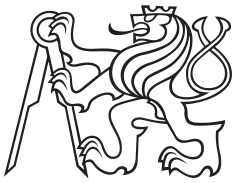
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická

Manipulátory pro editaci transformačních matic a skriptování v I3T

Daniel Gruncl

Školitel: Ing. Sloup Jaroslav
Květen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Gruncl** Jméno: **Daniel** Osobní číslo: **483765**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Manipulátory pro editaci transformačních matic a skriptování v I3T

Název bakalářské práce anglicky:

Transformation manipulators and scripting in I3T

Pokyny pro vypracování:

Prostudujte existující možnosti interaktivní editace transformací / transformačních matic přímo v 3D scéně pomocí manipulátorů (rotace, posun, změna měřítka, atd.) v dostupných modelářích (např. Maya, Blender, 3D Max). Na základě studie navrhněte a implementujte sadu manipulátorů, kterou zaintegrujete do existujícího nástroje pro výuku transformací I3T tak, aby rozšiřovala stávající možnosti editace specifických transformačních matic. Použitelnost implementovaných manipulátorů ověřte pomocí testů s uživateli.
Rozšiřte I3T o možnost skriptování a spouštění externích skriptů. Skriptovaci jakyk nechť obsahuje takovou sadu operací, která umožní kompletní vytvoření scény, tj. vytvoření transformačních matic, sekvencí, operátorů a jejich vzájemné propojení pomocí hran do acyklického grafu. Skriptování využijte k ukládání a načítání popisu celé scény. Skriptem bude tedy možné popsat vytvoření celé scény stejně, jako by byla načtena ze stávajícího formátu souborů pro popis scén v I3T.

Seznam doporučené literatury:

[1] Michal Folta: Teaching of Transformations. Diplomová práce, ČVUT FEL, 2016.
[2] Petr Felkel, Alejandra Magana, Michal Folta, Alexa Gabrielle Sears, Bedřich Beneš: I3T: Using Interactive Computer Graphics to Teach Geometric Transformations. Eurographics Education Papers 2018, http://www.i3t-tool.org/
[3] Daniel Aschermann: Editor geometrie v systému VRUT. Bakalářská práce, ČVUT FEL, 2020.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jaroslav Sloup, Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **12.02.2021** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **30.09.2022**

Ing. Jaroslav Sloup podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání Podpis studenta

Poděkování

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Abstrakt

Cílem této práce je návrh manipulátorů interaktivní úpravy transformačních a projekčních matic a začlenění těchto manipulátorů do pro nástroj pro výuku transformací I3T. Návrh je učiněn na základě prostudování manipulátorů jiných 3D prostředí. Návrh je podroben testům s uživateli. Součástí práce je také rozšíření I3T o možnost zpracovávání externích skriptů.

Klíčová slova: I3T, skriptování, 3D transformace, manipulátory transformací, interaktivní transformace

Školitel: Ing. Sloup Jaroslav
E-413,
Karlovo náměstí 13,
12000 Praha 2

Abstract

The goal of this work is design of manipulators for interactive modification of transformation and projection matrices and the integration of these manipulators into I3T – software for teaching transformations. The design is based on study of manipulators in other 3D environments. The design is subjected to tests with users. Part of the work is also extending I3T with functionality of executing external scripts.

Keywords: I3T, scripting, 3D transformations, transformations manipulators, interactive transformations

Title translation: Transformation manipulators and scripting in I3T

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Nástroj pro výuku 3D transformací I3T umožňuje editaci transformačních matic s vizualizací provedených změn v 3D scéně, ale chybí funkcionalita opačná, tedy provádět editace v 3D scéně a sledovat změny transformačních matic. Cílem této je práce navrhnout manipulátory, neboli ovládací prvky pro nástroj I3T, které by poskytovaly tuto funkcionalitu. Takové manipulátory mají pomoct uživatelům aplikace I3T spatřit souvislost mezi transformacemi a jejich maticovou reprezentací.

V práci se postupně budu zabývat rozborem existujících řešení, poté jejich zhodnocením a návrhem vlastního řešení, začleněním návrhu do nástroje I3T a implementací a uživatelskými testy. Návrh obsahuje manipulátory pro úpravu rotace, translace, změny měřítka (škálování), projekčních matic, matice lookAt, což je speciální případ rotační matice, a pro úpravu jednotlivých sloupců obecné matice.

Dalším cílem je rozšířit I3T o interpretr externích skriptů a definovat pro tento interpretr sadu operací, umožňující kompletní uložení a načtení stavu scény I3T.

1.1 O I3T

I3T je program sloužící k výuce 3D transformací vyvíjený na katedře počítačové grafiky a interakce. V době psaní této práce je I3T v rozpracovaném stavu a mnoho funkcionalit chybí, proto zde popíšu funkcionality I3T na jeho starší verzi, vytvořené v rámci diplomové práce Michala Folty [1]. Na obrázku 1.1 vidíme hlavní okno I3T, kde se zobrazuje 3D scéna a záložka workspace – pracovní plocha.

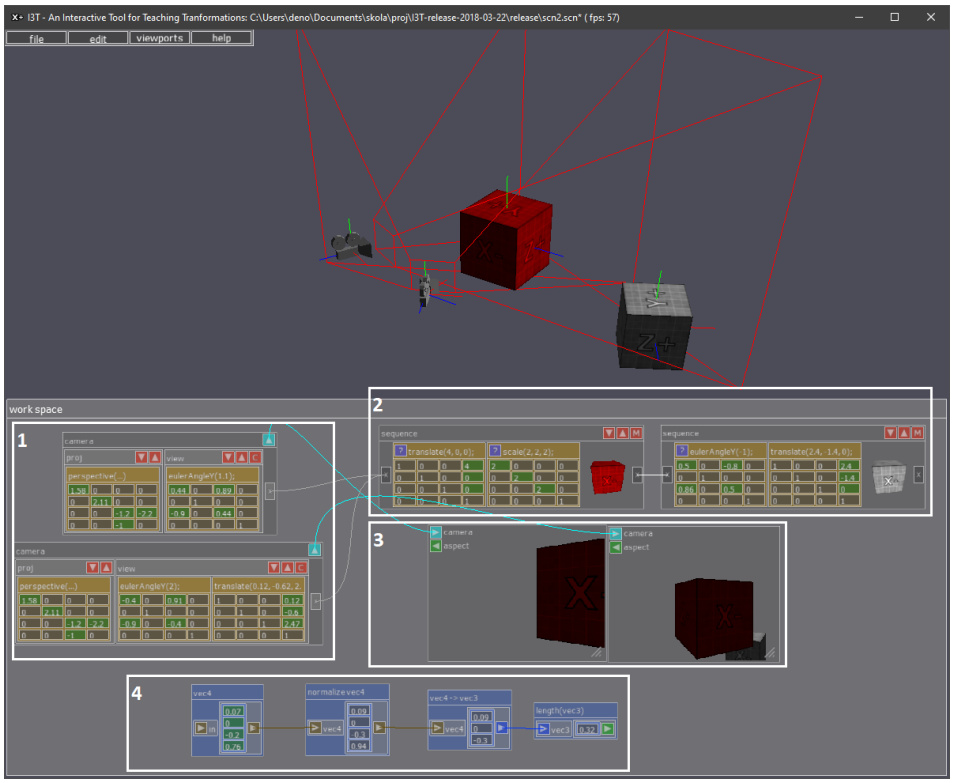
V rámečku 1 jsou dvě sekvence pro nastavení kamery, obsahující matici projekce a matici pro směr pohledu. V rámečku 3 se zobrazují záběry kamer.

V rámečku 2 jsou dvě propojené sekvence transformačních matic. Změny transformačních matic jsou v reálném čase zobrazeny ve 3D scéně. Výsledná transformace každé sekvence vznikne pronásobením všech jejích matic s výslednou transformací sekvence, která je jejím předchůdcem, tedy připojená k ní zleva.

V rámečku 4 vidíme ukázky několika matematických operátorů. I3T obsahuje množství takovýchto krabiček, umožňující provádět různé matematické

operace a konverze na vektorech, maticích i skalárních hodnotách. Operátory v rámečku 4 jsou, zleva doprava, tyto: vektor4, normalizace, konverze na vektor 3 a délka vektoru 3.

Políčka matic, zobrazená zeleně, lze upravovat. K sekvenci lze připojit 3D model, jak také vidíme na obrázku, který je pak zobrazen výslednou transformací své sekvence. Na obrázku 1.1 také vidíme, že existují různé typy



Obrázek 1.1: Prostředí I3T.

matic, povolující pouze transformace jejich typu. Výčet typů transformačních matic v I3T je tento:

- free – matice, kterou je možno upravovat bez omezení.
- translation – matice translace.
- eulerAngleX, eulerAngleY, eulerAngleZ – matice, umožňující rotaci kolem jedné osy.
- rotate – matice, která je inicializována osou rotace a úhlem.
- quat – matice rotace kolem tří os, která je zadávaná jako kvaternion, hodnoty matice nelze přímo měnit.
- scale, uniform scale – matice pro škálování a uniformní škálování.
- lookAt – matice rotace a translace, jejíž osa Z směřuje k zadanému bodu v prostoru. Transformační matice jsou podrobněji popsány v další sekci.

- ortho – matice ortografické projekce.
- perspective – matice perspektivní projekce.
- frustum – matice perspektivní projekce, kde střed symetrie (bod, ke kterému ubíhají vzdalující se linky) nemusí ležet ve středu obrazovky a nemusí ani ležet v obrazovce. Využití takové projekce je například, když chceme vykreslit snímek o vyšším rozlišení, než je limit OpenGL. V takovém případě se snímek složí ze snímků několika kamer, které mají jeden společný střed symetrie [3].

I3T ve své finální podobě má také umožňovat různé maticových a vektorové operace, například normalizace vektoru, inverze matice, násobení matice vektorem..., ale ty podrobněji rozebírat nebudu, protože jsou mimo rozsah této práce.

1.2 Úvod do maticových transformací

Matice se násobí zprava doleva, jinými slovy na matici se aplikují transfromace matic zleva od ní. V praxi to znamená, chceme-li transformovat v globálním prostoru, transformační matici umístíme vpravo od transformované matice, chceme-li transformovat v prostoru dané matice, násobíme zleva. V případě hiearchie matic to znamená, že v sekvenci matic rodiče matice najdeme vlevo od ní, matici potomka pak vpravo.

1.2.1 Typy transformací

Matice rotace. Matice, jejíž první tři sloupce jsou ortonormální a jejich souřadnice w je 0 a čtvrtý sloupec je čtvrtým sloupcem matice identity. Ortogonální matice obecně je rotace a zrcadlení, ale zda matice obsahuje i zrcadlení je těžké pohledem poznat.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos & -sin & 0 \\ 0 & sin & cos & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} cos & 0 & sin & 0 \\ 0 & 1 & 0 & 0 \\ -sin & 0 & cos & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} cos & -sin & 0 & 0 \\ sin & cos & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Matice rotací[2] v osách X, Y a Z, zleva doprava

Translační matice. Matice identity, k jejímuž čtvrtému sloupci je přičten vektor translace v osách X,Y,Z.

$$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

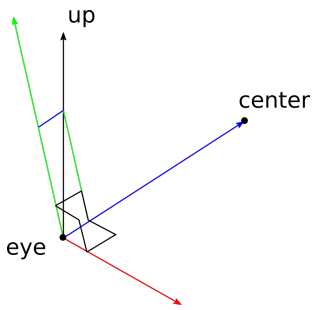
Matice translace

Matice změny měřítka. Matice identity, jejíž první tři sloupce jsou vynásobeny nenulovým skalárem.

$$\begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

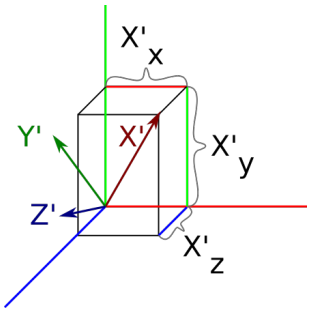
Matice změny měřítka v osách X, Y, Z

Matice lookAt. Matice rotace a translace, jejíž osa Z (třetí sloupec) směřovuje do zadaného bodu v prostoru, neboli, transformuje objekt nacházející se na zadaném bodu v prostoru tak, aby se „díval“ na zadaný druhý zadaný bod v prostoru. Směr Z je tedy rozdíl těchto dvou bodů, jak je zobrazeno na obrázku 1.2.



Obrázek 1.2: Význam matice lookAt.

Matice volné transformace. Matice, umožňující libovolnou transformaci bázových vektorů. První tři sloupce této matice jsou vektory nové báze vyjádřené v souřadnicích původní báze, poslední sloupec je translace oproti původní bázi. Na obrázku 1.3 vidíme vektory Y' a Z', představující druhý a třetí sloupce této matice. X', u kterého jsou zobrazeny i jednotlivé složky X'_x, X'_y a X'_z reprezentuje první sloupec této matice.



Obrázek 1.3: Význam volné transformace.

Projekční matice. Matice, která provádí posun a škálování, čímž se určuje která a jak velká část světa bude zobrazena na obrazovce. Převrácený poměr škálování v osách X a Y by měl odpovídat poměrům stran obrazovky, aby obraz nebyl stlačený nebo roztáhnutý. Poměr je převrácený, protože aby byla zobrazena část světa o šířce w, škálování projekční maticí v ose X bude $1/w$.

Perspektivní projekční matice je navržena tak, aby po vynásobení vektorem výsledný vektor měl ve své w souřadnici hodnotu $-z$ původního vektoru. Výsledný vektor se poté vydělí svou hodnoutou w, což je $-z$ původního vektoru. Toto způsobí perspektivní zkreslení – souřadnice x a y jsou vyděleny $-z$, což znamená, že čím dál je bod od kamery, tím více se přiblíží ke středu symetrie, což je bod $(0, 0)$ v prostoru viewportu a zároveň je to i střed obrazovky v případě symetrické perspektivní matice.

Hodnota z je se záporným znaménkem proto, že v OpenGL poloosa $Z-$ směřuje vpřed, tudíž všechny body před kamerou mají souřadnici z zápornou, což by po perspektivním dělení $1/w$ způsobilo převrácení obrazu.

Perspektivní matice je symetrická právě tehdy, když

$$(l, b) = -(r, t), \tag{1.1}$$

kde l je levá hranice záběru, r pravá, b spodní a t vrchní.

$$\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Matice ortografické a perspektivní projekce[3]

Kapitola 2
Návrh manipulátorů

Při návrhu manipulátorů je vhodné se pro lepší uživatelskou přívětivost držet zažitých konvencí, proto by měl návrh manipulátorů vycházet z implementací v běžných 3D prostředích. Pozorovanými prostředími jsou Maya, Blender, Unity a Unreal engine.

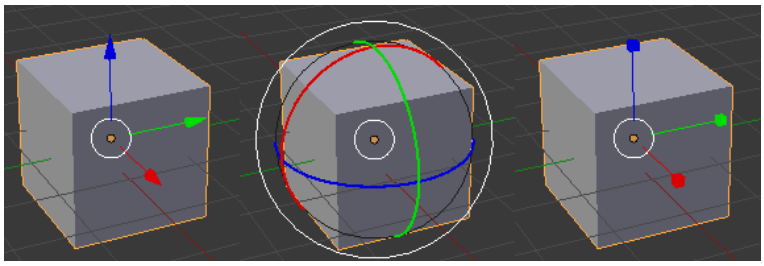
Různá prostředí používají různé souřadné systémy, pro přehlednost se budu v celé práci držet systému, kde osa Z směřuje vpřed, osa X doprava a osa Y nahoru.

2.1 Rozbor jiných 3D prostředí

V této části rozeberu vlastnosti manipulátorů různých typů transformací v prostředích Unity, Blender, Unreal Engine a Maya. Zatímco prostředí Unity a Blender znám jako uživatel, při rozboru manipulátorů v Unreal Engine a Maya jsem vycházel z online dokumentací a manuálů[1][4][5].

Standardními vlastnostmi manipulátorů zmíněných prostředí je transformace v globálním a modelovém prostoru, škálování uniformní nebo v jednotlivých osách, translace v jednotlivých osách, rotace v jednotlivých osách.

Na obrázku 2.1 vidíme obvyklou podobu manipulátorů pro transfromace v jednotlivých osách. Tyto vlastnosti nebudu rozepisovat u každého zvlášť, vypíšu pouze ty odlišné od zde popsaného standardu.



Obrázek 2.1: Obvyklá podoba manipulátorů translace, rotace, škálování.

[1]Obrázky manipulátorů Maya a Unreal Engine taktéž pocházejí ze zdrojů [4][5].

2.1.1 Blender

Blender kromě prostoru globálního a modelového nabízí také prostor pohledový, tedy prostor z pohledu kamery. Při editování transformace Blender skryje manipulátory, které pak nezakrývají model, který uživatel edituje.

Rotace

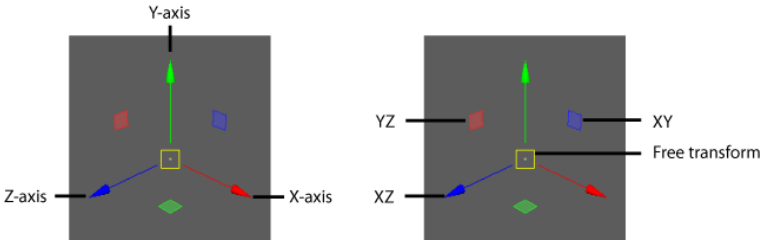
Okolo kruhů pro manipulaci rotace v jednotlivých osách je ještě čtvrtý kruh, umožňující transformaci v ose Z z pohledu kamery bez ohledu na zvolený prostor transformace. Blender umožňuje také rotaci v prostoru „Gimbal“, česky Kardanův závěs. V tomto prostoru jsou rotační osy na sobě závislé, nemusí tedy být navzájem kolmé. Rotace se chová jako by model byl zavěšen v soustavě tří prstencu navzájem zavěšených do sebe.

2.1.2 Maya

Maya nabízí možnost editace v prostoru lokálním, což je prostor prvku, nadřazeného danému modelu v hiearchii scény. Je-li model v kořeni scény, je tento prostor totožný s globálním.

Translace

V počátku šipek pro transformaci v jednotlivých osách se nachází čtvereček, umožňující transformaci v rovině XY v pohledu kamery bez ohledu na zvolený prostor transformace. Po bocích šipek se nachází další čtverečky, celkem tři, které umožňují translaci v rovině dvou šipek, přiléhajících k danému čtverečku.



Obrázek 2.2: Manipulátory translace v Maya.

Rotace

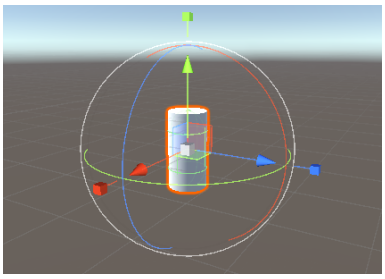
Okolo kruhů pro manipulaci rotace v jednotlivých osách je ještě čtvrtý kruh, umožňující transformaci v ose Z z pohledu kamery bez ohledu na zvolený prostor transformace.

Škálování

Škálování je možné také v rovině, stejně jako u translace (obrázek 2.2). V počátku ružice šipek pro škálování se nachází kostička, tahem za kterouž se provádí uniformní škálování.

2.1.3 Unity 3D

V Unity lze klávesami W, E, R, T, Y přepínat mezi manipulátory translace, rotace, škálování, škálování ve dvou osách současně (popsáno níže) a poslední možnost, obrázek 2.3, zobrazí první tři typy manipulátorů zároveň. Při najetí myší na manipulátor se manipulátor vybarví bíle, čímž se naznačí uživateli, že daný prvek je interaktivní, což je užitečné pro prvky, u kterých není zcela očividné, že se dají ovládat, například transformace a škálování ce dvou osách současně, jak je popsáno níže.



Obrázek 2.3: Kombinované manipulátory v Unity.

Translace

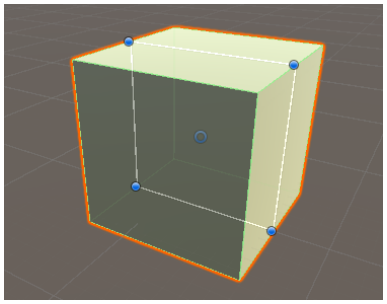
Po bocích šipek se nachází další čtverečky, celkem tři, které umožňují translaci v rovině dvou šipek, přiléhajících k danému čtverečku, podobně jako v Mayi.

Rotace

Okolo kruhů pro manipulaci rotace v jednotlivých osách je ještě čtvrtý kruh, umožňující transformaci v ose Z z pohledu kamery bez ohledu na zvolený prostor transformace.

Škálování

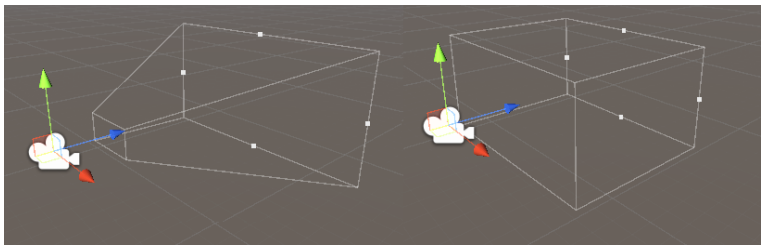
V počátku šipek pro škálování se také nachází kostička, tahem za kterouž se provádí uniformní škálování. Unity 3D také nabízí speciální manipulátor pro škálování ve dvou osách současně. Na obrázku 2.4 je vidět obdélník ležící v rovině aktivního prostoru transformací, v té, která je nejkolmější na směr pohledu kamery. Tahem za rohy tohoto obdélníku se provádí škálování ve dvou osách současně.



Obrázek 2.4: Manipulátory škálování v rovině v Unity 3D.

Projekční matice

Unity nabízí také manipulátory pro projekční matice, i když s omezenou funkcionalitou. Jak lze vidět na onrázku níže (2.5), na hranách přední stěny kvádru reprezentujícího zorné pole kamery, se nachází čtverečky, kterými lze měnit šíři a výšku záběr. Poměr stran je pevně daný poměrem stran obrazovky. V případě perspektivní projekce se mění úhel záběru.

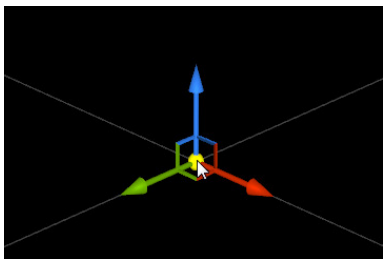


Obrázek 2.5: Manipulátory projekčních matic v Unity.

2.1.4 Unreal Engine

Translace

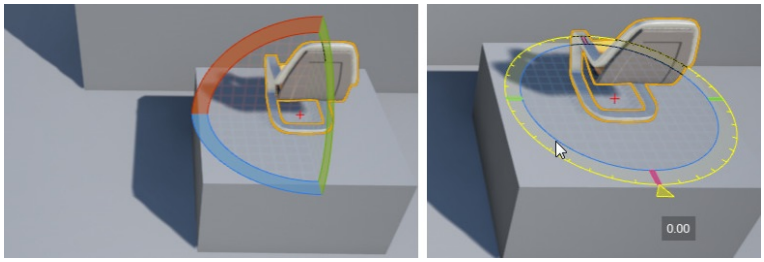
Po bocích šipek se nachází další čtverečky, celkem tři, které umožňují translaci v rovině dvou šipek, přiléhajících k danému čtverečku, podobně jako v Mayi. Pomocí kuličky uprostřed se provádí translace v rovině obrazovky. Stejně jako v Unity, manipulátory se barevně zvýrazní při najetí myší.



Obrázek 2.6: Manipulátory translace v Unreal engine.

Rotace

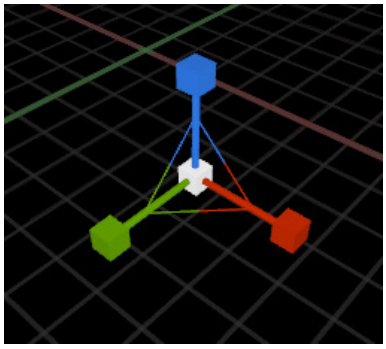
Manipulátory pro rotaci jsou jiné než v ostatních pozorovaných prostředích, jak ukazuje obrázek 2.7. Při provádění rotace se zobrazí pouze jeden kruh, ležící v rovině kolmé na osu rotace, zobrazující úhel, o kolik je model rotován oproti stavu před započetím provádění rotace.



Obrázek 2.7: Manipulátory rotace v Unreal engine.

Škálování

Škálování je možné také v rovině, pomocí tří trojúhelníku, nacházejících se v počátku šipek pro škálování, jak ukazuje obrázek 2.8. V počátku šipek pro škálování se také nachází kostička, tahem za kterouž se provádí uniformní škálování.



Obrázek 2.8: Manipulátory škálování v Unreal engine.

2.1.5 Souhrn

Manipulátory pro úpravu rotace, translace a měřítka mají ve všech pozorovaných prostředích stejnou podobu, až na rotaci v Unreal Engine. V Unity, Mayi a Unreal Enginu jsou také manipulátory úpravy měřítka či translace ve dvou osách současně. S výjimkou škálování v Unity jsou udělané vždy stejným způsobem. Unity má svůj vlastní manipulátor pro škálování v rovině, ale vždy se zobrazuje pouze pro jednu ze tří rovin – je navržený speciálně pro vytváření 2D scén, což se pro I3T nehodí.

Kombinované manipulátory jako v Unity nebude možné implementovat, opět z důvodu že I3T umožňuje na každé matici jeden určitý typ transformace.

Zajímavá je ale možnost přepínání mezi manipulátory různých transformací pomocí klávesnice. Při přepnutí na jiný typ manipulátorů by se výběr (focus) ve workspace přesunul na matici umožňující onen typ transformace, která je zároveň potomkem nebo předkem vybrané matice v sekvenci transformací, pokud taková matice existuje.

Pozorovaná prostředí nabízejí možnost uniformího škálování (s výjimkou Blenderu). Prostředí do určité míry umožňují i manipulaci v prostoru obrazovky, například rotace podle osy pohledu kamery (s výjimkou Unreal Engine) nebo translace v prostoru obrazovky (Maya, Unreal Engine).

V Unity je manipulátor pro projekční matice, v ostatních prostředích manipulátory pro méně obvyklé transformace, jako projekční matice, nebo lookAt (což je speciální případ rotační matice) vůbec nejsou.

2.2 Vlastní návrh

Manipulátory pro rotaci, škálování a translaci v jedné ose budou mít podobu jako na obrázku 2.1 (strana 7), je to obvyklá podoba a není důvod ji měnit.

Možnost výběru prostoru transformace nebude možné začlenit do I3T kvůli limitujícím vlastnostem. Systém I3T je totiž navržen tak, že každá matice má svůj typ (rotace okolo osy X, škálování, translace, ...) a lze na ní provádět pouze transformace jejího typu, přechod do obecného prostoru by tedy byl možný pouze s obecnou maticí (v I3T typ free). Navíc transformace v obecném prostoru je pro výukové účely nevhodná, protože mění velkou část matice, kvůli čemuž není snadné vidět souvislost mezi maticí a transformací.

2.2.1 Ovládání manipulátorů

Při vybrání matice ze sekvence (klikem myši) se zobrazí manipulátory pro daný typ matice. Umožňuje-li vybraný typ matice transformaci jenom v jedné ose, pak se manipulátory pro ostatní osy zobrazí vyšeděné, neaktivní. Pro matice rotace se zobrazí manipulátory rotace, pro translaci manipulátory translace, pro škálování manipulátory škálování, pro matici typu free se zobrazí manipulátory pro upravování jednotlivých sloupců, pro ortografické a perspektivní projekční matice se zobrazí manipulátory odpovídající jejich typům.

Při najetí myší se manipulátor barevně zvýrazní, aby bylo uživateli zřejmé, že jde o interaktivní prvek. Blender při probíhající editaci (poté, co uživatel klikne na manipulátor) skryje manipulátory, což se hodí pro modelování, ale v I3T jsou editovány matice, ke kterým často nebude navázaná žádná geometrie a pak by uživatel neměl přehled co dělá, proto jsem se rozhodl manipulátory neskrývat.

Manipulátory budou vždy upravovat pouze jednu matici sekvence a budou se zobrazovat ve scéně na pozici matice. Pozici matice definujeme jako translační část (první tři hodnoty třetího sloupce) matice P, vzniklé vynásobením matic

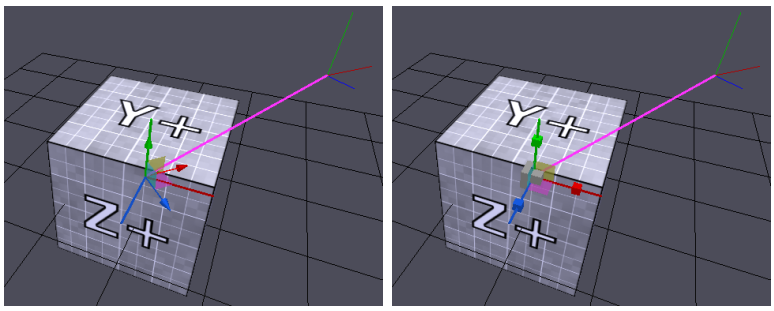
$$P = (S_1M_1 \cdot S_1M_2 \cdot \ldots \cdot S_1M_x) \cdot (S_2M_1 \cdot S_2M_2 \cdot \ldots \cdot S_2M_y) \cdot \ldots \cdot (S_zM_1 \cdot S_zM_2 \cdot \ldots \cdot S_zM_w), \tag{2.1}$$

kde S_aM_b je b-tá matice a-té sekvence , jak je popsáno v úvodu (strana 1) a S_zM_w je editovaná matice.

Manipulátory se budou ovládat levým tlačítkem a tahem myši. Při najetí myší manipulátor zvýrazní a zároveň se změní kurzor myši na ručičku. V některých případech totiž nemusí být zřejmé, že některý prvek vykreslený na obrazovce je manipulátor, například manipulátor pro uniformní škálování.

2.2.2 Translace a škálování

Škálování a translace budou také možné ve dvou osách současně, jako v Unreal engine nebo Maya. Metodu škálování v rovině použitou v Unity 3D jsem se rozhodl nepoužít, protože vždy zobrazuje manipulátor pouze pro jednu rovinu (Obrázek2.4). Také to znamená mít dvě separátní sady manipulátorů pro škálování, jednu pro škálování v jedné ose a druhou pro škálování v rovině, což by bylo nekonzistentní s ostatními manipulátory. Uniformí škálování se bude provádět kostičkou ve středu manipulátorů, jako v Unity 3D, Maya a Unreal engine. Zde na obrázku 2.9 vidíme podobu návrhu:



Obrázek 2.9: Manipulátory pro translaci a škálování v I3T.

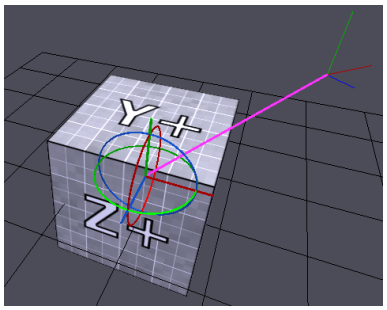
2.2.3 Rotace

Rotace se provádí klikutím a táhnutím za kruh, který lze vidět na následujícím obrázku 2.10. Jsou různé způsoby jak pohyb, popřípadě pozici myši převést na rotaci. V Blenderu se používá metoda, kde uživatel krouží myší okolo bodu na obrazovce, kde se nachází objekt. Tato metoda není začátečníkovi zřejmá, proto Blender poskytuje nápovědu tím, že při rotaci se kurzor myší změní v ikonku naznačující rotaci okolo bodu. Grafická knihovna ImGui, kterou I3T používá, ale neumožňuje vytváření vlastních kurzorů myši.

V ostatních prostředích se používá metoda, kdy míra rotace je úměrná tahu myši podél osy. Tuto metodu lze také implementovat různými způsoby. Nejjednodušší možnost je měřit vzdálenost od místa kliknutí, ale to umožňuje rotaci pouze jedním směrem (vzdálenost je vždy kladná). Další možnost je měřit pohyb podél kolmice na osu rotace. Toto funguje dobře, pokud se úhel mezi osou rotace a rovinou obrazovky blíží 0. Tehdy se manipulátor rotace, tedy kruh, promítá na obrazovku jako velmi protáhlá elipsa, podobná čáře, a uživatel bude intuitivně táhnout myší ve směru protažení elipsy, což

je totéž co kolmice na osu rotace. V případě že se tento úhel ale blíží pravému úhlu, kruh se promítá na obrazovku jako kruh a uvedená intuice zde již nezapůsobí.

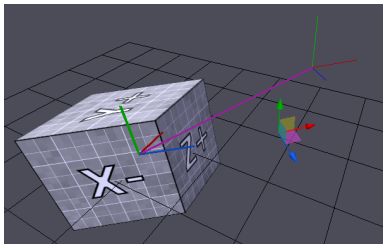
V tomto případě je nejintuitivnější táhnout myší ve směru tečny na kruh v místě kliknutí. Tato metoda pokrývá i předchozí případ, protože pro velmi protáhlou elipsu je na většině její hranice tečna blízká kolmici na osu rotace.



Obrázek 2.10: Manipulátory pro rotaci v I3T.

2.2.4 lookAt

Manipulátory pro editaci matice lookAt nejsou v pozorovaných prostředních implementovány, je to totiž dost specifický případ transformace, který se v praxi používá zřídka. Vzhledem k tomu, že lookAt se „dívá“ na nějaké místo v prostoru, rozhodl jsem se znovupoužít manipulátory pro translaci – matice se bude transformovat tak, aby její osa Z ukazovala na místo, kde se nachází ružice šipek – manipulátor translace. Manipulátorem lze volně pohybovat tahem za jeho šipky a plošky.



Obrázek 2.11: Manipulátory pro LookAt v I3T.

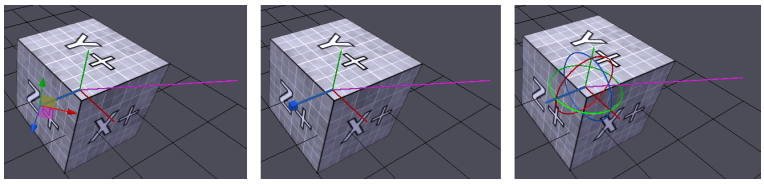
2.2.5 Volná transformace

Volná transformace umožňuje upravovat libovolné políčko matice bez jakýchkoliv omezení. Manipulátory pro takové editace se v pozorovaných prostředích nevyskytují z obdobného důvodu jako pro matici lookAt. Proto jsem opět přistoupil k vlastnímu návrhu, který spočívá v tom, že se zobrazí čtyři linky, reprezentující čtyři sloupce matice. Linky reprezentující první tři sloupce

začínají v pozici p_1, což je pozice daná transformační sekvencí končící upravovanou maticí včetně a končí v pozici $p_1 + s_i$, kde s_i jsou první tři hodnoty prvního, druhého, nebo třetího sloupce.

V případě čtvrtého sloupce, reprezentujícího translaci, je provedena drobná úprava, kdy linka začíná v p_1 ale končí v $p_1 - s_4$, kde s_4 jsou první tři hodnoty čtvrtého sloupce. Čtvrtá linka tedy končí v pozici dané transformační sekvencí končící předkem upravované matice včetně.

Vybraný sloupec (reprezentovaný linkou) lze škálovat, rotovat, nebo posunem jeho konce provádět obecnou úpravu. Manipulátory pro tyto tři operace jsou zobrazeny separátně a pouze pro jednu osu současně. Mezi režimem transformace lze přepínat klávesami `r`, `s` a `t`, transformovanou osou lze přepínat klávesami `x`, `y`, `z` a `w`, nebo kliknutím na osu. Na obrázku 2.12 jsou zobrazeny všechny tři módy při transformaci osy Z. Osy X, Y, Z, W jsou reprezentované popořadě červenou, zelenou, modrou a fialovou linkou.

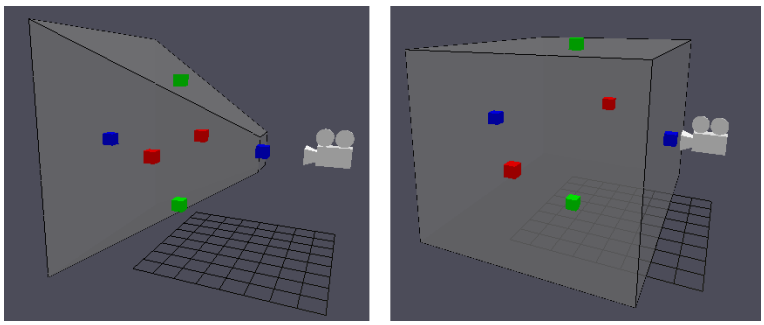


Obrázek 2.12: Manipulátory pro úpravu jednotlivých sloupců matice.

2.2.6 Projekční matice

Manipulátory pro editaci projekčních matic jsou implementovány pouze v Unity 3D, a navíc ještě umožňují pouze editaci šíře záběru. Proto jsem navrhl vlastní sadu manipulátorů pro editaci matic perspective, frustrum a orthographic (obrázek 2.13).

Při editaci projekční matice se zobrazí poloprůhledný komolý jehlan pro perspektivní projekci nebo kvádr pro ortografickou projekci, reprezentující zorné pole kamery. Modrými kostičkami uprostřed přední a zadní plochy tvaru se upravuje far a near, červenými kostičkami se edituje left a right (width pro projekci typu perspective), zelenými kostičkami nahoře a dole se upravuje up a bottom (height pro projekci typu perspective).



Obrázek 2.13: Manipulátory pro úpravu projekčních matic.

Kapitola 3

Začlenění manipulátorů do I3T

Při začlenění manipulátorů do aplikace I3T je potřeba navázat na ostatní části aplikace, jako renderování 3D scény, kde se budou manipulátory zobrazovat a workspace, které uchovává matice, které budou upravovány manipulátory. Tyto matice jsou uchovávány v krabičkách, které byly popsány v úvodu. Z těchto krabiček lze vyčíst jejich typ, lze vyčíst transformační matici a měnit ji, je-li nějaká.

3.1 Začlenění do scény

Pro začlenění manipulátorů do scény bylo potřeba rozšířit existující scénu o nové shadery a možnost renderování s využitím stencil bufferu – kvůli detekci interakce s táhly manipulátorů ve scéně. Původní renderovací systém I3T jsem shledal přílíš složitým a nepružným. Přidání jakékoliv nové funkcionality by vyžadovalo zásah do mnoha různých tříd, čímž by rychle vznikl nepřehledný a neudržovatelný kód. Vzhledem k těmto problémům jsem se rozhodl nahradit ho vlastním systémem, který se zakládá na modelu GameObject – Component (obecně Entity – Component). Tento model je často využíván při vývoji her, je například využit v Unity Engine. Tento model byl navržen právě k překonání takových problémů[6]. Nyní následuje výčet a popis tříd implementujících 3D scénu založenou na tomto modelu.

3.1.1 GameObject

Třída `GameObject` obsahuje ukazatel na rodiče a pole ukazatelů typu `GameObject`, což umožňuje sestavit strom scény jako graf s hranami průchodnými oběma směry, a pole ukazatelů typu `Component`, což umožňuje přidávat bloky kódu, poskytující nějakou funkcionalitu.

GameObject obsahuje může být inicializován bez geometrie jako `GameObject()` nebo s geometrií jako `GameObject(const pgr::MeshData mesh, struct Shader* shader, GLuint texture)`. Za parametr `texture` lze dosadit 0. V tomto případě GameObject použije výchozí texturu `World::whiteTexture`. Geometrie GameObjectu je vykreslena funkcí `draw(glm::mat4 parentTransform)`. V této funkci jsou přistoupeny statické proměnné třídy `World`, obsahující

transformaci kamery, která je pro renderování potřebná. Ostatní informace potřebné pro renderování jsou uchovány v GameObjectu.

Třída také obsahuje funkce pro manipulování grafu scény, jako přidání, odebrání potomka nebo změnění rodiče. Tyto funkce přebírají parametr `bool keepTransform`. Při změně rodiče GameObjectu se totiž změní prostor, ve kterém se tento GameObject nachází, což nemusí být žádoucí. Takto vypadá výsledná transformace GameObjectu T_{full} před přesunem v hiearchii scény:

$$T_{full} = P_{old} \cdot T.$$

Po přesunu

$$T_{newfull} = P_{new} \cdot T.$$

Změnu prostoru lze kompenzovat jako

$$T_{newfull} = P_{new} \cdot T_c \cdot T,$$

kde T_c je

$$T_c = P_{new}^{-1} \cdot P_{old} \cdot T. \tag{3.1}$$

T je původní transformace GameObjectu, P_{old} je transformace původního rodiče, P_{new} je transformace nového rodiče a T_c je kompenzovaná transformace. Vidíme, že výsledná transformace GameObjectu zůstala po dosazení T_c stejná:

$$T_{newfull} = P_{new} \cdot P_{new}^{-1} \cdot P_{old} \cdot T,$$

Po úpravě:

$$T_{newfull} = P_{old} \cdot T,$$

tedy $T_{newfull} = T_{full}$.

3.1.2 Component

Komponenta má pět členských funkcí, `void start()`, `void update()`, `void render(glm::mat4* parent,bool renderTransparent)`, `void GUI()`, `const char* getComponentType` a jednu statickou funkci `static const char* componentType()`. Třída `Component` je abstraktní třída, která je implementována konkrétními třídami.

Funkce `void start()` je volána na všech komponentách scény před prvním snímkem. Zde se provádí kód, který z nějakého důvodu nemůže být v konstruktoru, například zasahování do jiných částí scény.

Dále je zde funkce `void update()`, která je volána v každém snímku na všech komponentách ve scéně. V této funkci se odehrává logika scény, „herní logika“.

Funkce `void render(glm::mat4* parent,bool renderTransparent)` provádí kreslení na obrazovku. Parametr `parent` je matice nadřazeného GameObjectu v hiearchii scény a `renderTransparent` říká zda je na řadě renderování průhledné či neprůhledné geometrie – rozebráno v sekci 3.1.5. Tato funkce je volána kamerou – kamera je také komponenta. Renderování scény se děje při

zavolání `void update()` na kameře. Kamera pak zavolá funkci renderování na všech komponentách scény.

Funkce `void GUI()` je určena k vykreslování prvků uživatelského rozhraní ImGUI v okně 3D scény.

Nakonec, funkce `const char* getComponentType()` a `static const char* componentType()` vrací ukazatel na char, který jednoznačně určuje komponentu, což umožňuje vyhledávání konkrétního typu komponenty v poli ukazatelů na obecné komponenty (funkce `Component* getComponent(const char*)`).

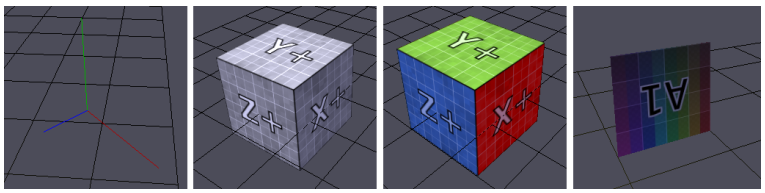
3.1.3 Shader

Tato struktura uchovává shader program i lokaci jeho atributů a uniformních proměnných, jak je vypsáno v ukázce níže. Struktura obsahuje atributy pozice vertexu, normála a texturovací souřadnice. Struktura obsahuje uniformní proměnné pozice kamery v globálních souřadnicích, transformaci modelu v globálních souřadnicích, které jsou potřeba pro výpočet osvětlení. Dále jsou zde uniformní matice `PVMmatrix`, která která obsahuje plnou transformaci objektu včetně projekce kamery, uniformní matice `VNmatrix`, obsahující transformaci normál a uniformní vektor 4 `color`, obsahující barvu modelu. Barva modelu v každém pixelu x, y je dána jako $tex(x, y) \cdot color$.

```
struct Shader{
    GLuint program;     ///<GL shader program
    GLuint camera;      ///<uniform camera position before rotation
    GLuint Mmatrix;     ///<uniform matrix of model's transform in shader
    GLuint PVMmatrix;   ///<uniform matrix view*model's transform in shader
    GLuint VNmatrix;    ///<uniform matrix for adjusting model's normals in
     shader
    GLuint color;       ///<uniform color (vec4)
    GLint attr_pos;     ///<vertice position attribute
    GLint attr_norm;    ///<vertice normal vector attribute
    GLint attr_uv;      ///<vertice texture coords attribute
};
```

3.1.4 World

Tato třída obsahuje definuje 3D scénu jako strom GameObjectů (pole `GameObject* sceneRoot`) a pomocné statické i nestatické funkce a proměnné. Funkce `void onUpdate()` je volána aplikací každý snímek. Funkce `void onStart()` musí být zavolána před `void onUpdate()`. V této funkci se volá `void start()` na všech komponentách scény. Ve funkci `void onGUI()` je voláno `void GUI()` na všech komponentách scény. Funkce `void onGUI()` musí být volána až po vykreslení 3D scény na obrazovky, jinak by prvky GUI byly překryty obrazem scény. Dále je zde interface pro komunikaci s `WorkspaceWindow`. Jsou to funkce `GameObject* addModel(const char* name)` a `bool removeModel(GameObject* g)`. Parametr `name` říká, jaký model se má ve scéně vytvořit. Jsou předdefinované čtyři možnosti – `PlainAxis`, `CubeGray`, `CubeColor` a `ColorGrid`,jak lze vidět na obrázku 3.1.



Obrázek 3.1: Předdefinované modely.

Funkce `void handlesSetMatrix(matnode,parent)` přebírá parametr `matnode`, což je krabička vybraná ve workspace a `parent`, což je sekvence, do které krabička náleží, nebo `NULL`, nenáleží-li krabička do žádné sekvence.

Statické funkce `bool init()` a `void end()` slouží k inicializování a uvolnění statických dat třídy, což jsou shadery a textury. Data musí být inicializována před vytvořením instance `World`. `World` má tři shadery, které jsou uchovány společně se svými parametry ve struktuře `Shader` Jsou to `shader0`, který slouží k vykreslování modelů ve scéně a `shaderHandle`, který slouží k vykreslování manipulátorů. Od `shader0` se liší tím, že nevykresluje textury a má méně výrazné stínování. Nakonec je zde `shaderProj`, který slouží k vykreslování zorného pole kamery a od předchozích dvou shaderů se liší tím, že přijímá dvě projekční matice – jednu editovanou manipulátory, jejíž zorné pole je vykresleno, a druhou je projekce kamery scény, jako v ostatních shaderech. Všechny shadery zapisují do kanálu alfa. Libovolným shaderem lze tedy vykreslovat průhlednou geometrii, průhlednost závisí na nastavení GameObjectu, která jsou shaderu předávána – parametry `glm::vec4 color` a `GLuint texture`.

Funkce `Shader World::loadShader(const char* vs_name, const char* fs_name)` načte a zkompiluje shader. Předpokládá se, že shader program obsahuje všechny požadované proměnné správného typu a se správným názvem.

Parametry osvětlení jsou napevno napsané v shaderu. V konstruktoru `World()` jsou inicializovány manipulátory (podrobněji rozepsáno v sekci 3.2) a je vytvořena prázdná scéna, ve které se nachází mřížka, která umožňuje orientaci v prostoru v prázdné scéně (možno vidět např. na obrázku 3.1).

3.1.5 Základní komponenty

Komponenty zde popsané jsou nutné k renderování a prohlížení 3D scény.

Camera. Tato komponenta slouží k renderování zadané 3D scény do GL framebufferu. Kamera má vlastnost `m_mainCamera`, říkající zda daná kamera je **hlavní kamerou** (ve scéně může být víc kamer). Je-li kamera označená jako hlavní, potom bude renderovat na obrazovku a bude ukládat své transformační matice (view a projection) do statických proměnných třídy World, které slouží k výpočtu různých operací mezi obrazovkou a 3D scénou, například projekce z 3D do 2D. Tyto operace jsou definovány v souboru `World/Transforms.cpp`. Hlavní kamera také každý snímek aktualizuje svoji projekční matici podle rozměrů obrazovky, které jsou uchovány v třídě World. Ve scéně musí být právě jedna hlavní kamera.

Kamera má dva konstruktory, `Camera(float viewingAngle, GameObject* sceneRoot,RenderTexture*renderTarget)` a `Camera(float viewingAngle, GameObject* sceneRoot)`. Třída RenderTexture je wrapper okolo GL framebufferu, obsahující dodatečné informace, jako je výška a šířka framebufferu v pixelech a GL textury, ze kterých se framebuffer skládá – `color attachment`, `depth attachment` a `stencil attachment`. Není-li tento parametr zadán (druhé přetížení konstruktoru), bude kamera předpokládat, že nějaký GL framebuffer je již aktivovaný, zároveň se kamera také nastaví jako hlavní.

Parametr `viewingAngle` udává šíři záběru kamery ve stupních. Projekční matice kamery je uložena ve veřejné proměnné `m_perspective`. Není-li kamera označena jako hlavní, lze zde uložit jakoukoliv uživatelskou matici. V opačném případě lze také uložit vlastní matici, ale ta se pak před renderováním přepíše perspektivní maticí dané poměrem stran obrazovky `World::width/World::height` a `viewingAngle`.

Kamera provádí renderování ve své funkci `void update()`, kde volá `void render(glm::mat4* parent,bool renderTransparent)` na všech komponentách zadaného stromu GameObjectů (parametr konstruktoru `sceneRoot`). Renderování se provádí ve dvou krocích – v prvním kroku se renderují všechny komponenty, které mají nastaveno `isTransparent=false`. V druhém kroku se renderují všechny komponenty, které jsou označeny jako průhledné `isTransparent=true`. Kamera nerozhoduje, kdy se má renderovat která geometrie. Kamera pouze na komponentě zavolá funkci `render` s parametrem říkajícím zda je nyní potřeba renderovat průhlednou či neprůhlednou geometrii a potřebná logika je implementována v oné komponentě.

Renderer. Tato komponenta ve svém `void render(glm::mat4* parent, bool renderTransparent)` volá `void draw(glm::mat4 parent)` na GameObjectu, který je jejím vlastníkem. Komponentě se dá nastavit, zda je průhledná, zda má renderovat se šablonou (GL stencil) a zda má renderovat jako trojúhelníky (GL_TRIANGLES) nebo jako wireframe (GL_LINES). `Renderer` označený jako průhledný renderuje pouze tehdy, platí-li `renderTransparent`. Zmíněné parametry se dají nastavit přímo členskými proměnnými `bool m_isTransparent`, `bool m_useStencil` a `bool m_drawLines`, nebo parametrem konstruktoru `Renderer(unsigned int flags=0)`. Parametr `flags` je bitově maskován s konstantami `Renderer::IS_TRANSPARENT=1<<0`, `Renderer::USE_STENCIL=1<<1` a `Renderer::DRAW_LINES=1<<2` – první bit nastaví průhlednost, druhý bit aktivuje šablonu a třetí bit nastaví renderování na `GL_LINES`.

CameraControl. Tato komponenta podle vstupu z klávesnice a myši rotuje a posouvá GameObject, který je jejím vlastníkem. Předpokládá se tedy, že hlavní kamera scény je připnutá ke stejnému GameObjectu jako komponenta `CameraControl`. Šipkami se pohybuje kamerou v rovině XZ, při přidržení Ctrl v rovině XY v lokálním prostoru kamery. Přidržením levého tlačítka myši a tahem se rotuje kamera, prostředním tlačítkem myši a tahem se pohybuje kamerou opět v v rovině XY v lokálním prostoru kamery.

Kolečkem myši se lze pohybovat dopředu a zpět, stejně jako šipkami.

3.2 Navázání na workspace

Pokud je změněn výběr ve workspace, workspace zavolá funkci scény `void World::handlesSetMatrix(*node, *parent)`, které předá jako první parametr vybranou krabičku, nebo `NULL` v případě, že je výběr zrušen, nebo je-li vybráno více krabiček zároveň. Druhý parametr je sekvence, ve které se krabička nachází. Tento paremetr je NULL, není-li vybraná žádná krabička, nebo není-li vybraná krabička součástí žádné sekvence. Pro zobrazení manipulátorů je třeba totiž znát pozici matice krabičky (uvažujeme pouze krabičky reprezentující nějakou maticovou transformaci), protože na téže pozici se mají zobrazit manipulátory, a její lokální prostor, protože v tomto prostoru budou manipulátory pracovat – tento prostor určuje, kterým směrem budou ukazovat táhla manipulátorů.

Například, pokud máme vybranou matici rotace okolo osy Y (vertikální osa), ale předkem této matice je rotace okolo osy X o pravý úhel, osu Y budeme ve scéně pozorovat jako horizontální. K zjištění pozice matice a její lokálního prostoru potřebujeme znát všechny matice předcházející vybranou matici v hiearchii workspace, což se dozvíme právě ze sekvence, ve které se vybraná matice nachází.

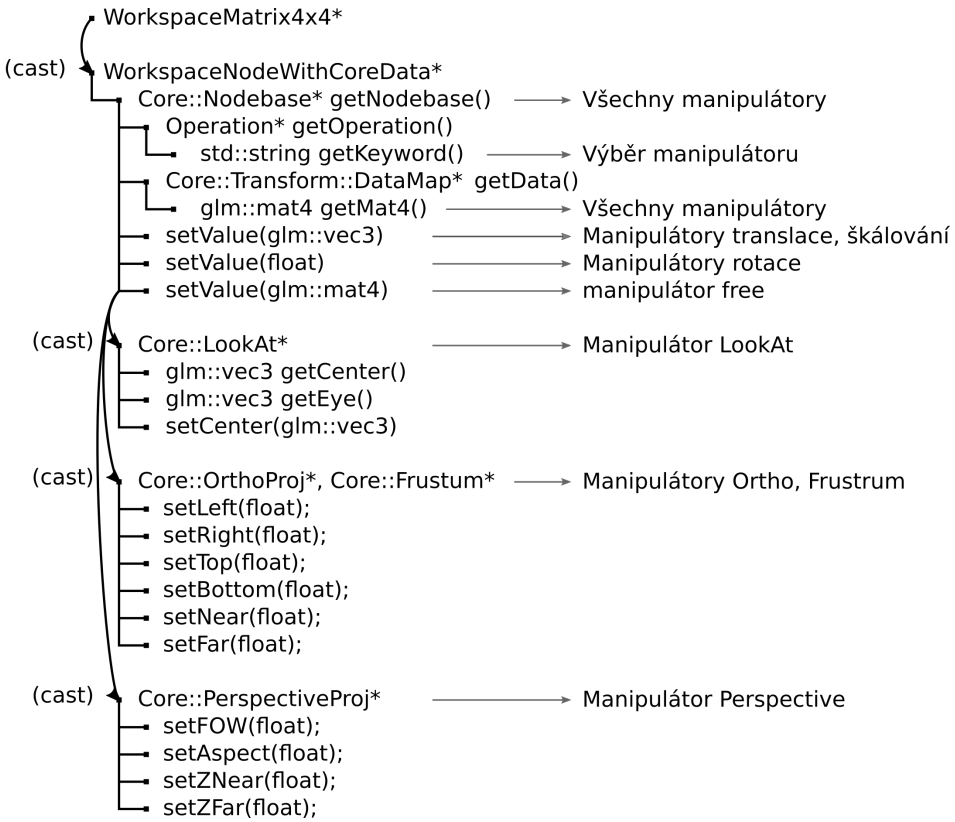
Manipulátor každého typu transformace je samostatná komponenta. Každá z těchto komponent má veřejné proměnné `Ptr<Core::NodeBase> m_editednode` a `Ptr<Core::Sequence> m_parent;`. Třída `World` si drží ukazatele na tyto proměnné pro každou komponentu manipulátoru v členské proměnné `std::map<std::string, Manipulator> manipulators`, kde `std::string` je název operace krabičky, která je navázaná na daný manipulátor a `Manipulator` je takováto struktura:

```
struct Manipulator {
    Ptr<Core::NodeBase>*editedNode;
    Ptr<Core::Sequence>*parent;
    Component*component;
};
```

Proměnné `*editedNode`, `*parent` jsou ukazatele na proměnné komponent manipulátorů `m_editednode` a `*m_parent`, zatímco `*component` je ukazatel na samotnou komponentu manipulátoru.

Funkce `void World::handlesSetMatrix(*node, *parent)` deaktivuje všechny manipulátory a poté aktivuje manipulátor, který se nachází na místě `manipulators[keyword]` a skrze strukturu `Manipulator` nastaví proměnné onoho manipulátoru. `keyword` je textový řetězec, jednoznačně popisující funkci krabičky, který získáme pomocí getteru, jak je také zobrazeno v následujícím diagramu.

Krabičky jsou složeny z několika úrovní dědičnosti. V případě manipulátorů se budeme zbývat pouze krabičkami, dědícími z `WorkspaceMatrix4x4`. V diagramu 3.2 jsou zobrazené funkce nastavující a získávající vlastnosti krabičky, které potřebujeme pro jejich propojení s manipulátory.



Obrázek 3.2: Vlastnosti přistupované manipulátory.

Kapitola 4

Implementace manipulátorů

Komponenty, vytvářející manipulátory pro jednotlivé typy transformací jsou umístěné v hiearchii scény. Vždy je aktivní buď jedna komponenta obsahující manipulátory pro vybranou krabičku z workspace, nebo žádná, není-li vybrána žádná krabička, je-li vybráno několik krabiček nebo nexistují-li manipulátory pro vybranou krabičku. Aktivace a deaktivace těchto komponent se děje ve funkci `void handlesSetMatrix(matnode, parent)`.

4.1 Translace a škálování v jedné ose

Při škálování nebo translaci dějící se v jedné ose je tato osa promítnuta z 3D na obrazovku a potom je měřen pohyb myši podél této osy a míra tohoto pohybu potom určuje míru transformace. Průmět osy na obrazovku a_{scr} se získá jako rozdíl mezí průmětem bodu p, což je pozice upravované matice, a průmětem bodu $p + a_i$, kde a_i je osa, ve které škálujeme nebo posunujeme. Dalším krokem je převést pohyb myši m_c z kanonické báze do na pohyb m_a v bázi tvořené normalizovanými vektory a_{scr} a $a_{scr}^{\perp}$:

$$m_a = (a_{scr}\ a_{scr}^{\perp})^{-1} \cdot m_c. \tag{4.1}$$

4.2 Škálování ve dvou osách

V případě škálování ve dvou osách současně jsou na obrazovku promítnuty tyto dvě osy a_1 a a_2, pohyb myši v kanonické bázi je převeden na pohyb v bázi tvořené průmětem těchto dvou os na obrazovku. Pohyb myši se převede do této báze podobně jako v předcházejícím případě:

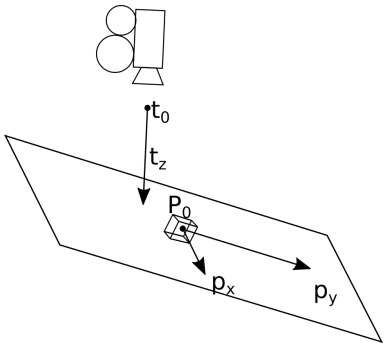
$$m_a = (a_{1scr}\ a_{2scr})^{-1} \cdot m_c. \tag{4.2}$$

4.3 Translace ve dvou osách

V případě translace ve dvou osách je míra translace určena na základě výpočtu pozice průniku paprsku vycházejícího z myši s virtuální rovinou ve 3D scéně, jak je popsáno v článku [7]. V mé modifikaci je paprsek promítnut

do obrazovky, k průmětu je přičten pohyb myši v tomto snímku a výsledný bod promítnut zpět do 3D scény. Učinil jsem tak proto, že na plošku pro manipulaci lze kliknout na různých místech, a potom dojde ke skokové změně pozice, kdy se transformační matice změní tak, aby se počátek plošky nacházel na kurzoru myši. Shrnuto – modifikace způsobuje, že i zde míra transformace je dána pohybem myši, ne její pozicí.

Na obrázku 4.1 vidíme paprsek, který vychází z pozice kamery[1] t_0, prochází kurzorem myši a má směr t_z. Vidíme také rovinu mající počátek v pozici modelu p_0 danou vektory p_x, p_y, což jsou osy, ve kterých provádíme translaci. V původní verzi výpočtu t_z spočteme jako rozdíl dvou bodů $(x\ y\ z_1)$ a



Obrázek 4.1: Průnik paprsku z kurzoru s rovinou translace.

$(x\ y\ z_2)$, kde x, y jsou dány převodem pozice kurzoru myši z prostoru obrazovky do prostoru viewportu a $-1 < z_1 < z_2 < 1$. V upravené verzi souřadnice x a y jsou dány převodem pozice modelu do prostoru obrazovky, přičtením pohybu myši v tomoto snímku a převedením zpět do prostoru viewportu. Jak lze vidět na obrázku 4.1, potřebujeme najít lineární kombinaci vektorů $a \cdot t_z + b \cdot p_x + c \cdot p_y$ takovou, abychom se dostali z t_0 do p_0, potom $b \cdot p_x$ a $c \cdot p_y$ budou míry posunu v jednotlivých osách translace. Toto lze zapsat jako maticovou rovnici

$$\begin{pmatrix} t_z & p_x & p_y \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} p_0 - t_0 \end{pmatrix}, \tag{4.3}$$

po úpravě

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} t_z & p_x & p_y \end{pmatrix}^{-1} \cdot \begin{pmatrix} p_0 - t_0 \end{pmatrix}. \tag{4.4}$$

Potom nová pozice posouvaného modelu je $p_0 + a \cdot t_z$.

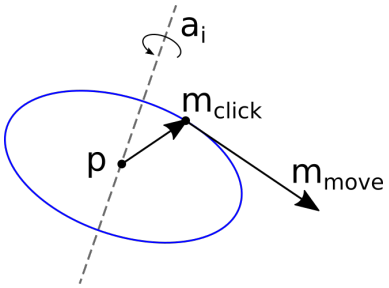
4.4 Rotace

Pohyb myši se převádí na rotaci jako míra pohybu myši podél tečny na kruh v místě kliknutí, kruh představující manipulátor rotace. Nejdříve je potřeba

[1] Pozice kamery je posunutí světa v opačném směru, proto $\vec{T_0}$ je mínus pozice kamery

převést pozici myši z prostoru obrazovky do 3D prostoru, což se opět udělá průnikem paprsku myši s rovinou, v tomto případě rovinou, ve které leží kruh, na který bylo kliknuto. Poté se tečna spočítá jako vektorový součin osy kruhu a_i a vektoru daném rozdílu mezi pozicí (středem) kruhu p a místem kliknutí v 3D prostoru m_{click}. Výsledný vektor m_{move} se promítne na obrazovku a poté se měří pohyb myši podél tohoto vektoru, stejně jako v případě pohybu nebo škálování v jedné ose.

$$m_{move} = a_i \times \Big(m_{click} - p\Big) \tag{4.5}$$



Obrázek 4.2: Tečna na manipulátor rotace.

4.5 lookAt

Při editaci lookAt se editovaná matice rotuje tak, aby její osa Z směřovala na zadané místo prostoru, kterým se dá pohybovat manipulátory translace. Matice lookAt je generována čtyřmi vektory

$$\begin{aligned} Z &= p - p_{trg} \\ Y &= Z \times \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \\ X &= Z \times Y, \end{aligned}$$

a čtvrtým vektorem reprezentujícím translaci. p_{trg} je pozice manipulátorů a p je pozice matice. Normalizované vektory X, Y, Z pak tvoří první tři hodnoty prvních třech sloupců nové matice. První tři hodnoty čtvrtého sloupce obsahují translaci. Ostatní hodnoty jsou hodnoty jednotkové matice. Funkce `glm::lookAt`, která je v I3T používána k generování matice, ji ale generuje v invertovaném tvaru. Tato funkce je totiž navržena k generování transformace kamery a rotace a translace kamery je v OpenGL implementována jako rotace a translace světa v opačném směru. Proto manipulátory vnitřně pracují s inverzí výsledku `glm::lookAt`.

4.6 Projekční matice

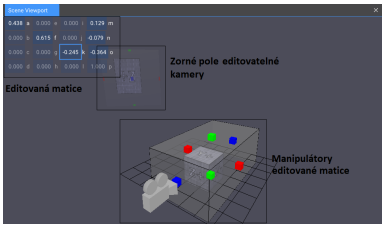
Vizualizace záběru kamery se provede transformováním krychle $\langle -1; 1 \rangle^3$ inverzí editované projekční matice. Opět, inverzí proto, že projekční matice

provádí škálování a translace světa v opačném směru. Například, chceme-li zobrazit velkou část světa, tak projekční matice „zmenší“ svět, aby se na obrazovku promítla jeho větší část, ale záběr kamery, který vizualizujeme, se tím naopak zvětší. V případě perspektivní projekce je škálování závislé na vzdálenosti od obrazovky – čím dál je předmět od obrazovky, tím je menší a šíře záběru je naopak větší. V případě perspektivní matice je při transformování vrcholů vizualizující krychle potřeba provést perspektivní dělení, které je potřeba provést ve vertex shaderu, proto se vizualizace záběru vykresluje speciálním shaderem `viewproj-vs.glsl`, který zpracovává i projekční matici editovanou manipulátory.

Kapitola 5

Uživatelské testy

Do uživatelských testů jsou zahrnuti uživatelé, kteří systém I3T již znají, i uživatelé, kteří se s ním setkávají poprvé. Toto jsem zohlednil i v dotazníku, ve kterém uživatelé odpoví na otázky týkající se jejich zkušeností s manipulátory v jiných 3D prostředích i s aplikací I3T. Dotazník začíná úvodem, popisujícím základní funkcionality I3T, jeho ovládání a popis scény(obrázek 5.1). Poté následují otázky ke zkušenostem uživatele s I3T nebo jinými 3D prostředími. Poté je uživatel postaven před sadu úloh, které má vyplnit a nakonec následují otázky ohledně uživatelské zkušenosti s vyplňováním úloh a na záběr možností manipulátorů využitých uživatelem. Dotazník je součástí práce jako příloha.



Obrázek 5.1: Popis scény.

5.1 Testovací úlohy

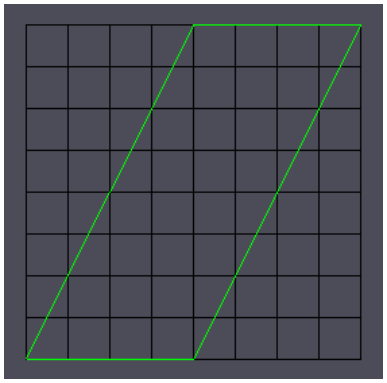
Uživatel je postaven před sedm testovacích úloh, které vyžadují použití všech transformačních matic pro které jsou implementované manipulátory. Zde je výčet těchto úloh:

1. Rotujte editovatelnou kameru pomocí lookAt.
2. Posuňte šedou kostičku, tak, aby byla v zorném poli editovatelné kamery (obrázek 5.1).
3. Nastavte zorné pole editovatelné kanery tak, aby kostička vyplňovala co největší část zorného pole. Zkuste to pro matice ortho, perspective i frustrum.

4. Rotujte kostičku tak, aby se editovatelná kamera dívala na stranu šedé kostičky Y+.

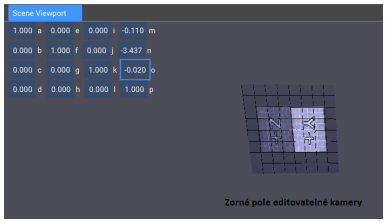
5. Škálujte a rotujte šedou kostičku, tak, aby přesně pokryla celou mříž.

6. Pomocí volné transformace zkoste a škálujte hrany kostičky tak, aby kostička pokrývala mříž takovýmto způsobem:



Obrázek 5.2: Obrázek k 6. úloze.

7. Nastavte ortografickou kameru a transformaci kostičky tak, aby zorné pole editovatelné kamery vypadalo takto:



Obrázek 5.3: Obrázek k 7. úloze, zobrazující zorné pole editovatelné kamery.

5.2 Průběh testování

Úlohy byly testovány v několika kolech, kde v každém kole je testována nová verze, která reaguje na připomínky k minulé verzi. Stručný souhrn výsledků testování s každým uživatelem je uveden v tabulkách. Řádek „Obtížnost úloh“ říká, jak uživatel hodnotil obtižnost úloh 1 – 7 na stupnici od jedné do pěti, kde pět je nejobtížnější. Následující řádky ukazují, jak uživatel využil nebo rozpoznal možnosti manipulátorů. Řádky k transformaci jednotlivých os říkají, jak uživatel využil manipulátory volné translace, které jsou popsány na straně 14. V řádku „Neintuitivní“ jsou vypsány manipulátory transformací, jež uživatelé označili za neintuitivní nebo matoucí. V řádku „Zlepšit“ jsou návrhy uživatelů na zlepšení manipulátorů.

5.2.1 První testování

Po prvním kole testů se ukázalo, že nejvíce problémů uživatelům dělají manipulátory volné transformace a lookAt (tabulka 5.1). Pouze Alikhan zjistil, že lze přepínat ve volné transformaci škálování, rotaci a translaci. Tato možnost je dost skrytá, protože lze přepínat pouze klávesnicí. Rozhodl jsem se tedy přidat nápovědu - při manipulaci volné transformace se v levém dolním rohu zobrazí text s klávesovými zkratkami. Stejně jsem postoupil v případě lookAt (obrázek 5.4), kde uživatelům nebylo jasné, co vlastně upravují. Při manipulaci lookAt se zobrazí nápověda, že jde o posun bodu *center*, do kterého se má dívat kamera.

Zároveň Damir odhalil chybu, že kamera se ve skutečnosti dívá opačným směrem, než je *center*. Chybu jsem opravil – ve skutečnosti šlo pouze o to, že manipulátory pro posun *center* se zobrazovaly na opačném místě oproti počátku, než měly, tedy v bodě $-center$.

Oběma uživatelům chyběla možnost resetovat matice do původního stavu. Tamto možnost by ale měla být součástí krabičky ve Workspace, jak to bylo v staré verzi I3T (obrázek 1.1, strana 2). Co se týče ukazatele os, ten je zobrazen, pouze si ho Damir nevšiml.

	Alikhan	Damir
Pracoval dříve s I3T	Ne	Ne
Pracoval s jinými 3D prostředími	Ano	Ano
Obtížnost úloh	2, 1, 1, 1, 2, 2, 1	3, 1, 1, 3, 1, 1, 2
Škálování/translace v rovině	Ano	Neobjevil
Uniformní škálování	Ano	Neobjevil
Škálování jednotlivých os	Ano	Ano
Zkosení jednotlivých os	Ano	Neobjevil
Rotace jednotlivých os	Ano	Ano
Neintuitivní	LookAt	LookAt
Nerozpoznáno		Volná transformace
Zlepšit	Návrat matice do výchozích hodnot	Návrat matice do výchozích hodnot, přidat ukazatel os

Tabulka 5.1: Výsledky prvního testování.

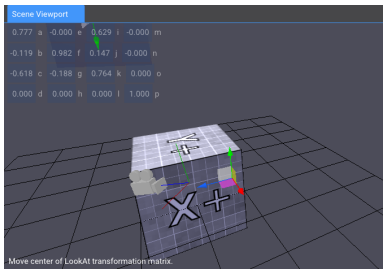
5.2.2 Druhé testování

Poté jsem upravenou verzi opět testoval s uživateli. Miroslav také neobjevil možnosti manipulátorů volné transformace a nevšiml si nápovědy pro lookAt i pro manipulátory volné transformace. Dále také objevil chybu zmizení ikony kamery po nastavení nulových hodnot v projekčních maticích. Zde šlo o to, že do transformační matice této ikony se dostaly neplatné hodnoty NaN, která vznikne při neplatné číselné operaci, v tomto případě při pokusu o dělení nulou. Po seznámení s manipulátory volné translace Miroslav zjistil, že při pokusu o škálování osy nulové délky se v transformované matici objeví hodnoty NaN.

Dále jsem se rozhodl k manipulátorům lookAt přidat ikonu kamery (obrázek 5.4), aby i bez nápovědy bylo zřejmější, co transformace provádí, ale nenapadlo mě žádné zlepšení volné transformace. V tabulce 5.2 je opět přehled výsledků testování.

	Miroslav	Tomáš
Pracoval dříve s I3T	Ano	Ne
Pracoval s jinými 3D prostředími	Ano	Ano
Obtížnost úloh	3, 2, 2, 1, 2, 2, 4	1, 1, 2, 2, 2 ,5 ,5
Škálování/translace v rovině	Ano	Ano
Uniformní škálování	Neobjevil	Ano
Škálování jednotlivých os	Neobjevil	Ano
Zkosení jednotlivých os	Neobjevil	Ano
Rotace jednotlivých os	Neobjevil	Ne
Neintuitivní		
Nerozpoznáno	Volná transformace	
Zlepšit	Pozice bodu lookAt v prostoru je špatně rozeznatelná	

Tabulka 5.2: Výsledky druhého testování.



Obrázek 5.4: Manipulátory LookAt po úpravě.

5.2.3 Třetí testování

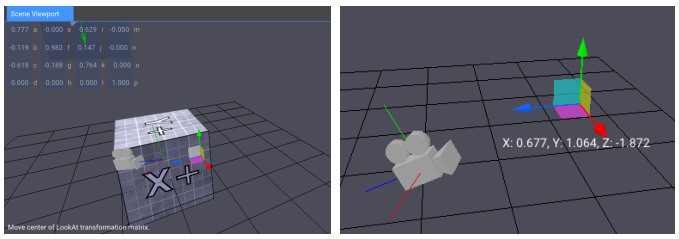
Po těchto opravách jsem pokračoval v testování s dalšími uživateli. Mírně jsem zvětšil manipulátory, protože jsou opravdu malé. Ukázalo se, že rozpoznání manipulátorů volné transformace je hlavním problémem. I přes textovou nápovědu většina uživatelů objevila možnosti těchto manipulátorů pouze náhodou, nebo je neobjevila vůbec, jak je vidět v tabulce 5.3, a úlohu zkosení (6. úloha) dělala přímou úpravou hodnot matice. Při přepnutí na matici volné transformace jsou manipulátory v módu rotování os, díky čemuž zřejmě uživatelé považovali tento manipulátor za obyčejnou rotaci. Proto jsem jako výchozí stav použil zkosení translací (obrázek 2.12, strana 15). Zároveň jsem také přidal funkcionalitu, že při změnění hodnoty libovolného sloupce matice se manipulátor přepne do módu editování osy odpovídající změněnému sloupci, stejně jako by se přepínala editovaná osa klávesami `x`, `y`, `z` a `w`, jak je popsáno v sekci 2.2.5 na straně 14. Přepínání os by mělo upoutat pozornost uživatele na manipulátor, aby jej použil.

	Sofie	Martin
Pracoval/a dříve s I3T	Ano	Ano
Pracoval/a s jinými 3D prostředími	Ano	Ano
Obtížnost úloh	3, 2, 2, 1, 2, 2, 4	2, 1, 1, 2, 1, 3, 1
Škálování/translace v rovině	Ano	Ano
Uniformní škálování	Ne	Ano
Škálování jednotlivých os	Neobjevila	Neobjevil
Zkosení jednotlivých os	Ano	Neobjevil
Rotace jednotlivých os	Ano	Neobjevil
Neintuitivní		
Nerozpoznáno	Volná transformace, popisky	Volná transformace, popisky
Zlepšit	Přidat nápovědu, kde by byly vypsané klávesové zkratky	Větší nebo tučnější manipulátory

Tabulka 5.3: Výsledky třetího testování.

5.2.4 Čtvrté testování

I po posledních úpravách většina uživatelů stále nerozpoznala možnosti volného manipulátoru. Rovněž uživatel Petr podotkl, že by se hodilo, kdyby nápověda k aktivnímu manipulátoru byla zobrazena i když na manipulátoru neprobíhá žádná interakce. Nápovědu manipulátoru jsem tedy ponechal zobrazenou, pokaždé, když je zobrazený manipulátor. Stejně tak jsem doplnil možnost upravovat pozici bodu *eye* matice lookAt (tabulka 5.5) a ikonu kamery jsem umístil přímo na pozici *eye*, aby nedocházelo k chybnému dojmu, že pohybování *center* se pohybuje také kamera (5.5). Také byla odhalena chyba, kvůli které se při určitém složení rotací manipulátoru rotace rotuje opačným směrem, než odpovídá směr tahu myši. Toto chybu jsem odstranil. Dále se opět objevila připomínka, že pozice bodu *center* transformace lookAt je špatně rozeznatelná (tabulka 5.4). Proto jsem pod manipulátor přidal popisek s aktuální pozicí *center*, jak je také vidět na obrázku 5.5.



Obrázek 5.5: Manipulátory LookAt před a po úpravě

	Jaroslav	Max
Pracoval/a dříve s I3T	Ano	Ne
Pracoval/a s jinými 3D prostředími	Ne	Ano
Obtížnost úloh	2, 1, 1, 1, 1, 3, 4	1, 1, 2, 1, 1, 3, 2
Škálování/translace v rovině	Ano	Ano
Uniformní škálování	Ne	Ano
Škálování jednotlivých os	Neobjevil	Neobjevil
Zkosení jednotlivých os	Neobjevil	Ano
Rotace jednotlivých os	Neobjevil	Neobjevil
Neintuitivní		
Nerozpoznáno	Volná transformace	Volná transformace
Zlepšit	Pozice bodu lookAt v prostoru je špatně rozeznatelná	

Tabulka 5.4: Výsledky čtvrtého testování.

	Petr
Pracoval/a dříve s I3T	Ano
Pracoval/a s jinými 3D prostředími	Ne
Obtížnost úloh	2, 3, 3, 3, 1, 1, 2
Škálování/translace v rovině	Ano
Uniformní škálování	Ano
Škálování jednotlivých os	Ano
Zkosení jednotlivých os	Ano
Rotace jednotlivých os	Ne
Neintuitivní	LookAt, rotace
Nerozpoznáno	
Zlepšit	Nelze pohybovat bodem eye u matice lookAt

Tabulka 5.5: Výsledky čtvrtého testování 2.

5.2.5 Výsledky testování

Po odstranění několika chyb a postupném vylepšování manipulátorů lookAt zůstal jediný problém – většina uživatelů neobjeví manipulátory volné transformace vůbec, nebo jenom částečně. Po diskusi s uživateli vyšlo najevo, že problém je v tom, že táhla manipulátorů volné transformace vypadají stejně jako u translace, rotace, nebo škálování, a proto je velká část uživatelů ignoruje. Také zde byla opakující se stížnost, že nelze zoomovat kolečkem myši ve scéně, což jsem napravil, i když to nesouvisí s ovládáním manipulátorů.

Kapitola 6
Skriptování

Interpretr skriptů má umožňovat v první řadě načítání a ukládání scény, později by mohl být použit i pro vytváření nových krabiček a ovládání a úpravu scény z konzole, což by ocenili zkušenější uživatelé. Existuje řada volně dostupných interpretrů, které by bylo možné napojit na I3T, proto jsem se rozhodl použít již hotové řešení.

Mým cílem byl interpretr, který by byl úsporný, multiplatformní, snadno rozšiřitelný a hlavně, který by mohl být napojen k I3T. Užitečná by byla také možnost začlenit interpretr přímo do kódu I3T, čímž by odpadla starost o multiplatformnost. Mezi zvažovanými interpretry byl python, jax, c4 a picoc (Pico C). Python je velmi rozsáhlý a populární interpretr, který podporuje množství knihoven třetích stran a jeho jazyk má bohatou syntaxi. I když lze rozšířit interpretr vlastními knihovnami, nehodí pro požadované účely kvůli své rozsáhlosti.

Interpretry picoc, jax a c4 jsou interpretry jazyka c napsané v jazyce c, což umožňuje jejich zahrnutí do kódů I3T, čímž odpadá starost o multiplatformnost – interpretr je zkompilován pro požadovanou platformu společně s ostatním kódem I3T.

Jax a c4 jsou velmi úsporné (58 a 22kB zdrojového kódu) ale nenabízejí žádné rozhraní pro rozšíření o uživatelské funkce, naproti tomu picoc je rozsáhlejší (310kB zdrojového kódu), ale nabízí velmi jednoduché rozhraní pro přidání uživatelských funkcí, proto jsem se rozhodl použít ten.

6.1 Picoc

Picoc byl původně navržen pro programování dronů, robotů a různých embedded zařízení[8], proto minimálně spoléhá na standardní knihovny, neboť ty nejsou na některých takových platformách dostupné. Stav interpretru je definovaný strukturou `Picoc`, která obsahuje všechny požadované datové struktury a proměnné. Je tedy možné současně mít v jedné aplikaci více instancí interpretru, které jsou na sobě nezávislé. Na ukázce níže je zobrazen životní cyklus interpretru.

```
Picoc pc;
PicocInitialise(&pc,PICOC_STACK_SIZE);
if(PicocPlatformSetExitPoint(&pc)){
```

```
    PicocCleanup(&pc); return pc.PicocExitValue;
}
//Add custom functions and types
IncludeRegister(&pc,"I3T.h",NULL,&PlatformLibraryI3T[0],defs);
//Expose application variable to script
VariableDefinePlatformVar(&pc,NULL,"free",&pc->IntType,(union AnyValue*)&
    mat4typefree,FALSE);
PicocPlatformScanFile(&pc, filename);
PicocCleanup(&pc);
return pc.PicocExitValue;
```

V `PicocInitialise` se inicializuje slovník klíčových slov, datových typů, definují se funkce, přístupné ze skriptu.

`PicocPlatformSetExitPoint` určí místo v programu aplikace, kam interpretr skočí, když dojde k jakékoliv chybě (syntaktická chyba, nedostatek paměti, nenalezený skript...). V případě zobrazeném výše tedy při chybě interpretr skočí dovnitř podmínky `if (PicocPlatformSetExitPoint(&pc))`, kde je uvolněna paměť allokovaná interpretrem a vrácena jeho návratová hodnota.

`IncludeRegister` je funkce, která rozšíří interpretr o uživatelské funkce. `PlatformLibraryI3T` je pole dvojic zakončené nulou. Dvojice obsahuje ukazatel na funkci a její předpis jako string. Předpis funkce určuje, jak bude funkce volána ze skriptu. Argumenty, se kterými je funkce volána ze skriptu jsou potom předány funkci `saveW` argumtem Param, který obsahuje pole ukazatelů na strukturu Param, jejíž obsah lze číst jako některý ze základních datových typů, v příkladu níže jako ukazatel (`Param[0]->Val->Pointer`).

```
void saveW(struct ParseState* Parser, struct Value* ReturnValue, struct
    Value** Param, int NumArgs) {
    const char*filename = (char*)Param[0]->Val->Pointer;
    WorkspaceWindow * ww =
        (WorkspaceWindow*)I3T::getWindowPtr<WorkspaceWindow>();
    bool status=false;
    if(ww != NULL){status=SaveWorkspace(filename, &ww->WorkspaceNodes);}
    ReturnValue->Val->Integer = status;
}
struct LibraryFunction PlatformLibraryI3T[] ={
 {saveW,"bool save(char*);" },
    {NULL, NULL }
};
```

Funkce `VariableDefinePlatformVar` zpřístupní proměnné aplikace do skriptu. Parametry funkce jsou instance interpretr, pozice parseru (lze ponechat NULL), název proměnné ve skiptu, datový typ proměnné ve skriptu, ukazatel na proměnnou, zapisovatelnost proměnné ve skriptu.

Funkce `PicocPlatformScanFile` vykoná obsah souboru. Na místě této funkce lze také použít `PicocParse`, která vykoná obsah textového řetězce nebo `PicocParseInteractive`, která vykonává vstup z stdin. Tyto tři funkce lze použít na jedné instanci interpretru libovolněkrát v libovolném pořadí.

Nakonec se funkcí `PicocCleanup` uvolní veškerá paměť allokovaná interpretrem a z `pc.PicocExitValue` můžeme číst návratovou hodnotu interpretru, která indikuje, zda při běhu interpretru z nějakého důvodu, například syntak-

tické chyby ve zpracovávaném skriptu, došlo k chybě, tedy hodnota 1 označuje chybu, hodnota 0 bezchybný běh.

6.1.1 Úpravy interpretru

Z technických důvodů bylo potřeba upravit kód interpretru. Bylo potřeba přesměrovat výstup interpretru z stdout do std::cout, aby mohl být zobrazen v konzoli (kapitola 6.3.3), což také znamenalo upravit kód interpretru pro C++, protože syntaxe C není úplnou podmnožinou syntaxe C++, například deklarace struktur v C vyžaduje použití klíčového slova `struct` před typem proměnné, C++ zase vyžaduje typování návratové hodnoty funkce `malloc`.

Dále jsem z interpretru odstranil podporu standardních knihoven `stdlib.h`, `stdio.h` (s výjimkou funkce `printf`), `string.h`, `time.h`, `math.h`, `errno.h`, unixové knihovny `unistd.h` a knihovny `srv1.h`, která sloužila pro ovládání blíže neurčeného robota. Funkce těchto knihoven jsou pro I3T nepotřebné, a v některých případech, typicky funkce knihovny `unistd.h`, i nežádoucí, protože umožňují vytvořit skripty, které by páchaly škody na počítači uživatele.

6.2 Návrh skriptování

Přímočarý přístup při vytváření funkcí skriptu, kterými by bylo možno vytvářet a manipulovat workspace, by bylo vytvářené datové struktury a objekty zpřístupnit do skriptu, kde by s nimi manipulovalo dalšími funkcemi. Toto řešení má ale několik nevýhod. Znamenalo by to vytvořit strukturu, která by logicky kopírovala existující struktury tvořící workspace. Datové struktury tvořící workspace jsou složité a dosud prochází vývojem a jakákoliv případná změna by musela být reflektována redefinováním struktur používaných skriptem i funkcích skriptu s nimi pracujícími.

Také je třeba brát v úvahu, že skriptování nebude používáno pouze pro načtení a uložení scény z menu aplikace, ale bude používáno i uživateli. Plný přístup uživatele k těmto strukturám by také vyžadovalo neustálou kontrolu platnosti prováděných operací i přistupovaných dat. Například pokus o propojení nekompatibilních nebo neexistujících vstupů a výstupů vyvolá logickou výjimku v aplikaci I3T a ukončení aplikace, stejně tak ukládání nekompatibilních dat do krabiček (například ukládání vektoru 4 do krabičky násobící dvě matice). Zároveň by také nebylo možné zkontrolovat platnost přístupů do paměti. Například zápis do neinicializovaného ukazatele by mohl způsobit pád či nestabilitu aplikace.

Proto jsem se přišel s řešením, kdy veškerá data a krabičky vytvářené skriptem jsou skrytá. Dostupné jsou pouze identifikátory v případě krabiček, nebo indexy v polích v případě dat. Pole jsou taktéž skrytá. Pole mají známou délku, a proto při pokusu manipulovat nějakou datovou strukturu lze vždy rozhodnout, zda struktura zadaná indexem nebo identifikátorem existuje. Inicializace struktur skriptem i jejich skrytí před uživatelem umožňuje lépe kontrolovat jejich platnost. Při pracování s vytvořenými krabičkami se funkcím

skriptu pouze předá její identifikátor. Data se předávají jako index ve skrytém poli.

Dalším cílem návrhu je, aby byl skript snadno použitelný, rozšiřitelný a intuitivní. Skrytí vnitřních hodnot vytvářených struktur k tomuto také napomáhá. Uživatel se nemusí obávat způsobení chybového stavu, ani si nemusí pamatovat kudy má přistupovat jaké vlastnosti, ani jakým funkcím je předávat.

Stejně jsem postupoval při pojmenovávání funkcí a pojmenovaných konstant. Všechny funkce a hodnoty jsou pojmenováné se všemi písmeny malými (lower case) a zkratkovitým pojmenováním. Zapamatování si zkratek a jejich významu může ze začátku dělat potíže, ale při častějším používání je vhodnější mít krátká pojmenování.

6.3 Implementace

Funkce skriptu jsou definované v `libraryI3T.cpp`. Rozšíření interpretru o tyto funkce se provádí funkcí `PlatformLibraryInitI3T`, jak bylo popsáno v sekci 6.1. Funkce manipulující workspace přistupují k workspace přímo, bez zprostředkovávacích struktur. Předpokladem spouštění skriptu je tedy, že v aplikaci existuje workspace, i když není nutné, aby okno workspace bylo otevřené. V souboru `libraryI3T.h` jsou definovaná struktura `ScriptingData`, která je zobrazená na následující ukázce:

```
struct ScriptingData {
 Mat4types mat4Types;
 VecOperators vecOperators;
 FloatOperators floatOperators;
 ArithmeticOperators arithmeticOperators;
 Convertors convertors;
 QuatOperators quatOperators;
 NodeLODs nodeLODs;
 std::vector<glm::mat4>nodeData;
};
```

V `ScriptingData` vidíme definováno několik struktur a vektor typu `glm::mat4`. Ve strukturách jsou definované konstanty, které slouží jako parametry říkající funkcím skriptu vytvářejících nějaký typ operátoru, jaký podtyp oprátoru má být vytvořen. Například, funkce `int mat4oper(int,int,int,char*)` má podtypy `inverse`, `transpose` a další. Úplný výčet podtypů se nachází v sekci 6.3.4 na straně 44.

Struktura `Mat4types` obsahuje výčet typů transformací a operátorů s maticemi 4x4, podobně `VecOperators` pro vektory 3 a vektory 4 a`FloatOperators` pro krabičky pracující se skalárními proměnnými. Struktura `ArithmeticOperators` obsahuje výčet typů základních aritmetických operací, například sčítání a násobení, které můžou být použity na maticích, vektorech i skalárech, poté jsou zde struktury `Convertors`, obsahující výčet konvertorů výstupů krabiček a `QuatOpers`, obsahující výčet kvaternionových operátorů. Poslední struktura `NodeLODs` obsahuje výčet úrovní detailnosti vykreslování krabičky.

Vektor `nodeData` obsahuje data vytvářená skriptem, která jsou potom předávána jako index v tomto vektoru funkcím vytvářející krabičky zobrazující tato data. Kopie dat se předá vytvořené krabičce, která tak není nijak závislá na tomto vektoru. Do tohoto vektoru se ukládají data vytvářená spuštěním skriptu ze souboru i data vytvářená z konzole.

Načtení a ukládání workspace se neděje přímo spuštením skriptu, ale voláním funkci `loadWorkspace` a `saveWorkspace` v souboru `Scripting.cpp`, jejichž předpis je zobrazen na následující ukázce:

```
bool loadWorkspace(const char* filename);
bool saveWorkspace(const char* filename, std::vector<Ptr<
    WorkspaceNodeWithCoreData>>* _workspace);
```

Funkce `saveWorkspace` uloží zadané pole krabiček, neboli scénu, do zadaného souboru jako skript, který provedený interpretrem vytvoří opět tu samou scénu. Funkce `loadWorkspace` spustí zadaný skript a poté smaže data, která byla vytvořena v `nodeData` skriptem. Načtení a uložení je podrobněji rozepsáno v následujících sekcích.

Funkce načítání a ukládání byly testovány manuálním vytvořením grafu ve workspace a poté uložením a opětovným načtením. Nezměnil-li se graf operací, je ověřené, že načtení i ukládání funguje správně. Takto jsem testoval všechny funkce, které popisují workspace, to znamená ukládání vnitřních hodnot krabiček, propojení krabiček, konfigurace vzhledu krabiček a vkládání matic do sekvencí.

6.3.1 Načítání workspace

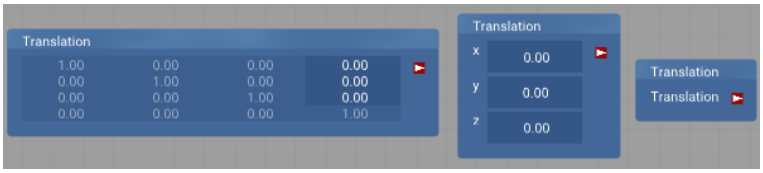
Pro popisování workspace slouží tři hlavní druhy funkcí. Jsou to funkce vytvářející data, funkce vytvářející krabičky a funkce propojující krabičky. Lze vytvářet data typu matice 4x4, vektor 3, vektor 4 a skalár. Tyto funkce vracejí číslo, který je indexem vytvořených dat v poli dat, které je pro uživatele skriptu nedostupné.

Funkce vytvářející krabičky mají parametry označující jejich pozici ve workspace, jejich data a případné další parametry, záležící na typu krabičky. Tyto funkce vrací identifikátor krabičky. Tento identifikátor se pak předává jako parametr jiným funkcím, které nějak pracují s krabičkami.

Třetí typ funkcí je reprezentován například funkcí `plugnodes`, která má jako parametry identifikátory dvou krabiček, které mají být propojeny a číslo vstupu a výstupu, které mají být propojeny. Funkce vrací hodnotu bool, určující, zda krabičky byly propojeny. Propojení může selhat z různých důvodů - neexistující krabičky, neexistující vstup nebo výstup, nebo rozdílné datové typy vstupu a výstupu. Další takové funkce jsou třeba `seqAdd`, která vloží krabičku transformace do sekvence, nebo `unplugInput`, která odpojí zadaný vstup zadané krabičky.

Nakonec je zde pomocná funkce `confnode`, která nastavuje zobrazovací vlastnosti krabičky, jako je počet zobrazených desetinných míst a LOD – detail vykreslení (obrázek 6.1).

Workspace lze načíst jako novou scénu (předchozí obsah je zahozen) pomocí



Obrázek 6.1: Úrovně detailu krabičky.

funkce `bool save(char*filename)`, nebo jako doplnění současné scény o nový obsah pomocí `bool append(char*filename)`.

6.3.2 Ukládání workspace

Při ukládání workspace je potřeba rozebrat obsah workspace a převést každou krabičku a její konfiguraci na textový řetězec, reprezentující kus kódu v jazyce C, který by byl zpracovaný interpretrem. Obsah workspace je pole krabiček `std::vector<Ptr<WorkspaceNodeWithCoreData»`. Pro uložení krabičky je potřeba znát informace, které jednoznačně popisují typ krabičky, vizuální stav krabičky i vnitřní (logický) stav. Logický stav krabičky je dán připojenými vstupy a případnými vnitřními daty krabičky. Vnitřní data jsou matice, vektor, nebo skalár. Vizuální reprezentace krabičky je dána pozicí ve workspace, počtem zobrazených desetinných míst na hodnotách zobrazených krabičkou a úroveň detailu vykreslování, jak je ukázáno na obrázku 6.1.

Pro zjištění typu krabičky a jejího logického stavu je třeba znát její jádro, které lze získat getterem `Ptr<Core::NodeBase> WorkspaceNodeWithCoreData::getNodebase()`. Z jádra pak lze získat následující informace:

```
const DataStore&     data       = nodebase->getData();
const Operation*  operation  = nodebase->getOperation();
const char*             keyword    = operation->keyWord.c_str();
std::vector<Core::Pin>  inputs     = nodebase->getInputPins();
```

Konstanta `DataStore` obsahuje vnitřní data krabičky, má-li nějaké. Z `operation` získáme `keyword`, což je textový řetězec, který umožní identifikovat typ krabičky. Pole `inputs` obsahuje všechny vstupní piny. Pro to, abychom dokázali zapsat propojení pinů, musíme pro každý vstupní pin najít krabičku a její vstup, do kterého je vstupní pin připojený, je-li někam připojený. Kód v následující ukázce zkontroluje, zda je i-tý vstup krabičky někam zapojený, a když je, následně najde krabičku do něj připojenou, na dalším řádku pin krabičky, kam je aktuálně zpracovávaný pin připojený, a na dalším řádku získá jeho index. Smyčka na konci potom najde index krabičky, do které je zpracovávaný vstup připojený.

```
if(inputs[i].isPluggedIn()){
    Ptr<Core::NodeBase> parent = Core::GraphManager::getParent(nodebase,i);
 const Core::Pin*parentpin=inputs[i].getParentPin();
 int indexout=parentpin->getIndex();
 for (int j = 0; j < workspace->size(); j++) {
  if (parent.get() == workspace[j]->getNodebase().get()) {parentindex = j;}
```

```
 }
}
```

Nyní k vizuální reprezentaci krabičky. Přesnost zobrazených hodnot je uložená přímo v krabičce, dostupná getterem `int WorkspaceNodeWithCoreData::getNumberOfVisibleDecimal()`, stejně jako úroveň detailu: `WorkspaceLevelOfDetail WorkspaceNodeWithCoreData::getLevelOfDetail()`. Pozice krabičky ve workspace je uložená v NodeEditoru, kterému se zadá id krabičky, jak je zobazeno na ukázce níže:

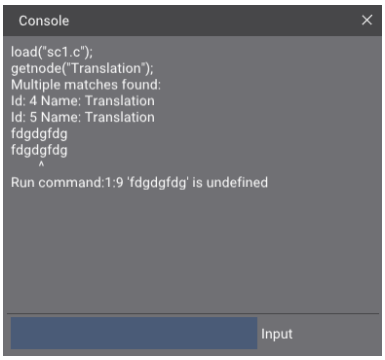
```
WorkspaceNodeWithCoreData* workspacenode= &workspace[i];
ImVec2 pos = ne::GetNodePosition(workspacenode->getId());
```

Ukládání provádí funkce `saveWorkspace(const char* filename, std::vector<Ptr<WorkspaceNodeWithCoreData>>* _workspace)`, kde `_workspace` je vektor krabiček, které se mají uložit. Toto umožňuje uložit pouze zadanou část workspace, čehož využívá funkce skriptu `bool savesel(char*filename)`, která ukládá pouze vybranou část workspace.

6.3.3 Konzole

Konzole umožňuje načítání a ukládání scény a upravování workspace. Workspace lze modifikovat přidáváním a propojováním krabiček, což se děje stejnými funkcemi jako v případě načítání workspace. Při upravování krabičky ve workspace je potřeba získat index této krabičky, k čemuž slouží funkce `int getnode(char*)`. Parametr funkce je popisek v hlavičce krabičky krabičky. V případě, že zadaný název má více krabiček, funkce zároveň vypíše do konzole index všech krabiček s tímto názvem. Návratová hodnota funkce se potom předává jako parametr funkcí, které umožňují propojování, odpojování a mazání krabiček.

Konzole uchovává historii příkazů, kterou lze procházet klávesami šipka nahoru a šipka dolu. V konzoli se zobrazují příkazy zadané uživatelem a také výstup z `std::cout`, kam je přesměrován výstup z picoc. Pokud je potřeba v nějaké funkci skriptu potřeba vypisovat do konzole, například kvůli informování uživatele o chybových stavech, stačí pouze, aby tato funkce zapisovala do `std::cout`.



Obrázek 6.2: Okno konzole v I3T.

6.3.4 Seznam funkcí

Zde je seznam všech funkcí pro práci s I3T přístupných ze skriptu. Funkce, které vrací výsledek operace jako `bool` vrací `true`, byla-li operace úspěšně provedena, `false` jinak. Pro všechny funkce pro vytváření krabiček existuje funkce s příponou `c` (centered), která nemá parametry `x` a `y` a vytvořenou krabičku automaticky umisťuje do středu workspace, například `int mat4oper(int type,int x ,int y,char* header)` – `int mat4operc(int type, char* header)`.

int mat4oper(int type,int x ,int y, char* header). Vytvoří maticový operátor zadaného typu a vrátí jeho id. Možné typy jsou `determinant`, `inverse`, `matadd` (sčítání matic), `matmul` (maticové násobení), `matmulvec`, `vecmulmat`, `floatmulmat`,`free`, (zobrazí vstupní matici), `trackball`, `transpose`, `ortho`, `perspective`, `frustrum`, `axisangle`, `rotatex`, `rotatey`, `rotatez`, `scale`, `lookat`, `translate`.

int mat4(int type, int data, int x, int y, char* header). Vytvoří maticovou transformaci zadaného typu a vrátí její id. Možné typy jsou, `free`, `scale`, `uniscale` (uniformní škálování), `rotatex`, `lookat`, `rotatey`, `rotatez`, `axisangle`, `translate`, `ortho`, `perspective`, `frustrum`.

int vec4oper(int type, int x ,int y, char* header). Vytvoří operátor zadaného typu na vektoru 4 a vrátí jeho index. Možné typy jsou `dot`, `perspdiv` (perspektivní dělení), `norm`, `vecmulfloat`, `add`, `sub` a `mix`.

int vec4(int data, int x, int y, char* header). Vytvoří krabičku pro zadání vektoru 4 a vrátí její id.

int vec3oper(int type, int x ,int y, char* header). Vytvoří operátor zadaného typu na vektoru 3 a vrátí jeho id. Možné typy jsou `cross`, `dot`, `norm`, `vecmulfloat`, `length`, `add`, `sub`, `show` a `mix`.

int vec3(int data, int x, int y, char* header). Vytvoří krabičku pro zadání vektoru 3 a vrátí její id.

int scalaroper(int type, int x, int y, char* header). Vytvoří operátor zadaného typu na skaláru a vrátí jeho index. Možné typy jsou `asinacos`, `sincos`, `cycle` (hodnota ubíhající v čase), `pow`, `clamp`, `signum`, `mul`, `add`, `div`, `mix`.

int scalar(int data, int x, int y, char* header). Vytvoří krabičku pro zadání skaláru a vrátí její id.

int quatoper(int type, int x, int y, char* header). Vytvoří operátor zadaného typu na skaláru a vrátí jeho id. Možné typy jsou `scalarvec3_quat`, `angleaxis_quat`, `vec3vec3_quat`, `quat_scalarvec3`, `quat_angleaxis`, `scalarmulquat`, `quat_euler`, `euler_quat`, `slerp`, `longslerp`, `lerp`, `conjugate`, `qvq`, `inverse`, `norm`, `length`.

int quat(int data, int x, int y, char* header). Vytvoří krabičku pro zadání kvaternionu a vrátí její id.

int convertor(int type, int x, int y, char* header). Vytvoří konverzi zadaného typu vrátí její id. Možné typy jsou `mat_tr` (rozložení matice na rotaci a translaci), `tr_mat` (složení translační a rotační matice), `mat_vecs4`, `mat_quat`, `mat_scalars`, `vecs4_mat`, `vec4_vec3`, `vec4_scalars`, `vecs3_mat`, `vec3_vec4`, `vec3_scalars`, `quat_mat`, `quat_scalars`, `scalars_mat`, `scalars_vec3`, `scalars_vec4`, `scalars_quat`.

int sequence(int x, int y,char* header). Vytvoří krabičku sekvence, do které je možné vkládat transformační matice a vrátí její id.

bool seqadd(int sequence,int node). Vloží krabičku transformace do sekvence.

bool plugnodes(int lnode, int rnode, int output, int input). Připojí výstup `output` krabičky `lnode` do vstupu `input` krabičky `rnode` a vrátí výsledek operace.

bool unpluginput(int node, int input). Odpojí vstup `input` krabičky `node` a vrátí výsledek operace

int getnode(char* header). Vrátí id poslední krabičky se zadaným názvem, nebo -1, není-li nalezena žádná.

bool delnode(int node). Smaže krabičku se zadaným id, existuje-li, a vrátí výsledek operace.

bool confnode(int node, int precision, int lod). . Konfiguruje zadanou krabičku. Povolené hodnoty `lod` jsou `full`, `setvalues` a `label`.

int datamat4(float f1, float f2,...,float f16). Vytvoří matici 4x4 a vrátí její id. Matice je zadávána po sloupcích.

int datavec4(float f1, float f2, float f3, float f4). Vytvoří vektor 4 a vrátí jeho index.

int datavec3(float f1, float f2, float f3). Vytvoří vektor 3 a vrátí jeho index.

int datascalar(float f). Vytvoří skalár a vrátí jeho index.

bool load(char* filename). Načte workspace a vrátí výsledek operace.

bool append(char* filename). Přidá do workspace nový obsah při ponechání předchozího obsahu workspace a vrátí výsledek operace. Nově přidané krabičky jsou označené.

bool save(char* filename). Uloží workspace a vrátí výsledek operace.

bool savesel(char* filename). Uloží pouze krabičky vybrané ve workspace a vrátí výsledek operace.

bool run(char* filename). Spustí zadaný skript a vrátí výsledek operace. Funkce je stejná jako `bool append(char* filename)`, s tím rozdílem, že data vytvořené funkcemi `datamat4`, `datavec4`, ..., nejsou po skončení skriptu smazána a případný přibyvší nový obsah workspace není označený.

Kapitola 7

Závěr

Cíle práce byly až na drobné nedostatky dosaženy. Byly implementovány manipulátory všech transformaci, s výjimkou manipulátorů pro operátor trackball, který nebyl v době průběhu této práce dostupný. Zpětné vazby uživatelů i průběžné konzultace poskytly podněty k úpravám a zlepšením manipulátorů. Během testování bylo odhaleno několik chyb a došlo k zlepšení manipulátorů lookAt, nicméně manipulátory volné transformace nebyly většinou uživatelů odhaleny i přes přidání nápověd, jak je popsáno na straně 33. Načítání a ukládání workspace pokrývá většinu krabiček v I3T, ale není kompletní z toho důvodu, že v době implementace načítání a ukládání workspace tyto krabičky nebyly připravené, nicméně doplnění jejich podpory je jednoduché, jako vzor může posloužit implementace ukládání a načítání již podporovaných krabiček v souborech `Scripting.cpp` a `libraryI3T.cpp`.

7.1 Další úpravy a rozšíření

Přetrvávajícím nedostatkem zůstává nerozpoznání manipulátorů volné transformace většinou uživatelů. Možnosti jsou zásadně přepracovat vzhled a ovládání těchto manipulátorů, nebo do I3T přidat tutoriál k ovládání manipulátorů. Je třeba také brát v úvahu, že testy trvající pouze deset až patnáct minut nepředstavují reálné případy užití, a je možné, že uživatel prozkoumávající a používající I3T vlastní iniciativou bude více experimentovat a rozpozná tyto manipulátory.

Dále bylo během uživatelských testů navrženo vytvořit manipulátor pro trackball, který by bylo možno použít pro editování kvaternionů nebo operátoru trackball.

7.1.1 Zobrazení měr při používání manipulátorů

Uživatel Miroslav navrhl zobrazování osy nebo roviny, ve které probíhá transformace. Toto se týká manipulátorů škálování, translace a lookAt. Také při konzultacích s vedoucím bylo navrženo u manipulátorů rotace zobrazit úseč kruhu, o kterou byla matice rotována při probíhající transakci, jako je tomu u manipulátorů rotace v Unreal Engine (obrázek 2.7, strana 11).

7.1.2 I3T pro počítačovou grafiku

V případě rozšíření I3T pro výuku nejen 3D transformací, ale i ryze grafických úloh, jako je osvětlování a texturování, by bylo potřeba rozšířit systém scény, konkrétně třídu `World` o možnost načtení uživatelských shaderů, textur i modelů. Třída `World` by také obsahovala nastavení světelných zdrojů. Pro texturování by se hodilo, kdyby `GameObject` obsahoval vektor textur místo pouze jedné textury, což by umožnilo přistupovat více textur v uživatelských shaderech.

Literatura

[1] FOLTA, Michal. *Systém na výuku transformací.* Praha, 2016. Diplomová práce. ČVUT v Praze, Fakulta elektrotechnická, Katedra počítačové grafiky a interakce.

[2] FELKEL, Petr a Jaroslav SLOUP. Transformations. In: *PGR: Programování grafiky* [ONLINE]. České vysoké učení technické, 2020. [cit. 19.4.2021]. Dostupné z: https://cent.felk.cvut.cz/courses/PGR/lectures/04_Transform_1.pdf

[3] FELKEL, Petr a Jaroslav SLOUP. Transformations II. In: *PGR: Programování grafiky* [ONLINE]. České vysoké učení technické, 2020. [cit. 18.12.2020]. Dostupné z: https://cent.felk.cvut.cz/courses/PGR/lectures/05_Transform_2.pdf

[4] *Transform Objects* [ONLINE]. Autodesk, inc. 2019. [cit. 8.10.2020]. Dostupné z: https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2019/ENU/Maya-Basics/files/GUID-9622730D-3D21-451C-8BEE-E01BCC979F91-htm.html

[5] *Transforming Actors* [ONLINE]. Epic Games, inc. 2009. [cit. 8.10.2020]. Dostupné z: https://docs.unrealengine.com/en-US/Engine/Actors/Transform/index.html

[6] MARTIN, Adam. Entity Systems are the future of MMOG development – Part 1. In: *T-machine* [ONLINE]. 3.9.2007 [cit. 15.3.2021]. Dostupné z: http://t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1/

[7] KATIC, Steve. Interactive Techniques in Three-dimensional Scenes (Part 1): Moving 3D Objects with the Mouse using OpenGL 2.1. In: *Code Project* [ONLINE]. 2009. [cit. 19.10.2020]. Dostupné z: https://www.codeproject.com/Articles/35139/Interactive-Techniques-in-Three-dimensional-Scenes

[8] POIRIER, Joseph. picoc: A very small C interpreter. In: *GitHub* [ONLINE]. 2015. Dostupné z: https://github.com/jpoirier/picoc

Přílohy

A. Uživatelský dotazník

Úvod

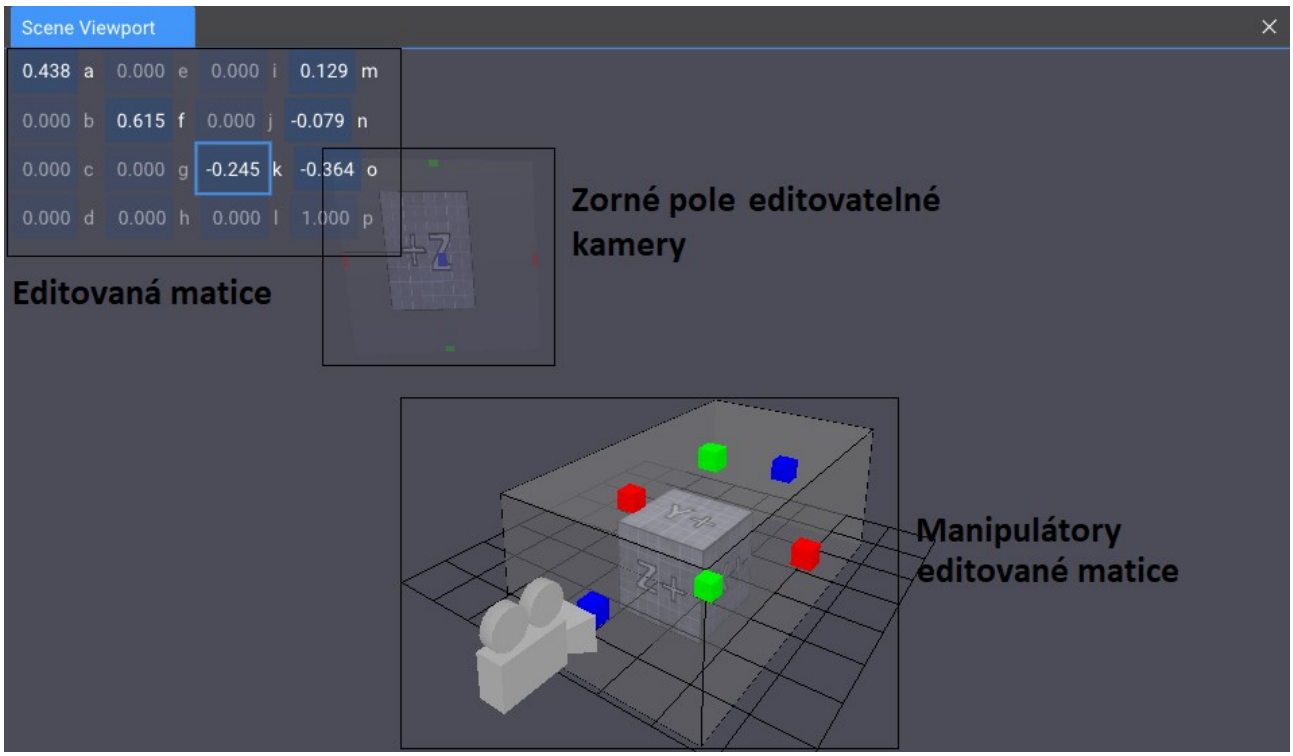
I3T je nástroj sloužící pro výuku 3D transformací. I3T obsahuje panel Workspace, kde je možno vytvářet maticové a vektorové transformace a operátory a propojovat je navzájem. K maticím a sekvencím matic lze připojit 3D model, který se transformovaný sekvencí, ke které je připojen, zobrazí v panelu Viewport.
Matice bude možné upravovat přímo z workspace a pozorovat změny transformovaného objektu ve viewportu, nebo upravovat matici pomocí manipulátorů ve viewportu a sledovat změny ve workspace.
I3T je zatím v rozpracovaném stavu. Ve finální verzi bude možné vytvářet matice libovolného typu ve workspace a upravovat je manipulátory, v současné verzi je matice upravovaná manipulátory zadána napevno a zobrazuje v levém horním rohu viewportu a její typ lze přepínat klávesnicí.

Typy transformačních matic jsou tyto: Rotace okolo osy X, Rotace okolo osy Y, Rotace okolo osy Z, změna měřítka, posun, volná editace (možnost libovolně měnit políčka matice), lookAt (matice rotace, jejíž osa Z směřuje do zadaného místa v prostoru), matice ortografické projekce, matice perspektivní projekce, matice frustrum (perspektivní projekce se volným středem symetrie - vzdalující se předměty nemusí ubíhat ke středu obrazovky).

Ovládání

Typ této matice lze přepínat klávesami shift+jedna z kláves { s (škálování) , t (translace) , p (perspektivní projekce) ,f (frustrum),o (orografické projekce) ,x (rotace okolo x) ,y (rotace okolo y) ,z (rotace okolo z) ,g (volná transformace),l (lookAt)}
Ve scéně se lze pohybovat šipkami a myší.

Šedá kostička na následujícím obrázku je zobrazená maticemi translace, rotace X, rotace Y, rotace Z, škálování, volná transformace – v tomto pořadí.
Editovanou matici lze upravovat kliknutím na její políčko a táhnutním nebo také *manipulátory editované matice*.
Směr pohledu editovatelné kamery, také zobrazené na obrázku, je dán maticí lookAt
Zorné pole editovatelné kamery je dáno jednou z matic perspective, frustrum a ortho – tou, která je právě aktivní.



Úvodní dotazník
Používal jste někdy dříve nástroj I3T?

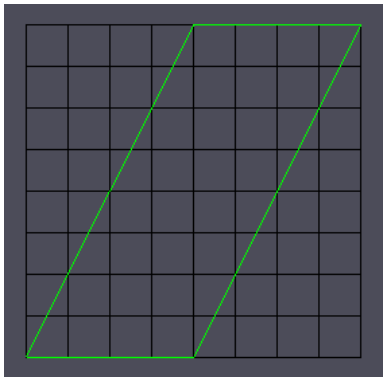
Pracoval jste někdy s prostředími pro 3D počítačovou grafiku? Pokud ano, s jakými?

Pracoval jste někdy s maticemi 3D transformací?

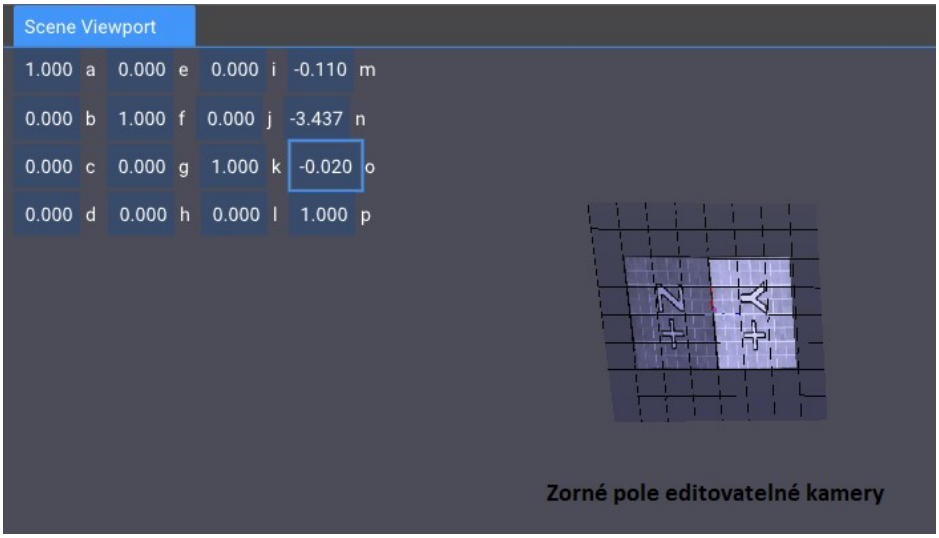
Úvodní úloha
Seznamte se s funkcionalitou prvků zobrazených na předchozím obrázku. Vyzkoušejte si ovládání.

Testovací úlohy

1. Rotujte editovatelnou kameru pomocí lookAt.
2. Posuňte šedou kostičku, tak, aby byla v zorném poli editovatelné kamery.
3. Nastavte zorné pole editovatelné kanery tak, aby kostička vyplňovala co největší část zorného pole. Zkuste to pro matice ortho, perspective i frustrum.
4. Rotujte kostičku tak, aby se editovatelná kamera dívala na stranu šedé kostičky Y+.
5. Škálujte a rotujte šedou kostičku, tak, aby přesně pokryla celou mříž.
6. Pomocí volné transformace zkoste a škálujte hrany kostičky tak, aby kostička pokrývala mříž takovýmto způsobem:



7. Nastavte ortografickou kameru a transformaci kostičky tak, aby *zorné pole editovatelné kamery* vypadalo takto:



Výsledky

1. Pomohlo vám I3T najít souvislosti mezi transformacemi a jejich maticemi?

2. Ohodnoťte obtížnost úloh na stupnici od 1 do 5: (1 – nejlehčí, 5 – nejtěžší)
   1.
   2.
   3.
   4.
   5.
   6.
   7.

3. Byly některé úlohy těžké? Proč?

4. Použil jste škálování/translaci v rovině?

5. Použil jste uniformní škálování?

6. Použil jste škálování os při volné transformaci?

7. Použil jste při volné transformaci zkosení os pomocí šipek translace?

8. Použil jste při volné transformaci zkosení os pomocí rotace?

9. Bylo chování a ovládání manipulátorů nějakým způsobem neočekávané? Pokud ano, jak?

10. Zlepšil byste nějak manipulátory? (citlivost ovládání, způsob ovládání, tvar manipulátorů, klávesové zkratky, atd.)

11. Narazil jste na chybu aplikace nebo manipulátorů? Pokud ano, jakou?

B. Kompilování.

I3T je vyvíjeno prostřednictvím repozitářového systému git. Repozitář projektu je potřeba klonovat rekurzivně, pomocí přepínače gitu --recurse-submodules. Tak se naklonují i všechny externí závislosti I3T. V kořenu projektu i3t-bunny je soubor \CMakeLists.txt, což je popis projektu, podle kterého program CMake vygeneruje projektové soubory. Požadovaná verze CMake je 3.13 nebo vyšší. Spustí-li se CMake ve složce i3t-bunny, potom stačí zadat parametry kořen projektu a cílové vývojové prostředí, například takto:
cmake -S . -G "Visual Studio 16 2019".
Příkaz cmake --help zobrazí seznam všech podporovaných vývojových prostředí. Kód I3T používá vlastnosti C++20, proto v případě generování souborů pro vývojové prostředí, které nemá standard C++20 jako výchozí, je potřeba ho explicitně povolit. Výsledkem kompilace je I3T.exe ve složce Binaries/Debug.