VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PROCEDURÁLNĚ GENEROVANÝ LENS FLARE EFEKT PRO BLENDER

PROCEDURALLY GENERATED LENS FLARE EFFECT FOR BLENDER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR VOLF**
AUTHOR

VEDOUCÍ PRÁCE **TOMÁŠ CHLUBNA, Ing.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Volf Petr**
Program: Informační technologie
Název: **Procedurálně generovaný lens flare efekt pro Blender**
Procedurally Generated Lens Flare Effect for Blender
Kategorie: Počítačová grafika

Zadání:

1. Naučte se základy práce s 3D modelovacím programem Blender (verze 2.8 a výše)
2. Seznamte se se skriptovacím Blender Python API
3. Nastudujte princip funkce objektivů u kamer a fotoaparátů
4. Navrhněte způsob simulace odlesků způsobujících lens flare artefakty a definujte uživatelské parametry výsledného add-onu
5. Add-on implementujte a demonstrujte jeho použití na různých typech scén
6. Zdokumentujte a zveřejněte výsledný add-on pro použití dalšími uživateli
7. Vytvořte video, reprezentující výsledky vaší práce

Literatura:

- Hullin, Matthias, et al. "Physically-based real-time lens flare rendering." *ACM SIGGRAPH 2011 papers*. 2011. 1-10.
- Blender Python API Official Documentation

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 4, experimenty vedoucí k bodu 5.

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

Vedoucí práce: **Chlubna Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Lens flare je obrazový efekt způsobený nechtěným odrazem světla v čočkovém systému fotoaparátu. Cílem této práce je simulovat tento efekt v aplikaci pro tvorbu 3D obsahu Blender. Generovaný efekt je potom vložitelný do výsledného obrázku. Projekt je navržen jako doplněk do aplikace. Doplňěk nabízí uživateli několik parametrů pro nastavení žádaného výsledku.

Abstract

Lens flare is an image artefact caused by unintended reflection of light in camera's lens system. The goal of this project is simulation of this phenomena in open source 3D content creation suite Blender. The generated effect is then composited into final image. The project is designed as an add-on. The addon allows user to customize several parameters to achieve the desired result.

Klíčová slova

Odlesk čočky, Blender, Python, OpenGL, clona, objektiv, disperze, shader

Keywords

Lens flare, Blender, Python, OpenGL, aperture, lens, dispersion, shader

Citace

VOLF, Petr. *Procedurálně generovaný lens flare efekt pro Blender*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Tomáš Chlubna, Ing.

Procedurálně generovaný lens flare efekt pro Blender

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Chlubny. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

. . . . . .
Petr Volf
10. května 2021

Poděkování

Chtěl bych poděkovat všem co mě během práce podporovali. Dále bych chtěl poděkovat panu Ing. Tomáši Chlubnovi za tipy na zlepšení doplňku a rychlou zpětnou vazbu při práci. Nakonec bych chtěl poděkovat všem co mi pomohli doplněk otestovat.

Obsah

Kapitola 1

Úvod

Jedním z mnoha cílů počítačové grafiky je docílit fotorealisticky vypadajícího výsledku. K tomu slouží mnoho různých realitu napodobujících metod, od simulace světla, po vytváření komplexních modelů různých materiálů. Tyto modely nabízí perfektní obraz reality. Někdy je ale potřeba simulovat i vady na realitě. Jednou z takových vad je lens flare, což je artefakt způsobený nechtěným odrazem světla ve snímací technologií kamer a fotoaparátů. Způsobuje vady na pořízeném vizuálním materiálu, jako jsou kruhy, prstence a jiné. Někdy jsou tyto vady ale v počítačové grafice žádané. Pokud by například v oblasti filmového průmyslu bylo potřeba udělat nějakou scénu celou počítačově generovanou, je vhodné takový efekt použít pro zachování konzistence mezi počítačově generovanými záběry a záběry pořízenými fyzickou kamerou.

Cílem práce je simulovat tento efekt v aplikaci Blender, která se věnuje vytváření 3D scén, modelů a generováním vizuálního obsahu z nich. Výsledkem práce je doplněk, který je možné do Blenderu nainstalovat. Blender obsahuje vestavěnou podporu pro podobné efekty, ale jejich funkcionalita je značně omezená a nejsou příliš dobře nastavitelné uživatelem. Doplněk je do velké míry nastavitelný uživatelem.

Pro toto téma jsem se rozhodl, protože v Blenderu mám zkušenosti s vytvářením scén, 3D modelů a jejich vykreslením. Nikdy jsem si ale nezkoušel práci s programovacím rozhraním Blenderu nebo vytvořením dodatečné funkcionality ve formě doplňku do jakékoliv aplikace. Obojí by mohly být užitečné zkušenosti do budoucnosti.

V kapitole 2 je detailnější popis lens flare efektu a jeho prvků, popis Blenderu, jeho uživatelské a aplikační rozhraní. V kapitole 3 je popsán návrh řešení. Kapitola 4 se zabývá detailním popisem implementace vykreslování efektu a uživatelského rozhraní. V kapitole 5 je popis testování doplňku a metodika testování. V příloze A jsou snímky s efektem vykreslené pomocí doplňku.

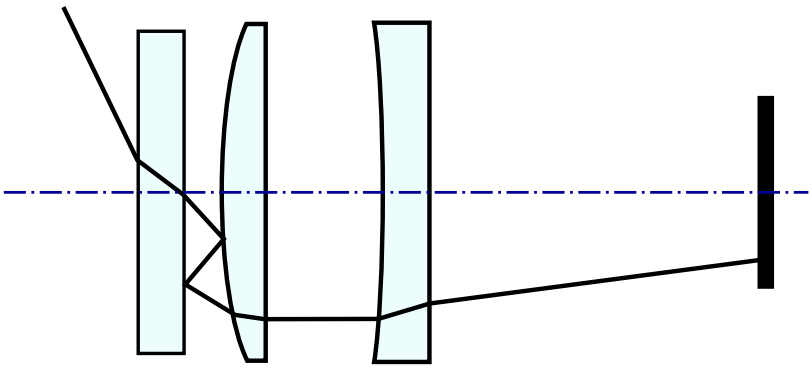
Kapitola 2

Teorie

Zde je popsán efekt lens flare a jeho prvky. Dále je uveden popis aplikace *Blender* a jeho uživatelského a aplikačního rozhraní. Nakonec je zmíněna knihovna *OpenGL*.

2.1 Lens flare

Lens flare (česky odlesk čočky) je obrazový efekt způsobený nechtěným odrazem světla mezi čočkami fotoaparátu. Hlavním důvodem pro tento jev je nedokonalý materiál použitý na výrobu čoček. Místo toho, aby došlo k lomu světla v čočce, se světlo odrazí. Pokud se takhle světlo odrazí vícekrát, může se pak stát že dorazí na snímač a způsobí daný efekt. Tento efekt způsobuje nedokonalosti na výsledném obrazu, jako jsou čáry, kruhy, zamlžení a paprsky. V oblasti filmu, fotografie a počítačových her jsou tyto efekty někdy žádané, pro vytvoření žádaného dojmu.[11, 10] Mohou tak např. indikovat silné zdroje světla. Na obrázku 2.1 je znázorněno, jak může takový odraz v optickém systému probíhat.



Obrázek 2.1: Ukázka možné cesty světla tvořící odlesk. Světlo přichází z levé strany do čočkové soustavy, kde se láme a odráží. Nakonec dopadá na snímač vpravo.

Silné zdroje světla tento efekt způsobují nejvíce. Je také možné že efekt je způsoben silným zdrojem světla mimo snímek, např. sluncem. Tyto odlesky lze eliminovat jednoduše pomocí sluneční clony. [5] Na obrázku 2.2 je ukázka sluneční clony.

Obrázek 2.2: Sluneční clona. Výřezy ve cloně jsou z důvodu zorného pole fotoaparátu.[1]

Výsledek lens flare efektu je odlišný pro každou soustavu čoček. Závisí také na pozici světla na snímku. Nedokonalosti v čočkách mohou způsobit že světlo o různé vlnové délce se pohlcuje pod jiným úhlem. [7] Tento fenomén se nazývá *disperze*. Disperze na snímku způsobuje duhové artefakty. To je pak jedna z příčin různobarevných odlesků. Klasickým příkladem disperze může být skleněný hranol rozkládající světlo na barevné spektrum.

Některé čočky mají povrchovou úpravu, která snižuje míru odrazu světla a tím snižuje sílu lens flare efektu pro určité vlnové délky světla. Také díky tomu mohou být některé artefakty zabarvené. Pokud se světlo odrazí vícekrát, vznikne z toho více artefaktů. Větší počet odrazů ale způsobí menší intenzitu artefaktu. [6] Objektivy určené pro velkou určenou přiblížení jsou kvůli velkému počtu čoček na tyto efekty velmi náchylné.

Anamorfní čočky jsou používané při snímání širokoúhlého záběru na film, který není pro širokoúhlý formát dělaný. Tyto čočky způsobují že odlesky jsou roztáhnuté po horizontální ose. [4] Na obrázku 2.3 je ukázka anamorfního lens flare efektu.

Obrázek 2.3: Ukázka anamorfního lens flare efektu. Snímek efektu je z filmu *Star Trek*.[2]

[1] Obrázek převzat z https://unsplash.com/photos/OFh9fSIpQbA

[2] Obrázek převzat z https://www.businessinsider.com/why-star-trek-has-so-much-lens-flare-2015-11

2.1.1 Prvky lens flare efektu

Na místě, kde je na snímku zobrazený zdroj světla, je nejvíce viditelná záře okolo zdroje světla. Na některých soustavách čoček může být přítomných několik paprsků zářících směrem od zdroje světla. Tento efekt je způsoben nedokonalostí čoček, nebo nízkou kvalitou optického systému.

Dalším prvkem lens flare efektu je fenomén označovaný jako *ghosts* (duchové). [8] Projevuje se jako mnoho tvarů, nejčastěji kruhů, rozmístěných v linii mezi zdrojem a středem snímku. Tvar útvarů závisí na tvaru závěrky. Na obrázku 2.4 je popsaná struktura komplexního efektu. Tento efekt je právě důsledkem odrazu světla mezi čočkami. *Ghosts* jsou v optické soustavě dále zkreslovány kvůli tvaru čoček. Může také nastat, že *ghosts* budou mít silnější okraj.

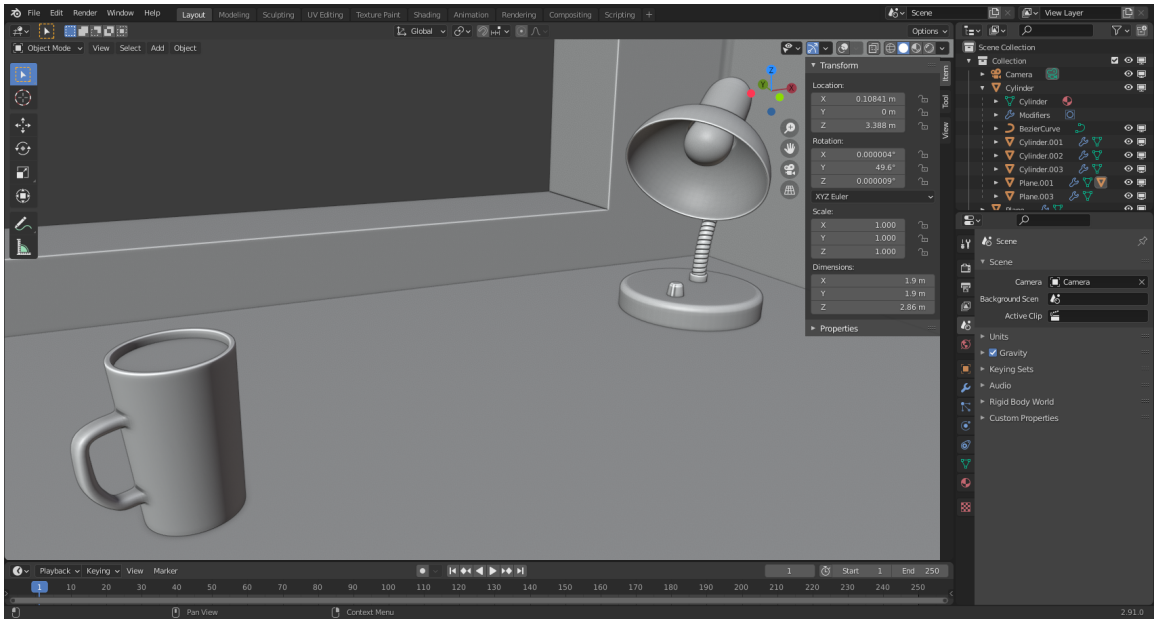
Ve optických soupravách může nastat, že *ghost* bude částečně useknutý, což může být způsobeno tím, že ne všechno světlo se odrazí tak aby se dostalo na senzor. [9] Míra takového zkreslení je zpravidla větší poblíž okrajů snímku.

Obrázek 2.4: Struktura složitého lens flare efektu. **A** - Zdroj světla s mírnou září a silnými paprsky, **B** - *ghosts*, **C** - zvýrazněný tvar jednoho artefaktu, je z něj poznat tvar závěrky, **D** - artefakty blíže k okrajům jsou zkreslené jak tvarově, tak i barevně.
Obrázek převzat z [6].

2.2 Blender

Blender je open-source aplikace pro tvorbu a vykreslování 3D scén. Podporuje širokou škálu činností, od vytváření modelů, textur (viz. sekce 2.3) a animaci, po střih videa, vytváření simulačních efektů a dalších věcí. Blender nabízí 2 hlavní renderery (vykreslovací back-end), EEVEE (real-time renderer používající rozhraní OpenGL pro vytváření snímků) a Cycles (path-tracing renderer zaměřený na fyzikálně správné výsledky).

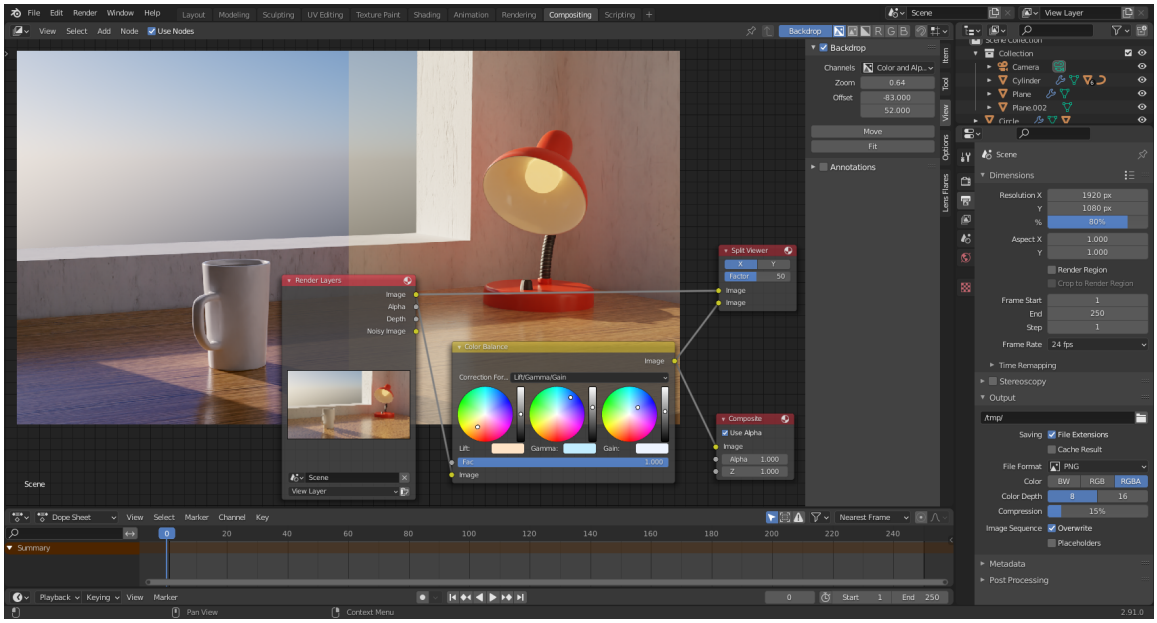
Rozhraní Blenderu je tvořeno z podoken, což jsou samostatné editory. Každý editor má vlastní funkcionalitu a slouží ke specifickému účelu (např. editor *3D viewport* slouží k manipulaci s 3D objekty ve scéně, editor *Properties* slouží k editaci vlastností objektů, nebo i celé scény, atd.). Některé editory mají vizuální rozhraní, tzv. node editor, který pracuje na principu uzlů a propojení mezi nimi. Blender uživateli umožňuje rychle se pohybovat mezi různými přednastavenými rozloženími podoken pro různé činnosti (např. *Layout* pro rozmístění objektů ve scéně, *Compositing* pro výsledné úpravy, atd.). Na obrázku 2.5 je vidět typické rozhraní Blenderu při tvorbě scény.



Obrázek 2.5: Ukázka rozhraní Blenderu verze 2.91 s jednoduchou scénou

Některé editory pracují v několika odlišných módech. Např. editor *3D Viewport* má jako výchozí mód *Object Mode*, ve kterém je možné hýbat s objekty ve scéně. Pokud je ale nutné upravit geometrii objektu, je potřeba se přepnout do módu *Edit Mode*. *3D Viewport* nabízí dále módy na malování vertexů, scuplting, a další.

Blender nabízí možnost výsledné snímky dále upravit. Lze vyvážit barvy, nebo přidat některé efekty. Pro tyto úpravy slouží editor *Compositor*. Ten umožňuje upravovat výsledný snímek pomocí uzlů. V uzlových editorech je možné vytvářet skupiny uzlů a ty pak znovu použít mezi různými soubory. Tyto skupiny je však možné tvořit jen z existujících uzlů. Lens flare efekty se zpravidla v počítačové grafice přidávají do snímku dodatečně, takže kompozitor je pro to ideální místo. Rozhraní kompozitoru je na obrázku 2.6.



Obrázek 2.6: Ukázka rozhraní editoru *Compositor* s příkladem efektu vyvážení barev

2.2.1 Blender Python API

Blender nabízí uživatelům kromě svých vestavěných nástrojů možnost vytvářet novou funkcionalitu přes své aplikační rozhraní. [1] Toto rozhraní je postaveno na programovacím jazyce Python ve verzi 3.x. Python je interpretovaný programovací jazyk. Je dynamicky typovaný a podporuje mnoho programovacích paradigmat. Přes toto rozhraní je možné modifikovat objekty ve scéně, měnit jejich vlastnosti, ovládat uživatelské rozhraní, nebo vytvářet vlastní funkcionalitu. Blender obsahuje integrovaný interpretr jazyka Python, takže lze spouštět skripty pracující s rozhraním přímo v aplikaci skrz integrovanou konzoli, nebo použít editor *Text Edtor*.

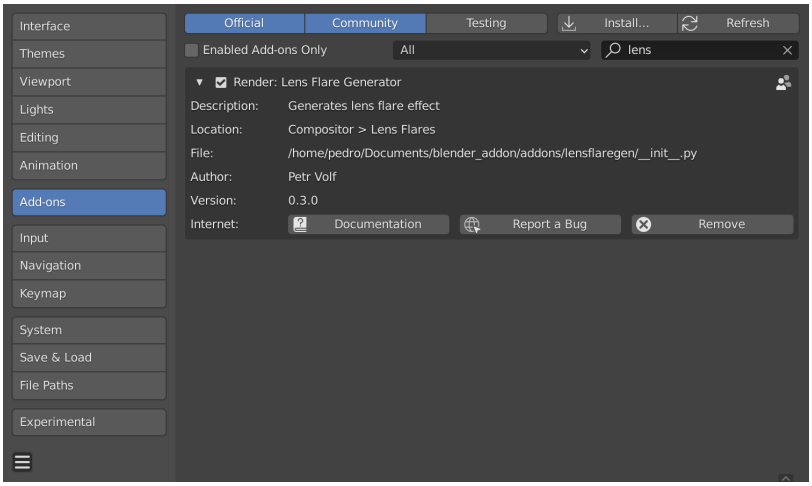
Interakce API a uzlových editorů spočívá v automatizování akcí, co by jinak dělal uživatel. Je možné vytvořit nové uzly daných typů, měnit jejich parametry, vytvářet propojení a slučovat uzly do skupin.

S API se pracuje hlavně pomocí modulu *bpy* a jeho podmodulů, přes které je možné volat akce, nebo přistupovat k datům. Mezi tyto moduly patří např.:

- *bpy.data*, pro přístup ke všem objektům a jejich datům, např. *bpy.data.objects["Cube"]*
- *bpy.context*, pro přístup k objektům co má uživatel právě vybrané, např. *bpy.context.object*
- *bpy.ops*, pro přístup k operátorům a jejich volání, např. akce jako *Flip Normals*
- *bpy.props*, pro rozšiřováních existujících typů novými vlastnostmi, nebo vytvářeních nových datových skupin

Aby bylo možné použít třídy vytvořené ve skriptu, je nutné je v API registrovat, potom případně odregistrovat. Díky tomu je možné doplňky zapínat a vypínat. Na obrázku 2.7 je znázorněné rozhraní nastavení doplňků.

Jednotlivé akce v *Blenderu* jsou reprezentovány jako tzv. operátory. Operátory jsou implementovány jako třídy dědící z *bpy.types.Operator*. Tyto operátory mohou konat svou



Obrázek 2.7: Rozhraní nastavení Blenderu s otevřeným menu na správu doplňků

funkci na základě parametrů, nebo např. podle aktuálně vybraného objektu. Operátory mohou být volány přes kód doplňku, v konzoli nebo kliknutím na tlačítko spárované s operátorem. Doplňky mohou definovat vlastní operátory. Operátory mohou mít jíné chování na základě zdroje volání. Například operátor na vykreslení snímku (*bpy.ops.render.render*) při volání z uživatelského rozhraní otevře nové okno a tam interaktivně zobrazuje postup vykreslování. Pokud je ale volaný z kódu, tak po dobu vykreslování zamrzne uživatelské rozhraní a okno s výsledkem se neotevře. Třídy implementující operátory mohou specifikovat název operátoru, který je rozdílný od jména třídy. Je tak možné uživatelsky vytvořené operátory seskupovat s vestavěnými operátory.

Uživatelské rozhraní se přes API vytváří pomocí panelů. Je potřeba vytvořit novou třídu (potomek třídy *bpy.types.Panel*) a na ní metodu *draw*. Aby Blender věděl, kam panel vložit, je nutné ve třídě panelu specifikovat několik proměnných jako *bl_label*, *bl_space_type* atd. V metodě *draw* se pak pracuje s položkou *layout*. Pomocí *layoutu* je možné vytvářet tlačítka na aktivaci operátorů, slidery pro nastavení různých hodnot a nebo pomocné prvky na řádkování, popisky, atd. Na obrázku 2.8 je vidět ukázka rozhraní a kód co je tvoří.

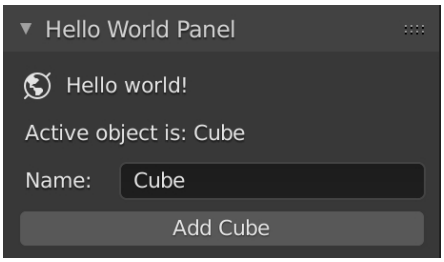
```
def draw(self, context):
    layout = self.layout

    obj = context.object

    row = layout.row()
    row.label(text="Hello world!", icon='WORLD_DATA')

    row = layout.row()
    row.label(text="Active object is: " + obj.name)
    row = layout.row()
    row.prop(obj, "name")

    row = layout.row()
    row.operator("mesh.primitive_cube_add")
```



Obrázek 2.8: Ukázka uživatelského rozhraní a kód kterým je vytvořeno. Kód převzat z [1].

Doplňky mohou i přidávat vlastní data. *Blender* umožňuje rozšířit některé datové bloky (např. objekt, scénu, geometrii, materiál, obrázek) o vlastní hodnoty. Definice takovýchto nových hodnot se provádí pomocí typů z modulu *bpy.props*. Je takhle možné přidávat vlastní struktury i kolekce. Všechny vlastnosti jsou silně typované. Lze specifikovat výchozí hodnoty, minima a maxima, popisek do rozhraní a další parametry. Několik doplňků takto může sdílet data mezi sebou.

2.3 OpenGL

OpenGL je multiplatfomní knihovna pro programování grafických aplikací. Umožňuje použít grafické akcelerátory pro vykreslování 2D i 3D grafiky. Lze jej použít z různých programovacích jazyků. Rozhraní knihovny je definované jako sada funkcí, které pak uživatel knihovny volá ve svém programu.

Blender pro své uživatelské rozhraní používá knihovnu *OpenGL* a pomocí API pak umožňuje doplňkům vlastní použití knihovny. Většina *OpenGL* funkcí je dostupná z API modulu *bgl*. Protože je *Blender* v procesu přechodu na modernější grafickou knihovnu [3], některé části knihovny *OpenGL* jsou ve třídách obalující určitou funkcionalitu. Tyto třídy jsou dostupné z modulů *gpu* a *gpu_extras*.

Proces vykreslení zpravidla následuje tyto kroky:

- procesor nahraje na grafickou kartu geometrii pro vykreslení
- pro každý bod geometrie se vypočítá transformovaná pozice do relativních koordinátů okna
- grafická karta provede rasterizaci a vygeneruje fragmenty
- pro každý fragment se vypočítá barva, zvoleným způsobem se zamíchá s již existujícím pixelem a zapíše se do paměti

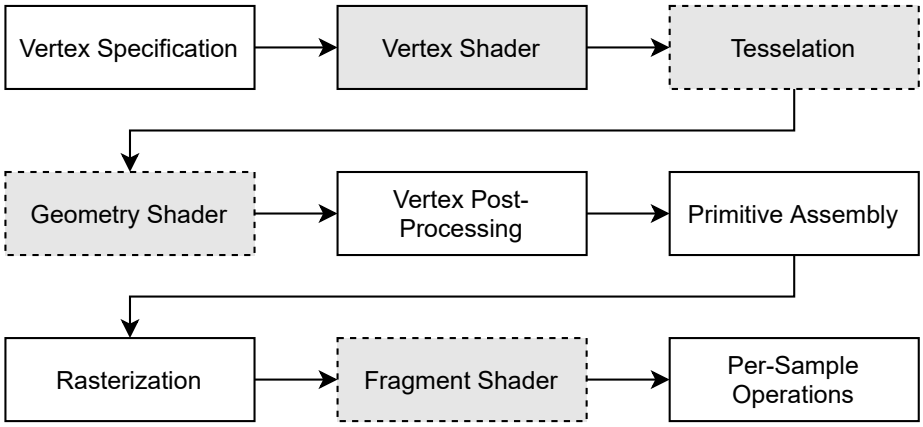
Sekvenci kroků pro vykreslení objektu se říká *rendering pipeline*. Na obrázku 2.9 je diagram kompletní *pipeline*.

Geometrie se na grafickou kartu nahrává jako pole čísel s plovoucí řádovou čárkou. Je na uživateli aby správně nastavil adresaci takových dat. Takto si uživatel knihovny může vybrat jestli chce vykreslit body, čáry, nebo trojúhelníkové plošky. Každý bod pak může kromě pozice obsahovat i vlastnosti jako barvu, atd.

Obrázek je pro použití v *OpenGL* nutné předat do *textury*. *Textura* je objekt obsahující jeden nebo více obrázků stejného formátu. Specifikuje jak bude obrázek použitý, jak se bude chovat mapování koordinátů na plochu obrázku, nebo jak se bude chovat při změně velikosti. *OpenGL* umožňuje použít více textur na během jednoho příkazu na vykreslení. Každá textura musí být pro použití přiřazená do texturové jednotky. Počet texturovacích jednotek závisí na hardware, na kterém program běží.

Místo v paměti grafické karty kam *OpenGL* vykresluje se nazývá *framebuffer*. Formát takového místa může být různý podle potřeb programu. *Framebuffer* může být tvořen i z více textur, např. může mít dodatečnou texturu pro zápis hloubky ve 3D scéně. *OpenGL* umožňuje také vykreslovat do textury a následně tuto texturu použít jako zdroj pro další vykreslování.

[3] Vulkan, viz. https://code.blender.org/2021/04/blender-2021-roadmap/



Obrázek 2.9: *OpenGL rendering pipeline*. Šipky naznačují postup kroků. Kroky s čárkovaným okrajem označují nepovinné části. Podbarvené kroky jsou programovatelné pomocí *shaderů*, viz. sekce 2.3.1. Kroky převzány z [3]

2.3.1 Shader

Na grafické kartě při použití knihovny běží uživatelské programy, tzv. *shadery*. Pomocí nich se řídí samotné vykreslování geometrie. V moderních verzích *OpenGL* nebo i jiných API existují různé druhy *shaderů* pro ovládání různých kroků vykreslování. Tyto shadery pak např. vypočítají pozici geometrie po transformaci, barvu pixelu, nebo mohou generovat novou geometrii. Mezi hlavní *shadery* patří *vertex shader* a *fragment shader*.

Vertex shader je spouštěn pro každý vertex a jeho výsledkem je nová pozice vertexu na obrazovce. Může mít i více výstupů, které jsou pak dál v jiných *shaderech* využity. Je tak třeba možné předat barvu vertexu, koordináty textury a další parametry.

Fragment shader je spouštěn pro každý fragment. *Fragment* je sbírka hodnot vytvořená během rasterizace. Zpravidla jde o soubor hodnot pro jeden pixel, ale může existovat i více fragmentů pro jeden fyzický pixel. Během běhu *fragment shaderu* se z fragmentu vypočítá barva, která se zapíše do *framebufferu*. Je možné psát do více *framebufferů* zároveň, nebo fragment zahodit. Fragmenty se během zápisu do *framebufferu* promíchávají podle způsobu určeným uživatelem knihovny. Díky tomu je možné vykreslit i poloprůhledné objekty.

Pro použití v *OpenGL* je potřeba *shader* zkompilovat do programu. Tyto programy mohou mít i více shaderů, viz. obrázek 2.9. Tyto programy pak běží paralelně na grafické kartě.

2.3.2 OpenGL Shading Language (GLSL)

V *OpenGL* se na psaní *shaderů* používá jazyk *GLSL*. Tento jazyk je založen na jazyce *C*. Na obrázku 2.10 je krátká ukázka *GLSL* kódu. *Shadery* mají zpravidla několik vstupů a výstupů. Parametry označeny klíčovým slovem `in` jsou vstupy shaderu, parametry označené `out` jsou výstupy. Parametry označené klíčovým slovem *uniform* jsou stejné pro všechny volání daného *shaderu*. *Uniformy* se nastavují na straně procesoru ve klientském jazyce, ze kterého je knihovna *OpenGL* volána. Pomocí *uniform* je tak možné nastavovat parametry daného shaderu. *Uniformy* lze použít jen jako vstupy. Mohou být seskupeny do struktur podobně jako v jazyce C.

GLSL podporuje vektorové a maticové datové typy. Mezi tyto typy patří např. `vec2`, `vec3` nebo `mat4`. Práce s těmito typy umožňuje tzv. *swizzling*. *Swizzling* je název pro mož-

```
uniform vec2 offset;

in vec2 position;
in vec2 uv;

out vec2 uvInterp;

void main() {
    uvInterp = uv;
    gl_Position = vec4(position + offset, 0.0, 1.0);
}
```

Obrázek 2.10: Ukázka *vertex shaderu* zapsaného v GLSL pro transformaci geometrie. *Shader* vezme vstupní pozici vertexu, přičte k ní posun a výsledek zapíše. UV koordináty předává beze změny dále.

nost adresovat vektor po jednotlivých komponentách v libovolném pořadí. Lze to chápat jako rozšíření běžného přístupu do datové struktury. Na obrázku 2.11 je ukázka takového přístupu.

```
vec3 test = vec3(1.0, 2.0, 3.0);
vec3 test_copy = test.zyx; \\ obsahuje hodnoty (3.0, 2.0, 1.0)

vec2 pos = test.xz; \\ obsahuje (1.0, 3.0)
vec4 color = pos.xxyy; \\ obsahuje (1.0, 1.0, 3.0, 3.0)
```

Obrázek 2.11: Ukázka GLSL s použitím *swizzlingu*.

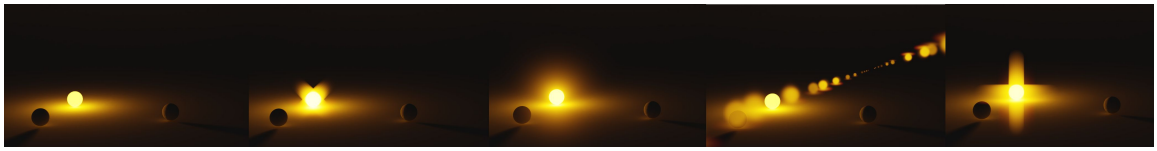
Pro přístup k barevným datům textury se používá vestavěná funkce `texture`. Prvním parametrem je objekt typu `sampler`, který odpovádá jedné texturovací jednotce. Nejčastějším *samplerem* je `sampler2D`. Existují i *samplery* pro jiné typy textur, např. `sampler3D` pro 3D textury, *samplerCube* pro textury na pozadí scén nebo odrazy, které je také možné použít ve funkci `texture`.

Druhým parametrem jsou koordináty, pro které funkce vrátí barvu. Koordináty nabývají formy vektoru (`vec2` pro běžné textury, `vec3` pro 3D textury). Zpravidla se označují jako UV koordináty. Jde o relativní koordináty. Například dvojice $(0.5, 0.5)$ označuje střed textury. Mohou ale nabývat i hodnot mimo rozsah 0.0 až 1.0. Jak se textura bude chovat mimo tento rozsah lze nastavit pomocí parametrů na straně procesoru.

2.4 Existující řešení

Lens flare efekty lze simulovat mnoha způsoby. Je možné použít metody jako photon mapping (mapování fotonů). [8] Photon mapping metody zpočívají v simulaci miliónů fotonů a jejich následném zpracování. Jiné metody používají ray-tracing (sledování paprsku), někdy i kombinovaný s rasterizací pro akceleraci výsledku. [6]. Některé tyto metody je možné i akcelerovat takovým způsobem, že jsou vhodné i pro real-time aplikace jako hry nebo architektonické vizualizace ve virtuální realitě. [9] Tyto metody jsou zpravidla používány kvůli přesnosti, takže pracují s matematickým popisem čoček. Je možné využít i jednodušší, avšak nepřesné metody jako kompozice statických obrázků. Tuto metodu využívá např. placený doplněk pro *Blender Flared*, který lens flare vkládá přímo do scény. [2]

Blender obsahuje v kompozitoru uzel *Glare*, který umí simulovat některé vlastnosti lens flare. Tento uzel umí simulovat pět různých efektů. Je na něm možné nastavit kvalitu efektu, což v praxi sníží rozlišení výsledného efektu a tím může nechtěně zvětšit nebo zmenšit velikost efektu. *Glare* může mít jako výstup jen jeden aktivní efekt, takže na zkombinování více efektů dohromady je potřeba více uzlů. Je možné na uzlu nastavit intenzitu efektu nebo práh od kterého se zdrojový pixel považuje za světlo. Výstupy uzlu jsou znázorněny na obrázku 2.12.



Obrázek 2.12: Ukázka efektů uzlu *Glare*: zleva doprava
a) žádný efekt, b) *Simple Star*, c) *Fog Glow*, d) *Ghosts*, e) *Streaks*

Kapitola 3

Návrh

Zde je popsán návrh simulace efektu, jeho vstupní parametry a očekávané chování. Dále je zde návrh uživatelského rozhraní a návrh použití doplňku.

3.1 Návrh simulace lens flare efektu

Jako způsob simulace je použito generování textur s jednotlivými artefakty, které se pak spojí do jednoho obrázku. Vybraný způsob simulace není fyzikálně založený, ale snaží se vizuálně přiblížit k reálným efektům. *Blender* se zpravidla využívá pro offline rendering (vykreslování předem do obrázku), takže rychlost generování lens flare by se měla pohybovat v řádu sekund. Doplněk vyčte tvar závěrky z aktivní kamery ve scéně, nebo nabídne uživateli možnost manuální kontroly.

Efekt je možné definovat jako kombinaci dvou uživatelsky nastavitelných efektů:

- záři a paprsky okolo zdroje světla
- *ghosts*, jejich počet, barvy, pozice, tvar a další parametry

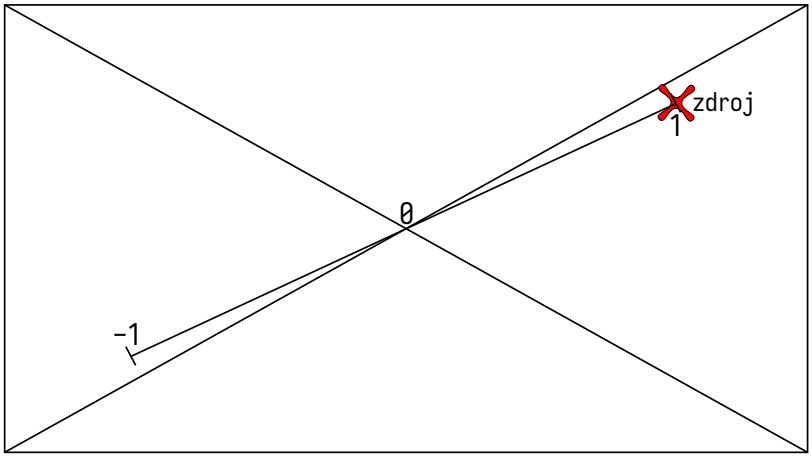
Paprsky bývají výraznou částí efektu. Během pohybu kamery jsou zase více výraznější *ghosts*. Doplněk proto simuluje obojí.

Díky tomu, že *ghosts* jsou vždy v rovné linii mezi zdrojem světla a středem snímku, je možné jejich polohu popsat jen jedním parametrem. Na obrázku 3.1 je znázorněno fungování toho parametru. Pro větší kontrolu nad výslednou pozicí artefaktu je možné použít posun kolmý ke zmíněné linii.

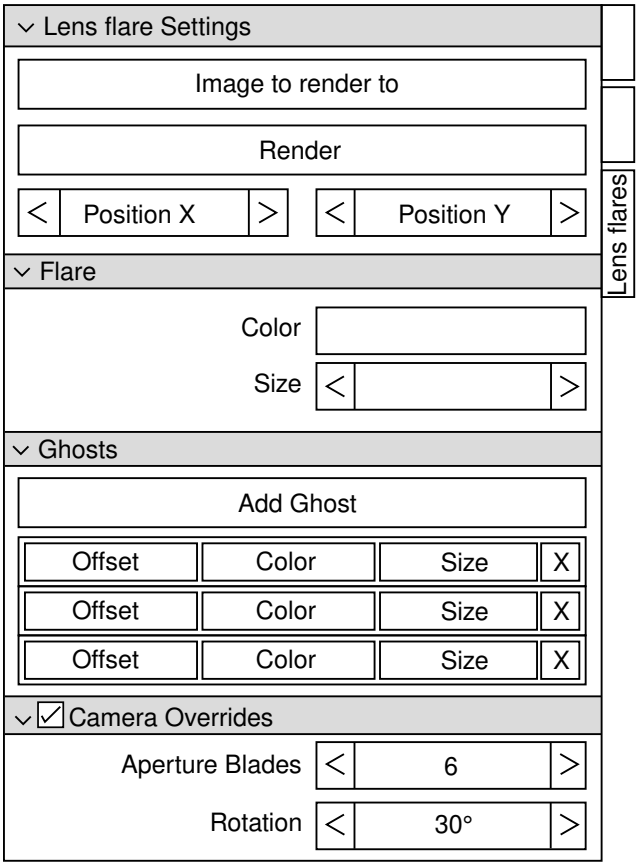
Velikost vykreslovaných artefaktů by měla být relativní k poměru stran snímku, aby jí změny rozlišení snímku neovlivnily. Velikost jednotlivých *artefaktů* je pak možné popsat jako počet procent šířky snímku (tzn. *ghost* o velikosti 100 je pak široký přes celý snímek).

3.2 Návrh uživatelského rozhraní

Hlavní způsob, jak přidat vlastní uživatelské rozhraní do kompozitoru je přes vytvoření nového panelu. Panely ve výchozím rozložení bývají v pravém horním rohu podokna. Na obrázku 3.2 je návrh uživatelského rozhraní pro doplněk.



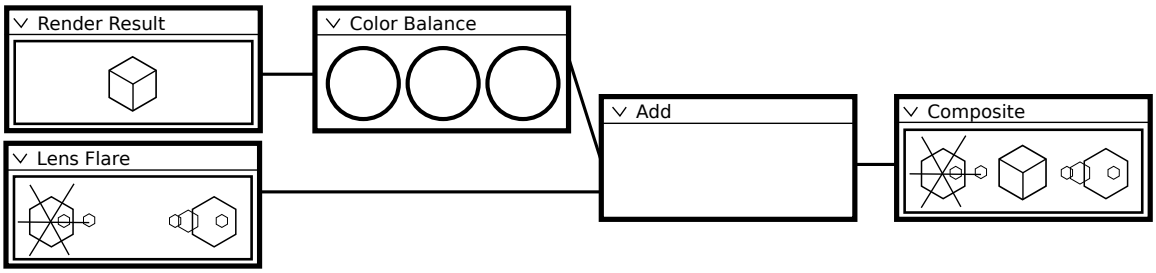
Obrázek 3.1: Diagram pozice artefaktů na základě parametru posunu. Pokud je hodnota parametru 1.0, tak je artefakt na pozici zdroje světla. Pokud je hodnota opačná, tj. -1.0, tak je artefakt umístěn středově symetricky ke zdroji světla. Tento parametr je lineární a může nabývat libovolných hodnot pro větší uměleckou kontrolu.



Obrázek 3.2: Návrh uživatelského rozhraní pro doplněk. V horní části jsou obecné parametry efektu jako pozice, cílový obrázek, atd. Potom následuje část pro nastavení záře okolo světelného zdroje. Pod ní je oblast věnující se nastavení *ghostů*, s možností definice pozice, barvy, velikosti, atd. Nakonec je dole oblast pro přepsání parametrů kamery.

3.3 Návrh použití efektu

Protože v Blenderu není možné v kompozitoru vytvářet úplně vlastní chování aniž by se modifikoval zdrojový kód Blenderu, efekt je generován do obrázku. Ten je následně poskytnut kompozitoru, kde je pomocí uzlů na míchání barev spojen se zdrojovým obrázkem. Uživatel má tak volnost efekt podle potřeby transformovat. Může si tak i zvolit, v jaké fázi post-processingu efekt do snímku vloží (např. až po efektu filmového zrna). Na obrázku 3.3 je návrh použití efektu v kompozitoru.



Obrázek 3.3: Návrh použití doplňku. Obrázek s efektem je dostupný přes uzel *Image*. Uživatel má kontrolu nad tím, kdy a jak efekt přidá do snímku.

Kapitola 4

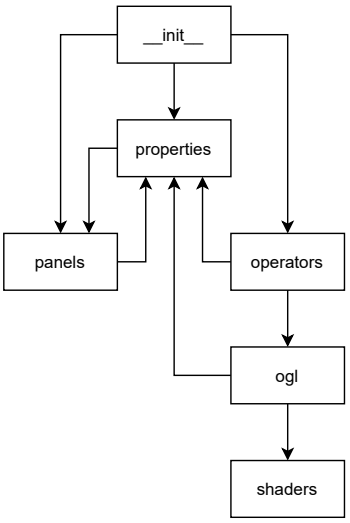
Implementace

V této kapitole je popsán výsledek praktické části práce. Doplněk je možné rozdělit na dvě hlavní části, uživatelské rozhraní a vykreslovací jádro.

Pro obě části je společný kód pro inicializaci doplňku a definice dat. Doplněk je rozdělen do modulů podle funkcionality:

- `__init__`, kde se doplněk inicializuje, případně čistí zdroje při deaktivaci
- `panels`, kde jsou definice uživatelského rozhraní
- `operators`, kde jsou funkce, které doplněk nabízí
- `properties`, kde jsou definice dat co doplněk potřebuje
- `ogl`, kde je kód pro OpenGL vykreslování
- `shaders`, kde jsou *shadery* pro modul `ogl`

Na obrázku 4.1 jsou znázorněny vazby mezi jednotlivými moduly.



Obrázek 4.1: Architektura doplňku. Šipky znázorňují vazby mezi moduly. Např. šipka mezi *ogl* a *shaders* znamená že modul *ogl* importuje věci z *shaders*.

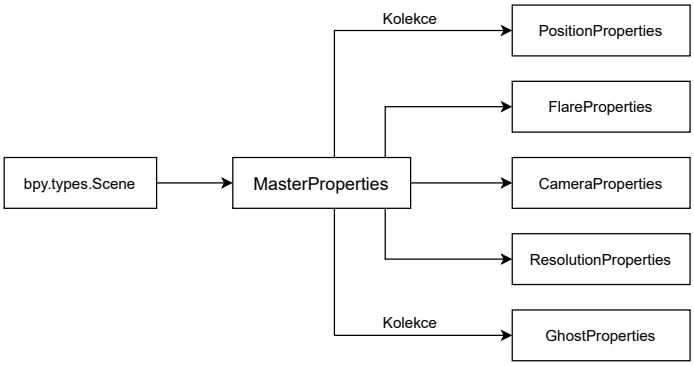
Všechny moduly kromě *___init___* importující datové definice z modulu *properties* je importují jen kvůli typové kontrole při vývoji doplňku. Cyklická vazba mezi *panels* a *properties* je pak způsobena potřebou generovat náhledové obrázky jednotlivých *ghostů* v obou modulech.

Modul *___init___* se stará o inicializaci a deaktivaci doplňku. Při aktivaci zajistí, že scéna bude obsahovat potřebná data pro správnou funkci doplňku. *Blender* má integrovaný operátor na restart sktriptů a doplňků. Pokud toto nastane, tak modul *___init___* zajistí, aby se znovu načetly všechny moduly a nepoužila se verze z vyrovnávací paměti.

V modulu *operators* jsou implementovány operátory doplňku. Hlavním operátorem doplňku je třída `OGLRenderOperator`. Tento operátor zajistí že všechny požadovaná data pro vykreslení existují a jsou validní. Například pokud uživatel chce použít výchozí parametry kamery a nemá ve scéně kameru, tak operátor nelze spustit. Také jej nelze spustit pokud chybí výstupní obrázek. Dalším operátorem je třída `RenderAnimationOperator`, která řeší vykreslení animací. Zbývající operátory jsou hlavně vytvořeny pro práci se seznamy.

V modulu *ogl* je kód pro komunikaci s *OpenGL API*. Hlavní funkce v tomto modulu jsou `render_lens_flare` a `render_debug_cross`. Ostatní funkce jsou volány z těchto dvou funkcí. Modul *shaders* obsahuje třídu starající se o načítání *shaderů* a jejich kombinování do požadovaných programů.

Doplněk si vlastní data ukládá tak, že rozšiřuje datový blok se scénou o položku s typem `MasterProperties` z modulu *properties*. Struktura dat doplňku je hierarchická. Každá třída obsahuje část celkového nastavení. Třída `MasterProperties` obsahuje hlavní parametry efektu a parametry, které by byly nevhodné jinde. Na obrázku 4.2 je znázorněno jak vypadá struktura dat.



Obrázek 4.2: Architektura dat doplňku. Šipky znázorňují vazby mezi jednotlivými třídami. Šipky označené slovem kolekce naznačují, že třída obsahuje kolekci jiných tříd.

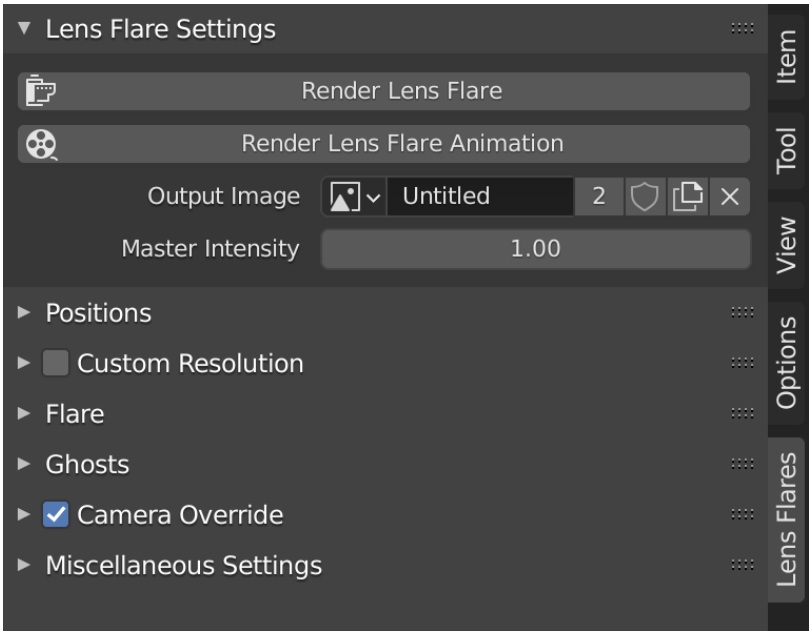
Při prototypování efektů jsem použil nástroje Shadertoy [1] a Godot Engine [2].
Oba nástroje umožňují velmi rychlé prototypování *fragment shaderů*.

[1] https://www.shadertoy.com/
[2] https://godotengine.org/

4.1 Uživatelské rozhraní

Uživatelské rozhraní doplňku bylo implementováno tak, aby zapadalo mezi ostatní prvky uživatelského rozhraní v *Blenderu*. Rozhraní je rozděleno na několik panelů. Každý panel obsahuje nastavení některé části efektu. Stejně jako ve zbytku rozhraní *Blenderu* je možné panely minimalizovat nebo vyměnit jejich pořadí. Všechny panely jsou implementovány v modulu `panels`.

Rozhraní doplňku je dostupné z kompozitoru, protože se předpokládá že uživatel doplňku bude efekt přidávat při finální kompozici. Je však i možné efekt použít jako texturu jako každou jinou. Doplněk tak po každém vykreslení efektu zajistí, že se rozhraní kompozitoru překreslí, aby uživatel hned viděl svoje úpravy. Všechny panely jsou dostupné ze stejné záložky v kompozitoru. Na obrázku 4.3 je pak výsledné uživatelské rozhraní se zavřenými panely.



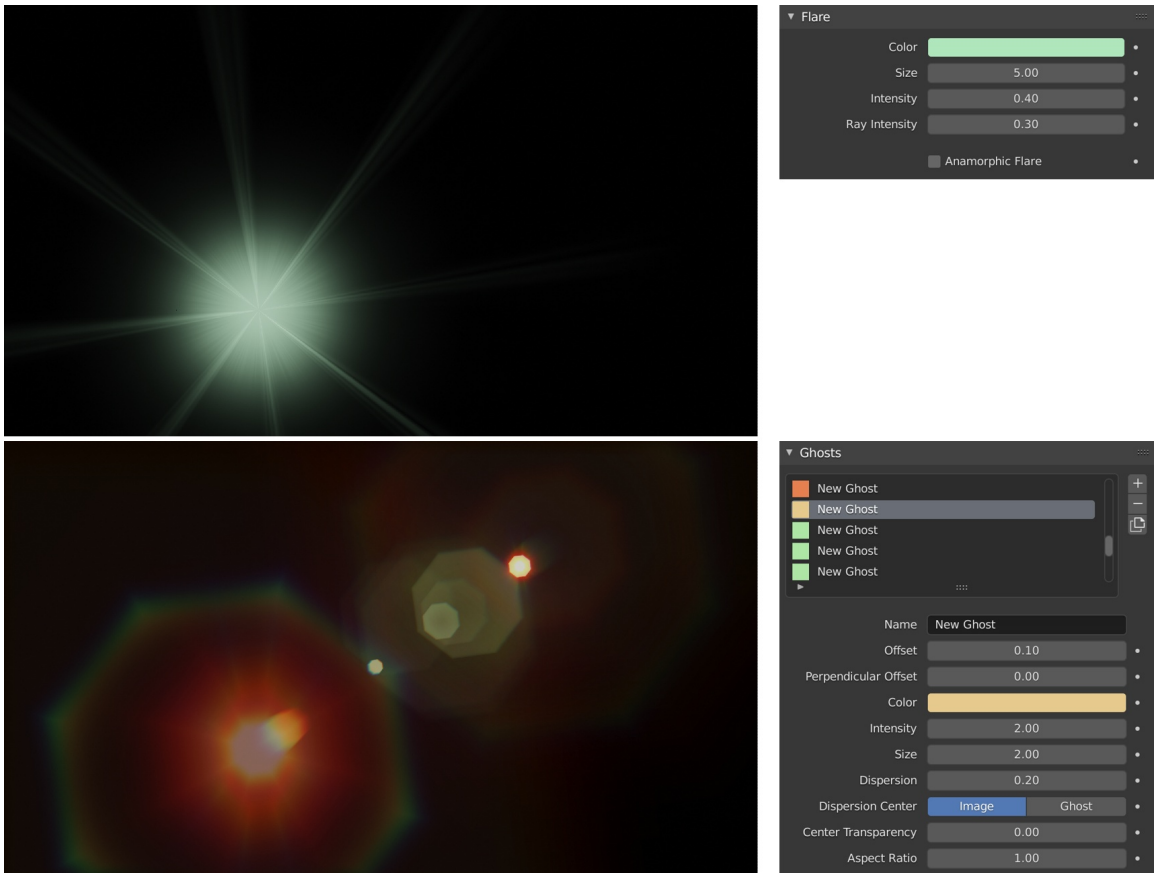
Obrázek 4.3: Ukázka hotového uživatelského rozhraní se zavřenými panely.

Pokud je to dává u dané položky smysl, vykresluje se hodnota položky ve speciálním formátu. Například rotace závěrky je ve stupních, rozlišení efektu v pixelech, atd. Většina položek má dva sety limitů. Pokud je vstup použit jako *slider*, používá jiné maxima a minima než když uživatel napíše hodnotu přímo do vstupu. Toto chování je možné najít i na jiných místech v *Blenderu*, např. některé vstupy pro barvy umožňují zadat i hodnoty mimo rozsah 0.0 až 1.0.

Ve vrchní části je panel *Lens Flare Settings*. Z tohoto panelu se spouští vykreslování efektu a animace. Na tomto místě se nastavuje cílový obrázek do kterého se efekt vykreslí. Pokud obrázek není zadaný, tlačítko na animace je neaktivní a nelze jej zmáčknout. V tomto panelu se také nastavuje celková intenzita efektu. Pro nastavení obrázku doplněk používá šablonový prvek, ze kterého je možné cílové obrázky přejmenovat, smazat. atd. Tento prvek je na více místech v uživatelském rozhraní *Blenderu*.

4.1.1 Panely pro vzhled efektu

O hlavní parametry efektu se starají panely *Flare* a *Ghosts*. Na obrázku 4.4 je znázorněno rozhraní panelů a kterou část efektu ovládají.



Obrázek 4.4: Panely ovládající efekt. Ve vrchní části je panel *Flare* a vedle něj část efektu s září a paprsky. Dole je panel *Ghosts* a artefakty které se z něj nastavují.

V panelu *Flare* se nastavují parametry záře a paprsků. Kvůli zjednodušení práce je od sebe oddělena intenzita a barva efektu. Intenzitu paprsků a intenzitu záře je možné nastavit zvlášť. Je zde taky možné nastavit anamorfní styl záře a paprsků.

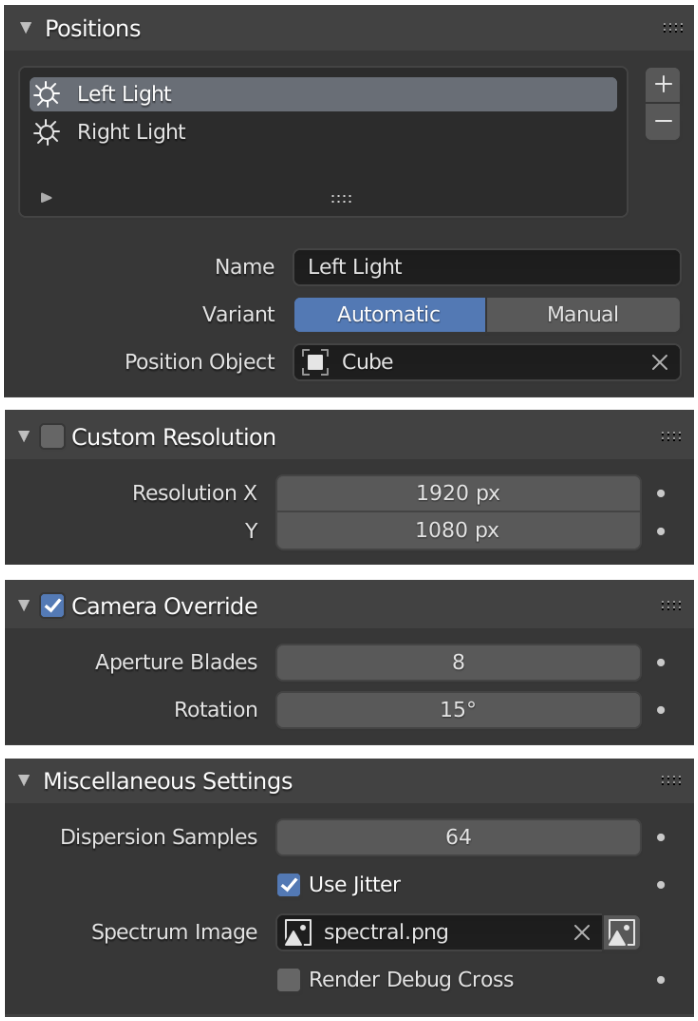
Panel *Ghosts* slouží k editaci zbývajících artefaktů. Na rozdíl od návrhu je zde použit seznam jednotlivých *ghostů*. Během vývoje bylo zřejmé že narůstající počet parametrů a počet *ghostů* by způsobil, že původně navržené rozhraní se rychle stane nepřehledným. Použitý seznam je navržen podle seznamů existujících jinde v uživatelském rozhraní *Blenderu*.

Vedle seznamu *ghostů* jsou tlačítka na přidávání a odebírání jednotlivých artefaktů. Pro zrychlení práce má seznam také tlačítko na duplikaci aktivního *ghosta*. Kvůli přehlednosti má každý *ghost* v seznamu u jména malý čtverec se svou barvou. Náhledy *ghostů* v seznamu jsou generovány jako obrázky v rozlišení 2×2 pixelů. Tyto obrázky jsou při každé změně barvy nějakého z *ghostů* znovu vygenerovány.

Pokud je nějaký *ghost* vybraný, tak se pod seznamem zobrazí políčka pro editaci jednotlivých atributů. Každý *ghost* může mít uživatelem přiřazené jméno pro lepší organizaci, které je zde upravitelné.

4.1.2 Zbývající panely

Na obrázku 4.5 jsou zbývající panely. Tyto panely nastavují parametry, které ovlivňují chování všech vykreslených artefaktů. Panely *Custom Resolution* a *Camera Override* lze aktivovat a deaktivovat tlačítkem vedle jména panelu.



Obrázek 4.5: Zbývající panely doplňku. Panely ovládají kvalitu, rozlišení a pozice efektu.

Panely jsou ve výchozím rozložení v tomto pořadí:

- Panel *Positions*, kde je možné nastaví pozici efektu na obrázku. Efektů může být v obrázku několik, což je užitečné když je ve snímku několik zdrojů světla. Pozice na obrazovce mohou být zadány buď manuálně, nebo automaticky podle pozice objektu na obrazovce.
- Panel *Custom Resolution* umožňuje použít na vykreslení efektu jiné rozlišení než je rozlišení výsledného snímku. Je tak možné zvýšit kvalitu efektu (na každý pixel výsledného snímku se pak použije více pixelů efektu).
- Panel *Camera Override* zase umožňuje potlačit nastavení kamery. Hodnoty v tomto panelu se chovají úplně stejně jako obdobné hodnoty nastavení kamery, které jsou

pomocí tohoto panelu možné potlačit. Vstup na počet hran závěrky nedovoluje vložit hodnoty 1 a 2, které nejsou validní (závěrka o jedné hraně nedává smysl).

- Panel *Miscellaneous Settings*, kde je možné nastavit kvalitu a režim disperze, nastavit vlastní texturu barevného spektra nebo vykreslit ladící kříž s polohou efektu.

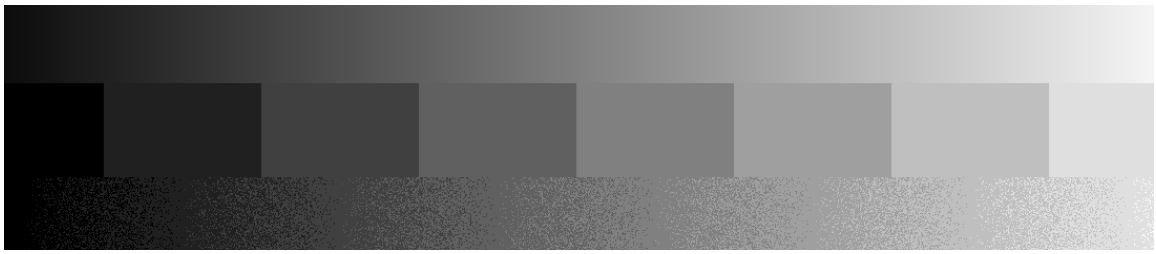
4.2 Vykreslovací jádro

Vykreslování bylo realizováno pomocí knihovny *OpenGL* s použitím *Blender Python API*. Efekt se vykreslí mimo obrazovku a je následně nakopírován do obrázku, kde je s ním možné v *Blenderu* dále pracovat. Vykreslování probíhá v tomto pořadí:

- prvně se vykreslí *ghost* do vedlejšího *framebufferu*
- následně se nakopíruje do hlavního *framebufferu*, během tohoto kroku se simuluje disperze (viz. 2.1)
- tento proces se opakuje pro všechny *ghosty*
- nakonec se vykreslí záře a paprsky

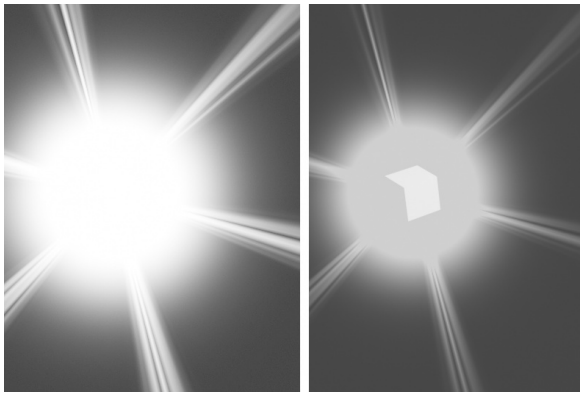
Mezi jednotlivými *ghosty* se vedlejší *framebuffer* nastaví na černou barvu. Při vykreslování se na sebe všechny vykreslené artefakty jednoduše aditivně přidávají.

Všechny *framebuffery* poskytnuté API třídou `gpu.types.GPUOffScreen` mají osmibitový formát. Díky tomu je možné, že vykreslenému efektu bude chybět barevná přesnost a nastane *banding*. *Banding* je nežádoucí jev způsobený nedostatečnou přesností digitálního formátu obrázku. Obrázek trpící *bandingem* má místo hladkých barevných přechodů přechody, které se jeví „schodovitě“. Banding je možné potlačit pomocí *ditheringu*. *Dithering* zpočívá v přidání cíleného šumu do obrázku takovým způsem, aby nenarušil obsah, ale narušil strukturu barevných pruhů vytvořených kvůli *bandingu*. Na obrázku 4.6 je ukázka *bandingu* a *ditheringu*.



Obrázek 4.6: Ukázka bandingu a ditheringu. Ve vrchní části obrázku je přechod z černé do bílé. V prostřední části je ten samý přechod, jen s nedostatečnou přesností. Ve spodní části je přechod se stejnou přesností jako prostřední, ale šum v přechodu dodává pocit plynulejšího přechodu.

Zároveň efekt nepodporuje vysoký rozsah hodnot. Po vykreslení efektů jsou hodnoty ve *framebufferu* ořízntuty na rozsah 0 až 255. Při kompozici efektu v *Blenderu* se pak může stát že efekt bude mít nižší intenzitu než vykreslovaný snímek do kterého se efekt kompozituje. *Blender* od verze 2.92 v kompozitoru má uzel *Exposure*, který tento problém částečně obchází. Na obrázku 4.7 je výsledek efektu a vliv uzlu *Exposure*.

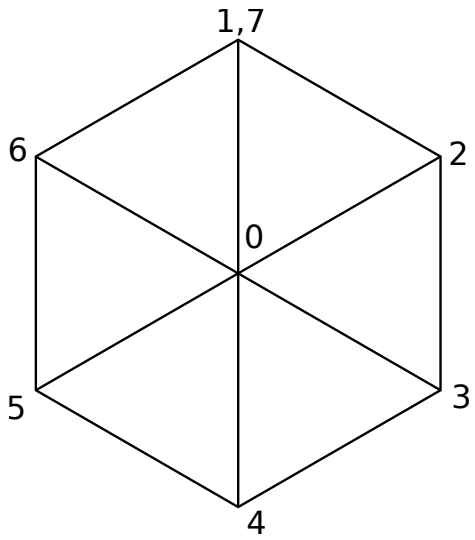


Obrázek 4.7: Ukázka uzlu *Exposure*. Efekt je na stejné pozici jako kostka, protože ale na kostku dopadá silné světlo, je jasnější než efekt. Efekt je na obou snímcích vykreslován s maximální intenzitou. Vlevo je použit uzel *Exposure* před kompzozicí do obrázku. Intenzita záře je uzlem posílena bez existence jiných artefaktů způsobených malou přesností obrázku.

4.2.1 Vykreslování *ghostů*

Kvůli tomu, že každý *ghost* může mít úplně jiné parametry se vykresluje každý zvlášť. Geometrie *ghosta* se předem vygeneruje na straně procesoru. Na obrázku 4.8 je ukázka takové geometrie. Protože je využitý *TRIANGLE_FAN* (viz. 2.3) mód na sestavení primitivních objektů, tak je jeden vertex obsažen dvakrát. Centrální bod má jinou barvu, aby bylo možné vykreslit hranu *ghosta* s menší intenzitou.

Počet stran *ghosta* je určen tvarem závěrky. Pokud je závěrka kruhová, *ghost* se vykreslí jako pravidelný mnohoúhelník s 256 stranami. Všechny instance artefaktu *ghost* sdílí jednu geometrii. Transformace geometrie jsou dělány pro každý artefakt zvlášť.



Obrázek 4.8: Ukázka geometrie *ghosta*

Geometrie *ghosta* se při vykreslování se podle parametrů transformuje do žádané pozice, velikosti a tvaru. Pozice je určena podle parametru posunu a dodatečného kolmého posunu. Pokud je nastavena rotace závěrky, *ghost* se otočí do žádané pozice. Transformační matice

pro tuto transformaci jsou vypočítány na straně procesoru a předány do *shaderu* přes uniformní proměnnou.

V této fázi se i nastaví žádaný poměr stran artefaktu dle parametrů. Doplněk poměr stran určuje podle parametru, který se chová stejně jako parametr *Ratio* v nastavení hloubky pole kamery v *Blenderu*. Pokud je parametr např. 2.0, tak *ghost* má poloviční šířku, pokud je 0.5, tak *ghost* je dvakrát širší.

Po vykreslení ghosta do vedlejšího *framebufferu* se nakopíruje obrázek do hlavního *framebufferu*. Během tohoto kroku se simuluje disperze. Efekt simulace disperze je rastrový, jako vstup bere texturu s vykresleným *ghostem* z předchozího kroku. Geometrie pro vykreslení takového efektu je ploška tvořená ze dvou trojúhelníků, která pokrývá celou obrazovku.

Pro simulaci disperze je použita řídící textura, která určuje, jak bude barevné spektrum vypadat. Tato textura je nastavitelná uživatelem. Na obrázku 4.9 je zobrazena výchozí textura spektra použitá v doplňku. Textura byla navržená tak, aby celková intenzita červené, zelené a modré barvy bylo přibližně stejná. Textura tak nezkreslí barvu *ghosta* když disperze není použitá.

Obrázek 4.9: Ukázka textury spektra

Doplněk umožňuje dva režimy disperze, disperze od středu snímku a od středu *ghosta*. Míra disperze je daná uživatelem jako parametrem s hodnotami od -1.0 do 1.0. Je tak možné spektrum invertovat.

Efekt disperze je vypočítán pomocí vzorkování. Počet vzorků disperze je definovaný uživatelem (viz. kapitola 4.1.2). Každý vzorek má jeden klíčový parametr X, podle kterého je určeno jakou barvu bude vzorek mít a ze kterého se určí jeho pozice. Tento parametr má hodnoty v rozsahu 0.0 až 1.0. Vzorky pro jeden pixel mají rovnoměrné rozložení v tomto rozsahu. Parametr X je určen podle vzorce 4.1.

$$X = \frac{\text{číslo vzorku} + \text{náhodný posun}}{\text{celkový počet vzorků}} \tag{4.1}$$

Protože výsledek by měl ostré okraje způsobené nízkým počtem vzorků, každý pixel je vzorkován s mírným posunem pomocí textury šumu. Náhodný posun je stejný pro všechny vzorky na jednom pixelu. Efekt je díky tomu hladší za cenu šumu. Počet vzorků však tento šum dokáže potlačit. Tento posun vzorkování je možné vypnout.

Pozice na zdrojovém snímku, odkud vzorek bere barvu je dána podle vzorce 4.2. Střed disperze, pozice pixelu a cílová pozice vzorku jsou dvourozměrný vektor.

$$\text{pozice vzorku} = ((\text{pozice pixelu} - \text{střed disperze}) \times \text{disperze vzorku}) + \text{střed disperze} \tag{4.2}$$

Disperze vzorku je jednorozměrný parametr, který bere v potaz míru disperze, kterou uživatel zadal a parametr X daného vzorku. Disperze vzorku je definovaná podle vzorce 4.3.

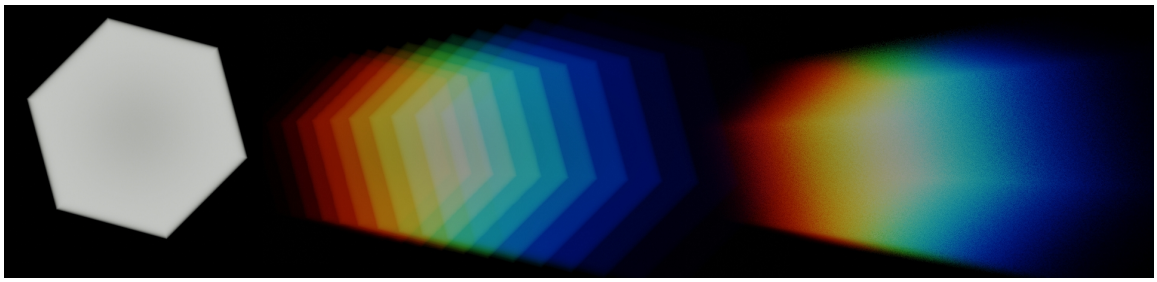
$$\text{disperze vzorku} = (X - 0.5) \times 2.0 \times \text{míra disperze} + 1.0 \tag{4.3}$$

Nabývá hodnot v rozsahu od 0.0 do 2.0. Střed disperze je vždy na pozici *ghosta* před vykreslením disperze.

Barva vzorku je určena barvou zdrojového snímku vynásobenou s barvu získanou z textury spektra. Protože parametr vzorku je v rozsahu od 0.0 do 1.0, je možné tento parametr jednoduše převést na UV koordináty textury spektra.

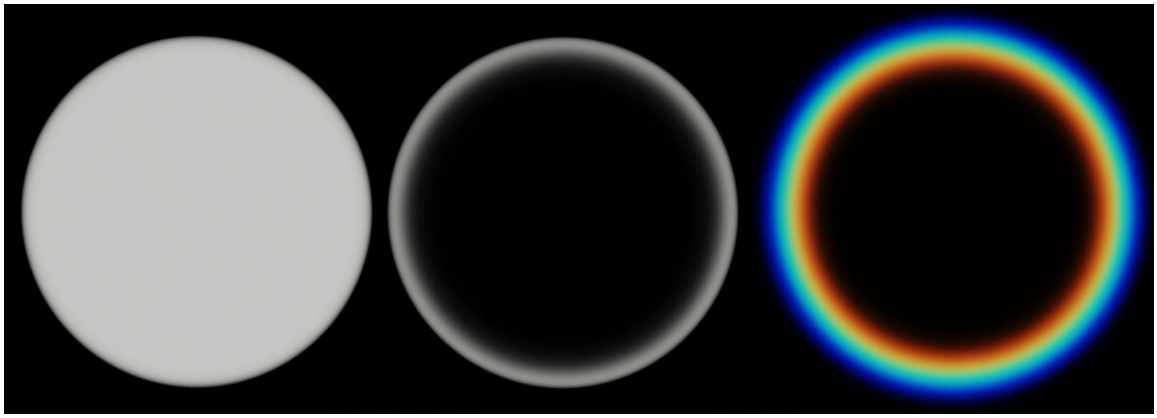
Pokud je disperze nulová, všechny vzorky se vykreslí na stejné místo. To má pak za následek, že *ghost* je prakticky vynásobený celkovou intenzitou barevného spektra.

Po sečtení všech vzorků se celková intenzita fragmentu vydělí počtem vzorku, aby celková intenzita zůstala stejná, nezávisle na počtu vzorků. Na obrázku 4.10 je znázorněno fungování simulace disperze.



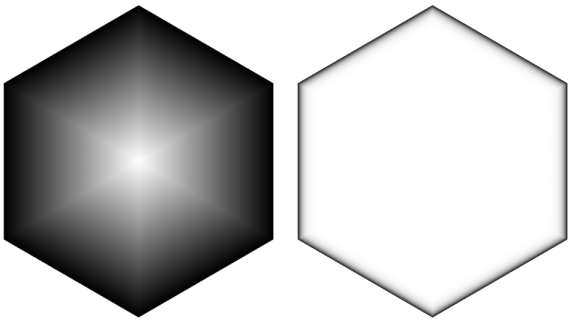
Obrázek 4.10: Simulace disperze. Zleva doprava - a) *ghost* bez disperze, b) *ghost* s disperzí, c) *ghost* s disperzí a náhodným vzorkováním

Každý *ghost* má parametr na průhlednost ve svém středu. Tento parametr byl zvolen kvůli snaze vykreslovat duhové kruhy, které se někdy mohou v reálných efektech objevit. Pokud je tento parametr vhodně zkombinován s disperzí, je možné takových artefaktů dosáhnout. Na obrázku 4.11 je znázorněno postupné vykreslení takového efektu. Protože *ghosts* mohou mít nízký počet hran, je tento efekt počítán až ve *fragment shaderu* z pozice fragmentu relativně ke středu artefaktu.



Obrázek 4.11: Simulace duhových kruhů. Zleva doprava - a) *ghost* bez průhledného středu, b) *ghost* s průhledným středem, c) *ghost* s průhledným středem a disperzí

Vnější okraje *ghostů* mají sníženou intenzitu aby výsledný *ghost* neměl příliš ostré okraje. Tohoto efektu je dosaženo pomocí mapování barev předaných společně s geometrií. Na obrázku 4.12 je znázorněna geometrie *ghosta* s barvou geometrie.

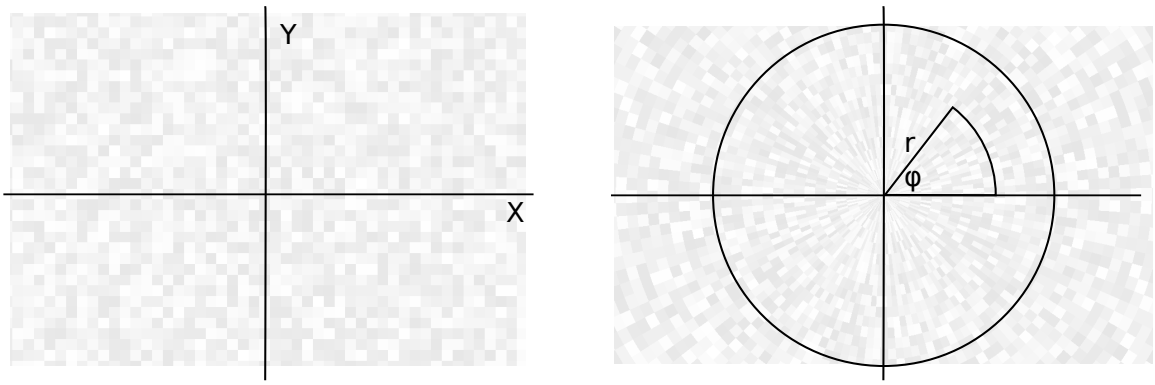


Obrázek 4.12: Vizualizace okraje *ghosta*. Vlevo je *ghost* s lineárně interpolovanou barvou, vpravo *ghost* s namapovanou barvou. Barvy jsou invertovány kvůli přehlednosti.

4.2.2 Vykreslování záře a paprsků

Po vykreslení všech *ghostů* se vykresluje záře a paprsky přímo do hlavního bufferu. Tento efekt je vykreslován pro každý pixel na snímku, takže lze znovu použít geometrii použitou pro vykreslení disperze. Na vykreslení normálních i anamorfních paprsků je použitý stejný *shader*. Záře okolo středu je simulována pomocí Gaussovy funkce.

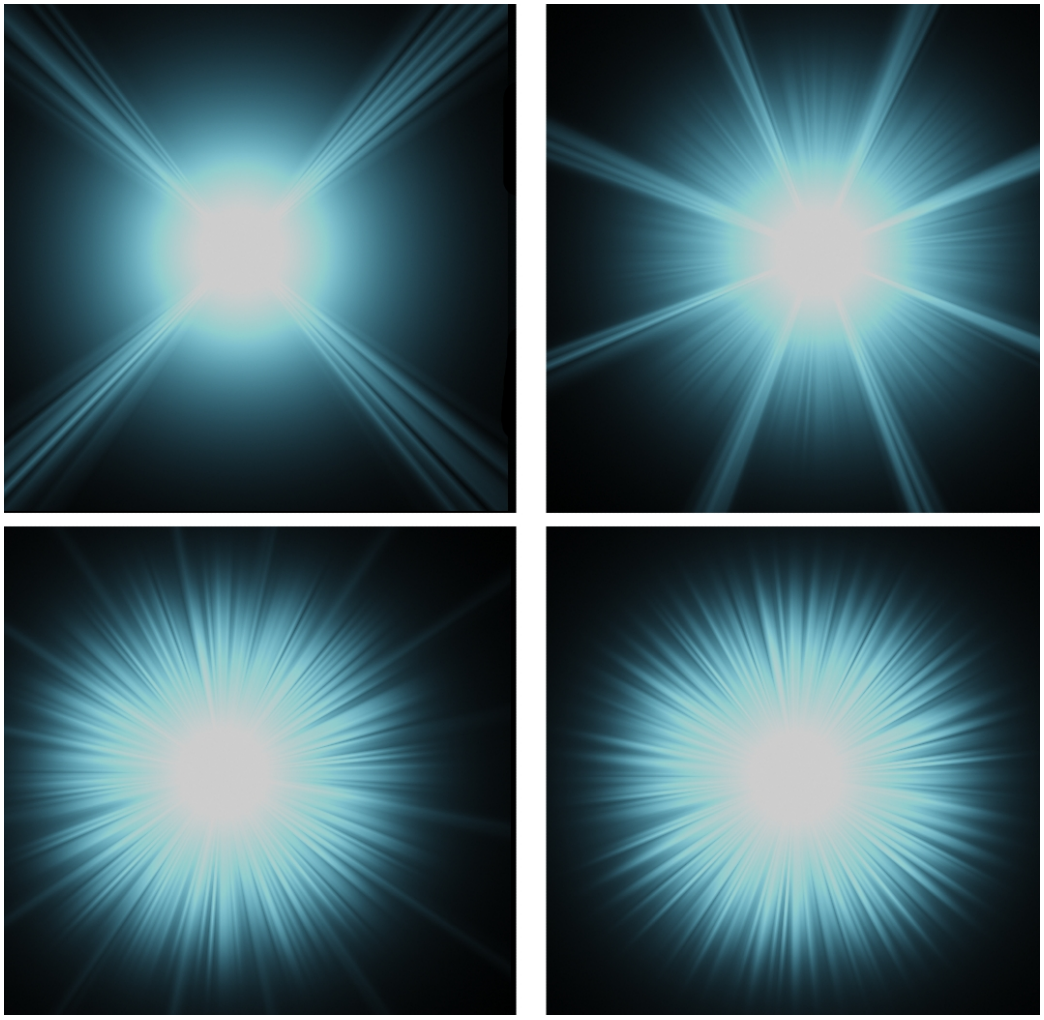
Na vykreslení paprsků je použita textura náhodného černobílého šumu, která je pak namapována na polární koordináty. Na obrázku 4.13 je znázorněno mapování na polární koordináty.



Obrázek 4.13: Eulerovy a polární koordináty. V pozadí je znázorněno, jak by se na takové koordináty mapovala textura. Textura šumu převzata z Shadertoy.

Při použití normálních paprsků je jejich síla modulována počtem hran závěrky. Pokud je závěrka kruhová, paprsky jsou rozmístěné rovnoměrně okolo záře. V případě polygonální závěrky jsou paprsky soustředěny do několika silnějších paprsků. Počet silnějších paprsků závisí na počtu hran závěrky. Na obrázku 4.14 je znázorněno toto chování.

Při vykreslování anamorfních paprsků nemá tvar závěrky na tvar paprsků žádný vliv. Tento efekt je tvořen kombinací krátkých paprsků rozmístěných rovnoměrně okolo zdroje světla. K tomu je přidán hlavní silný paprsek charakteristický pro anamorfní čočky. Tento paprsek je horizontální. Na obrázku 4.15 je ukázka anamorfního paprsku. Horizontální paprsek je vykreslen jako pruh, který je širší okolo zdroje záře světla. Kruh paprsků okolo středu záře je vykreslen pomocí již zmíněných polárních koordinátů.



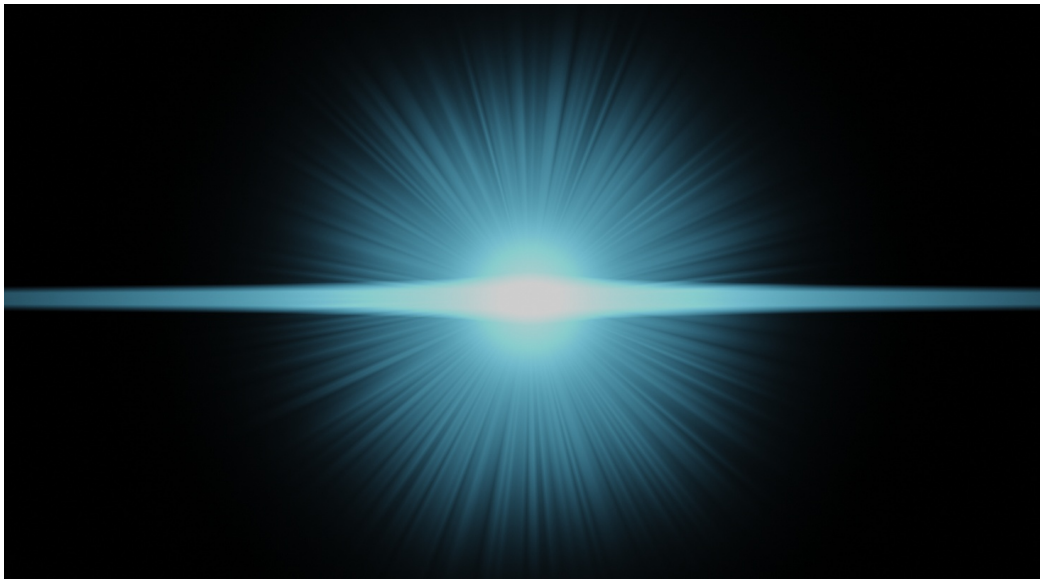
Obrázek 4.14: Chování paprsků při měnícím se tvaru závěrky. Zleva doprava, shora dolů - počet hran závěrky - 4, 8, 16, kruhová.

4.3 Ostatní funkcionalita

Protože je střed efektu zadávaný jako dvojice koordinátů v relativních jednotkách, obtížně se určuje přesná pozice efektu na snímku. Z tohoto důvodu je možné zapnout ladící režim, který místo efektu vykreslí kříž, pomocí kterého je pak možné efekt přesněji umístit.

4.3.1 Animace

Jednou z menších priorit při tvorbě doplňku byla podpora animací. Všechny atributy jsou díky tomu animovatelné. Každý snímek animace může být jiný, takže efekt se vykresluje pro každý snímek znovu. *Blender* umožňuje doplňkům reagovat na akce uživatele pomocí definování odpovědí na určité události v aplikaci. Před každým novým snímkem animace by tak bylo možné efekt překreslit. *Blender* jenže neposkytuje *OpenGL* kontext při těchto reakcích, takže na animace má doplněk vlastní chování. Animace se sice vykreslí, ale rozhraní *Blenderu* po dobu animace neodpovídá. Výsledek se uloží do jednotlivých snímků pro další zpracování.



Obrázek 4.15: Výchozí anamorfní paprsek a záře. Efekt má silný horizontální paprsek a slabší kruh paprsků okolo zdroje záře.

4.4 Publikace doplňku

Doplněk je zdarma dostupný na *GitHubu*[3]. Vybrané verze mají tag, pomocí kterého je možné stáhnout určitou verzi doplňku. Tu je pak možné jednoduše nainstalovat. Jednotlivé verze doplňku nemusí být plně kompatibilní. Na GitHubu je rovněž krátký návod k použití.

[3] https://github.com/TheNumerus/blender-lensflaregen

Kapitola 5

Testování

Většina testování probíhala současně s vývojem doplňku. Ke konci vývoje jsem doplněk poskytl několika spolužákům na vyzkoušení.

5.1 Vlastní testování

Vlastní testování probíhalo tím způsobem, že po implementaci nějaké nové funkcionality jsem v *Blenderu* hned danou funkci zkoušel. Některé testy probíhaly tím způsobem, že jsem z internetu získal snímek s efektem a ten jsem následně zkusil napodobit za pomocí doplňku. Jeden z takových testů je na obrázku 5.1.

Obrázek 5.1: Výsledek napodobení efektu. Vlevo je výsledek doplňku, vpravo originál z [6]

5.1.1 Chyby odhalené vlastním testováním

Při vývoji jsem narazil na několik zajímavých chyb:

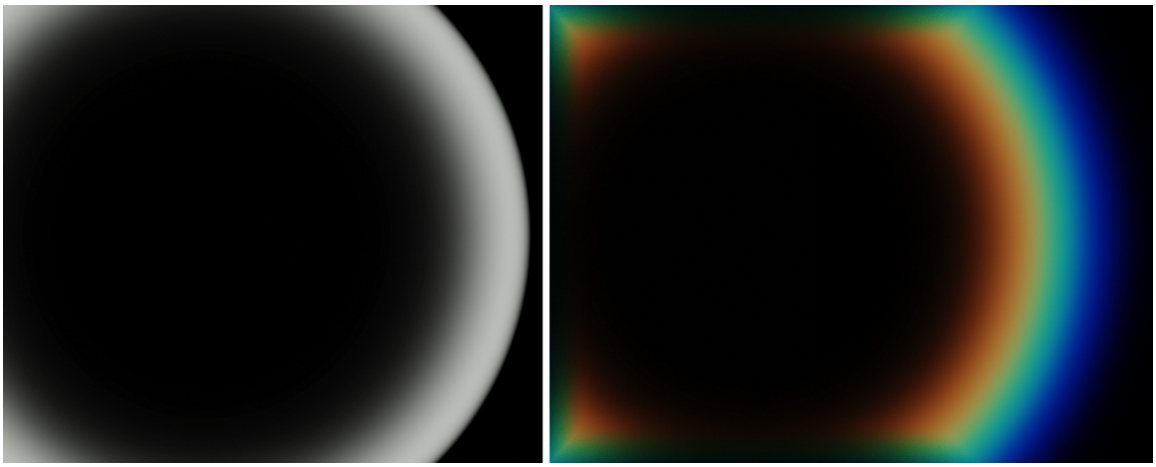
- Narazil jsem na neočekávané chování v *OpenGL* implementaci. Ve Windows jedna texturovací jednotka nereagovala na některé příkazy. V Linuxu se chovala správně. Textura šumu použitá na několika místech v programu má rozlišení 64×64 pixelů. Při vzorkování disperze je mapována tak, aby každý pixel snímku odpovídal přesně jednomu pixelu textury. Textura by se tak mnohokrát opakovala na snímku, aby ho pokryla. Texturovací jednotka ale nereagovala na nastavení opakování textury. Textura spektra nepotřebuje opakovat, takže program nakonec využívá texturovací jednotky v jiném pořadí.

- Dále jsem narazil na chybu v *Blenderu*, kde při načtení obrázku v rozlišení 1×1 pixelů mohl nastát pád aplikace. Po zjištění že jde o nahlášenou, ale zatím nevyřešenou chybu [1], jsem náhledy jednotlivých *ghostů* vykresloval v rozlišení 2×2 pixely.

5.1.2 Nedostatky v simulaci efektu

Doplněk má také několik nedostatků v simulaci artefaktů:

- Doplněk nesimuluje všechny artefakty správně se všemi nedostatky. Chybí podpora pro částečný odraz (jenom část světla dorazí na snímek) a zkreslení způsobené tvarem čoček.
- Doplněk nepodporuje tvar závěrky, který není pravidelný polygon, nebo kruh.
- Může se stát při určitých počtech vzorků disperze, že v artefakty jsou viditelné střídavé oblasti s velkou mírou šumu a oblasti bez viditelného šumu. Tento problém by mělo být možné řešit jinou metodou vzorkování.
- Jedna nevýhoda zvolené metody simulace disperze je, že pracuje jen s daty co už na snímku jsou a nedokáže obstarat data mimo snímek. Může se tak při velké míře disperze stát že data budou chybět a efekt bude vypadat „useknutý“. Na obrázku 5.2 je znázorněna taková obrazová vada.



Obrázek 5.2: Vada v simulaci disperze. Vlevo je snímek *ghosta* bez disperze. Vpravo je tentýž *ghost* vykreslen s disperzí. Protože horní, dolní a levý okraj artefaktu je mimo snímek, disperze nedokáže obstarat potřebná data a *ghost* se jeví oseknutý. V rozích je pak poznat, jak velká disperze je použita.

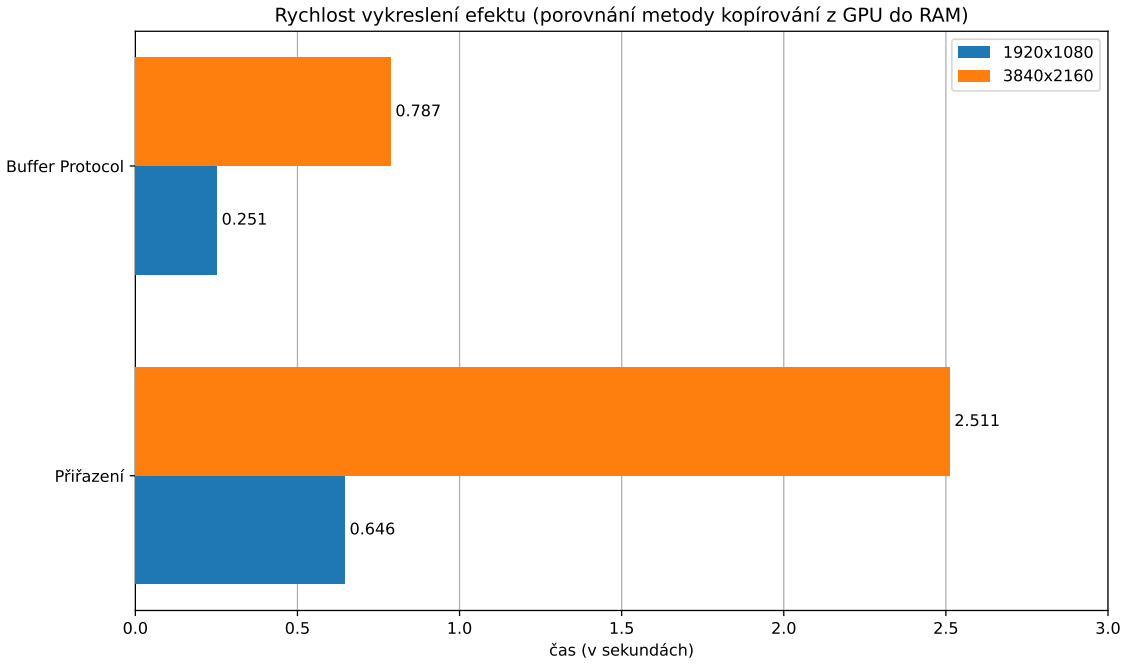
[1] Bug report dostupný z https://developer.blender.org/T70356

5.2 Profiling

Profilování doplňku probíhalo na dvou různých PC a dvou operačních systémech. Jeden PC má dedikovanou grafickou kartu a druhá je notebook s integrovanou grafickou kartou. Mezi testované OS patří Windows 10 v sestavení 19042 a Manjaro Linux.

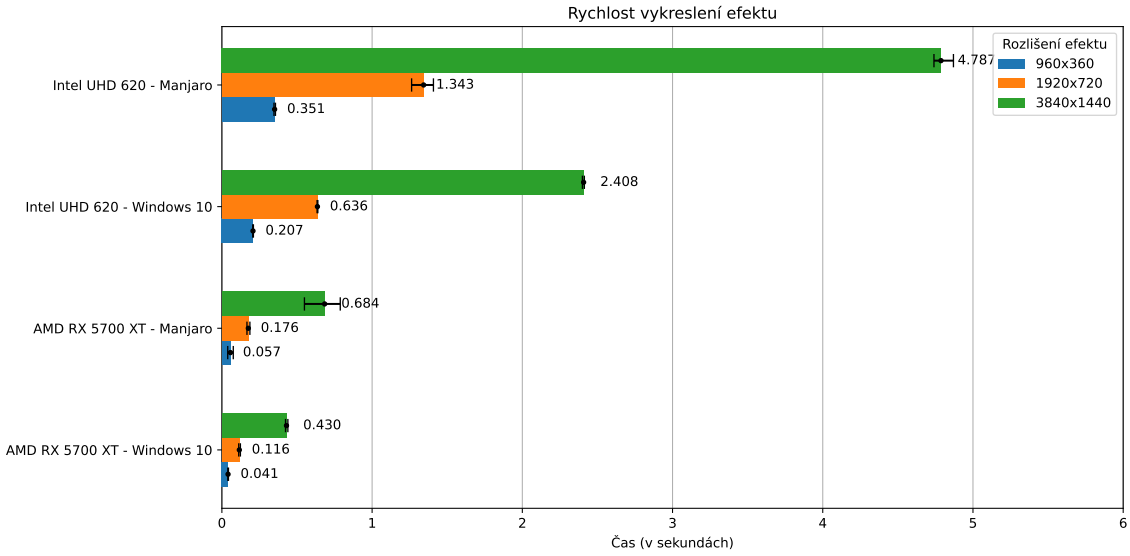
Jeden z důvodů implementace vykreslovacího jádra v *OpenGL* byla rychlost vykreslování. První verze doplňku měly vykreslování implementované pomocí manipulace obrázkových dat v *Pythonu*. Tento způsob byl nepřijatelně pomalý i při použití velmi jednoduchých útvarů. Vykreslení jednoho kruhového *ghosta* trvalo i několik sekund. Efekt může obsahovat i desítky takových artefaktů, takže bylo potřeba efekt přepsat. Vykreslování geometrických tvarů by bylo zbytečně složité.

Po dalším profilování vykreslovacího jádra jsem zjistil, že většinu času doplněk strávil kopírováním výsledného snímku z paměti grafické karty do paměti *Blenderu*. *Blender* ve verzi 2.83 a novějších obsahuje funkci, která toto kopírování značně zrychluje. Nová funkce implementuje *Python Buffer Protocol*, takže kód na kopírování běží na nižší úrovni blíže k hardware. Doplněk tuto funkci využívá, a kvůli tomu je 2.83 minimální podporovanou verzí. Na obrázku 5.3 je graf s porovnáním metod kopírování dat. Toto testování proběhlo se scénou A.3 na PC s Manjaro Linuxem.



Obrázek 5.3: Výsledky testování rychlosti kopírování dat.

Níže 5.4 je graf časů vykreslování s použitím finální verze doplňku. Na obrázku A.4 je znázorněna scéna, na které byly zaznamenány následující časy. Všechny výsledky byly pořízeny s 64 vzorky disperze. Každá varianta testu byla spuštěna třikrát a zaznamenány jsou prostřední časy. Měřen je celkový čas běhu operátoru, tzn. alokace *framebufferů*, kompilace *shaderů* pro grafickou kartu, samotné vykreslení efektu, kopírování do paměti RAM a uklizení zdrojů.



Obrázek 5.4: Výsledky testování výkonu doplňku.

Podle dat lze usoudit že efekt je použitelný i na méně výkonném hardwaru. Na Linuxu je menší výkon pravděpodobně způsoben implementací *OpenGL* v ovladačích grafické karty. Efekt by pravděpodobně bylo možné použít v *real-time* aplikacích jako hry, atd.

5.3 Testování jinými uživateli

Doplněk jsem poskytl dvěma skupinám uživatelů. Jedna skupina uživatelů již měla s tvorbou vlastního obsahu v *Blenderu* zkušenosti. Druhá skupina nikdy předtím *Blender* nepoužila. Všechno testování probíhalo s osobním dohledem.

Uživatelé se zkušenostmi byli schopní po přečtení návodu na stránkách doplňku velmi rychle vytvořit vlastní snímek s efektem.

Uživatelé bez zkušeností s *Blenderem* potřebovali pochopitelně pomoct s orientací v uživatelském rozhraní. Pro tyto uživatele jsem připravil soubor s prázdným efektem pro editaci. Po zorientování v rozhraní doplňku byli nakonec schopní nějaký efekt vytvořit.

Díky tomuto testování jsem zjistil že uživatelé měli možnost nastavit na některých místech extrémní hodnoty, které měly za následek zamrznutí aplikace (milióny vzorků disperze). Dále jsem díky tomuto testování zjistil že by bylo vhodné přidat podporu pro přednastavené efekty, které by sloužily jako výchozí bod pro editaci, ať uživatelé nemusí začínat s prázdným snímkem.

Při předávání souborů s již existujícím efektem nastalo při testování k problému, kde uživatel, který otevíral soubor vytvořený jiným, mohl mít chybějící texturu s barevným spektrem. Díky tomu simulace disperze pracovala s výchozí texturou, což je v případě *Blenderu* růžová textura. Výsledný efekt je pak celý růžový. Problému se dá předejít když uživatel, co efekt vytváří, zabalí texturu se spektrem do souboru. Lepší řešení by bylo automaticky doplňkem přibalovat výchozí texturu, aby problém nenastal.

Uživatelům se stávalo, že omylem klikali na tlačítko pro vykreslení animace. Díky tomu jim rozhraní zamrzlo, z čehož byla zřejmá frustrace. Bylo by vhodné se uživatelů zeptat pomocí vyskakujícího okna jestli opravdu chtějí zahájit vykreslování animace. Alternativně nabídnout možnost nějakým způsobem vykreslování animace zrušit.

Kapitola 6

Závěr

Cílem práce bylo simulovat lens flare efekt a umožnit uživatelům kontrolu nad jeho parametry. Tohoto záměru bylo dosaženo.

Lens flare efekt jsem po nastudování rozdělil na záři, paprsky a obrazové artefakty *ghosts*. Následně jsem těmto artefaktům přiřadil parametry, které jsou nastavitelné uživatelem. Doplněk pomocí knihovny *OpenGL* generuje efekt do obrázku. Tento obrázek je pak využitelný pro další účely v *Blenderu*.

Efekt simuluje řadu artefaktů vznikajících v optické soustavě. Vykresleným artefaktům je možné nastavit barvu, pozici, poměr stran a další parametry. Je možné kontrolovat jejich tvar pomocí parametrů kamery. Doplněk simuluje uživatelem nastavitelnou disperzi. Disperze je řízená texturou znázorňující spektrum barev. Doplněk podporuje simulaci záře okolo zdroje světla způsobenou tvarem kruhových i anamorfních čoček.

Doplněk je schopný generovat vizuálně zajímavé výsledky v řádu desítek milisekund. Je použitelný i na systémech bez dedikované grafické karty. Podporuje *Blender* verze 2.83 a novější. Doplněk je veřejně dostupný ke stažení z internetu.

Práce mě naučila prototypovat a psát složitější *GLSL shadery* tvořící některé efekty. Dále jsem se naučil psát doplňky do *Blenderu* rozšiřující již existující funkcionalitu.

V budoucnu bych chtěl po opravě známých nedostatků dále simulovat zkreslení generovaných artefaktů způsobené tvarem čoček. Dále bych chtěl modulovat vykreslené artefakty texturou a simulovat tím např. škrábance a prach na čočce. Jako další rozšíření by bylo zajímavé oddělit vykreslovací jádro od doplňku a mít jej samostatně jako knihovnu, aby efekt bylo možné použít i mimo *Blender*.

Literatura

[1] *Blender 2.91.0 Python API Documentation — Blender Python API.* [cit. 2021-01-18]. Dostupné z: https://docs.blender.org/api/current/index.html.

[2] *Blender Lens Flare Generator add-on in Realtime - Flared add-on.* [cit. 2021-01-18]. Dostupné z: https://www.blenderlensflare.com/.

[3] *Rendering Pipeline Overview - OpenGL Wiki.* [cit. 2021-05-09]. Dostupné z: https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview.

[4] *Understanding Anamorphic Lenses.* [cit. 2021-01-21]. Dostupné z: https://www.red.com/red-101/anamorphic-lenses.

[5] *Understanding Camera Lens Flare.* [cit. 2021-01-21]. Dostupné z: https://web.archive.org/web/20210112210430/https://www.cambridgeincolour.com/tutorials/lens-flare.htm.

[6] Hullin, M., Eisemann, E., Seidel, H.-P. a Lee, S. Physically-Based Real-Time Lens Flare Rendering. In: *ACM SIGGRAPH 2011 Papers.* New York, NY, USA: Association for Computing Machinery, 2011. SIGGRAPH '11. DOI: 10.1145/1964921.1965003. ISBN 9781450309431. Dostupné z: https://doi.org/10.1145/1964921.1965003.

[7] Johnson, M. K. a Farid, H. Exposing digital forgeries through chromatic aberration. In: *Proceedings of the 8th workshop on Multimedia and security.* 2006, s. 48–55.

[8] Keshmirian, A. *A physically-based approach for lens flare simulation.* 2008. Disertační práce. UC San Diego.

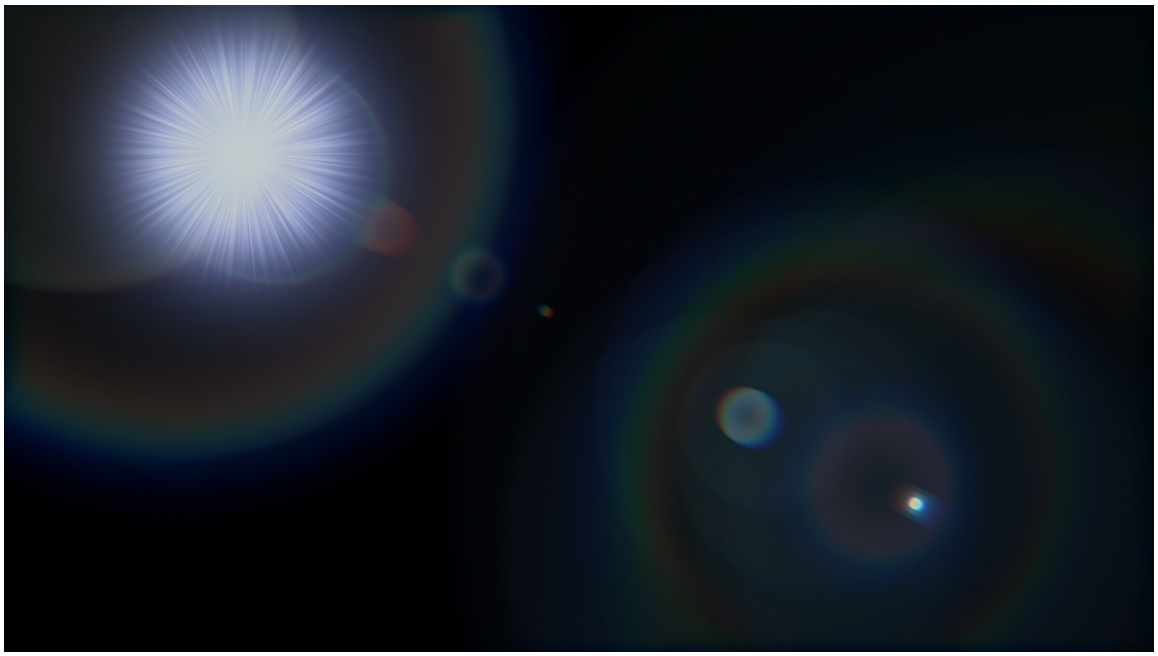
[9] Lee, S. a Eisemann, E. Practical real-time lens-flare rendering. In: Wiley Online Library. *Computer Graphics Forum.* 2013, sv. 32, č. 4, s. 1–6.

[10] Pekkarinen, E. a Balzer, M. Physically Based Lens Flare Rendering in "The Lego Movie 2". In: *Proceedings of the 2019 Digital Production Symposium.* New York, NY, USA: Association for Computing Machinery, 2019. DigiPro '19. DOI: 10.1145/3329715.3338881. ISBN 9781450367998. Dostupné z: https://doi.org/10.1145/3329715.3338881.

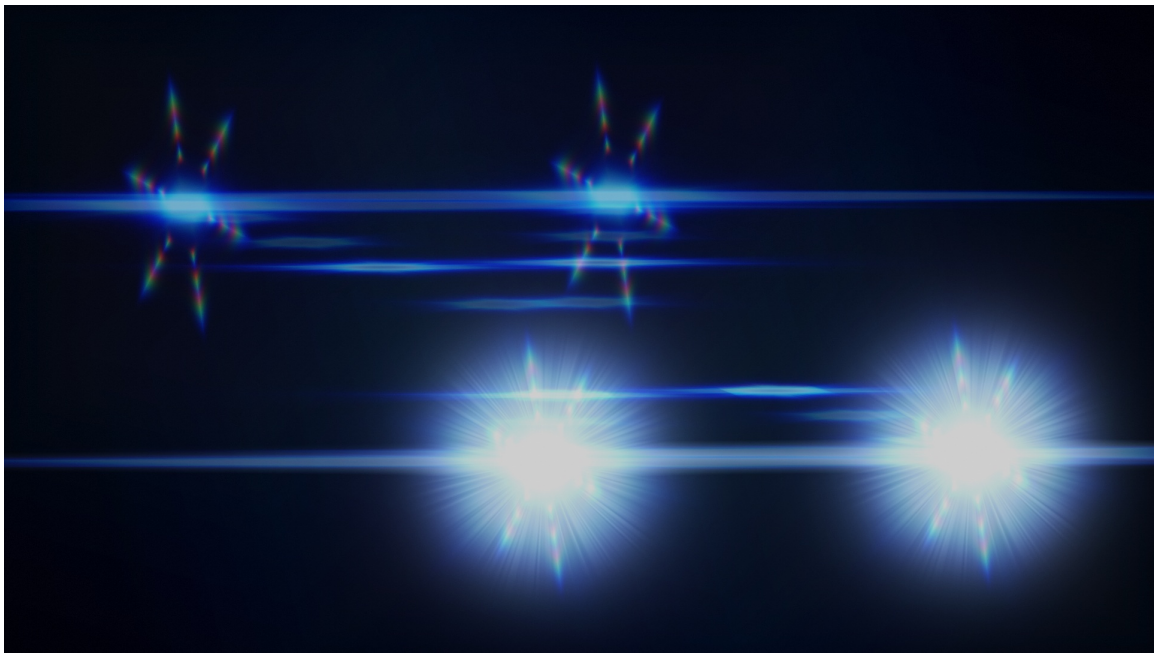
[11] Purse, L. *Digital Imaging in Popular Cinema.* Edinburgh University Press, 2013. ISBN 9780748675623. Dostupné z: https://books.google.cz/books?id=RbVvAAAAQBAJ.

Příloha A

Obrázky s vykresleným efektem

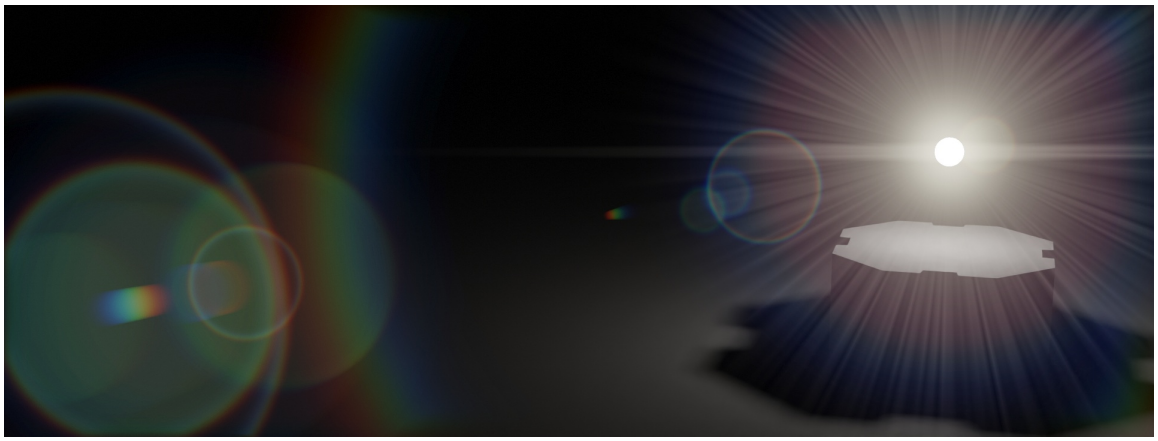


Obrázek A.1: Efekt s kruhovou závěrkou, normální čočkou a duhovými kruhy.



Obrázek A.2: Efekt s šestiúhelníkovou závěrkou, anamorfní čočkou a částečnými duhovými kruhy. Efekt obsahuje dva zdroje světla. Lze jej tak použít např. na světla auta.

Obrázek A.3: Efekt s osmiúhelníkovou závěrkou, normální čočkou a částečnými duhovými kruhy.



Obrázek A.4: Testová scéna pro profilování efektu. Pozadí je vykresleno přímo v *Blenderu* a efekt je přidán pomocí kompozitoru. Efekt obsahuje 24 *ghostů* s různými parametry.

Obrázek A.5: Scéna s jedoucím autem. Pozadí je vykresleno přímo v *Blenderu* a efekt je přidán pomocí kompozitoru. Textury použité ve scéně jsou z https://hdrihaven.com/ a https://www.textures.com.

Obrázek A.6: Ukázka kompozice do fotky. Foto vlastní, upraveno.

Příloha B

Obsah paměťového média

Paměťové médium obsahuje následující soubory:

- `video.mp4`, kde jsou výsledky práce v animované formě
- `thesis.pdf`, což je tento dokument v elektronické podobě
- `doc.pdf`, který obsahuje návod k instalaci a rychlou příručku
- `lensflaregen.zip`, který obsahuje doplněk ve formě archivu připraveného k instalaci do *Blenderu*

Dále jsou obsažené tyto složky:

- `flares`, kde jsou předpřipravené efekty k nahlédnutí
- `lensflaregen`, kde jsou zdrojové texty programu
- `bc-thesis`, kde je tento dokument ve zdrojové formě