Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Videomapping v Unity

Plzeň 2016 Tomáš Chmelík

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 9. června 2016

Tomáš Chmelík

Abstract

Videomapping is a subject that requires relatively large amount of completed resources for it to succeed. This works is concentrated on creation of these resources. Game engine Unity 3D will be used as a base for this project. Using of this engine will simplify creation of said resources and their subsequent usage.

Abstrakt

Videomapping je téma, které vyžaduje poměrně veliký počet hotových prostředků pro úspěšné provedení. Tato práce se zabývá vytvořením těchto prostředků, které jsou nutné pro úspěšný videomapping. Práce bude vytvářena pomocí herního enginu Unity 3D, který umožní snadné vytváření těchto prostředků a následně i jejich použití.

Obsah

1 Úvod

Videomapping je typ rozšířené reality. Projektory místo promítání na obvyklé projekční plátna promítají na neobvyklé, nepravidelné objekty. Z objektů jako například fasády domů, rozprášených vodních kapek, nebo soch se stanou projekční plátna. Videomapping vyžaduje podrobnou znalost fyzické scény, na kterou se bude promítat. Tato znalost se využívá k promítaní buď nových textur objektů nebo virtuální scény, jejíž objekty mohou vystupovat do prostoru. První známé použití videomappingu je v roce 1969 v Disneylandu, kde se promítalo na sochy pro iluzi pohybu.

Oblast použití videomappingu se od prvního použití rozšířila a metody jeho provádění se zlepšily. V dnešní době existují firmy, které zprostředkují videomapping vysoké kvality. Promítání virtuálních objektů, které vystupují do prostoru, doprovázené hudbou se pro nového diváka stanou nezapomenutelnou zkušeností. Prostředky pro vytvoření projektu menšího měřítka jsou stále méně dostupné a projekt se většinou musí vytvořit z velkých částí od znova.

Cílem této práce bylo usnadnit přípravu projektu pro videomapping. Vytvořit nástroje, které umožní jednoduché vytvoření jednotlivých scén, nastavení parametrů projektorů a provedou uživatele procesem kalibrace. Práce bude provedena v herním enginu Unity 3D.

2 Rozšířená realita

Jak bylo řečeno videomapping je jedním z typů rozšířené reality. Na stupnici smíšené reality je rozšířená realita před půlkou mezi fyzickým světem a virtuálním světem [6]. Rozšířená realita nějakým způsobem rozšiřuje fyzický svět o virtuální objekty. Rozšířená realita ale nemusí být jenom o přidávání virtuálních objektů do scény, ale i odebírání fyzických objektů ze scény. To může být provedeno překreslením objektů virtuálním pozadím.

Rozšířená realita není omezena pouze na vizuální podněty. Například s použitím sluchátek, které mají na vnějšku mikrofony, je možné přidávat syntetizované 3D zvuky.

Rozšířená realita má rozsáhlé užití. Například umění, sport, počítačové hry, vojenství, lékařství a výuka jsou oblasti, které používají rozšířenou realitu pro předání větší množství užitečných informací a poskytují vysokou úroveň interakce.

2.1 Používané přístupy

Systémy rozšířené reality se rozdělují na dvě skupiny podle přístupu jakým je docíleno spojení fyzického a virtuálního světa.

2.1.1 Přímé metody

Přímé metody umožňují uživateli sledovat fyzickou scénu přímo. Virtuální objekty jsou přidány nějakým průhledným způsobem. Jako příklad tohoto přístupu jsou Google Glasses nebo podobné produkty. Virtuální objekty jsou přidány promítáním na nějaký průhledný materiál. Když je rozšířená realita vypnutá, uživatel stále vidí fyzickou scénu.

2.1.2 Nepřímé metody

Nepřímé metody rozšířené reality úplně zabraňují uživateli sledovat fyzickou realitu. Fyzická realita je snímaná kamerou, zpracovaná počítačem, kde jsou přidány virtuální objekty, a poté zobrazena uživateli. Tento přístup se blíží více k virtuální realitě, protože odděluje uživatele od fyzické reality. Místo promítání čistě virtuální scény je promítána i fyzická scéna, takže se přístup vrací k rozšířené realitě.

2.2 Porovnání přístupů

Oba přístupy mají své výhody a své nevýhody. Nejdříve jsou vysvětleny výhody přímých metod oproti nepřímým.

1. Rozlišení - Fyzická scéna přímých metod není promítaná zařízením a proto není omezená rozlišením zařízení zobrazující virtuální scénu.

2. Bezpečnost - Přímé metody neodstřihávají uživatele od fyzické reality a když je systém neaktivní, uživatel stále vidí fyzickou scénu.

3. Pozice očí - Nepřímé metody snímají fyzickou scénu kamerou. Tato kamera nemůže být přesně na místě očí uživatele. Uživateli tedy přijde, že má oči umístěné na jiném místě. Další problém může vytvářet rozdíl rozteče očí uživatele a kamer.

Nepřímé metody mají následující výhody.

1. Větší kontrola - Zpracování i fyzického obrazu počítačem umožňuje aplikovat různé filtry na obraz fyzické scény, které by při použití přímých metod nebylo možné použít. Některé přímé metody také trpí tím, že virtuální objekty jsou zobrazeny z části průhledné, což je možné napravit zpracováním počítačem.

2. Odezva - Přímé metody většinou trpí odezvou mezi změnou pozice nebo orientace uživatele a pohybem virtuálních objektů. Nepřímé metody umožňují odezvy virtuální a fyzické scény sjednotit. Stále bude existovat odezva mezi pohybem uživatele a pohybem scény, ale virtuální a fyzické objekty se budou pohybovat vůči sobě.

2.3 Problémy rozšířené reality

Hlavním problém rozšířené reality je geometrická registrace. Fyzické a virtuální scény musejí být zarovnány vzhledem k sobě, aby iluze sloučení scén fungovala. V opačném případě by virtuální scéna klouzala po fyzické scéně nebo by na sebe objekty nepasovaly. Při použití dostatečně přesného modelu fyzické scény by správně provedená geometrická registrace překryla každý fyzický objekt jejím virtuálním ekvivalentem.

Tento problém je navíc názornější pro přímé metody rozšířené reality, kde nejde upravit obraz fyzické scény, zatímco u nepřímých metod je možné obraz zpracovat počítačem a opravit menší chyby. Pokud uživatel sleduje fyzický objekt a jeho virtuální ekvivalent přímou metodou, lidské oko dokáže rozeznat i velice malé chyby registrace.

2.4 Videomapping

Videomapping se řadí mezi přímé metody rozšířené reality. Používají se projektory pro promítání virtuální scény na fyzické objekty. Pro úspěšný videomapping je potřeba znát přesné parametry všech použitých projektorů a zároveň mít přesný model fyzické scény. Bez obou bude geometrická registrace nepřesná a videomapping nepřesvědčivý.

Projektory vyžadují temné prostředí, aby byl kontrast mezi osvětlenými a neosvětlenými oblastmi co největší. Fyzické objekty, které mají být vidět tedy musejí být osvětleny bílou barvou. To může být provedeno vytvořením kopie fyzických objektů ve virtuální scéně s bílou barvou. Zde je důležité aby registrace byla dostatečně přesná, aby virtuální objekty a jejich fyzické ekvivalenty byly na stejném místě.

Videomapping, kde všechny virtuální objekty leží na stejném místě jako jejich fyzické ekvivalenty, může být sledován z libovolné pozice bez zkreslení. Takto se vytváří jeden z hlavních efektů videomappingu a to texturování fyzických objektů. Když virtuální objekty neleží na místě fyzických objektů, promítaný obraz bude zkreslený mimo místo správné perspektivy. Tato iluze perspektivy je druhý hlavní efekt videomappingu.

Pro videomapping, který podporuje pouze texturování objektů, stačí aby projektory promítaly přímo jejich pohled do virtuální scény. Pokud má být podporována i iluze perspektivy, obrazy projektorů musejí být vypočítané. Virtuální scéna také tehdy musí obsahovat nový objekt označující pozici uživatele. Z této pozice ve fyzickém světě nebude promítaná scéna zkreslená.

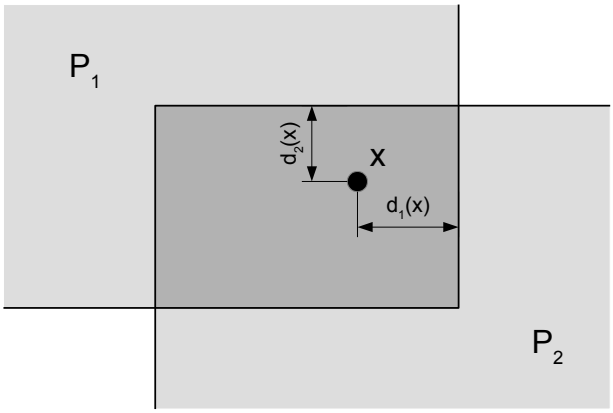
Jednou z možností výpočtu obrazů projektorů je modelovat fyzické objekty ve virtuální scéně. Tyto objekty jsou v tomto dokumentu pojmenovány jako *reálné*. Virtuální scéna je geometricky promítána na reálnou scénu. Takto se zjistí jak má být fyzické scéna osvícena, aby virtuální objekty nebyly zkreslené z pozice uživatele. Projektory tedy promítají jejich pohled na reálnou scénu.

2.4.1 Problémy videomappingu

Pro úspěšný videomapping je potřeba překonat několik problémů.

Kalibrace projektorů

Z problému geometrické registrace se při videomappingu objevuje problém zjištění parametrů projektorů. Pro přesné splynutí virtuálních a fyzických



Obrázek 2.1: Veličiny používané pro výpočet osvětlovacích map pro dva projektory.

objektů je potřeba, aby projektory promítaly na přesně určenou pozici. Tento problém se nazývá kalibrace projektorů a zabývá se jím kapitola 4.

Registrace projektorů

Další problém vycházející z geometrické registrace je problém s překryvy projektorů. Místa, které jsou v zorných polí více než jednoho projektoru, by byly více osvětlené a byly by tak jasně rozeznatelné. Pro dosáhnutí rovnoměrného osvětlení se musí intenzita osvětlení pixelů snížit na překryvu jednotlivých projektorů.

Budeme-li uvažovat pixel x na překryvu několika projektorů označených P_i (viz obrázek 2.1 pro příklad s 2 projektory). Intenzity, které jednotlivé projektory přispívají k celkovému osvětlení, se označí $\lambda_i(x)$. Aby osvětlení překryvu bylo rovnoměrné, musí platit $\sum_i \lambda_i(x) = 1$. Hodnoty $\lambda_i(x)$ se vypočítají podle vzorce (2.1), kde $d_i(x)$ označují vzdálenost pixelu x od nejbližší hrany průmětny projektoru P_i. Z jednotlivých hodnot λ_i projektoru P_i se vytvoří mapa snížení osvětlení LAM (Luminosity Attenuation Map). Úkol registrace projektorů je vypočítat pro každý projektor jejich LAM.

$$\lambda_i(x) = \frac{d_i(x)}{\sum_j d_j(x)} \tag{2.1}$$

Vypočítané LAM jsou závislé na vzájemné pozici fyzických objektů a projektorů. Pokud jsou pozice projektorů statické a fyzická scéna se nepohybuje, LAM je konstantní. V opačném případě se musí LAM neustále přepočítávat, což je časově náročné. Pozice virtuálních objektů a pozice správné perspektivy tímto nejsou zatížené a můžou se během videomappingu volně pohybovat.

Texturování objektů

Pro jednoduché texturování objektů je potřeba, aby fyzické objekty, které se budou texturovat, měly bílou barvu. Pokud objekty mají jinou barvu, nastává problém s jejich osvětlením. Tento problém vychází z definice barvy objektů. Červený objekt absorbuje všechny nečervené paprsky světla a červené světlo rozptyluje. Když se tedy osvětlí červený objekt zelenou barvou, objekt by měl být černý, protože všechno světlo je absorbováno.

Nechť fyzický objekt má po celém povrchu barvu v barevném systému RGB s hodnotami (r_f, g_f, b_f). Tyto hodnoty omezují barvy, které je možné na tento objekt promítat. Nová textura může mít RGB barvu (r_t, g_t, b_t) s hodnotami podle vzorce (2.2).

$$\begin{aligned} r_t &\in \langle 0, r_f \rangle \\ g_t &\in \langle 0, g_f \rangle \\ b_t &\in \langle 0, b_f \rangle \end{aligned} \tag{2.2}$$

S tímto problémem se setkává i u schovávání fyzických objektů. Pokud by měl být obraz virtuálního objektu promítán na fyzický objekt, který má být schován, musí se brát v potaz barva fyzického objektu pro výpočet promítané barvy také podle vzorce (2.2). Podle kombinace barvy fyzického a virtuálního objektu nemusí být možné vytvořit přesvědčivou iluzi schování fyzického objektu.

3 Model projektoru

Projektory je možné modelovat jako invertované kamery, které místo přijímání světla ho vydávají. Pinhole camera model tedy platí pro kamery i pro projektory. Pinhole camera model se zabývá projekcí bodů ve 3D prostoru na 2D průmětnu kamery [10]. Aby bylo možné provádět translaci vynásobením maticí, budou se používat homogenní vektory.

3.1 Souřadné systémy

Nejdříve je potřeba rozlišit jednotlivé souřadné systémy, které se v modelu pinhole camera používají.

Souřadný systém světa

Tento souřadný systém označuje polohu těles a kamery ve 3D prostoru. Považuje se za základní souřadný systém. Ve zbytku dokumentu bude označen spodním indexem w (souřadný systém S_w, osy x_w, y_w, z_w).

Souřadný systém kamery

Třídimenzionální souřadný systém s počátkem v centru kamery. Kladná osa X směřuje vpravo z pohledu kamery, osa Y směřuje nahoru a osa Z směřuje dopředu. Označen spodním indexem c (systém S_c, osy x_c, y_c, z_c).

Souřadný systém obrazu

Dvojdimenzionální souřadný systém. Označuje pozici obrazů objektů na průmětně. Pozice bodů se získá projektivní transformací bodů ze souřadného systému kamery. Označen spodním indexem s (systém S_s, osy x_s, y_s).

3.2 Projekce

Pro získání souřadnic bodů v souřadném systému obrazu musí být body převedeny do kamerového souřadného systému a poté promítnuty projektivní perspektivní transformací na průmětnu do souřadného systému obrazu.

3.2.1 Převod mezi souřadnými systémy

Zjistit pozici bodů jednoho souřadného systému ve druhém souřadném systému je poměrně jednoduché. Pro převod ze soustavy S_1 do soustavy S_2 je potřeba znát pozici počátku soustavy S_1 v souřadnicích soustavy S_2. Tento vektor má tři složky a pojmenujeme ho $\mathbf{T}_{21}$. Dále je potřeba znát rotační matici ze soustavy S_2 do soustavy S_1 a nazveme jí R_{21}. Tato matice má rozměry 3×3 a obsahuje promítnuté jednotkové vektory os prvního souřadného systému z pohledu druhého souřadného systému. Pro převedení bodů ze soustavy S_1 do soustavy S_2 stačí použít rovnici (3.1) [5].

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} I & \mathbf{T}_{21} \\ \mathbf{0}_3^T & 1 \end{bmatrix}}_{\text{Translace}} \times \underbrace{\begin{bmatrix} R_{21} & \mathbf{0}_3 \\ \mathbf{0}_3^T & 1 \end{bmatrix}}_{\text{Rotace}} \times \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} R_{21} & \mathbf{T}_{21} \\ \mathbf{0}_3^T & 1 \end{bmatrix}}_{M_{12}} \times \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} \tag{3.1}$$

Tento přístup je ale nepřirozený, protože pro vytvoření matice M_{12} pro převod ze soustavy S_1 do S_2 jsou potřeba hodnoty měřené v opačném směru ($\mathbf{T}_{21}$ a R_{21}). Proces převodu ze soustavy S_1 do S_2 je inverzní oproti procesu převodu na opačnou stranu. Matice M_{12} a M_{21} jsou tedy inverzní. Je tedy možné použít více pochopitelné hodnoty $\mathbf{T}_{12}$ a R_{12} v matici M_{21} pro odvození matice M_{12}, pokud jsme schopni vyjádřit inverzi matice M_{21}.

V rovnici (3.1) je vidět, jak jde rozložit matici M_{12} na matici translace a rotace. Translační matice je speciální typ matice: horní atomická trojúhelníková matice (jednotková diagonála, jeden sloupec může obsahovat nenulové hodnoty nad diagonálou, ostatní hodnoty nulové). Inverze této matice se rovná také horně atomické trojúhelníkové matici s opačnými (negativními) hodnotami ve vyplněném sloupci [8]. Rotační matice je podle její definice ortogonální a proto pro ní platí $R_{21}^T = R_{21}^{-1}$. Použitím znalosti vlastností těchto matic je možné odvodit definici M_{12} s použitím $\mathbf{T}_{12}$ a R_{12} (viz rovnice (3.2)).

$$\begin{aligned} M_{12} = \begin{bmatrix} R_{21} & \mathbf{T}_{21} \\ \mathbf{0}_3^T & 1 \end{bmatrix} = M_{21}^{-1} &= \begin{bmatrix} R_{12} & \mathbf{T}_{12} \\ \mathbf{0}_3^T & 1 \end{bmatrix}^{-1} \\ &= \begin{bmatrix} R_{12} & \mathbf{0}_3 \\ \mathbf{0}_3^T & 1 \end{bmatrix}^{-1} \times \begin{bmatrix} I & \mathbf{T}_{12} \\ \mathbf{0}_3^T & 1 \end{bmatrix}^{-1} \\ &= \begin{bmatrix} R_{12}^T & \mathbf{0}_3 \\ \mathbf{0}_3^T & 1 \end{bmatrix} \times \begin{bmatrix} I & -\mathbf{T}_{12} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \\ &= \begin{bmatrix} R_{12}^T & -R_{12}^T \mathbf{T}_{12} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \end{aligned} \tag{3.2}$$

Z této rovnice je poté možné odvodit vztahy (3.3) pro převod mezi rotačními maticemi a translačními vektory dvou souřadných systémů.

$$R_{21} = R_{12}^T \tag{3.3a}$$

$$\mathbf{T}_{21} = -R_{21}\mathbf{T}_{12} \tag{3.3b}$$

Sjednocením rovnic (3.1) a (3.3) je poté možné vytvořit více pochopitelnou rovnici (3.4) pro převod ze soustavy S_1 do S_2.

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} R_{12}^T & -R_{12}^T\mathbf{T}_{12} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} \tag{3.4}$$

3.2.2 Projektivní transformace

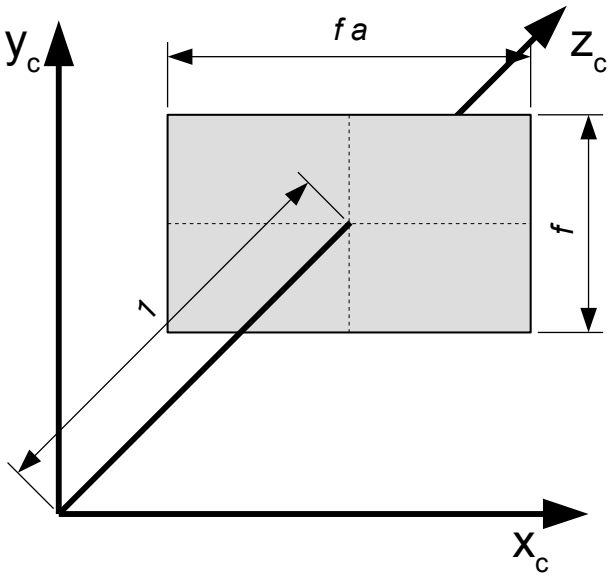
Pro model pinhole camera budeme uvažovat, že se jedná o perspektivní projekci s bodem projekce v počátku souřadném systému kamery.

Průmětna kamery je rovina v souřadném prostoru kamery na kterou se promítají obrazy bodů 3D prostoru. Tato rovina je v našem případě rovnoběžná s osami x_c a y_c. Její vzdálenost na ose z_c bude pro jednodušší výpočty vymezena na 1. Velikost průmětny není teoreticky omezená, ale parametry kamery jako ohnisková vzdálenost (f) a poměr stran (a) vymezují velikost obdélníku na průmětně, který je zorným polem kamery (viz obrázek 3.1). Vektor vyosení kamery ($\mathbf{o}$) tímto obdélníkem potom posouvá (viz obrázek 3.2) v rovině průmětny.

Body musí být převedeny do souřadného systému kamery před promítáním na průmětnu. Rovnice (3.5) poté převede body ze souřadného systému kamery do souřadného systému obrazu.

$$\begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{fa} & 0 & o_x \\ 0 & \frac{2}{f} & o_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \tag{3.5}$$

Tato rovnice převede body v zorném poli kamery do intervalu $(-1, 1) \times (-1, 1)$ v souřadném systému obrazu. Body mimo tento interval tedy nejsou viditelné kamerou s těmito parametry. Problém vytvářejí body za průmětnou ($z_c < 1$), protože se také promítnout na průmětnu, ale tyto body nemají být viditelné. Je proto potřeba sledovat souřadnice z_c a brát v potaz pouze body s vyšší hodnotou než 1 (body před průmětnou).



Obrázek 3.1: Pozice a velikost průmětny ve 3D prostoru kamery.

Zorný úhel

Z ohniskové vzdálenosti (f) a poměru stran (a) je možné vypočítat zorné úhly kamery. Z obrázku 3.1 je možné vytvořit vzorce (3.6) pro převod mezi vertikálními a horizontálními zornými úhly a parametry kamery (f, a).

$$fov_v = 2 \arctan\left(\frac{f}{2}\right) \tag{3.6a}$$

$$fov_h = 2 \arctan\left(\frac{fa}{2}\right) \tag{3.6b}$$

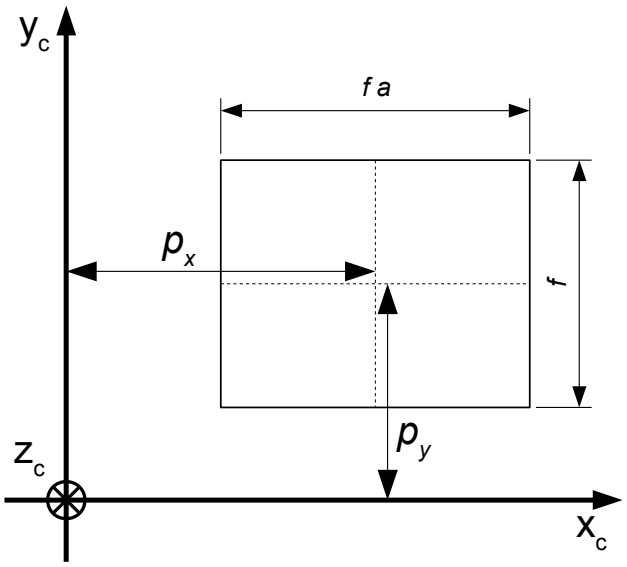
Vyosení

Vektor (p_x, p_y) označuje pozici středu průmětny v souřadném systému kamery 3.2. Převod mezi vektorem $\mathbf{p}$ a $\mathbf{o}$, použitým v matici projekce, je uveden v rovnici (3.7).

$$\begin{bmatrix} o_x \\ o_y \end{bmatrix} = - \begin{bmatrix} \frac{2}{fa} & 0 \\ 0 & \frac{2}{f} \end{bmatrix} \times \begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{3.7}$$

3.2.3 Vše dohromady

Použitím rovnic (3.3), (3.4) a (3.5) je možné vytvořit rovnici (3.8) pro převod bodů ze světového souřadného systému do souřadného systému obrazu [5]. Tato rovnice obsahuje 5 parametrů kamery.



Obrázek 3.2: Zobrazení vyosení kamery posuvem průmětny.

- **T** - pozice počátku souřadného systému S_c z pohledu souřadného systému S_w (pozice kamery v S_w)
- R - rotační matice převodu ze souřadného systému S_w do S_c (rotace kamery vůči S_w)
- f - ohnisková vzdálenost kamery
- a - poměr stran kamery
- **o** - vektor vyosení kamery

$$\begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \underbrace{\overbrace{\begin{bmatrix} I_2 & \mathbf{o} \\ \mathbf{0}_2^T & 1 \end{bmatrix}}^{\text{2D translace}} \times \overbrace{\begin{bmatrix} \frac{2}{fa} & 0 & 0 \\ 0 & \frac{2}{f} & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{\text{2D škálování}}}_{\text{vnitřní parametry}} \times \underbrace{\overbrace{\begin{bmatrix} I_3 & \mathbf{T} \end{bmatrix}}^{\text{3D translace}} \times \overbrace{\begin{bmatrix} R & \mathbf{0}_3 \\ \mathbf{0}_3^T & 1 \end{bmatrix}}^{\text{3D rotace}}}_{\text{vnější parametry}} \times \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{3.8}$$

Rodgiuova rotační rovnice

Pro jednodušší počítáni s rotační maticí R jde rotační matice vyjádřit jako třísložkový vektor pomocí Rodgiuovo rotační rovnice [1]. Složky vektoru **R** označují osu, kolem které se bude rotovat, zatímco velikost vektoru **R** označuje o kolik radiánů se kolem zadané osy bude rotovat. Takto se zmenší počet skalárních parametrů kamery na 10.

4 Kalibrace projektorů

Kalibrace projektorů je proces zjištění přesných parametrů projektorů. Parametry projektoru jako pozice a rotace v prostoru (vnější parametry) a ohnisková vzdálenost, poměr stran, vyosení (vnitřní parametry) jsou během kalibrace zjištěny. Další parametry jako radiální zkreslení je také možné vypočítat, ale tato práce se s nimi nebude zabývat.

Pro úspěšný videomapping je potřeba tyto parametry vědět velice přesně, protože i malá chyba některých parametrů se může znatelně zvětšit, když se promítá na vzdálené plochy. Ruční nastavení těchto parametrů tedy nepřipadá v úvahu a místo toho se nastavují pouze přibližné parametry, aby se proces kalibrace zrychlil a usnadnil.

Existuje několik metod pro kalibraci kamer. Kamery poskytují vizuální výstup jejich zorného pole, který jde použít pro kalibraci. Projektory tuto možnost nemají. Většinou se proto používají kamery společně s projektory pro kalibraci projektorů.

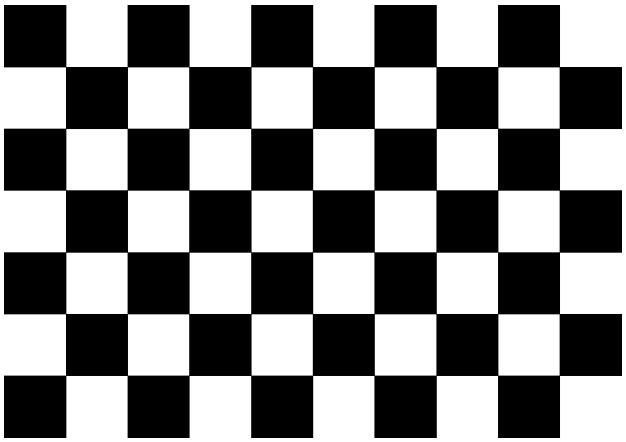
Metody se od sebe liší mírou spolupráce se člověkem, složitostí scény na kterou se bude promítat (od jedné plochy až po libovolnou scénu), zapojením kamery do kalibračního procesu atd.

4.1 Kalibrace detekcí scény

Metody kalibrace detekcí scény nepotřebují žádnou znalost scény. Během procesu kalibrace je fyzická scéna naskenovaná a vytvořen její model. Tento přístup vyžaduje, aby zorné pole každého projektoru bylo snímáno dvěma kamerami. Tyto kamery jsou používány pro odhadnutí modelu scény. Samotný proces kalibrace projektorů je rozdělen do tří kroků. Touto metodou se zabývá článek [3].

Kalibrace kamer

V prvním kroku je potřeba kalibrovat kamery. To je obvykle prováděno detekcí kalibračních vzorů. Vyhledáním a rozeznáním kalibračního vzoru v obrazu kamery se zabývá oblast počítačového vidění. Pro úspěšné nalezení kalibračního vzoru musí být celý v zorném poli kamery a musí být předem známá jeho geometrie. Kalibrační vzor je obvykle asymetrický, aby šla zjistit jeho orientace (viz obrázek 4.1). Během hledání kalibračního vzoru se detekují hrany mezi jednotlivými čtverci a kde se vertikální a horizontální hrany



Obrázek 4.1: Standardní kalibrační vzor.

protínají vznikají body kalibračního vzoru.

Nejvíce používané kalibrační vzory jsou 2D, ale je možné použít i 3D vzor. Pro úspěšnou kalibraci všech parametrů kamery je dokonce nutné použít kalibrační vzor, který nemá všechny body v rovině rovnoběžné s průmětnou. Parametry pozice kamery a ohniskové vzdálenosti v tomto případě nejde zjistit, protože posunutím kamery dopředu a zároveň snížením ohniskové vzdálenosti (zoom out) je možné promítnout kalibrační vzor na přesně stejnou pozici (efekt známý jako Dolly zoom). 2D kalibrační vzory je stále možné použít pro kalibraci, ale musejí být různě nakloněné, aby neležely rovnoběžně k průmětně.

Během kalibrace kamery není potřeba zjistit přímo parametry kamery, ale stačí pouze zjistit její matici (výsledek vynásobení jednotlivých matic v rovnici (3.8)). Tato výsledná matice má rozměry 3×4 s jedničkou v pravém spodním rohu. Během kalibrace je tedy potřeba zjistit 11 parametrů. Na to je potřeba alespoň 6 bodů v prostoru a jejich pozice na průmětně kamery, aby bylo možné kalibrovat kameru. Samotné zjištění parametrů je prováděno pomocí metody nejmenších čtverců.

Odhadnutí vzhledu scény

Druhý krok používá více kalibrovaných kamer pro odhadnutí scény. Scéna je postupně osvětlená projektory a kamery snímají obrazy na fyzických objektech. Vyhledáním korespondujících bodů v průmětnách více kamer je možné zjistit pozici bodu ve 3D prostoru. Takto se zjistí pro každý pixel projektoru jaký bod prostoru osvětluje.

Takto je vytvořen bodový model scény zorných polí jednotlivých projektorů. Ten je poté převeden do povrchové sítě pomocí 2D Delaunayho trian-

gulace. Pro spojení jednotlivých povrchových sítí je potřeba, aby se zorné pole projektorů z části překrývaly. Po vytvoření povrchových sítí se zjistí jak na sebe jednotlivé sítě navazují a jsou spojeny do jedné povrchové sítě.

Kalibrace projektorů

V předchozím kroku je zjištěna korespondence mezi body průmětny projektorů a pozicí v prostoru, kterou osvětlují. Tato informace je použita v tomto kroku pro kalibraci projektorů. Jsou vybrány body, které budou použit pro kalibraci pomocí metody nejmenších čtverců jako bylo provedeno u kamer v prvním kroku. Výsledná matice poté jde rozložit do jednotlivých vnitřních a vnějších parametrů. Problém nastává s výběrem použitých bodů, protože může nastat stejný problém s Dolly efektem jako u kamer.

Některé metody neprovádí druhý krok a používají promítaný kalibrační vzor pro kalibraci projektorů. Tehdy je kalibrační vzor detekován kamerami a zjištěna jeho pozice. Z pozice kalibračního vzoru na průmětně projektoru a pozice vzoru v prostoru se vypočítávají parametry projektoru.

Výhody a nevýhody

+ není potřeba interakce uživatele
+ model scény je naskenován během kalibrace
− jsou potřeba kamery
− rozlišení kamer omezuje přesnost kalibrace
− je vyžadován speciální software pro spolupráci kamer a projektorů

4.2 Kalibrace manuální detekcí vzorů

Kalibrace detekcí kalibračního vzoru používá znalost pozic bodů kalibračního vzoru ve 3D prostoru a pozic jejich obrazů na průmětně. Tato metoda používá pro detekci kalibračního vzoru uživatele místo kamer. Kroky kalibrace kamer a detekce scény se tedy přeskočí a rovnou se kalibrují projektory.

Kalibrace je prováděna umístěním kalibračního vzoru do scény. Uživatel určuje správnou pozici bodů kalibračního vzoru na průmětně projektoru. Stále se musí brát v potaz Dolly efekt při výběru kalibračního vzoru. Výsledné dvojice bodů kalibračního vzoru a obrazů na průmětně se použijí pro výpočet parametrů projektoru.

Výhody a nevýhody

+ nejsou potřeba kamery
+ kalibrační vzor nemusí být standardizovaný pro snadnou detekci kamerou
− vyžaduje interakci s uživatelem
− vyžaduje přesný kalibrační vzor
− nevytváří model scény

4.2.1 Kalibrace řešením nelineárních rovnic

Jedním z možných postupů zjištění parametrů kamer vychází ze vztahů mezi body v prostoru a jejich obrazy na průmětně. Vzorec (3.8) obsahuje všech 10 parametrů kamery (o_x, o_y, f, a, T_x, T_y, T_z, R_x, R_y, R_z).

Když se vyjádří vztahy pro x_s a y_s vzniknou dvě velice komplikované nelineární rovnice s 10 neznámými. Pro jeden kalibrační vzor jsou všechny tyto parametry stejné, a proto jdou takto dosadit do těchto rovnic hodnoty x_s, y_s, x_w, y_w a z_w 5 bodů kalibračního vzoru a vznikne tak soustava 10 nelineárních rovnic s 10 neznámými.

Rovnice s vyjádřenými x_s a y_x jsou složité, ale je možné vyjádřit jejich parciální derivace pro všechny neznámé. Takto jde vytvořit Jacobiho matici (J), která je využitá v iterativním výpočtu soustavy nelineárních rovnic Newton-Raphsonovou metodou. Vzorec (4.1) ukazuje jak takový iterativní výpočet bude probíhat, kde $\mathbf{x}_k$ označuje stávající odhad neznámých, $\mathbf{x}_{k+1}$ je odhad hodnot další iterace a f je vektorová funkce soustavy nelineárních rovnic [9].

$$\mathbf{x}_{k+1} = \mathbf{x}_k - J(\mathbf{x}_k)^{-1} f(\mathbf{x}_k) \tag{4.1}$$

I když je možné vyjádřit jednotlivé parciální derivace a poté invertovat Jacobiho matici, řešení se ukázalo jako příliš pomalé a náročné na implementaci.

4.2.2 OpenCV

OpenCV neboli Open Computer Vision je největší volně dostupná knihovna zabývající se počítačovým viděním. OpenCV obsahuje veliký počet modulů, například modul pro práci s obrázky, zpracování videa, rozpoznávání tváří. OpenCV také obsahuje modul pro detekci objektů a kalibraci kamer.

Knihovna OpenCV je implementována v jazyku C a pozdější moduly v jazyku C++. Dále jsou poskytnuty wrappery pro jazyky Python a Java. Knihovna je podporovaná na desktop platformách (Windows, Linux, MacOS) i na mobilních zařízeních (Android, iOS).

V této práci by byl použit pouze modul *calib*3*d*, který se zabývá kalibrací kamer. Tento modul poskytuje například funkce pro detekci kalibračního vzoru ale hlavně i funkci pro kalibraci kamer.

Následující funkce pro kalibraci kamery umožňuje přijmout hned několik kalibračních vzorů společně s jejich obrazy na průmětně. Více kalibračních vzorů je použito pro přesnější vypočítání vnitřních parametrů kamery, protože ty musejí být stejné pro všechny vstupy. Poté jsou vypočítány vnější parametry pro každý kalibrační vzor [2].

```
double cv::calibrateCamera(objectPoints, imagePoints,
            imageSize, cameraMatrix,  distCoeffs, rvecs,
            tvecs, flags, criteria)
```

- objectPoints (IN) - seznam kalibračních vzorů, každý vzor obsahuje seznam pozic kalibračních bodů
- imagePoints (IN) - seznam obrazů kalibračních vzorů na průmětně
- imageSize (IN) - velikost průmětny v pixelech
- cameraMatrix (IN/OUT) - odhad matice kamery, použit jenom při speciálním nastavení parametrů $flags$, po kalibraci vrací vypočítané hodnoty matice kamery (obsahuje poměr stran, ohniskovou vzdálenost a vyosení)
- distCoeffs (IN/OUT) - parametry radiálního zkreslení kamery
- rvecs (OUT) - seznam rotačních vektorů (viz Rodgiuova rotační rovnice) pro jednotlivé kalibrační vzory
- tvecs (OUT) - seznam translačních vektorů pro jednotlivé kalibrační vzory (translační vektor musí být orotován podle odpovídajícího rotačního vektoru pro získání skutečné pozice kamery)
- flags (IN) - nastavení chodu kalibrace (například napevno nastavit poměr stran, nepočítat radiální zkreslení, atd)
- criteria (IN) - zastavovací podmínky pro kalibraci

OpenCV dále poskytuje funkce pro převod mezi rotační maticí a rotačním vektorem (dle Rodgiuova rotační rovnice) a pro detekci kalibračního vzorů v zorném poli kamery. Vstupy a výstupy těchto funkcí jsou ve formátu, který funkce `calibrateCamera` dokáže zpracovat.

C# Wrapper

Knihovna nabízí wrappery pro několik jazyků, ale Unity 3D pracuje s jazykem C# a wrapper pro tento jazyk OpenCV nedodává. Jsou dostupné různé C# wrappery pro OpenCV, pro tuto práci byl vybrán *OpenCVSharp* [4]. Tento wrapper je dostupný na *NuGet* a je tak možné ho přímo přidat do Unity projektu pomocí vývojového prostředí Visual Studio.

4.3 Vybrané řešení

Pro zjednodušení procesu kalibrace nebudou použity kamery. Pro kalibraci projektorů tedy bude nutná přítomnost uživatele. Uživateli budou postupně zadány kalibrační body v prostoru a jeho úkolem bude určit pozici na průmětně projektoru, kde by měl daný bod být promítán.

Samotné vypočítání parametrů projektorů bude prováděno pomocí knihovny OpenCV. OpenCV bylo vybráno kvůli vysoké rychlosti výpočtu parametrů a protože je to obecně vyzkoušená knihovna. Pro komunikaci mezi OpenCV (C++) a Unity 3D (C#) bude použit wrapper *OpenCVSharp*.

5 Unity 3D

Unity 3D je herní engine dostupný na velikém množství platforem. Umožňuje vytvářet hry pro desktopy, konzole, mobilní zařízení a dokonce i web. Na stránkách Unity jsou dostupné detailní návody pro většinu funkcí Unity, společně s detailní programovou dokumentací a fóry pro spolupráci s komunitou.

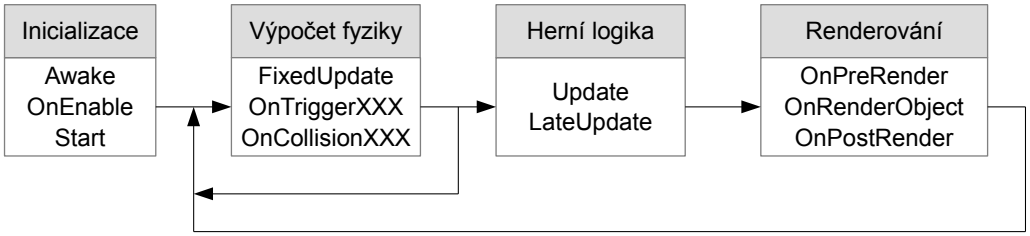
Unity bylo vytvořeno tak, aby používalo speciální prostředky, které každá platforma poskytuje. Různé knihovny pro zobrazení grafického výstupu budou použity na různých platformách. Unity shadery můžou obsahovat více verzí kódu, ze kterých bude vybráno podle dostupného hardwaru.

Samotné Unity je psáno v jazycích C, C++ i C#. Unity umožňuje vytvářet skriptů v jazycích JavaScript a C#. Pro vytváření skriptů v práci byl vybraný jazyk C#.

5.1 Engine

Unity engine pracuje jako diskrétní simulace s proměnným časovým krokem primárně zaměřen na vývoj her. Engine pracuje v uzavřené smyčce (*Unity Execution Loop*) volání metod událostí. Skripty přiřazené objektům scény obsahují metody, které upravují chod simulace.

Skripty, které obsahují metody z obrázku 5.1, budou volány když se herní simulace dostane do daného kroku. Je brán veliký důraz, aby volaná metoda netrvala příliš dlouho, protože Unity volá danou metodu jako normální metodu, a dlouhé trvání přímo zpomalí běh simulace a sníží obnovovací frekvenci programu/hry.



Obrázek 5.1: Zjednodušená výpočetní smyčka Unity (úplná dostupná na [7]). Metody končící XXX značí, že existuje více metod se stejným začátkem.

Unity při startu programu projde všechny načtené skripty a pomocí reflexe hledá metody, které se jmenují stejné jako události v *Unity Execution Loop*. Vstupní body těchto metod jsou zapamatovány pro zrychlení volání těchto metod během každého snímku.

Unity se snaží udržet minimální hodnotu FPS (Frames per second) nad hranicí viditelnou lidským okem. Obvyklé FPS, které se snaží Unity udržet je 60, neboli přibližně 16 milisekund mezi snímky. Jednoduchá scéna vyžaduje méně než 16 milisekund pro výpočet změn mezi snímky. Zbytek času mezi snímky Unity jednoduše čeká, aby zbytečně nezatěžovalo hardware.

5.1.1 Koprogramy

Pro delší nebo asynchronní operace (delší než zbývající čas mezi snímky) je teoreticky možné vytvářet vlastní vlákna, ale všechny objekty vytvořené vláknem Unity jsou chráněné proti zásahu z jiného vlákna. Jednodušší způsob vytváření asynchronních procesů jsou *Coroutines*, neboli koprogramy. Ty využívají speciální metody konstrukce jazyku C# a to iterátory.

Iterátory jsou metody, které vracejí svůj výsledek postupně, jak je vyžadován (*lazy execution*). Návratová hodnota iteračních metody musí být *IEnumerator*, *IEnumerable*, nebo jejich generické varianty a jejich deklarace nesmí obsahovat *out* nebo *ref* parametry. Při procházení výsledků těchto metod (například pomocí *foreach* smyčky) je další prvek vypočítán těsně předtím než je použit. Iterování je možné jenom směrem dopředu, protože iterátor uchovává pouze hodnotu stávajícího iterovaného prvku a místo v kódu, kde se bude pokračovat při další iteraci.

Koprogramy v Unity mají omezení, že musejí vracet negenerickou variantu *IEnumerator*. Metoda koprogramu musí obsahovat volání *return yield*, v opačném případě by metoda proběhla celá v jednom kuse, během jednoho snímku. Volání *return yield* označují místa, kde se výpočet zastaví mezi snímky simulace. V koprogramech je možno vrátit instance dvou tříd:

- *WaitForFixedUpdate* - výpočet bude pokračovat další snímek simulace (pro zkrácení je možné vrátit hodnotu *null*)
- *WaitForSeconds* - výpočet bude pokračovat po uběhnutí zadaného počtu sekund

Nový koprogram se spustí voláním metody *StartCoroutine*, které je předán název metody koprogramu. Každý snímek simulace, během fáze výpočtu herní logiky v *Unity Execution Loop* (mezi *Update* a *LateUpdate*), jsou vyvolány všechny koprogramy, aby vrátily další výsledek. Umístění *return*

yield v koprogramech je tedy velice důležité, protože sám koprogram musí vrátit řízení zpět do Unity.

5.1.2 Scény

Unity scéna je kolekce Unity objektů společně s jejich nastavením. Scény jsou ukládány do souborů s příponou *unity*. Scény, které mají být dostupné za běhu programu, musejí být přidané mezi kompilované scény. Po startu programu se automaticky načte scéna s indexem 0. Poté je možné načítat ostatní scény.

Ve starší verzi Unity bylo možné za běhu načítat scény buď nahražením doposud načtené scény, nebo přidáním všech objektů načítané scény do aktuální scény (aditivní načtení). Po aditivním načtení scény nebylo možné scény rozlyšit a proto ani odnačíst určitou scénu. V editoru nebylo možné nastavovat reference na objekty v jiných scénách.

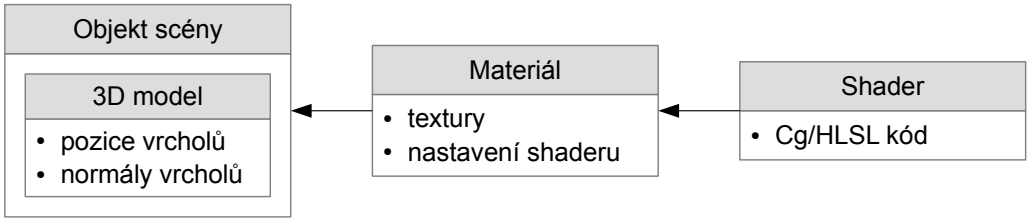
Od verze 5.3 Unity podporuje načítání a pracování s více scénami najednou. Byly přidány nástroje pro snadnější načítání, slučování a odnačítání scén. Editor byl také rozšířen o možnost načtení více scén. Jakmile je načteno více scén, je možné nastavovat reference mezi jednotlivými scénami.

5.1.3 Objekty scény

Každý objekt scény obsahuje jeho jméno, vrstvu, tag a potom může obsahovat libovolný počet komponent. Komponenty jsou skripty, které jdou přiřadit objektům scény. Většinou tyto komponenty obsahují nějaké parametry, které jdou upravovat v editoru, a/nebo metody, které jsou volané z jiných komponent nebo během *Unity Execution Loop* popsané v kapitole 5.1.

Speciální komponentu $Transform$ musí mít každý objekt. Tato komponenta obsahuje pozici, rotaci a velikost objektu. Pro snadné upravování pozice obsahuje metody jako posunutí podél vektoru (relativně k sobě nebo ke světu). Po upravení rotace má metody pro rotaci kolem libovolné osy nebo nasměrování objektu na zadaný bod. Dále jsou dostupné metody pro převod bodu z relativních souřadnic do absolutních a naopak. Nakonec $Transform$ také obsahuje informace o hierarchii scény: referenci na předka a seznam potomků společně s metodami na úpravu této hierarchie.

Vrstva objektu určuje skupinu, do které objekt patří. Tyto skupiny jde poté prohledávat. Jedno využití vrstev je u kamer, kde jde nastavit jakou vrstvy objektů bude kamera snímat. Ostatní objekty jsou pro kameru neviditelné. Dále se vrstvy používají během *Ray Tracingu*, kde se nastavuje,



Obrázek 5.2: Vztah mezi objekty scény, materiály a shadery.

které vrstvy se mají brát v potaz při výpočtu kolizí. Podobnou roli rozdělení do skupin mají tagy. Tagy mají jen jiné využití, jako například nalezení všech objektů ve scéně s určitým tagem.

5.1.4 Materiály, shadery a textury

Materiály, shadery a textury jsou použity v Unity pro výpočet grafického výstupu. Textury jsou bitmapové obrázky. Textury nemusejí reprezentovat jenom barvy objektu, ale například normal, bump, displacement nebo reflection mapu. Shadery jsou malé programy, běžící na grafických kartách, které vypočítávají barvu každého renderovaného pixelu. Materiály potom spojují shadery a textury a definují jak bude povrch objektů renderován. Materiál má přiřazený shader, který používá, společně s jeho nastavením a obsahuje reference na textury, které používá shader pro výpočet barev pixelů. Materiály jsou poté přiřazeny objektům ve scéně (viz obrázek 5.2).

Unity shadery jsou jednou z nejsložitější částí Unity a nebudou zde popsané detailně. Unity shadery jsou tvořeny z definice vlastností a podshaderů. Vlastnosti umožňují parametrizovat funkci shaderu (nastavit texturu, volnou nebo intervalem omezenou skalární hodnotu, barvu a vektor). Tyto vlastnosti potom je možné nastavit/upravit v editoru v materiálu, kterému je daný shader přiřazen.

Shader může mít více podshaderů, každý speciálně pro každou platformu, která ho bude používat. Při startu programu Unity prochází postupně podshadery a hledá první, který je kompatibilní s dostupnou grafickou kartou. Každý podshader obsahuje meta informace, které zkracují samotné tělo podshaderu.

Tělo shaderů je psáno ve speciálním jazyku Cg/HLSL, který je podobný jazyku C. Vlastnosti shaderu jsou použity pro nastavení hodnot z editoru, ale pro použití v tělu shaderu je potřeba je znovu definovat ve variantách, které grafické karty podporují. Unity je poté propojí a zkopíruje do paměti grafické karty.

Jazyk Cg/HLSL umožňuje psát pragma, které dokážou zkrátit délku

kódu (podobné makrům v C). Pragma jsou také použity pro určení jmen důležitých funkcí (viz níže). Další důležitou částí jsou sémantiky funkcí a proměnných (podobné atributům v C#). Jména sémantik nejsou standardizované, protože každý výrobce grafických karet si určil vlastní, tak několik různých sémantik může znamenat totéž. Unity shadery jde rozdělit na následující dva typy.

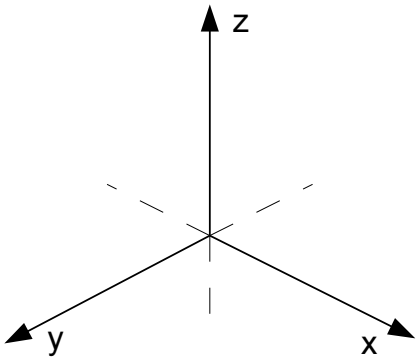
Surface shader

Surface shadery jsou používané hlavně pro práci se světlem a stíny. Tyto shadery mají vytvořený snadný přístup k Unity světelné pipelině. Surface shadery automaticky podporují přímé a odložené osvětlení. Tento typ shaderu jde napsat několika pragma řádky v jazyku Cg/HLSL a skutečný kód bude automaticky vygenerován. Tímto vygenerování se surface shader automaticky převede do vertex a fragment shaderu.

Vertex a fragment shader

Vertex a fragment shadery jsou hlavně používané, pokud je potřeba vytvořit nějaký speciální efekt. Většinou jsou více specializované, protože se musejí napsat skoro celé ručně v jazyku Cg/HLSL. Tyto shadery jsou rozšířeny o nutnost definicí speciálních funkcí pomocí direktivy pragma. Tyto funkce jsou poté prováděny v pipelinovém režimu grafickou kartou. Funkce *vertex* a *fragment* jsou povinné a většinou také jediné použité funkce. Těmto funkcím se musí napsat vstupní a výstupní struktury, jejich sémantiky a následně i jejich obsah. Při použití všech funkcí se budou vykonávat v následujícím pořadí v pipeline režimu.

- *vertex* - funkce pro transformaci vrcholů a jejich vlastností
- *hull* - funkce, která transformuje ovládací body daného kousku objektu (DX11)
- *domain* - funkce pro výpočet pozice vrcholu v její doméně (DX11)
- *geometry* - funkce, která může vytvářet nové vrcholy podle vrcholů v blízkém okolí (DX10)
- *fragment* - funkce pro výpočet barvy fragmentů (pixelů)



Obrázek 5.3: Směry kladných částí os v Unity 3D.

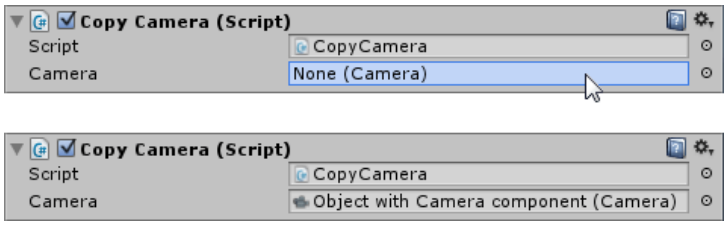
5.2 Editor

Unity editor je nástroj pro vytváření scén a nastavování vlastností objektů. Unity má levotočivý souřadnicový systém (viz obrázek 5.3). Pro nastavení vlastností objektů nabízí editor inspektor. Inspektor zobrazuje všechny komponenty právě vybraného objektu. Samotné komponenty potom mají zobrazené vlastnosti, které jde měnit. Inspektor v základu umožňuje nastavovat vlastnosti pouze Unity objektům. Pro zobrazení vlastností uživatelem vytvořených tříd je potřeba vytvořit skript, který rozšíří inspektor o informace jak zobrazovat daný typ v editoru.

Pomocí inspektoru je možné nastavovat i reference mezi jednotlivými objekty. Každý objekt i komponenta ve scéně mají přiřazené GUID, které je v tomto případě používané pro propojení mezi objekty. Nastavení referencí na jiný objekt nebo jeho komponenty se provádí přetažením daného objektu do pole inspektoru, které je nastavováno. Pokud dané pole vyžaduje komponentu, kterou přetažený objekt obsahuje, je automaticky nastavena reference na komponentu přetaženého objektu (viz obrázek 5.4). Pokud přetahovaný objekt má více komponent stejného typu, použije se reference na první (horní) komponentu. Pokud je žádána nižší komponenta, je potřeba otevřít další inspektor a přetáhnout přímo žádanou komponentu.

Během vytváření scény nebo běhu programu je někdy potřeba vytvářet složité objekty. Unity umožňuje vytvářet prefaby, které zastávají funkci šablony složitého objektu. Uložený prefab udržuje nastavení objektu, všechny jeho potomky a jejich nastavení. Prefaby jde potom přidávat do scény.

Za běhu programu nejsou v základu dostupné žádné prostředky z editoru (materiály, textury, prefaby, atd). Prostředky, které budou využívány během běhu programu, je potřeba přesunou do složky *Resources*. Během překladu Unity tyto prostředky přiloží k binárním datům. Skripty poté během běhu programu mají přístup k těmto prostředkům.



Obrázek 5.4: Nastavení reference na komponentu jiného objektu pomocí *Drag&Drop*. Nahoře stav komponenty před nastavením reference. Dole stav po přetažení objektu s žádanou komponentou na pole *Camera*.

5.2.1 GUI

Od verze Unity 4.6 jsou dostupné nové nástroje na tvorbu grafického rozhraní. Místo programování grafického rozhraní ve skriptech se rozhraní vytváří pomocí nových 2D objektů přímo v editoru (standartní $WYSIWYG$).

Všechny GUI prvky musejí mít někde v hierarchii jako předka objekt s komponentou *Canvas*. Tato komponenta umožňuje zobrazovat GUI prvky a nastavuje se v ní, kde se bude GUI zobrazovat.

- Screen space overlay - GUI bude kresleno přes celou obrazovku
- Screen space camera - GUI bude kresleno před vybranou kamerou
- World space - GUI bude umístěno do prostoru, viditelné ze všech kamer (například pro létající textové bubliny)

6 Vypracované řešení

Práce byla vypracovaná v Unity 3D Free verze 5.3.5f1 na operačním systému Windows 7 (64 bit). V rámci této práce byly vytvořeny nástroje pro zjednodušení a realizaci videomappingu. Projekt obsahuje skripty pro snadnou kalibraci projektorů, nastavení a zobrazení pozice uživatele v prostoru, pohybové akce proveditelné nad objekty scény, atd. Výstupem tohoto projektu bude *unitypackage* soubor, který bude možné importovat do jiných Unity projektů. Importem tohoto balíčku se do projektu přidají všechny objekty potřebné pro realizaci videomappingu.

Před zapnutím programu bude uživatel nejdříve vytvářet scénu. Toto nebude odlišné od normálního vytváření scény v Unity editoru. V editoru je nutné vytvořit virtuální scénu, která se bude promítat. Objekty virtuální scény je možné osvětlovat standardními světly, které Unity poskytuje. Reálnou scénu (objekty reálného světa, na které se bude promítat) je také možné vytvářet přímo v editoru, nebo poté za běhu programu importovat z *Wavefront obj* souboru. Jako poslední krok bude potřeba nastavit všem objektům speciální tag (viz kapitola 6.1 níže).

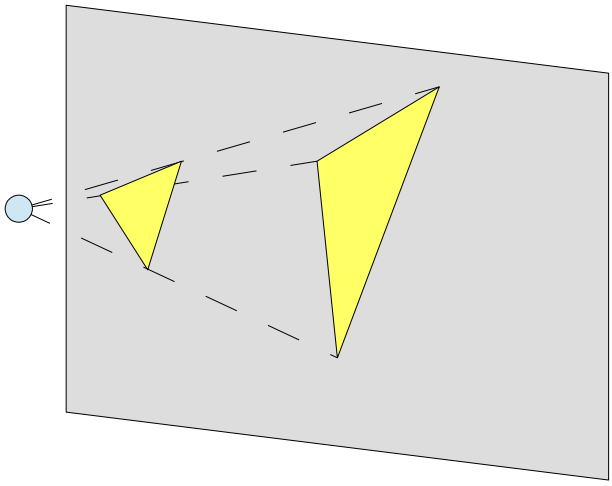
Pro další použití jsou objekty virtuální scény označeny OVS, objekty reálné scény jsou označeny ORS (reprezentace fyzických objektů ve Unity scéně) a objekty fyzické scény OFS (fyzické objekty, na které se promítá).

Do scény se také musejí přidat a nastavit objekt hlavní kamery a objekty projektorů. Tyto objekty jsou uložené jako prefaby (viz kapitola 6.4 níže). Některé parametry těchto objektů se musí nastavit pouze přibližné (později se budou kalibrovat), zatímco ostatní musejí být nastaveny přesně.

Program se poté může zkompilovat do spustitelného binárního souboru. Po spuštění bude uživatele provádět grafické rozhraní popsané v kapitole 6.3.

Samotný videomapping je potom prováděn v následujících krocích pomocí Unity komponent *Camera* a *Projector*. První dva kroky promítají (projekce v tomto případě znamená geometrickou projekci jednoho objektu na jiný objekt, viz obrázek 6.1) virtuální scénu na reálnou scénu z pozice hlavní kamery. Zatímco třetí krok pouze posílá obraz výsledku projektorům.

1. Unity *Camera* sleduje OVS a ukládá obraz do textury
2. Unity *Projector* promítá texturu z předchozího kroku na ORS ze stejné pozice jako jí *Camera* snímala
3. Unity *Camera* sleduje ORS a obraz posílá jako vstup projektoru



Obrázek 6.1: Ukázka projekce prováděné během videomappingu. Malý žlutý trojúhelník (virtuální scéna) je promítnutý na šedou plochu (reálné scéna) z pohledu modrého bodu (hlavní kamera).

6.1 Nové tagy a vrstvy

Při vytváření scény je potřeba odlišit určité objekty. Na to jdou použít vrstvy nebo tagy. Vrstvy by byly vhodnější, ale během importu *unitypackage* se vrstvy nepřidají do projektu. Toto by bylo možné obejít, ale jednodušší je použít tagy. Tagy se narozdíl od vrstev přidají do možných tagů na přiřazení po importování *unitypackage*. Jen aby se tagy správně importovali, musejí být použity v prefabech a proto byl vytvořen prefab Tag Holder.

Je vytvořeno 5 nových tagů. První dva tagy jsou použity v prefabech a uživatel by je neměl přenastavovat a žádný jiný objekt scény by je neměl mít. Další tři tagy jsou vytvořeny pro umožnění vytvářet scény uživatelem. Objekty ve scéně, které nemají žádný z těchto tagů nebudou nijak brány v potaz za běhu programu.

- User - nastaven hlavní kameře
- Projector - nastaven všem projektorům
- Real Object - označuje ORS, na které se bude promítat
- Scene Object - označuje OVS, které budou promítány na ORS
- Scene Light - označuje objekty, které budou osvětlovat virtuální scénu

Dále jsou vytvořeny dvě nové vrstvy: *Real layer* (hodnota 8) a *Scene layer* (hodnota 9). Po startu programu jsou objektům s danými tagy nastaveny jejich korespondující vrstvy. Uživatel by tyto vrstvy neměl ve zbytku

programu používat, protože objekty v těchto vrstvách jsou používány během videomappingu.

6.2 Rozdělení scén

Prvotní cíl projektu byl použití jedné Unity scény obsahující všechny objekty potřebné k videomappingu. Někdy je ale nutné pro stejnou reálnou scénu použít různé virtuální scény, nebo naopak. Kdyby vše bylo v jedné scéně, znamenalo by to při každé změně upravit scénu v editoru a zkompilovat celý projekt. S použitím Unity verze 5.3 je možné rozdělit tuto celou scénu do několika menších scén, ze kterých se poté může uživatel vybrat, kterou chce použít. Celá scéna by se mohla rozdělit do 3 typů menších scén.

- Hlavní scéna
- Reálná scéna
- Virtuální scéna

Hlavní scéna, která bude načtena po startu programu, musí obsahovat hlavní kameru, protože ovládá celý zbytek programu. Reálná scéna bude samozřejmě obsahovat ORS. Dále by měla obsahovat projektory, protože ty jsou pevně vázány na OFS, které jsou modelovány pomocí ORS. A nakonec virtuální scéna, obsahující OVS a jejich osvětlení.

Během běhu programu tedy bude nutné načíst až dvě scény. Toto rozdělení uživateli dává možnost načíst libovolnou kombinaci předem vypracovaných reálných a virtuálních scén. Pokud uživatel bude používat pouze jednu reálnou scénu, může přesunou její obsah do hlavní scény a pak během startu programu načíst pouze vybranou virtuální scénu. Totéž platí i pro použití jedné virtuální scény na několik reálných scén. Pro výběr jiné scény tedy uživatel nemusí překládat celý projekt, ale stačí pouze znovu zapnout program a načíst vybrané scény.

6.3 GUI

Po startu aplikace je vyžadována interakce s uživatelem pro správné nastavení projektorů. Z tohoto důvodu bylo vytvořeno grafické rozhraní, které provede uživatele procesem zapnutí aplikace. Celý proces zapnutí programu je rozdělen do 6 kroků.

Vybrání scény

Ihned po zapnutí programu se načtou a zobrazí jména všech dostupných scén. Uživatel má možnost vybrat dvě scény, které se načtou. Uživatel může vynechat načtení jedné nebo obou scén. Pro vynechání načtení scény musí uživatel vybrat první položku v seznamu scén. Když uživatel vybere stejnou scénu v obou výběrech, načte se pouze jednou.

Načtení *obj* souborů reálné scény

Druhý krok umožňuje načíst další ORS z *Wavefront obj* souborů. Tato možnost byla přidána pro umožnění dodefinovávat reálnou scénu po překladu projektu.

Načtení parametrů nebo kalibračních bodů

Třetí krok umožňuje načíst parametry projektorů z dřívějšího běhu programu po dokončené kalibraci. Také je možné načíst nové body kalibračního vzoru, který přepíše vzor, který byl vytvořený v editoru.

Kalibrace projektorů

Čtvrtý krok provádí samotnou kalibraci kamer. Postupně se prochází projektory a pro každý projektor se postupně zobrazují body kalibračního vzoru. Uživatel má za úkol kliknout na projektoru, kde by daný bod měl být promítán. Pokud je daný bod kalibračního vzoru mimo obraz projektoru, uživatel by měl kliknout na tlačítko *Next Point*. Uživatel také může přeskočit kalibraci libovolného projektoru kliknutím na tlačítko *Next Projector*. Pokud uživatel zadal minimální počet bodů dojde ke kalibraci při přechodu na další kameru nebo na další krok.

Uložení parametrů

Po kalibraci projektorů je možné uložit stávající parametry projektorů a kalibračního vzoru do souboru. Tento soubor poté jde načíst během kroku 2 a přeskočit kalibraci.

Registrace projektorů

Poslední krok provádí výpočet překryvů projektorů. Bez tohoto kroku by místa fyzické scény, kde se protínají zorné pole více projektorů, byly více osvětleny. Pro dosáhnutí rovnoměrného osvětlení se musí intenzita osvětlení pixelů snížit na překryvu jednotlivých projektorů.

Během registrace projektorů se budou procházet postupně pixely všech projektorů a vysílat se z nich paprsky do scény. Pokud paprsek protne ORS zjistí se bod dopadu tohoto paprsku. Projdou se ostatní projektory a testuje se, zda daný bod dopadu je v zorném poli projektoru. Takto se získají projektory, které přispívají k osvětlení daného bodu dopadu paprsku a pomocí vzorce (2.1) se vypočítá $\lambda_i(x)$. Intenzita osvětlení tohoto bodu ostatními projektory nejde v tuto chvílí vypočítat, protože obraz bodu dopadu nemusí být přesně uprostřed určitého pixelu a musí se počkat, až na tyto projektory dojde řada. Když paprsek neprotne objekt reálné scény nebo bod dopadu je viditelný jen z jednoho projektoru, intenzita pixelu je nastavena na 1.

Hodnoty všech LAM jsou poté upraveny pomocí gamma lookup table. Intenzity osvětlení na překryvech jsou upraveny, aby se počítalo s jednotlivými gamma korekcemi projektorů. Výsledná LAM je poté převedena do textury a zobrazena v GUI jednotlivých projektorů (více v kapitole 6.4.2). Celý tento proces je příliš časově náročný, než aby se provedl celý během jednoho snímku simulace. Proto je tento proces převeden do koprogramu, který se vykonává po dobu delší doby, zatímco je grafické rozhraní aktualizováno a postup je zobrazen progress barem.

Po dokončení těchto kroků se zobrazí scéna. Pokud scéna obsahuje alespoň jeden objekt obsahující akci, zobrazí se menu akcí. Toto menu zobrazuje jméno právě vybrané akce, umožňuje procházet jednotlivé akce a zapínat je. Zobrazí se i menu pozice hlavní kamery. Toto může být použito pro zjištění pozice správné perspektivy. Více o akcích v kapitole 6.6.

6.4 Prefaby

Některé objekty potřebné pro videomapping jsou příliš složité v jejich struktuře a nastavení. Z tohoto důvodu byly vytvořeny 3 prefaby.

6.4.1 Hlavní kamera

První prefab reprezentuje hlavní kameru. Do scény by měla být umístěna pouze jedna instance tohoto prefabu. Za běhu programu je tato kamera používaná pro určení pozice správné perspektivy. Uživatel, který sleduje obrazy promítané fyzickými projektory z místa hlavní kamery, bude vidět nezkreslené OVS, pokud byly parametry projektorů správně nastaveny a reálná scéna dostatečně přesně vymodelovaná. Tento prefab obsahuje hierarchii objektů používaných pro videomapping a GUI.

Základní komponenta, kterou tento prefab obsahuje je *Camera*. Zde by uživatel měl nastavit velikost zorné úhlu a viewportu kamery. Viewport ka-

mery vyznačuje obdélník, do kterého bude tato kamera zobrazovat výstup na obrazovce v normalizovaných souřadnicích. Ostatní parametry kamery by uživatel neměl měnit, protože jsou přednastavené, aby videomapping fungoval správně.

Komponenty *Camera* i *Projector* mají omezenou velikost zorného pole. Takže při použití pouze jednoho projektoru by záleželo jakým směrem je hlavní kamera orientovaná. Při videomappingu by mělo záležet pouze na pozici pozorovatele, nikoliv jeho orientaci. Z tohoto důvodu hlavní kamera obsahuje 6 dvojic *Camera-Projector*. Každá dvojice má nastavený zorný úhel na 90 stupňů a poměr stran na 1. Tyto dvojice jsou potom natočeny tak, aby vytvořily kostku (jedna dvojice dopředu, doprava, doleva, dozadu, nahoru a dolu). Takto je celý prostor kolem hlavní kamery neustále snímán a videomapping přestává být závislý na orientaci hlavní kamery.

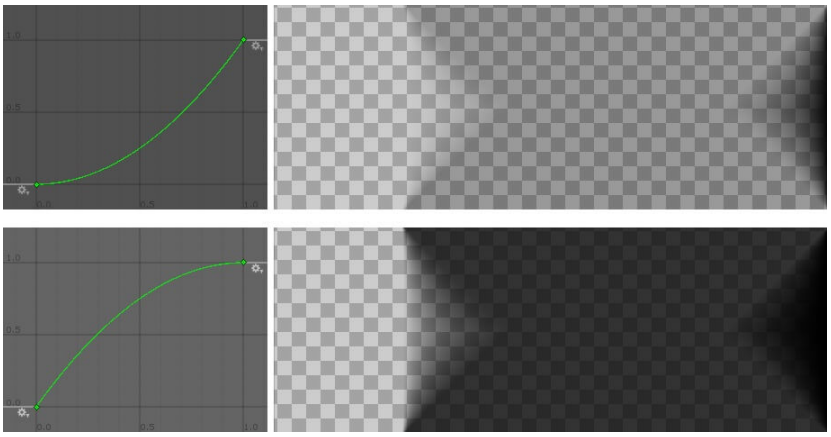
Grafické rozhraní aplikace je také umístěno v tomto prefabu. Toto grafické rozhraní je nastaveno do módu *Screen overlay camera* a přiřazeno hlavní kameře. Grafické rozhraní obsahuje hlavní menu (provádí uživatele při startu aplikace), akční menu (zobrazuje akce proveditelné nad objekty scény) a také zobrazuje pozici hlavní kamery, aby se uživatel mohl postavit na správné místo.

Tento prefab také obsahuje důležité skripty potřebné pro videomapping. Tyto skripty ovládají interakci grafického rozhraní, proces kalibrace, pohybování hlavní kamery a umožňují nastavit parametry kamery, které Unity jinak nedovoluje upravovat (vyosení a poměr stran). Kalibrační manager dovoluje nastavit pozice bodů kalibračního vzoru a velikost a barvu kalibrační koule (používaná pro zobrazení přibližné pozice kalibračního bodu). Nakonec ještě jde upravit rychlost otáčení a pohybu hlavní kamery.

6.4.2 Projektor

Tento prefab by měl být umístěn do scény na přibližné místo reálných projektorů (jedna instance na každý reálný projektor). Stejně jako hlavní kamera obsahuje komponentu *Camera*, které je potřeba nastavit před spuštěním aplikace zorný úhel, viewport, vyosení a poměr stran. Navíc oproti hlavní kameře obsahuje také komponentu pro nastavení gamma lookup curve projektoru.

Gamma lookup curve se používá, protože když se osvětluje oblast více projektory, intenzita osvětlení se snižuje s předpokladem, že projektor osvětluje toto místo jen jeden. Když tedy osvětlují dva projektory stejnou oblast, oba s poloviční intenzitou, výsledek nemá stejnou intenzitu jako kdyby místo osvětloval pouze jeden projektor. To je způsobeno gamma korekcí. Ga-



Obrázek 6.2: Ukázka dvou gamma lookup curve a jejich dopad na osvětlovací mapu projektoru. Textury napravo budou kresleny přes výstup projektoru pro ztmavení určitých pixelů.

mma křivky obou projektorů nejsou lineární a proto neplatí, že dvojnásobné zvýšení intenzity způsobí dvojnásobní osvětlení. Projektor s tímto počítá a vnitřně přepočítává novou intenzitu, aby obraz vypadal správně.

Když se poté kombinují osvětlení z dvou projektorů, projektory nepočítají s tím, že plocha je osvětlená ještě jiným projektorem a obraz má špatnou intenzitu osvětlení. Proto je použita Gamma lookup curve, která upravuje přepočet intenzit na překryvech projektorů, aby obraz byl osvícen rovnoměrně. Ukázka vlivu různých gamma lookup curve na LAM je dostupná na obrázku 6.2.

Samotného snížení intenzity pixelů projektoru je docíleno aditivním způsobem. Aditivní přístup snižuje intenzitu pixelů přidáním z části průhledné černé barvy (efektivně ztmavuje pixely). Toto jde použít, protože se jedná o fyzický projektor, kde promítání černé barvy odpovídá promítání průhledné barvy, jinak by se musel použít multiplikativní přístup, který upravuje průhlednost.

Toto přidání z části průhledné černé masky je prováděno pomocí GUI před komponentu *Camera*, která vytváří vstup pro projektory. Přes celé toto GUI je přetažen panel, kterému je nastavena textura masky intenzit osvětlení. Unity správně nepočítá s vyosením kamery a posunutím grafického rozhraní. Z tohoto důvodu byla každému projektoru vytvořena další kamera, která se stará o vykreslení grafického rozhraní a renderuje do stejného viewportu před kameru prefabu projektor.

6.4.3 Tag Holder

Tento prefab je vytvořen pouze, aby všechny nové tagy byly importovány společně s knihovnou, nemá proto žádný smysl ho přidávat do scény. Tyto tagy se poté využívají při vytváření scény uživatelem.

6.5 Skripty

Unity chápe skripty převážně jako komponenty přiřazené objektům scény. Projekt obsahuje i standardní C# skripty, které ale pouze obalují Unity objekty pro jednodušší práci s nimi. Speciální kategorie skriptů (komponent), které byly vytvořeny v této jsou akce, kterými se zabývá kapitola 6.6.

6.5.1 Komponenty

Main Menu

Tento skript se stará o interakci s uživatelem po zapnutí programu. Zobrazuje kroky podle potřeby a provádí akce uživatele voláním metod ostatních skriptů.

Action Menu

Skript zodpovědný za zobrazování, vybírání a zapínání všech dostupných akcí nad objekty scény. Zobrazuje jméno právě vybrané akce.

Calibration Manager

Tato komponenta obsahuje definici bodů kalibračního vzoru. Během kalibrace je zobrazen právě kalibrovaný bod vzoru jako bílá koule. Tato komponenta umožňuje měnit velikost i barvu této koule. Dále tato komponenta zpravuje proces kalibrace, pamatuje si doposud kalibrované body a obsahuje metody volání knihovny OpenCV.

Progress Bar

Unity v základu nenabízí progress bar a proto se musí vytvořit vlastní. Tato komponenta je použita při registraci projektorů pro zobrazení postupu registrace.

Camera Movement

Pro upravování pozice správné perspektivy byl vytvořen tento skript. Umožňuje pohybovat hlavní kameru scénou. Dají se zde nastavit rychlosti pohybu a rotace. Skript byl vytvořen tak, aby kopíroval chování Unity editoru, neboli pohyb se umožní při držení pravého tlačítka myši. Rotace kamery se poté ovládá pohybem myši zatímco pohyb je ovládán pomocí tlačítek W, S, A, D, $Space$ a C.

Camera Parameters

Parametry kamery, které Unity standardně nedovoluje nastavit, jako vyosení a poměr stran je možné nastavit v této komponentě. Upravené parametry jsou ihned zobrazeny na komolém jehlanu kamery ve scéně.

Gamma Lookup Curve

Pro upravení osvětlovací mapy po registraci projektorů byla vytvořena tato komponenta. Obsahuje definici křivky, která se používá pro vypočítání skutečného snížení osvětlení z vypočítaného snížení osvětlení.

Camera Position Copy

Komponenta používaní pro zjišťování pozice hlavní kamery. Tato pozice je poté zobrazena v grafickém rozhraní.

Camera Rect Copy

Tato komponenta kopíruje nastavení viewportu z jedné kamery do druhé. Toto je použito pro vykreslování osvětlovací mapy do projektorů. Kamera zodpovědná za vykreslení GUI musí mít nastavený stejný viewport, aby svým výstupem kreslila před kameru projektoru.

6.5.2 Ostatní

Tyto ostatní skripty nejsou Unity komponenty a zjednodušují práci s Unity objekty a doplňují funkcionalitu, kterou Unity v základu nemá.

Mesh Importer

Tento skript byl vytvořen pro import *obj* souborů během běhu programu. Načítá standardní $Wavefront\ obj$ soubor a vrací $Mesh$, kterou dokáže Unity použít pro vytvoření nového objektu.

Scene

Pro zjednodušení práce s Unity objekty ve scéně byl vytvořen tento skript. Umožňuje přistupovat ke hlavní kameře a kolekcím OVS, ORS, projektorů ve scéně, světel ve scéně a komponent akcí. Dále obsahuje metody pro zapnutí a vypnutí všech prvků těchto kolekcí. Nakonec ještě obsahuje metodu registrace projektorů.

Rect

Unity struktura *Rect* obsahuje veliké množství atributů a je nevhodný pro serializaci během ukládání parametrů kamer. Tento skript obsahuje pouze čtyři nutné atributy, které budou serializované.

Extension

Jazyk C# umožňuje přidávat metody uzavřeným třídám vytvořením statické metody (uvnitř nové statické třídy), která má první parametr označen parametrem *this*. V této metodě je možné přistupovat k veřejně dostupných vlastnostem a metodám prvního parametru. Toto nijak neporušuje zapouzdření objektů, pouze zpřehledňuje volání metody. Následující dvě volání jsou ekvivalentní.

```
instance.ExtensionMethod(parameter);
ExtensionClass.ExtensionMethod(instance, parameter);
```

Unity objekty *Gameobject* byly rozšířeny o metody pro jejich zapínání a vypínání. Místo obvyklého vypnutí celého objektu se vypínají pouze vybrané části jejich komponent. Toto bylo provedeno, protože normálně vypnutý objekt není už možné znovu zapnout, protože už nejde dohledat pomocí tagu. Pouze pomocí zapamatované reference je možné takový objekt znovu zapnout. Objekt vypnut nově vytvořenými metodami bude pořád vyhledatelný pomocí tagu a jeho funkčnost bude stejná jako kdyby byl normálně vypnut.

6.6 Akce

Akce jsou speciální komponenty, které jsou přiřazeny objektu, se kterým mají něco provést. Všechny objekty scény (OVS, ORS, hlavní kamera i projektory) můžou mít přiřazeno neomezený počet akcí. Tyto akce můžou provádět naprosto cokoliv s objekty. Po zapnutí programu se prohledají všechny objekty a shromáždí se všechny akce, které jde nad nimi provést. Tento seznam

akcí je poté zobrazen v grafickém rozhraní, kde je uživatel může procházet a zapínat.

V rámci tohoto projektu byla vytvořena základní kostra akcí (komponenta *ActionBehaviour*). Všechny akce musejí dědit od této třídy. V základu jde každé akci nastavit jméno, viditelnost za běhu (zda bude akce možná zapnout z grafického rozhraní) a referenci na akci, která se má zapnout po řádném skončení této akce. Některé akce jako změna barvy nebo materiálu je možné provést během jednoho snímku. Jiné akce jako pohyb objektu jsou prováděny po dobu několika snímků. Udržuje se tedy i stav akce, zda je právě aktivní.

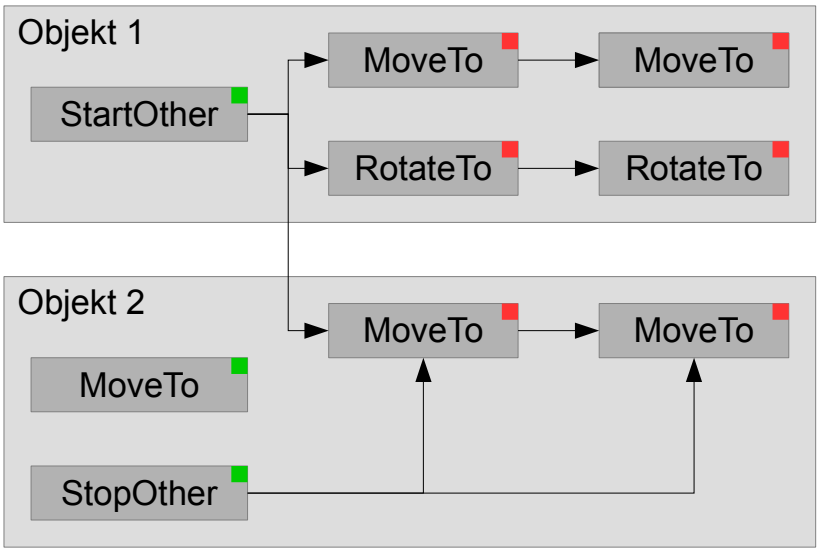
Akce se zapíná veřejnou metodou *StartAction* a předčasně se vypíná také veřejnou metodou *StopAction* (zastavení tímto způsobem nespustí další akci). Pro upravení chování potomků jsou vytvořeny *protected* virtuální metody *OnStartAction*, *OnUpdateAction* a *OnEndAction*. Pro ohlášení řádného ukončení akce je vytvořena *protected* metoda *EndAction*.

Pro účely testování funkčnosti projektu byly vytvořeny 4 akce. Tyto akce jde použít pro sestavení složitých grafů akcí jako například na obrázku 6.3. Unity editor umožňuje tyto akce vytvořit a nastavit naklikáním několika hodnot a přetažením referencí na ostatní akce.

- *StartOtherAction* - obsahuje seznam akcí, které budou zapnuty potom co se aktivuje tato akce
- *StopOtherAction* - obsahuje seznam akcí, které budou zastaveny (*StopAction*) při aktivaci této akce
- *MoveToAction* - po aktivaci začne posouvat objekt na určené místo po zadanou dobu
- *RotateToAction* - po aktivaci začne otáčet objekt do zadaného směru po zadanou dobu

6.7 Nový shader

Unity v základu neobsahuje shader, který funguje jako fyzický projektor. Po dlouhém hledání nebyl ani na internetu nalezen shader, který splňuje všechny nároky. Z tohoto důvodu se musel tento shader vytvořit. Na obrázku 6.4 je vidět rozdíl mezi obvykle nalezeným (vlevo) a požadovaným shaderem (vpravo) při promítání standardního kalibračního vzoru.

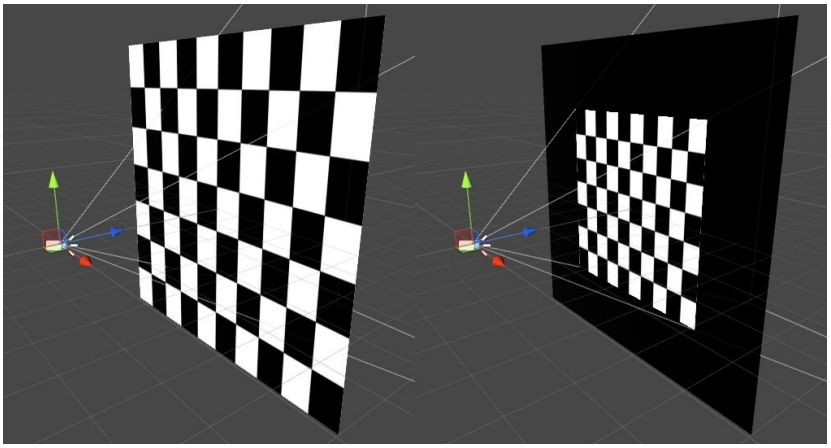


Obrázek 6.3: Příklad grafu akcí dvou objektů. Akce se zelenými čtverečky budou uživateli zobrazeny za běhu.

Nový shader (pojmenovaný *ProjectorAddidive*) je vertex fragment shader (shadery jsou vysvětlené v kapitole 5.1.4). Vlastnosti shaderu jsou promítaná textura a barva, která se má promítat mimo zorné pole projektoru. Protože se jedná o aditivní projektor, tato barva je v základu nastavena na černou. Většina práce je prováděna ve *fragment* funkci shaderu. Nejdříve se kontroluje jestli je bod, na který se promítá, před projektorem. Potom se kontroluje, že bod je uvnitř zorného pole projektoru. Následně se vypočítá barva z dodané textury a pozice bodu v zorném poli. Fragment funkce tuto barvu vrátí. Když alespoň jedna z těchto podmínek není splněná, automaticky se vrací barva nastavená ve vlastnostech shaderu.

6.8 Výstup projektu

Unity umožňuje zabalit libovolnou množinu souborů projektu do balíčku, který je poté možný importovat do jiného projektu. Výstupem tohoto projektu bude *unitypackage*, který bude obsahovat všechny knihovny, prefaby, materiály, skripty atd. Tento soubor se poté bude importovat od projektu, ve kterém se bude provádět videomapping. Uživatel poté použije importované zdroje pro snadné nastavení objektů videomappingu. Správné nastavení objektů, dostatečně přesně vymodelovaná fyzické scéna a dostatečně přesná kalibrace poté zaručí bezchybný videomapping.



Obrázek 6.4: Porovnání shaderů. Vlevo shader aplikuje texturu na objekt. Vpravo shader promítá texturu na objekt.

7 Testování a výsledky

Vytváření většiny skriptů, testování v editoru a zatěžové testy byly prováděny na následující domácí počítačové sestavě.

- CPU: Intel® Core™ i5-2500K (4 × 3.3 GHz)
- GPU: AMD Radeon HD 6950 (800MHz), paměť 2GB (1250MHz)
- Paměť: 20GB
- OS: Microsoft Windows 7 64bit

Funkčnost konečné verze projektu byla také testována v prostředí Oracle VM VirtualBox na stejném hostovacím zařízení se stejným operačním systémem. Testování na fyzické scéně bylo prováděno v prostorách KIV.

7.1 Fyzická scéna

Fyzická testovací scéna byla vytvořena ze stěn a nábytku v rohu místnosti (viz obrázek 7.1), model fyzické scény byl vytvořen ručně jako *Wavefront obj* soubor a následně importován do Unity. Reálná scéna se promítá dvěma projektory Benq W770ST se značným překryvem zorných polí pro testování výpočtu osvětlovacích map.

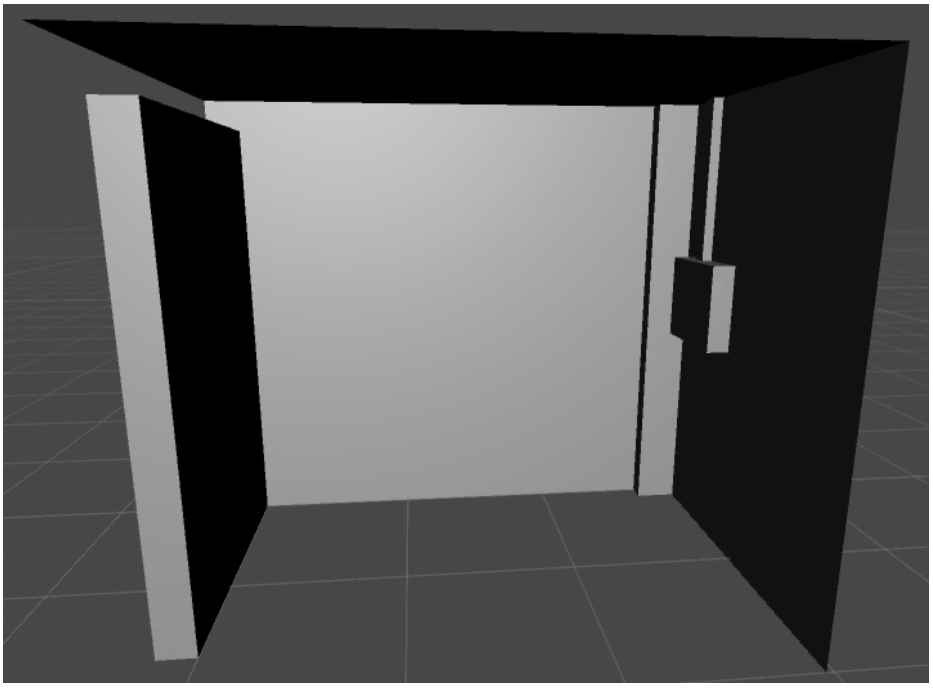
7.2 Virtuální scény

Pro testování byly vytvořeny dvě testovací virtuální scény.

7.2.1 Měřicí scéna

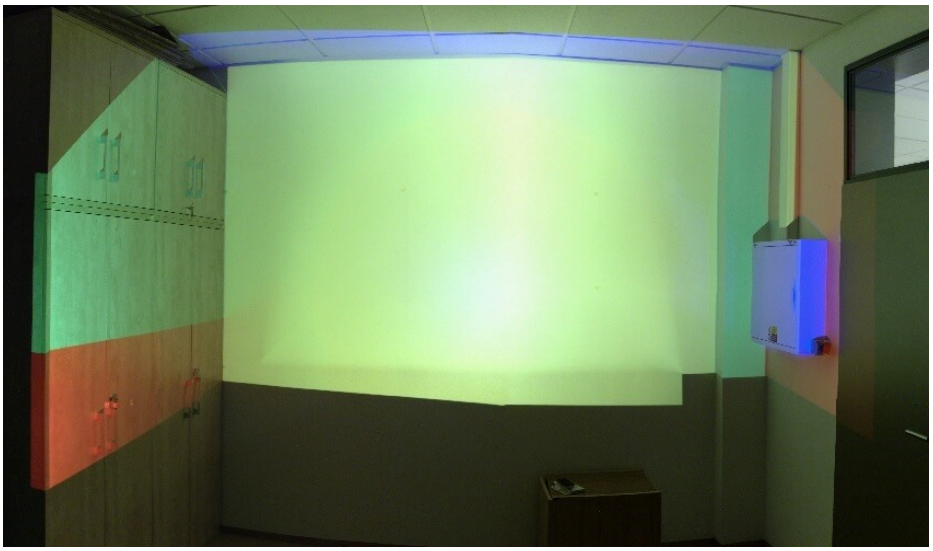
Pro jednoduché ověření funkčnosti kalibrace a samotného videomappingu byla vytvořena scéna, která kopíruje objekty reálné scény. Jednotlivé plochy byly různě obarveny, aby je bylo možné rozeznat (viz obrázek 7.2). Tato scéna je sledovatelná z libovolného bodu s nezkreslenými objekty, protože objekty nejsou umístěny v prostoru, ale přímo na pozicích reálných objektů.

(a) Testovací fyzická scéna, na kterou se bude promítat.



(b) Testovací reálná scéna (model fyzické scény).

Obrázek 7.1: Fyzická testovací scéna a její model v Unity.

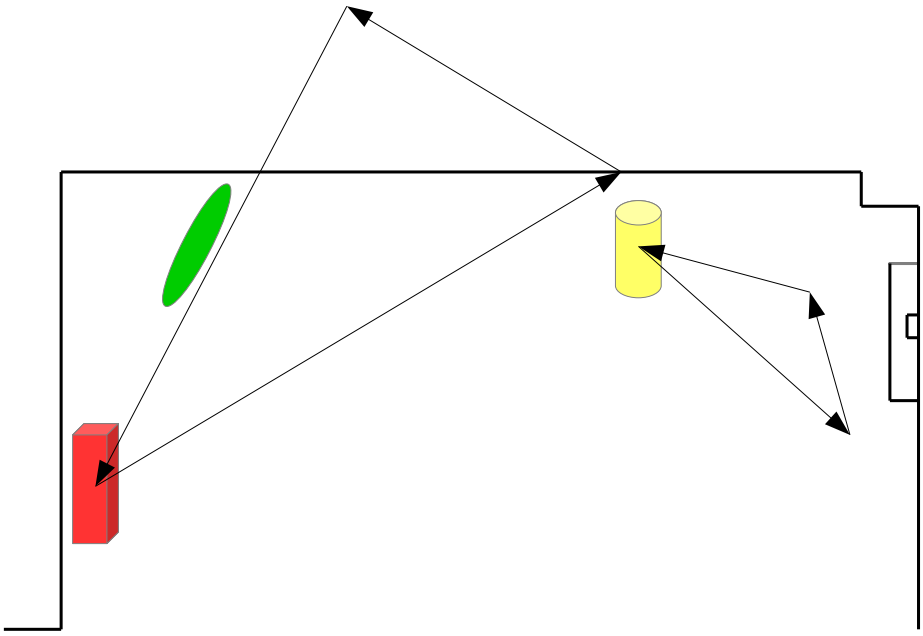


Obrázek 7.2: Měřící virtuální scéna zobrazující mapování ORS na OFS.

7.2.2 Zkušební scéna

Pro testování bodu správné perspektivy byla vytvořena tato scéna. Zde žádný objekt není umístěn na pozici reálných objektů, takže existuje pouze jeden bod, ze kterého nebude promítaná scéna zkreslená. Scéna obsahuje 3 objekty na místech (viz obrázek 7.3), kde je zkreslení maximálně vidět. Toto zkreslení je nejvíce vidět na objektech, které se promítají na více než jednu souvislou plochu. Ukázky správné a špatné perspektivy jsou v kapitole 7.5.

Těmto třem objektům byly vytvořeny grafy akcí. Válec a kvádr se pohybují mezi třemi body (viditelné na obrázku 7.3). Všechny tři objekty se také otáčejí kolem osy y pro vytvoření více pohybu. Dále byly vytvořeny akce pro hlavní kameru. Tyto akce přesunou hlavní kameru na předem určené místo, aby fyzické kamera byla v přesně stejných souřadnicích.



Obrázek 7.3: Půdorys virtuální zkušební scény. Šipky označují cestu po které se objekty pohybují.

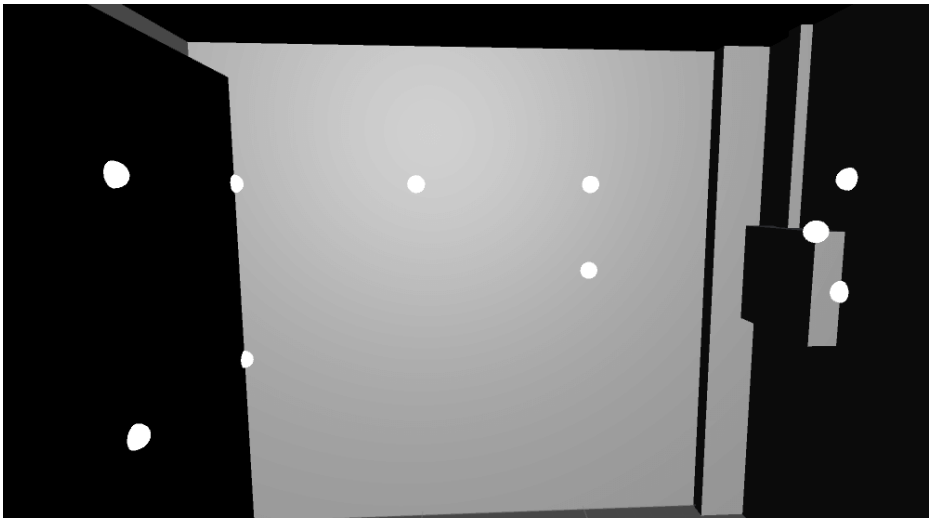
7.3 Kalibrace

Kalibrační vzor použitý pro kalibraci projektorů na reálnou scénu je zobrazený na obrázku 7.4. Testování ukázalo, že minimální počet bodů, které musí projektor vidět pro úspěšnou kalibraci je 6. Reálná scéna obsahuje dva projektory, projektor vlevo vidí 7 levých bodů kalibračního vzoru, zatímco pravý projektor vidí 6 pravých bodů kalibračního vzoru.

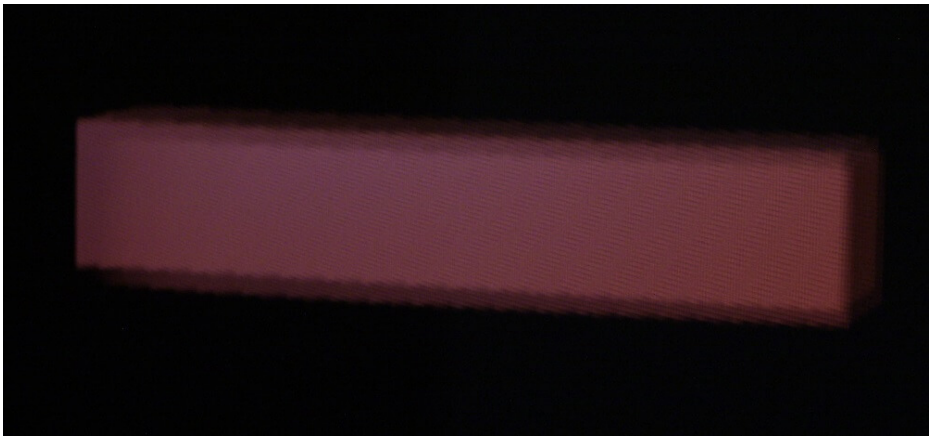
I když je 6 bodů minimum pro úspěšnou kalibraci, tyto body mají poměrně přísné podmínky pro jejich relativní pozici bodů. Proto je doporučeno, aby projektor viděl více bodů a tím se kalibrace zpřesní.

Samotná kalibrace probíhá poměrně rychle. Uživatel pro každý projektor prochází všechny body kalibračního vzoru a pro body, které jsou v zorném poli projektoru klikne co nejpřesněji na pixel, kde se má daný bod promítat. Tato scéna je možná nakalibrovat během přibližně půl minuty. Výsledek takové kalibrace je viditelný na obrázku 7.2. Kalibrace není úplně přesná, buď nebyla samotná fyzické scéna dostatečně přesně vymodelována, nebo kalibrace nebyla provedena dostatečně přesně. Výsledek je, že objekt na překryvu více projektorů není promítán na úplně stejné místo (viz obrázek 7.5).

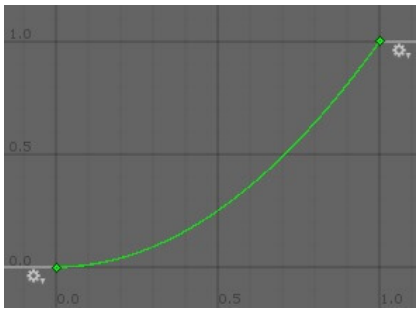
V tomto případě byly překryvy projektorů schválně značně veliké pro testování chyb překryvů. Při použití v praxi by překryvy byly značně menší a tato chyby by byla méně viditelné.



Obrázek 7.4: Body kalibračního vzoru v reálné scéně.



Obrázek 7.5: Nedokonalost modelu fyzické scény nebo kalibrace projektorů způsobuje neostré projekce objektů.

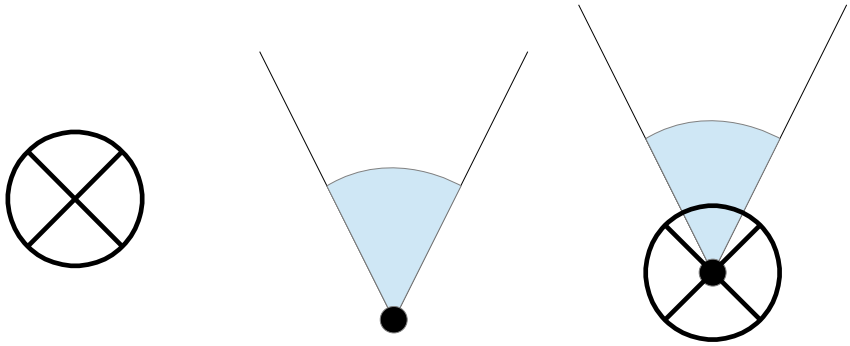


Obrázek 7.6: Nastavení gamma korekce obou projektorů při registraci projektorů.

Obrázek 7.7: Ukázka scény po výpočtu překryvů. Promítá se rovnoměrně bílá barva.

7.4 Registrace projektorů

Další krok startu aplikace je registrace projektorů. Projektory mají nastavenou křivku gamma korekce dle obrázku 7.6. Výsledek registrace s tímto nastavením je vidět na obrázku 7.7. Na této scéně se má promítat rovnoměrně bíla barva. V rozích překryvů je vidět mírné ztmavení.



(a) Pozice perspektivy. (b) Pozice kamery. (c) Kamera se správnou perspektivou.

Obrázek 7.8: Legenda obrázků půdorysů scén.

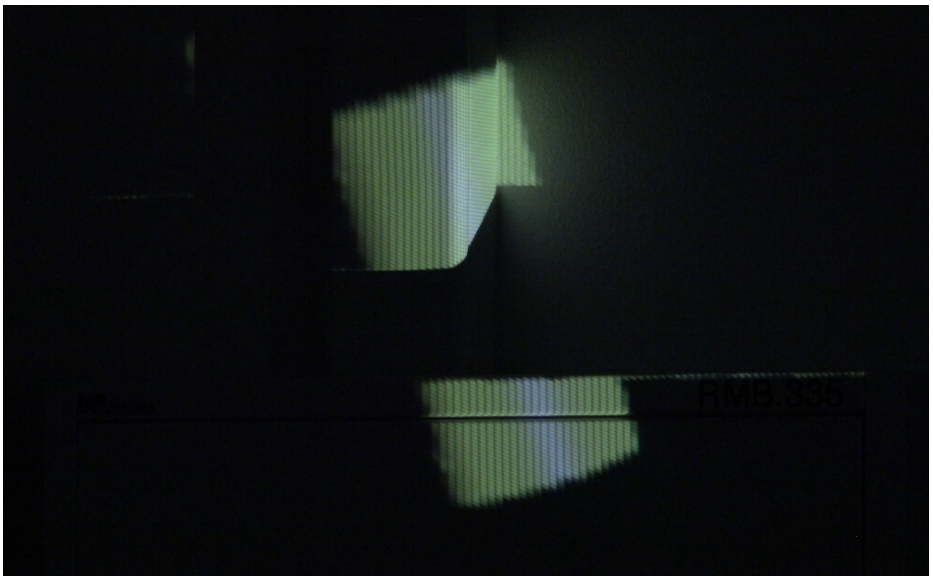
7.5 Správná perspektiva

Následující obrázky jsou tmavé, aby byly co nejlépe vidět promítané objekty. To způsobuje, že není jasné, kde se vyskytuje kamera a samotný objekt ve scéně. Proto byly vytvořeny obrázky půdorysu scény, kde jsou jejich pozice ukázány (legenda na obrázku 7.8).

První ze zobrazených objektů je válec. Tento válec je umístěn schválně na nejvíce problematickém místě scény. Pravá strana fyzické scény obsahuje větší počet objektů a jejich pozice zvýrazňuje chyby prováděné během kalibrace. Na obrázku 7.9 je vidět pohled ze správné perspektivy na válec. Prostřední část válce není promítána, protože je projektor umístěn níž než krabice rozvodu elektřiny. Tato krabice tedy vrhá stín na vedení kabelů a tato oblast nemůže být osvětlena. Kraje válce na sebe ale navazují. Velikost obrazu válce je stejná jako kdyby byl promítán na rovnou plochu. Na obrázku 7.10 je stejná scéna vyfocená z jiné pozice. Kraje na sebe nenavazují, velikost a tvar válce není správný.

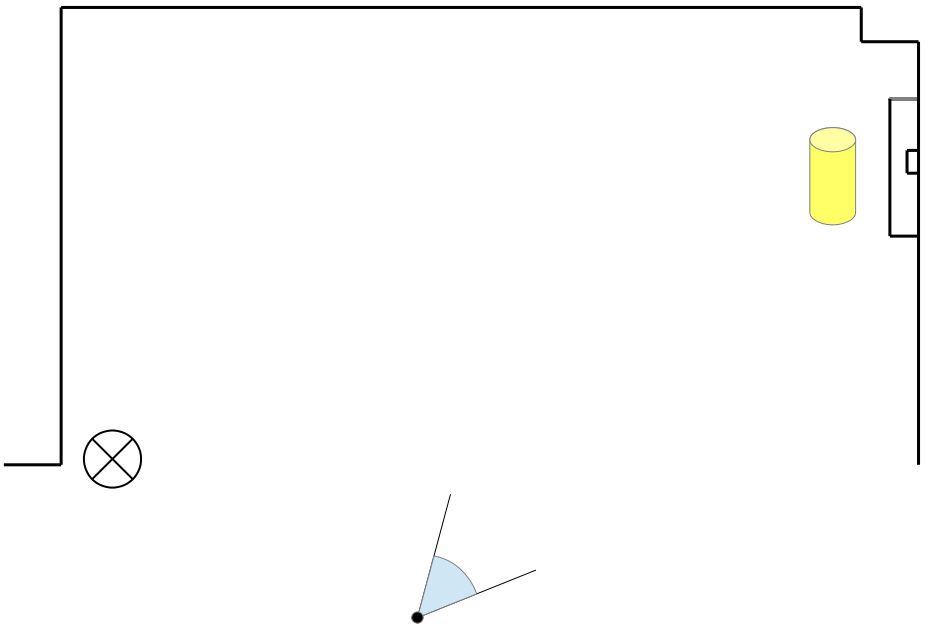
Další testovaný objekt je kvádr. Ten je umístěn na přechodu dvou na sebe kolmých reálných ploch. Kvádr je na levé ploše méně osvícen, protože je tato plocha umístěna vlevo od projektoru. Na tuto plochu tedy dopadá méně pixelů a tedy i méně světla. Z pozice správné perspektivy (na obrázku 7.11) má kvádr obraz převážně na levé ploše a není zdeformován. Z pozice špatné perspektivy (obrázek 7.12) je jeho obraz zdeformován tak, že většina obrazu je na pravé ploše a nevypadá jako kvádr.

(a) Půdorys scény.

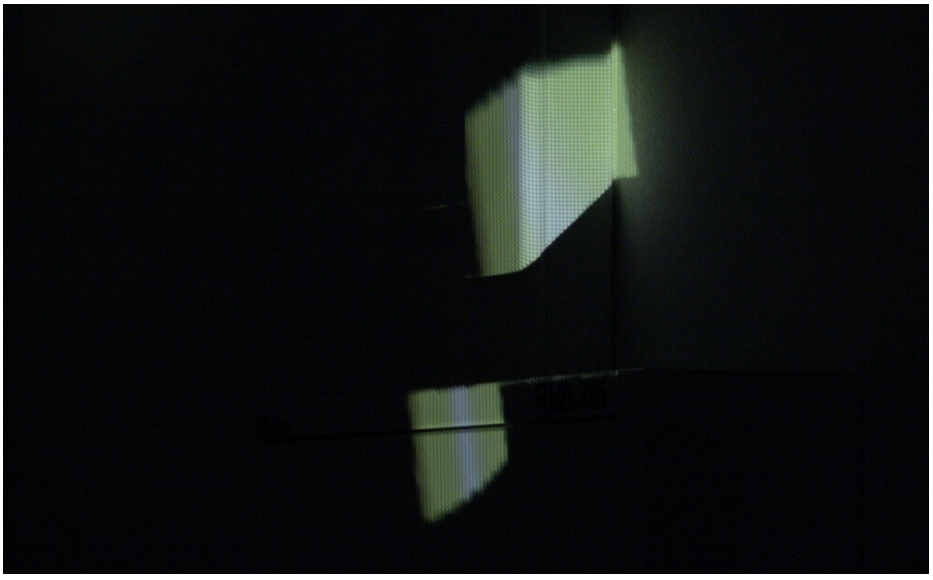


(b) Pohled z pozice kamery.

Obrázek 7.9: Pohled na válec ze správné pozice perspektivy.



(a) Půdorys scény.



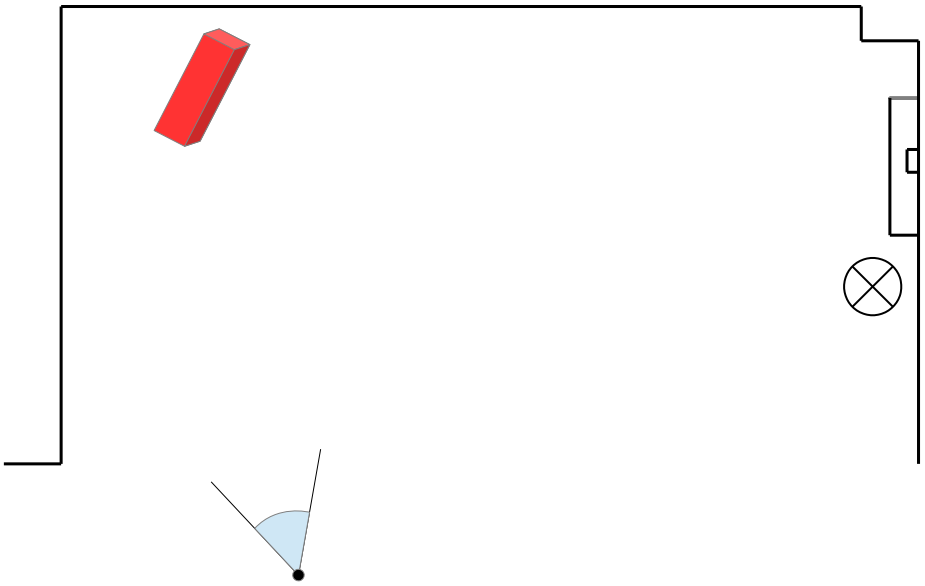
(b) Pohled z pozice kamery.

Obrázek 7.10: Pohled ze špatné pozice perspektivy.

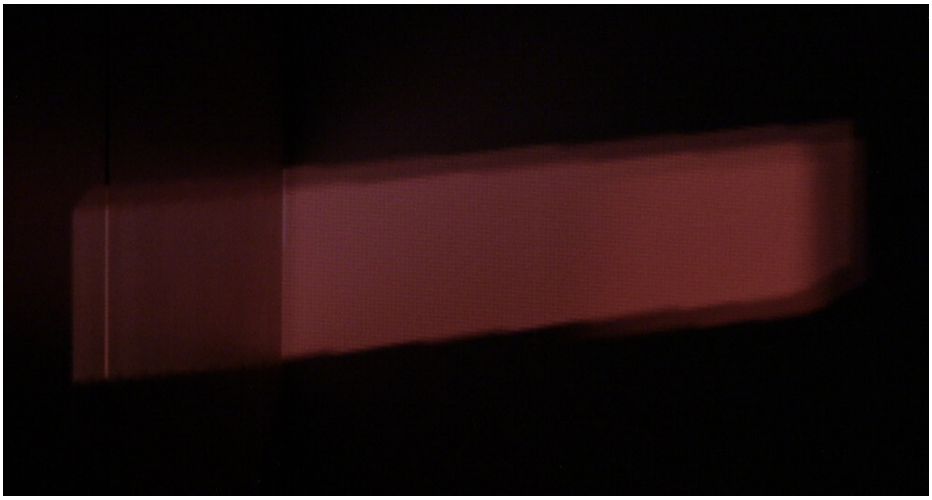
(a) Půdorys scény.

(b) Pohled z pozice kamery.

Obrázek 7.11: Pohled ze správné pozice perspektivy.

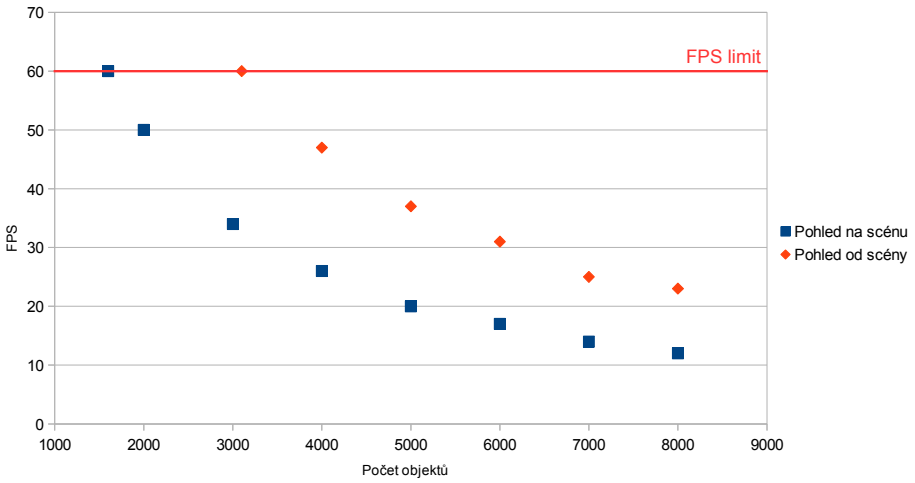


(a) Půdorys scény.



(b) Pohled z pozice kamery.

Obrázek 7.12: Pohled ze špatné pozice perspektivy.



Obrázek 7.13: Závislost počtu objektů scény na obnovovací frekvenci programu.

7.6 Zátěžový test

Poslední test zjišťuje závislost počtu objektů na obnovovací frekvenci programu. Unity engine je optimalizovaný a nerenderuje scénu, která není v zorném poli kamery a proto bylo měření provedeno při sledování scény hlavní kamerou a také při otočení kamery od scény. Během testování byla omezená obnovovací frekvence na 60 FPS a na grafech je zobrazena pouze první hodnota, pro kterou bylo dosaženo 60 FPS.

Graf 7.13 ukazuje, že scéna s více než 1600 objektů trpí snížením obnovovací frekvence. Když hlavní kamera nesleduje scénu, obnovovací frekvence se zvýší, ale projektorová kostka kolem hlavní kamery snímá celou scénu, takže scéna se stále renderuje celá a zvýšení FPS není tak razantní. Pokud se obnovovací frekvence omezí na 30 FPS, počet možných objektů naroste na 3300. Obvyklá scéna by bude obsahovat maximálně několik stovek objektů, takže pokles výkonu by neměl nastat. V krajním případě je možné otočit hlavní kameru od scény. Videomapping bude stále fungovat, jen se sníží počet kamer, které renderují scénu, a tím se zvýší FPS.

8 Závěr

V rámci diplomové práce byly vytvořeny prostředky pro usnadnění tvorby projektu videomappingu. Během řešení práce byly prozkoumány různé metody kalibrace projektorů. Jedna byla vybrána a použita. Dále byla vytvořena detailní uživatelská příručka popisující proces vytvoření scén, nastavení parametrů objektům a popisuje i uživatelské rozhraní zobrazené po startu aplikace.

Práce byla pečlivě testována v Unity editoru i na fyzické scéně. Byly vytvořeny dvě virtuální scény pro testování: měřicí scéna pro zhodnocení přesnosti kalibrace a zkušební scéna pro testování správné perspektivy. Tyto virtuální scény byly vyzkoušeny na fyzické scéně. Virtuální objekty sledovány z pozice správné perspektivy nebyly zkreslené. Toto platilo, i když se objekty ve scéně pohybovaly. Při sledování objektů ze špatné pozice byly objekty správně zkresleny.

Dále byla otestována přesnost kalibrace a registrace projektorů. Model fyzické scény nebo kalibrace projektorů nebyla dokonalá a proto na překryvu projektorů nebyly objekty promítány na přesně stejnou pozici. Nedokonalost kalibrace ovlivňovala i výsledek registrace projektorů. V rozích překryvu bylo vidět mírné ztmavení, které by se nemělo vyskytovat. Tyto problémy by se daly vyřešit přesnějším modelem fyzické scény a nebo přesnější kalibrací projektorů. Nakonec byl testován dopad počtu objektů na obnovovací frekvenci. Program běží plynule pro očekávaný počet objektů ve scéně.

Zadání diplomové práce bylo splněno. Práci by bylo možné rozšířit například umožněním zapínání akcí z mobilního zařízení, nebo napojením na software sledující polohu uživatele pro upravování polohy hlavní kamery.

Literatura

[1] OpenCV. *Rodrigues*, 2016. Dostupné z: http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#rodrigues.

[2] OpenCV. *Calibrate Camera*, 2016. Dostupné z: http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#calibratecamera.

[3] Raskar, R. et al. Multi-projector displays using camera-based registration. In *Visualization '99. Proceedings*, s. 161–522, Oct 1999. doi: 10.1109/VISUAL.1999.809883.

[4] Shimat. *.NET Framework wrapper for OpenCV* [online]. 2016. [cit. 26.5. 2016]. Dostupné z: https://github.com/shimat/opencvsharp.

[5] Simek, K. *Dissecting the Camera Matrix* [online]. 2016. [cit. 5.3. 2016]. Dostupné z: http://ksimek.github.io/2012/08/14/decompose/.

[6] Tamura, H. – Yamamoto, H. – Katayama, A. Mixed reality: future dreams seen at the border between real and virtual worlds. *IEEE Computer Graphics and Applications*. Nov 2001, 21, 6, s. 64–70. ISSN 0272-1716. doi: 10.1109/38.963462.

[7] Unity3D. *Execution Order of Event Functions*, 2016. Dostupné z: http://docs.unity3d.com/Manual/ExecutionOrder.html.

[8] Wikipedia. *Triangular matrix - Wikipedia, The Free Encyclopedia* [online]. 2016. [cit. 10.5. 2016]. Dostupné z: https://en.wikipedia.org/wiki/Triangular_matrix#Atomic_triangular_matrix.

[9] Wikipedia. *Newton's method - Wikipedia, The Free Encyclopedia* [online]. 2016. [cit. 26.5. 2016]. Dostupné z: https://en.wikipedia.org/wiki/Newton's_method#Nonlinear_systems_of_equations.

[10] Yannick, M. *Pinhole camera model* [online]. 2016. [cit. 17.2. 2016]. Dostupné z: http://www.epixea.com/research/multi-view-coding-thesisse8.html.

A Seznam zkratek

- LAM - Luminosity Attenuation Map (mapa snížení osvětlení)
- OpenCV - Open Computer Vision
- FPS - Frames Per Second (snímků za sekundu)
- GUID - Globally Unique IDentifier (globálě unikátní identifikátor)
- GUI - Graphical User Interace (grafické rozhraní)
- WYSIWYG - What You See Is What You Get (co vidíte, to dostanete)
- OVS - Objekty Virtuální Scény
- ORS - Objekty Reálné Scény
- OFS - Objekty Fyzické Scény

B Uživatelská příručka

Tato uživatelská příručka očekává, že uživatel je seznámený s Unity editorem. Na stránkách Unity jsou dostupné návody ovládání editoru.

B.1 Instalace

Práce byla vypracována a otestovaná v Unity 3D verze 5.3.5f1. Součástí instalace Unity je MonoDevelop, ve kterém je možné editovat skripty. Pro správný chod dále program vyžaduje nainstalovaný balíček Microsoft Visual C++ 2013 Redistributable. Toto bylo otestováno na čisté instalaci operačního systému Windows 7 (x64) na Oracle VM VirtualBox.

B.2 Unity Editor

B.2.1 Import balíčku

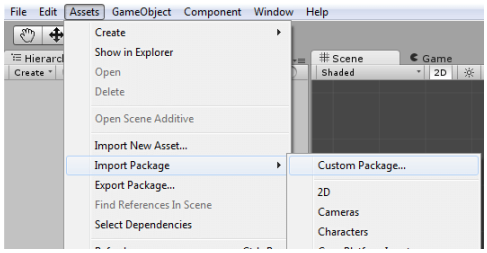
Po otevření nového projektu je potřeba importovat dodaný *unitypackage* soubor. To se provede kliknutím na záložku hlavního menu *Assets* −> *Import Package* −> *Custom Package...*. Otevře se dialog, kde se má vybrat žádaný *unitypackage*. Po vybrání balíčku se balíček rozbalí a uživateli ze zobrazí jeho obsah. Zde by uživatel měl vybrat všechny soubory tlačítkem *All* a přidat tyto soubory do projektu tlačítkem *Import* (viz obrázky B.1).

B.2.2 Vytváření scén

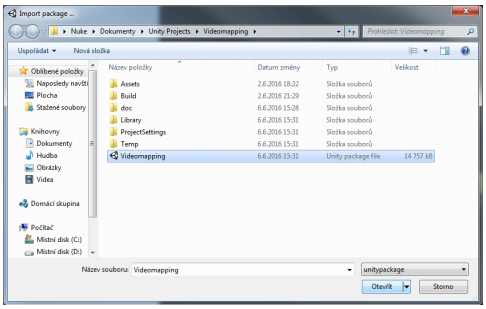
Po úspěšném importu balíčku se musí vytvořit scény. Celá scéna videomappingu jde rozdělit do tří scén. Toto rozdělení má smysl pouze pokud uživatel má v plánu používat více reálných virtuálních scén jedním programem. Když uživatel bude používat jednu reálnou scénu má smysl spojit hlavní a reálnou scénu. Naopak když bude uživatel používat jednu virtuální scénu pro více reálných, má smysl spojit hlavní a virtuální scénu.

Do těchto scén se budou přidávat jak obvyklé Unity objekty, tak i před připravené prefaby (viz kapitola 6.4). Po importu by tyto prefaby měly být dostupné mezi assety projektu v podsložce $Videomapping/Prefabs$.

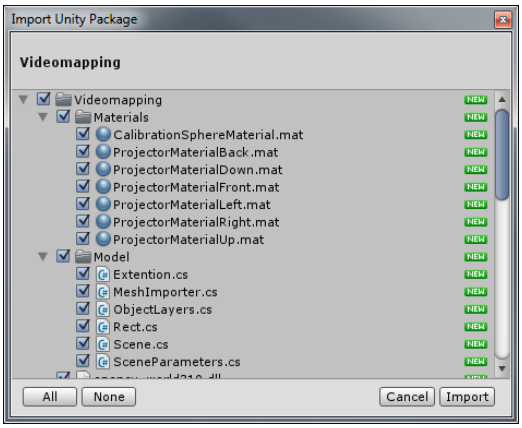
Uživatel by se také měl vyhýbat používání vlastních vrstev s čísly 8 a 9. Tyto vrstvy jsou používány pro označení reálných a virtuálních objektů pro filtrování co vidí kamery.



(a) Tlačítko v menu.



(b) Výběr balíčku.



(c) Výběr souborů z balíčku.

Obrázek B.1: Kroky importu *unitypackage* balíčku.

Hlavní scéna

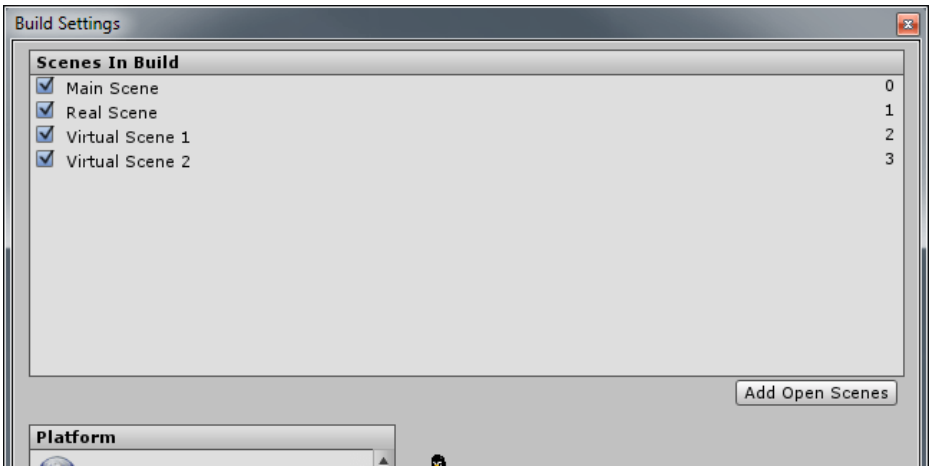
První typ scény je hlavní scéna. Tato scéna musí v programu být jen jedna a musí obsahovat prefab hlavní kamery.

Reálné scény

Reálné scény modelují fyzické scény, na které se bude promítat. Všechny objekty, které modelují fyzickou scénu, musejí mít nastavený tag na *Real Object*. Fyzické projektory jsou těsně vázané na fyzickou scénu, takže je vhodné umístit prefaby projektorů do těchto scén.

Virtuální scény

Virtuální scény obsahují virtuální objekty, které se budou promítat. Všechny tyto objekty musejí mít tag *Scene Object*, jinak nebudou brány v potaz během běhu programu.



Obrázek B.2: Výběr scén pro překlad.

B.2.3 Nastavování parametrů

Po vytvoření scén je potřeba nastavit použité prefaby, tedy hlavní kameru a projektory. Oba tyto prefaby obsahují komponentu *Camera*, které je potřeba nastavit viewport. Viewport označuje do jaké oblasti obrazovky (velikost obrazovky je normalizovaná do čtverce velikosti 1×1) bude tato kamera posílat svůj výstup. Dále se ještě může nastavit zorný úhel. Ostatní parametry kamery jako vyosení a poměr stran se nastavují v komponentě *CameraParameters*.

Hlavní kamera dále obsahuje komponenty pro nastavení kalibračního vzoru a barvy a velikosti kalibrační koule a rychlosti manuálního pohybu hlavní kamery (translace a rotace). Prefab projektoru na druhou stranu obsahuje komponentu pro nastavení gamma korekce fyzického projektoru pro správné nastavení osvětlení překryvů zorných polí projektorů.

B.2.4 Nastavování akcí

Uživatel si může vytvořit nové akce vytvořením nové třídy, která dědí od *ActionBehaviour* (více v kapitole 6.6). Akce jdou přiřazovat všem objektům v libovolné scéně, takže třeba i hlavní kameře.

B.2.5 Překlad

Při nastavení překladu je potřeba přidat všechny scény, které mají být během běhu programu používány. Tyto scény se přidávají přetažením do dialogu nastavení překladu jako na obrázku B.2. Hlavní scéna musí mít index 0, aby byla při startu programu automaticky načtená.

B.3 Běh programu

Po spuštění správně přeloženého projektu se zobrazí hlavní menu. Hlavní menu provází uživatele kroky před začátkem samotného videomappingu.

B.3.1 Hlavní menu

Hlavní menu obsahuje 7 kroků, které musí uživatel projít, aby byl videomapping připraven. Poslední krok pouze hlásí uživateli, že je všechno připraveno a nebude zde popsán. Náhled ostatních kroků je na obrázcích B.3.

Vybírání scény

Během prvního kroku si uživatel vybírá až dvě scény, které se mají načíst. Pokud chce načíst pouze jednu nebo žádnou, musí vybrat první volbu nenačtení žádné scény. Pokud program nenalezne žádnou další scénu, nastaví oba výběry na tuto volbu sám. Pokud uživatel vybere jednu scénu dvakrát, scéna se načte pouze jednou. K tomuto kroku není možno se vrátit, takže pokud uživatel udělal během tohoto kroku chybu, musí restartovat program.

Načtení *obj* souborů reálné scény

Další krok umožňuje importovat další objekty reálné scény z *obj* souboru. Po zadání cesty k souboru a kliknutí na tlačítko *Load* se program pokusí načíst soubor. Pokud se soubor podaří načíst, vymaže se zadaná cesta a je možno načíst další soubory.

Načtení parametrů nebo kalibračních bodů

V tomto kroku se uživatel rozhoduje zda bude provádět kalibraci projektorů nebo přesné parametry projektorů načte ze souboru. Pokud se rozhodne kalibrovat projektory, má ještě možnost načíst jiný kalibrační vzor ze souboru. Pokud se uživatel rozhodne načíst přesné parametry projektorů, další dva kroky se nebudou provádět.

Kalibrace projektorů

Během kalibrace projektorů se budou procházet všechny projektory v načtených scénách. Pro každý projektor se bude procházet a zobrazovat postupně všechny body kalibračního vzoru. Uživatel musí klikat na ploše obrazovky projektoru na místa, kde by právě zobrazený bod měl být promítán. Pokud

daný bod není možné zobrazit projektorem, protože je mimo zorné pole, kliknutím na *Next Point* se tento bod přeskočí.

Uživatel může ukončit kalibraci právě kalibrovaného projektoru kliknutím na tlačítko *Next Projector*. Pokud bylo nakalibrováno alespoň 6 bodů pro určitý projektor a uživatel ukončí kalibraci daného projektoru vybráním dalšího projektoru nebo kliknutím na tlačítko *Next*, program se pokusí daný projektor nakalibrovat.

Uložení parametrů

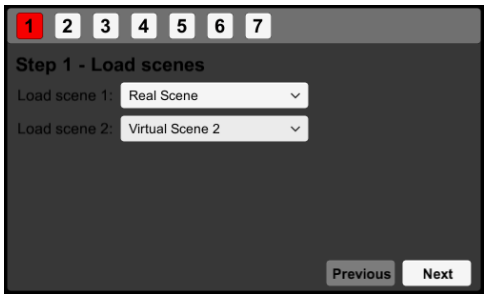
Tento krok má dva účely. Po skončení kalibrace zobrazí scénu pro rychlé zhodnocení úspěšnosti kalibrace projektorů. Pokud je uživatel spokojený s výsledky, má v tomto kroku možnost uložit kalibrované parametry do souboru společně s kalibračním vzorem, který byl použit. Tento soubor je při dalších zapnutí programu možno načíst v kroku 3 pro přeskočení kalibrace.

Registrace projektorů

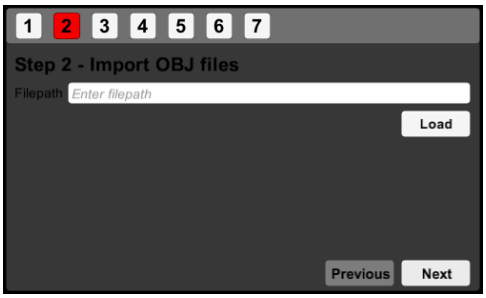
Pokud scéna obsahuje vysoký počet kamer, může registrace projektorů trvat delší dobu a proto byl vytvořen tento krok. Během registrace projektorů se zobrazuje progress bar, který ukazuje časový průběh registrace.

B.3.2 Videomapping

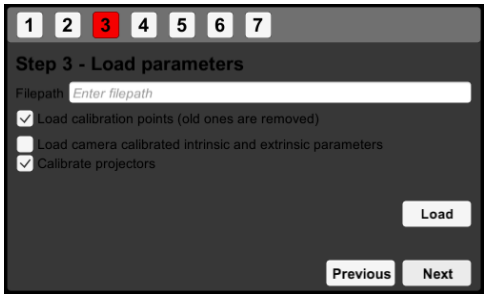
Po průchodu kroky hlavního menu se zobrazí scéna a videomapping může začít. Za běhu videomappingu jsou zobrazeny dvě menu (viz obrázek B.4). Jedno zobrazuje pozici hlavní kamery, neboli pozici správné perspektivy. Druhé menu se zobrazí pouze pokud načtená scéna obsahuje alespoň jednu akci. Toto menu zobrazuje všechny akce, které mají nastaveno, že mají být vidět za běhu programu. Uživatel je může procházet a podle jména vybrat akci, kterou chce aktivovat. Samotnou aktivaci provede tlačítkem *ExecuteAction*.



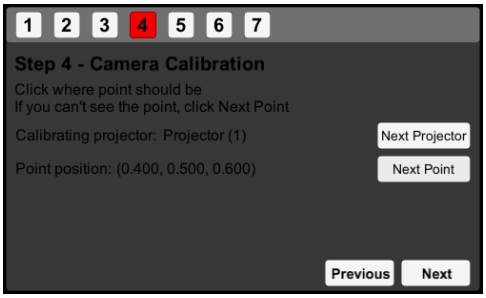
(a) Krok 1.



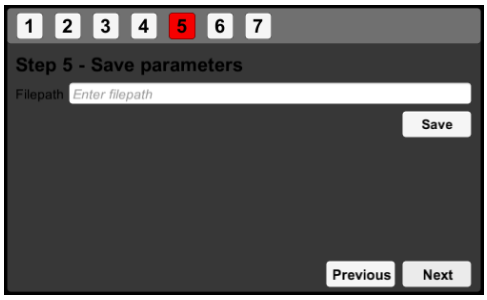
(b) Krok 2.



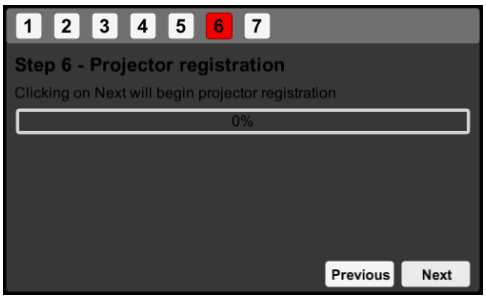
(c) Krok 3.



(d) Krok 4.

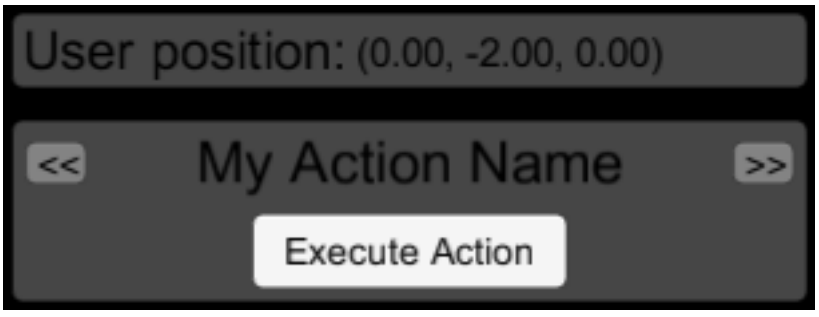


(e) Krok 5.



(f) Krok 6.

Obrázek B.3: Kroky hlavního menu.



Obrázek B.4: Menu zobrazené po dokončení kroků hlavního menu.