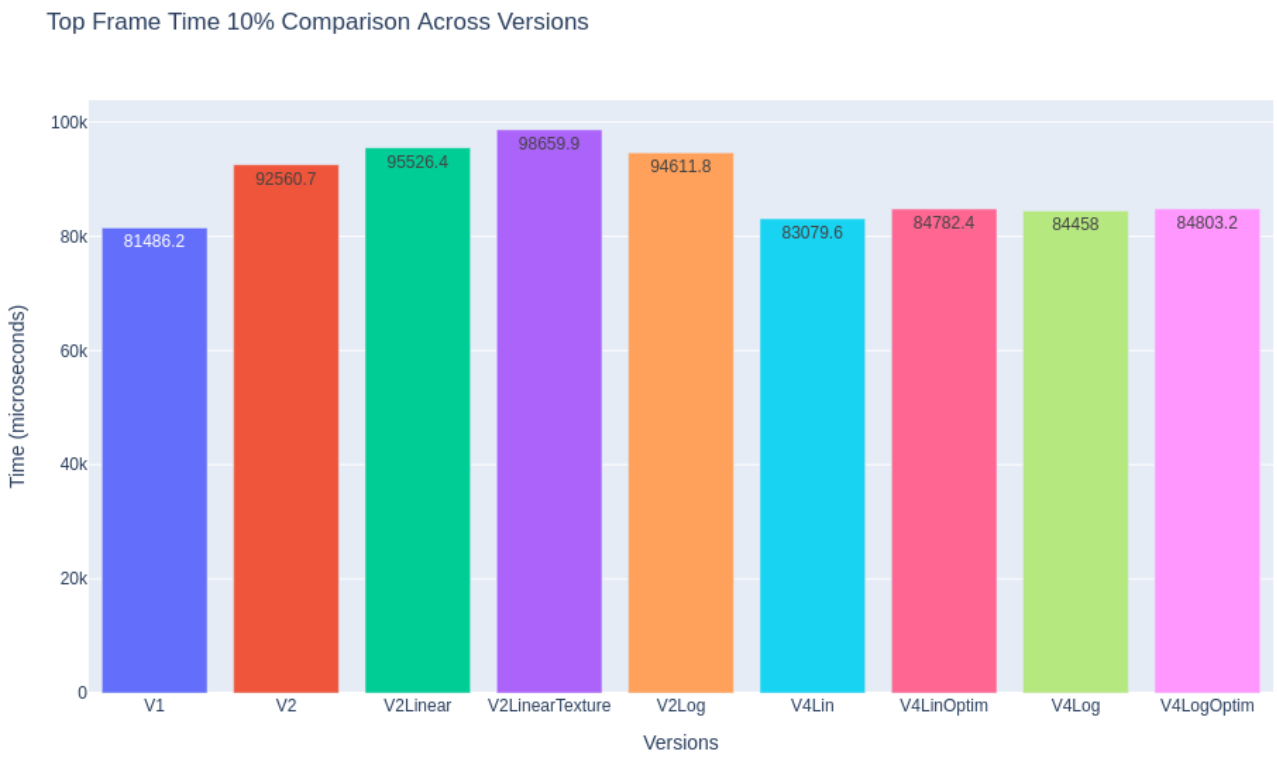
Max Frame Time



Bottom 10 % Frame Time



Top 10 % Frame Time



4.1.2 Úvod do Benchmarkingu

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

4.1.3 Testované Verzie Rendereru:

1. V1
2. V2
3. V2Log
4. V2Linear
5. V2LinearTexture
6. V4Log
7. V4Lin
8. V4LinOptim
9. V4LogOptim

4.1.4 Príprava Testovania

Testovacie Pozície Kamery:

- 3 rôzne priestorové pozície
- 10 sekund Interpolácia medzi poziciami
- Kamera sa počas testovania pohybuje medzi týmito pozíciami na základe interpolovaných nových pozícií v danom čase

Konfigurácia Parametrov:

- **Konštantné Parametre**:
  - Hlbka rekurzie: `3`
  - Rozptyl: `8`
  - Škálovací faktor: `2`
  - Gamma korekcia: `0.285`

4.1.5 Špecifiká Implementácie

Príprava Testovacích Dát

2. **Profiling Mechanizmus**
   - Generovanie CPU profilov pre každú verziu
   - Ukladanie profilov do `/profiles/`
   - Vytvorenie JSON súboru s nameranými časmi

Optimalizácie Pre Benchmark

- **Garbage Collection Vypnutý**:

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

3. Metriky Výkonu

4.1.6 Sledované Ukazovatele

1. **Priemerný Výpočtový Čas**
   - Pre každú verziu rendereru
   - Záznamy v mikrosekundách
   - Štatistická analýza výkonu
2. **Verzie Profilov**
   - Štandardné profily
   - Výkonnostné profily
   - Detailná analýza pre každú verziu

4.1.7. Výstup a Analýza

4.1.7.1 Výstupné Formáty

1. **CPU Profily**
   - Uložené vo formáte `.prof`
   - Pripravené pre analýzu nástrojmi ako `pprof`
2. **JSON Výkonnostné Dáta**
   - Uložené v `profiles/versionTimes.json`
   - Štruktúrovaný výstup pre ďalšiu analýzu

4.1.7.2 Postprocessing

- **Python Analýza**
   - Generovanie grafov
   - Štatistické vyhodnotenie
   - Porovnanie verzií

4.1.7 Kľúčové Výhody Systému

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

4.1.8. Záver

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

4.1.8.1 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(10*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)
    preformance := 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%s.prof", name)
        f, err := os.Create(profileFilename)
        if err != nil {
            log.Fatal(err)
        }

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

4.4 Vysledky testov

4.3 FresnelSchlick Funkcia

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

Účel

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

Parametre

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

Ako Funguje

1. Pri priamom pohľade na povrch (`cosTheta` blízko 1) je odraz blízky základnej odrazivosti materiálu (`F0`)
2. Pri pohľade z extrémneho uhla (`cosTheta` 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

Praktické Efekty

- Pre kovy (vodiče) je `F0` typicky vysoké (0.5-1.0), čo vedie k silným odrazom
- Pre nekovové materiály (dielektriká) je `F0` typicky nízke (0.02-0.05), s odrazmi viditeľnými hlavne pri extrémnych uhloch
- Vytvára efekt, kde sa povrchy ako voda, sklo alebo plast stávajú zrkadlovými pri pohľade z plochého uhla

4.3.1 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)
}
```

Účel

GGX distribučná funkcia modeluje **mikroploškovu distribúciu** povrchu, popisujúc, ako mikroskopické povrchové nepravidelnosti ovplyvňujú odraz svetla.

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)

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 umeleckým očakávaniam)
3. Distribúcia popisuje štatistickú pravdepodobnosť orientácie mikroploštiek v smere polovičného vektora
4. Pre hladké povrchy (nízka drsnosť) vytvorí úzky, intenzívny zrkadlový bod
5. Pre drsné povrchy (vysoká drsnosť) rozptýli odraz do väčšej plochy, vytvárajúc difúznejší vzhľad

Praktické Efekty

- Riadi veľkosť a intenzitu zrkadlových odleskov
- Hladké povrchy (nízka drsnosť) majú malé, jasné body
- Drsné povrchy (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.

5.0 Implementácia Voxel Renderingu

Voxel rendering spracováva každý voxel ako diskrétny, pevný prvok s definovanými hranicami. Implementácia využíva techniku ray-marchingu cez mriežku, 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 vzdialenosti 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 vzdialenosti
            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
}
```

5.0.1 Kľúčové Funkcie:

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

5.1 Implementácia Objemového Renderingu

Objemový rendering spracováva mriežku ako kontinuálne médium s premenlivými hustotami. Implementuje fyzikálne založené rozptyľovanie a absorpciu svetla cez participujúce 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 zaujatosť rozptylu
    )
```

```
    )

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

    var accumColor ColorFloat32
    transmittance := volumeMaterial.transmittance  // Počiatočná 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 rozptýleného svetla
        scattering := extinction * scatteringAlbedo * phaseFunction * 2.0

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

        // Akumulácia farby s príslušným fyzikálnym vážením
        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 finálnej farby, váženej aktuálnou priehľadnosťou
        accumColor = accumColor.Add(lightContribution.MulScalar(transmittance))

        // Optimalizácia predčasného ukončenia
        if transmittance < 0.001 {
            break
        }

        currentPos = currentPos.Add(stepSize)
    }

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

5.1.1 Kľúčové 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

5.2 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 priesečník lúča s ohraničujúcim boxom mriežky pred vykonaním detailného prechodu.
4. **Útlm Svetla Podľa Vzdialenosti**: Príspevok svetla je útlmený na základe vzdialenosti, poskytujúc realistický pokles bez náročných výpočtov.

5.3 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 dáta pre efekty dymu/hmly.
- **Úprava Materiálových Parametrov**: Hustota a priehľadnosť môžu byť nastavené pre rôzne vizuálne efekty.

5.4 Fyzikálne Modely Objemu

Aktuálna implementácia zahŕňa zjednodušený fyzikálny model založený na:

- **Beer-Lambertov Zákon**: Pre absorpciu svetla cez participujúce médiá
- **Henyey-Greensteinova Fázová Funkcia**: Pre anizotropný 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.

6.0 Implementácia Raymarchingu

Aktuálna implementácia raymarchingu je obmedzená na gule kvôli ich jednoduchej vzdialenostnej funkcii:

```
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
}
```

6.1.0 Aktuálny Stav

- Podporuje iba guľové primitívy
- Používa BVH pre akceleráciu
- Základná implementácia bez pokročilých funkcií

6.1.1 Plány Budúceho Vývoja

Rozšírenie Podpory Primitívov

Pre vylepšenie raymarchingových schopností plánujem implementovať ďalšie geometrické primitívy:

```
// 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/2, 0}
    return math32.Min(math32.Max(d.X, d.Y), 0) +
           Vector{math32.Max(d.X, 0), math32.Max(d.Y, 0), 0}.Length()
}
```

6.1.3 SDF Operácie

Pre umožnenie vytvárania komplexných objektov prostredníctvom operácií ako zjednotenie, priesečník a rozdiel:

```
// Zjednotenie dvoch SDF
func SdfUnion(d1, d2 float32) float32 {
    return math32.Min(d1, d2)
}

// Hladké zjednotenie s prelínaním
func SdfSmoothUnion(d1, d2, k float32) float32 {
    h := math32.Max(k-math32.Abs(d1-d2), 0.0)
    return math32.Min(d1, d2) - h*h*0.25/k
}

// Priesečník dvoch SDF
func SdfIntersection(d1, d2 float32) float32 {
    return math32.Max(d1, d2)
}

// Rozdiel SDF2 od SDF1
func SdfDifference(d1, d2 float32) float32 {
    return math32.Max(d1, -d2)
}
```

6.1.4 Implementačný Plán

1. **Rozšírenie Primitívov**
    - Implementácia základných primitívov (kocka, torus, valec)
    - Pridanie ovládacích parametrov v užívateľskom rozhraní pre každý typ primitívu
2. **SDF Operácie**
    - Implementácia Booleovských operácií (zjednotenie, priesečník, rozdiel)
    - Pridanie hladkého prelínania medzi tvarmi pre organické formy
3. **Optimalizácia Výkonu**

    - Rozšírenie BVH akceleračnej štruktúry pre všetky SDF primitívy
    - Implementácia priestorovej partície špecifickej pre raymarching

4. **Užívateľské Rozhranie**
    - Vytvorenie dedikovaného ovládacieho panelu raymarchingu
    - Pridanie vizuálnej spätnej väzby pre SDF operácie

5. **Pokročilé Funkcie**
    - Priestorová repetícia pre vytváranie vzorov
    - Deformácie založené na šume pre organické tvary
    - Priradenie materiálov pre SDF objekty

Tento rozšírený raymarchingový systém umožní vytváranie komplexných tvarov prostredníctvom konštruktívnej solid geometrie, čo používateľom umožní budovať zložité modely, ktoré by bolo ťažké dosiahnuť s tradičnou trojuholníkovou geometriou.

7.0 Podpora Post-Processing Shaderov

Úvod do Post-Processingu

Post-processing shadre predstavujú kľúčový nástroj pre vizuálne vylepšenie výstupného obrazu v ray-traceri, umožňujúci sofistikované úpravy renderovaného obrazu po jeho primárnom vygenerovaní.

Technologické Pozadie

7.0.1 Kage Shader Language

Pôvod: Vyvinutý súbežne s Ebiten 2D enginom

priklad syntaxu kage shadru

```
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)
}
```

Charakteristiky:

- Syntaxou inšpirovaná programovacím jazykom Go
- Zameraná na jednoduchosť a čitateľnosť
- Efektívna pre 2D a 3D grafické efekty

7.1 Podporované Post-Processing Efekty

7.1.0 Verzia V1: Základné Efekty

- Tint (farebný nádych)
- Contrast (kontrast)
- Bloom (svetelný efekt)

7.1.1 Verzia V2: Rozšírené Vizuálne Efekty

- Bloom V2: Vylepšená verzia svetelného efektu
- Sharpness: Zvýraznenie ostrosti obrazu
- Color Mapping: Limitácia počtu RGB hodnôt
- Chromatic Aberration: Farebná aberácia
- Edge Detection: Detekcia hrán pomocou Sobelovho filtra
- Lighten: Úprava RGB hodnôt s multivrstvovou podporou

7.1.2 Technické Charakteristiky

7.1.3 Shader Architektúra

- **Jazyk**: Kage Shader Language
- **Multipass Podpora**:
    - Umožňuje aplikáciu viacerých shaderov za sebou
    - Flexibilné reťazenie efektov
    - Postupné transformácie obrazu

7.1.4 Implementačné Detaily

- **Flexibilita**: Štruktúra pripravená na pridávanie nových shaderov
- **Výkonnosť**: Optimalizované pre rýchle spracovanie obrazu
- **Škálovateľnosť**: Jednoduchá rozšíriteľnosť efektov

7.1.5 Príklady Efektov

7.1.6 Color Mapping

- Redukcia farebnej hĺbky
- Kontrola presnosti farieb
- Umožňuje umělecké a štylizované vykresľovanie

7.1.7 Chromatic Aberration

- Simulácia optických nedokonalostí
- Pridáva vizuálnu dynamiku
- Efekt inšpirovaný optikou reálnych kamier

7.1.8 Edge Detection (Sobelov Filter)

- Zvýraznenie hrán v scéne
- Detekcia kontúr objektov
- Podpora pre analytické a umelecké vizualizácie

7.2 Výhody Implementácie

1. Vizuálna Flexibilita
2. Nízka Výpočtová Náročnosť
3. Jednoduché Rozšírenie
4. Umelecká Kontrola nad Obrazom

7.2.1 Budúci Vývoj

- Podpora komplexnejších efektov
- Rozšírenie kreatívnych možností post-processingu

7.3 Záver

Implementácia post-processing shaderov predstavuje sofistikovaný prístup k vizuálnemu vylepšeniu raytracerom generovaného obrazu, ponúkajúc bohatú škálu efektov s minimálnou výpočtovou réžiou.

8.0 Záver

Predložená maturitná práca predstavuje komplexný návrh a implementáciu 3D ray-tracingového engine-u, ktorý prekračuje tradičné hranice počítačovej grafiky. Projekt nie je iba technickým cvičením, ale ukazuje potenciál pre vytváranie sofistikovaných vizualizačných nástrojov s dôrazom na výkon, flexibilitu a užívateľskú rozšíriteľnosť.

8.1 Kľúčové prínosy práce

8.1.1 Technologická Inovácia

- Implementácia pokročilých ray-tracingových techník
- Podpora komplexných renderovacích algoritmov
- Flexibilný systém pre volumetrické a 3D zobrazovanie

8.1.2 Architektonické a Výkonnostné Riešenia

- Optimalizačné štruktúry ako BVH
- Efektívne využitie multiprocesingu
- Podpora štandardných 3D formátov
- Robustný benchmarkový systém pre kontinuálne meranie výkonu

8.1.3 Rozšírené Grafické Možnosti

- Pokročilý post-processing
- Podpora shaderových efektov
- 2D vrstvový systém pre následné úpravy
- Flexibilné nástroje pre manuálne a procedurálne úpravy obrazu

Projekt poskytuje nielen technické riešenie, ale aj platformu pre ďalší výskum a vývoj v oblasti počítačovej grafiky. Ukazuje, že moderné programovacie techniky a hlboké pochopenie grafických algoritmov môžu vyústiť do výkonného a adaptabilného grafického systému.

8.2 Perspektívy ďalšieho vývoja

- Integrácia pokročilých renderovacích techník
- Podpora real-time ray-tracingu
- Implementácia fyzikálne presnejších light transportných modelov
- Podpora komplexnejších animačných a dynamických scén

Implementovaný engine nie je len akademickým projektom, ale solidným základom pre budúci vývoj sofistikovaných grafických nástrojov. Demonstruje schopnosť navrhnúť komplexný systém, ktorý kombinuje výkonnosť, flexibilitu a inovatívny prístup k počítačovej grafike.

9.0 Zdroje

9.1 Online Knihy o Ray Tracingu

1. **Ray Tracing in One Weekend**
    - URL: https://raytracing.github.io/books/RayTracingInOneWeekend.html#overview
2. **Ray Tracing: The Next Week**
    - URL: https://raytracing.github.io/books/RayTracingTheNextWeek.html
3. **Ray Tracing: The Rest of Your Life**
    - URL: https://raytracing.github.io/books/RayTracingTheRestOfYourLife.html#cleaninguppdfmanagement/diffuseversusspecular

9.2 Technické Videá a Prezentácie

1. **Why you should avoid Linked List**
    - URL: https://www.youtube.com/watch?v=YQs6IC-vgmo
2. **Why is recursion bad?**
    - URL: https://www.youtube.com/watch?v=mMEmNX6aW_k
3. **How Big Budget AAA Games Render Bloom**
    - URL: https://www.youtube.com/watch?v=ml-5OGZC7vE
4. **Andrew Kelley Practical Data Oriented Design (DoD)**
    - URL: https://www.youtube.com/watch?v=IroPQ150F6c
5. **I redesigned my game**
    - URL: https://www.youtube.com/watch?v=PcMua73C_94