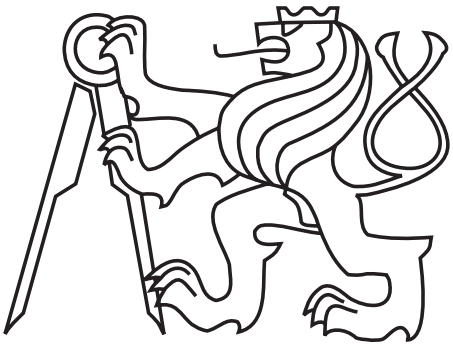
České vysoké učení technické
Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE



Michal Folta

Systém na výuku transformací

Katedra počítačové grafiky a interakce

Vedoucí diplomové práce: Ing. Petr Felkel, Ph.D.
Studijní program: Otevřená informatika
Studijní obor: Počítačová grafika a interakce

Praha 2016

Rád bych poděkoval svému vedoucímu Ing. Petru Felkelovi, Ph.D. za velmi zajímavé téma, ochotu, trpělivost, cenné rady a podporu, za níž moc děkuji i své rodině.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V ..... dne ..... Podpis autora:

Název práce: Systém na výuku transformací

Autor: Michal Folta

Katedra: Katedra počítačové grafiky a interakce

Vedoucí diplomové práce: Ing. Petr Felkel, Ph.D.

Abstrakt: Geometrické transformace jsou v počítačové grafice velmi důležité. Pro řadu lidí je však velkým problémem jim porozumět, což dokládá i velké množství tištěných publikací a tutoriálů na toto téma. Většina z nich trpí společným nedostatkem, a to žádným nebo jen nízkým stupněm interaktivity. Proto byla v rámci tohoto projektu vytvořena výuková aplikace obsahující interaktivní rozhraní založené na principu propojovatelných komponent, které umožňuje studovat transformace a jejich hierarchie názorným způsobem. Aplikace je využitelná jako demonstrační nástroj na přednáškách, pro vytváření úloh na cvičení a hlavně pro samostatné studium. Aplikace odstraňuje nedostatky existujících výukových aplikací, které jsou jen úzce specializované na jeden typ transformací, jsou zastaralé nebo popisují problém pouze ve dvou dimenzích.

Klíčová slova: transformace, výuka, aplikace, počítačová grafika, OpenGL, GLM

Title: Teaching of Transformations

Author: Michal Folta

Department: Department of Computer Graphics and Interaction

Supervisor: Ing. Petr Felkel, Ph.D.

Abstract: Geometric transformations in computer graphics are very important. For many people it is a big problem to understand, as evidenced by the large number of printed publications and tutorials on this topic. Most of them suffer from the common drawback of a low degree of interactivity or even no interactivity at all. Therefore, in this project developed educational application contains interactive interface based on the principle of connectable components, which enables the study of transformation and their hierarchy in an illustrative way. The application is usable as a demonstration tool in lectures, for creating tasks for exercise and especially for independent study. Application eliminates the shortcomings of existing educational applications that are narrowly specialized in one type of transformation are outdated or describe the problem in only two dimensions.

Keywords: transformations, application, teaching, computer graphics, OpenGL, GLM

Obsah

1. Úvod

Cílem práce je vytvořit aplikaci sloužící k výuce problematiky transformací v prostoru. Uživatel bude mít možnost názorným způsobem prostudovat a pochopit systém transformací a transformačního řetězce používaného k zobrazování objektů v počítačové grafice. Projekt je zaměřen na seznámení se systémem transformací, které používá knihovna Open Graphics Library (OpenGL) a OpenGL Mathematics (GLM), neboť tyto prostředí jsou hojně využívány při výuce. Princip transformací je ale obecný a aplikaci lze využít i k výuce v jiných prostředích. Cílem je tedy vytvořit nástroj, který interaktivním způsobem umožní prostudovat základní techniky používané v počítačové grafice.

Aplikace je určena především těm, kteří se setkávají s nesnázemi při studiu transformací nebo ve chvíli, kdy se je snaží použít v praxi. Problematika transformací a jejich matematická reprezentace může být pro mnohé z nás velmi neintuitivní. Přesto, že je k dispozici velké množství zdrojů, které se výuce transformací věnují, neoplývají dostatečnou možností interakce, která by proces učení ulehčila. Už samotné množství těchto dostupných zdrojů dokazuje, že se jedná o komplikované téma. Lze nalézt doslova stovky publikací, článků, internetových tutoriálů a fór, které se tématem transformací v počítačové grafice zabývají. Aplikací, které se tomuto tématu věnují, je však nedostatek. Ve většině případů se jedná o nekvalitní produkty, které postrádají jakoukoliv pedagogickou hodnotu. Nebo se jedná o programy, které jsou sice kvalitní, ale velice úzce zaměřené na konkrétní podproblém.

Při hledání dostupných zdrojů výuky jsem narazil na dva zajímavé. Jedním je Úvod do geometrických transformací na výukových stránkách Khan academy [Aca15]. Autor v několika lekcích vysvětluje naprosté základy transformací, jako co je to translace, rotace nebo škálování. Druhým je výukové video Model View Projection Matrices [Kin13], ve kterém autor vysvětluje principy modelové, pohledové a projekční transformace. V obou případech autoři používají jednoduchou aplikaci, na které vykládanou látku demonstrují. V prvním případě se jedná o grafické rozhraní umožňující aplikovat vybrané transformace na množinu bodů. V druhém pak o aplikaci, ve které je možné nastavit jednotlivé členy transformačních matic a pozorovat výslednou transformaci na geometrickém objektu. Zajímavé závěry však lze vyvodit z diskuze, které tyto zdroje obsahují. V prvním případě přibližně 38% relevantních příspěvků naznačuje, že jejich autoři probírané problematice vůbec nerozumějí. Dalších 48% má nějaké nejasnosti a 10% tazatelů by rádo vlastnilo použitou aplikaci. V druhém zdroji pak nepochopení vyjadřuje 16% čtenářů, dalších 11% má nejasnosti a použitou aplikaci by chtělo 40% dotazovaných. Ty hodnoty ukazují, že osvojit si principy transformací má problém velké množství lidí a že mnoho z nich by aplikaci umožnující dané téma studovat praktickým a interaktivním způsobem uvítalo.

V následujících kapitolách se budeme věnovat mnoha dostupným zdrojům výuky transformací. Velká pozornost bude kladena na již existující aplikace. Cílem je prostudovat metodiku výuky, které tyto zdroje používají a určit jejich klady a nedostatky. Na základě těchto informací nakonec navrhneme a implementujeme vhodnou výukovou aplikaci.

2. Dostupné zdroje a aplikace

V této kapitole se budeme věnovat dostupným zdrojům výuky transformací. Zaměříme se především na zdroje spojené s transformacemi v OpenGL. Jako většina lidí, nejprve zvolíme snadnější cestu a podíváme se na některé tutoriály běžně dostupné na internetu. Po zjištění, že tento zdroj nezachází dostatečně do hloubky, budeme potřebné informace hledat v odborných publikacích. Nakonec se podíváme na dostupné výukové aplikace, které by nás, při tvorbě programu, mohly inspirovat.

2.1 Tutoriály o transformacích v OpenGL

Ve snaze nastudovat problematiku transformací v počítačové grafice, ať už v OpenGL pomocí knihovny GLM nebo zcela obecně, se nejeden z nás jistě nevyhne tutoriálům, kterých je na internetu k dispozici nepřeberné množství. V následujících odstavcích postupně probereme tři typické zástupce.

Jedním z tutoriálů je [Ove15]. Ten se specializuje na osvětlení transformací v OpenGL za použití knihovny GLM. Hned v úvodu autor popisuje základní matematické operace s maticemi a vektory, jako je násobení a sčítání, což napovídá, že je tento tutoriál koncipován pro opravdové začátečníky. Poté následuje ukázka a jednoduché vysvětlení translace, škálování a rotace kolem bázových vektorů. O existenci bázových vektorů a tedy i významu prvků v transformační matici však zde nepadne jediná zmínka. Dozvídáme se tedy, že čtvrtý sloupec matice obsahuje translaci. Na hlavní diagonále se nacházejí složky škálování. A že existují tři rotace kolem os x, y a z. Dále následuje zmínka o možnosti jednotlivé transformace kombinovat tím, že dané matice vynásobíme. Autor toto téma popíše na šesti řádcích včetně ukázky transformace posunutí a škálování. Autor dále velmi zkráceně popisuje význam modelové, pohledové a projekční transformace a jejich umístění ve výsledném řetězci. Nakonec následují ukázky tvorby těchto matic za použití knihovny GLM a jejich aplikace v jednoduchém shaderovacím programu. Čitatel tohoto článku pravděpodobně zajásá a s přesvědčením, že o dané problematice ví dost, se vrhne na tvorbu své vlastní grafické aplikace. Celkem snadno přiměje svůj grafický objekt k pohybu a rotaci, ale brzo zjistí, že ve chvíli, kdy transformace daného objektu je již v nějakém obecném tvaru, jen stěží tento objekt přiměje například k pohybu nebo rotaci kolem globálních souřadnicových os. Pravděpodobně se nakonec dostane do fáze, kdy bude u svých objektů ukládat pozici pomocí vektoru a rotaci zas pomocí tří úhlů a výslednou transformaci vypočte až před vykreslením objektu. Ani tento přístup mu však nepomůže například k rotaci kolem obecného bodu a osy, nebo k pohybu tělesa v lokálním souřadném systému tělesa jiného.

Další článek [ot11] je součástí řady tutoriálů o vývoji aplikací v OpenGL. Opět se nejprve dozvíme jak násobit matice a vektory. Autoři navíc přidávají krátkou zmínku o existenci homogenních souřadnic. Opět následuje ukázka transformací posunutí, škálování a rotací pomocí eulerových úhlů, krátká zmínka o důležitosti pořadí aplikace transformací a významu modelové, pohledové a, o něco detailněji popsané, projekční transformaci. Na konec můžeme opět pozorovat příklad za použití knihovny GLM a aplikaci v jednoduchém shader programu. Tutoriál vesměs

nepřináší nic nového a danou problematiku opět popisuje jen zcela okrajově.

K dispozici je na internetu mnoho dalších tutoriálů a článků o dané problematice, které postupují podle naprosto stejného schématu [vO11], [Pro13], [Ahn13], [Ala13], [Dal12], [Gro10], [HC12], [Rat15]. Situace je o to horší, že na každý zdroj, který jsem zde uvedl, připadají další tři, které vysvětlují danou tématiku ještě pro staré verze OpenGL. Odkazy na ně zde ani neuvádím. Tutoriály obecně nezacházejí do hloubky a jejich nastudování nevede k dostatečnému pochopení problematiky transformací.

2.2 Literatura

Se zjištěním, že internetové tutoriály nestačí, je nutné vyhledat jiný zdroj informací. Mezi ty nejkvalitnější by zcela jistě měla patřit literatura, které je na toto téma k dispozici nepřeberné množství. Podle zaměření lze tyto knihy rozdělit na tři kategorie. Za prvé knihy, věnující se práci v OpenGL obecně, jako je například OpenGL Super Bible [Jr.11]. Popisují všechny možné aspekty práce s touto knihovnou a protože jsou transformace její nedílnou součástí, obsahují vždy i kapitolu, která se transformacím věnuje. Dále však existují publikace, které se přímo specializují na matematiku spojenou s počítačovou grafikou. Například 3D Math Primer For Graphics And Game Development [Par02], nebo Mathematics for 3D Game Programming and Computer Graphics [Len11]. Problematice transformací by tedy měly věnovat mnohem náležitější pozornost. Do poslední kategorie pak patří knihy, které se již nevěnují počítačové grafice a transformace vysvětlují čistě po matematické stránce. Například učebnice Lineární algebra [Ol07]. Postupně probereme zmíněné zástupce těchto publikací a podíváme se, jakým způsobem se problematiku transformací snaží vysvětlit.

Začneme publikací OpenGL Super Bible [Jr.11]. Jak již bylo řečeno, tato kniha popisuje práci s knihovnou OpenGL. Jedná se opravdu o vyčerpávající publikaci, která na svých skoro tisíci stránkách postupně projde danou problematiku od základních konceptů až po pokročilé techniky. Samozřejmě nechybí kapitola věnovaná transformacím. Autor však ihned v úvodu kapitoly zmiňuje míru komplexnosti transformací a čtenáře varuje, že se tomuto tématu věnuje pouze v základech a že pro hlubší pochopení toto pojednání stačit nebude. Na začátek opět přichází krátké pojednání o tom, co je to vektor a matice a jaké operace s nimi lze provádět. Poté nás autor postupně seznámí s konceptem modelové, pohledové a projekční transformace. V části pojednávající o modelové transformaci jsou krátce popsány transformace posunutí, otočení a škálování. Na pár řádcích je poté zmínka o možnosti skládání jednotlivých transformací násobením jejich reprezentačních matic. Je opravdu zajímavé, že tomuto tématu je v kapitole, která čítá přes padesát stránek, věnováno tak málo prostoru, přestože se jedná o tak fundamentální a zároveň velmi neintuitivní téma. Dále autor popíše funkci perspektivní a ortogonální projekce a pohledové transformace a jak tyto transformace nakonec použít. Zbytek kapitoly obsahuje ukázky samotné implementace, kterým je věnována podstatná část celého pojednání. Autor vše demonstruje za použití knihovny Math3d a samotné matice transformací ani nikde nezobrazuje. Jedná se tedy o pouze o velmi slabý úvod do problematiky transformací, který je svou hloubkou i stylem srovnatelný s internetovými tutoriály, které popisuji v předcházející kapitole. Troufnu si tvrdit, že většina čtenářů této publikace v

ní nenalezne o matematických principech transformací nic nového, protože tyto základy již dávno znají.

Naštěstí jsou však k dispozici publikace, které se specializují přímo na matematiku potřebnou v počítačové grafice. Lze tedy předpokládat, že nám o transformacích poskytnou všechny informace, které jsou nutné k jejich úplnému pochopení. Takovou knihou by mohla být například, již výše zmíněná, 3D Math Primer For Graphics And Game Development [Par02]. Kromě krátkého úvodu publikace velmi příhodně začíná zcela obecným popisem významu souřadných soustav a vztahů mezi nimi. Dále popisuje některé, pro počítačovou grafiku zajímavé, jako například souřadnou soustavu světovou, modelovou a soustavu kamery. Nikoho, kdo se v problematice transformací v prostoru orientuje, jistě nepřekvapí, že autor tomuto tématu, zatím zcela abstraktně, věnuje tolik prostoru. A to dokonce více prostoru než například výše popsané tutoriály věnují celému tématu.

Dále autor velmi vyčerpávajícím způsobem vysvětlí, co je to vektor a matice. Jaké operace nad nimi můžeme provádět a nakonec jakým způsobem může matice transformovat vektor. To celé čistě z matematického hlediska. Na přibližně sté stránce se konečně dostáváme k transformacím samotným. Celá kapitola nejprve popisuje princip samotných transformací. Velmi vhodně demonstruje dva přístupy chápání této problematiky. Za prvé může čtenář nahlížet na transformaci jako na transformaci objektu, přesněji jeho vrcholů. Nebo za druhé lze chápat transformaci jako transformaci souřadné soustavy, ve které jsou vrcholy objektu vyjádřeny. Nakonec je názorně demonstrováno, že se jedná o to samé. V této chvíli již má čtenář velmi dobrý teoretický základ o dané problematice. K pochopení transformací je totiž nutné plně rozumět významu souřadných soustav a uvědomit si, že transformace není jen například posun nebo rotace, ale hlavně právě přechod mezi různými souřadnými soustavami. Dále pak následuje vysvětlení všech možných základních transformací, jako jsou posuny a rotace, ale i různé projekce a podobně. Všechny tyto základní transformace jsou rozebrány velmi podrobně, včetně jejich odvození a tvarů výsledných matic.

Ve výsledku tato publikace poskytuje prakticky veškeré teoretické informace, které je k pochopení transformací nutné znát. Není divu, že autoři knihy OpenGL Super Bible [Jr.11], kterou popisuji výše, tuto publikaci ve svém díle doporučují. Čtenář by se však měl připravit na to, že se bude muset prokousat stovkami stránek, což jen dokazuje, jak je toto téma komplexní. Bude-li však chtít transformacím plně rozumět, podobné literatuře se zcela určitě nevyhne.

Jedno téma však zůstává otevřené. A to kombinování jednotlivých transformací. Této problematice autor věnuje přibližně jednu stránku, na které opět pouze poukáže, že lze jednotlivé transformace kombinovat násobením jejich matic. Je zajímavé, že si toto téma nezasloužilo více pozornosti v takto rozsáhlé a podrobné publikaci.

Do poslední kategorie publikací o transformacích patří různé učebnice především lineární algebry. Takovouto publikací je například Lineární algebra [Ol07] nebo Pěstujeme lineární algebru [Zah03]. Jako zdroj informací poslouží spíše jen tomu, kdo již o transformacích ví prakticky vše a chce svoje znalosti dotáhnout v teoretické rovině do konce. Pro člověka, který se chce pouze seznámit s transformacemi na takové úrovni, aby mohl vytvářet 3D aplikace, bude pravděpodobně mít tato literatura odrazující efekt. Troufnu si dokonce tvrdit, že ten, kdo ještě neprošel žádným kurzem lineární algebry, bude jen marně hledat spojitost mezi

problematikou transformací objektů v počítačové grafice a transformacemi, tak jak je tyto publikace popisují.

Popsané publikace pojednávají o problematice transformací sice v různých stupních detailu, ale velice podobnou cestou. Přesto, že se ve všech případech jedná o kvalitní literaturu, všechny tyto knihy trpí nedostatkem. A to je možnost si probíraný obsah prakticky vyzkoušet. Často se stává, že při čtení výukového textu má člověk pocit, že danému tématu rozumí, ale jakmile má nabyté znalosti uvést do praxe, ukáže se, že je tomu jinak. Z toho důvodu se v následující části textu budeme zabývat aplikacemi, které by tento nedostatek měly doplnit.

2.3 Aplikace

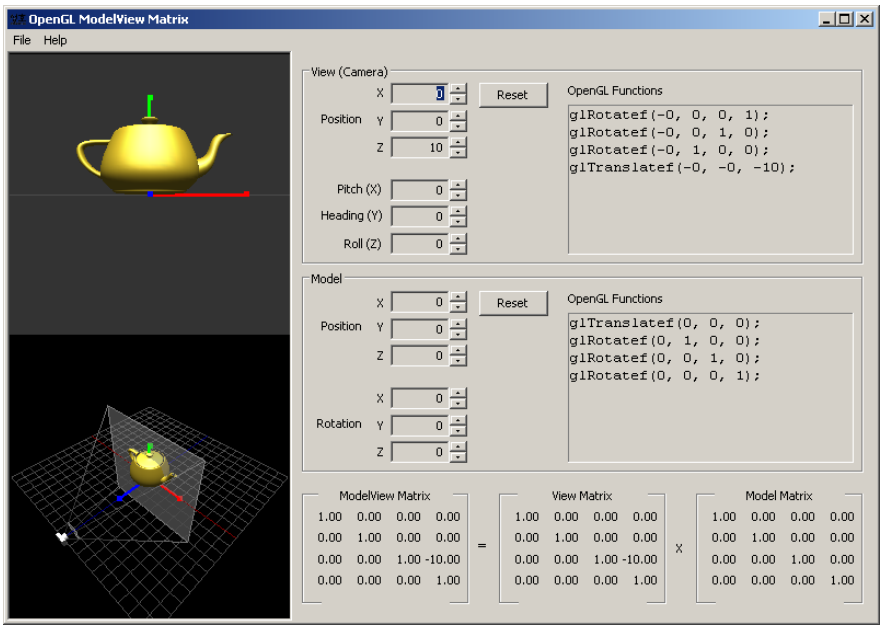
Dostupných aplikací věnujících se problematice transformací a jiným tématům v počítačové grafice není mnoho. Na následujících řádcích se budu věnovat zhodnocení kladů a záporů provedení dostupných aplikací, a to z toho důvodu, aby se výsledný program vyvaroval chyb a těžil z pozitiv, které tyto aplikace mají. Mezi hlavní zdroje patří aplikace od Song Ho Ahn [Ahn13] demonstrující funkci projekčních matic a výukové programy Siggraph od Nate Robinse [SIG01] zaměřené na pochopení práce s knihovnou OpenGL.

Ideální aplikace by měla splňovat následující. Zobrazovat jednotlivé transformační matice, jejich číselné hodnoty a pořadí, v jakém jsou aplikovány. Dále by měla ukazovat umístění vyšetřovaných transformací v celém řetězci transformací. Tedy jejich umístění vzhledem k ostatním transformacím jako je například projekce nebo pohledová transformace. Dále by měla danou problematiku demonstrovat za použití moderní verze OpenGL. Přesněji použitím knihovny GLM nebo i jiným způsobem za použití vertex shaderu. V poslední řadě musí vysvětlovat transformace v trojrozměrném prostoru a ne jen ve dvou rozměrech.

Shrnutí kritérií:

- zobrazuje transformační matice
- zobrazuje umístění transformací v transformačním řetězci
- demonstruje použití v moderní verzi OpenGL
- popisuje transformace ve 3D

Zdrojem pro výuku tvorby grafických aplikací v OpenGL jsou osobní stránky pana Song Ho Ahn [Ahn13], kde lze najít množství tutoriálů a příkladů. Mimo jiné také obsahují detailní tutoriál popisující transformace pomocí matic a dva výukové programy demonstrující použití projekční a pohledové transformace v OpenGL.

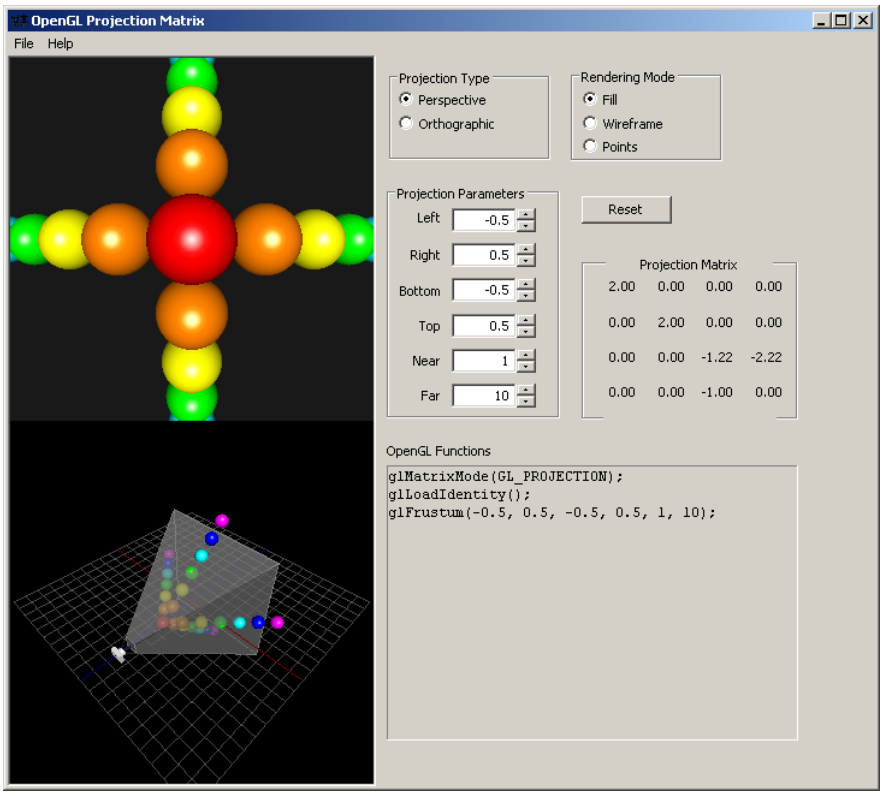


Obrázek 2.1: Song Ho Ahn: OpenGL ModelView Matrix

Prvním výukovým programem je OpenGL ModelView Matrix (obrázek 2.1). Jak již název napovídá, jedná se o aplikaci demonstrující funkci modelové a pohledové transformace. Uživatel může nastavit pozici a orientaci kamery i pokusného modelu zadáním translačního vektoru a tří eulerových úhlů. V pravé části lze pozorovat, jak z těchto hodnot lze pomocí funkcí v OpenGL vytvořit finální pohledovou a modelovou matici. Tyto matice jsou pak zobrazeny ve spodní části včetně výsledku jejich násobení. Celá situace je navíc zobrazována ve dvou náhledech, kde je vidět jak pohled z výsledné kamery, tak i nezávislý náhled celé scény. V tomto náhledu jsou velmi příhodně zobrazeny i osy hlavního souřadnicového systému a hlavně pak pohledový jehlan vyšetřované kamery s přední i zadní ořezávací rovinou. Přesto, že jsou náhledy matic k dispozici, jejich umístění v celém transformačním řetězci není nijak objasněno. Drobným nedostatkem je také skutečnost, že ačkoliv je ve skutečnosti matice pohledové transformace inverzí, tento fakt není nikde vysvětlen. Uživatel si možná ani nepovšimne, že při nastavení pozice a rotace kamery jsou ve skutečnosti do transformace dosazovány hodnoty opačné. Tyto nedostatky však můžeme omluvit s přihlédnutím k faktu, že program je doplňkem k vyčerpávajícímu tutoriálu. Program také bohužel používá starou verzi OpenGL a ukázka použití funkcí glRotatef a glTranslatef již tedy nemá příliš velkou pedagogickou hodnotu.

zobrazuje matice	umístění v řetězci	moderní OpenGL	3D
✓			✓

Tabulka 2.1: Song Ho Ahn: OpenGL ModelView Matrix



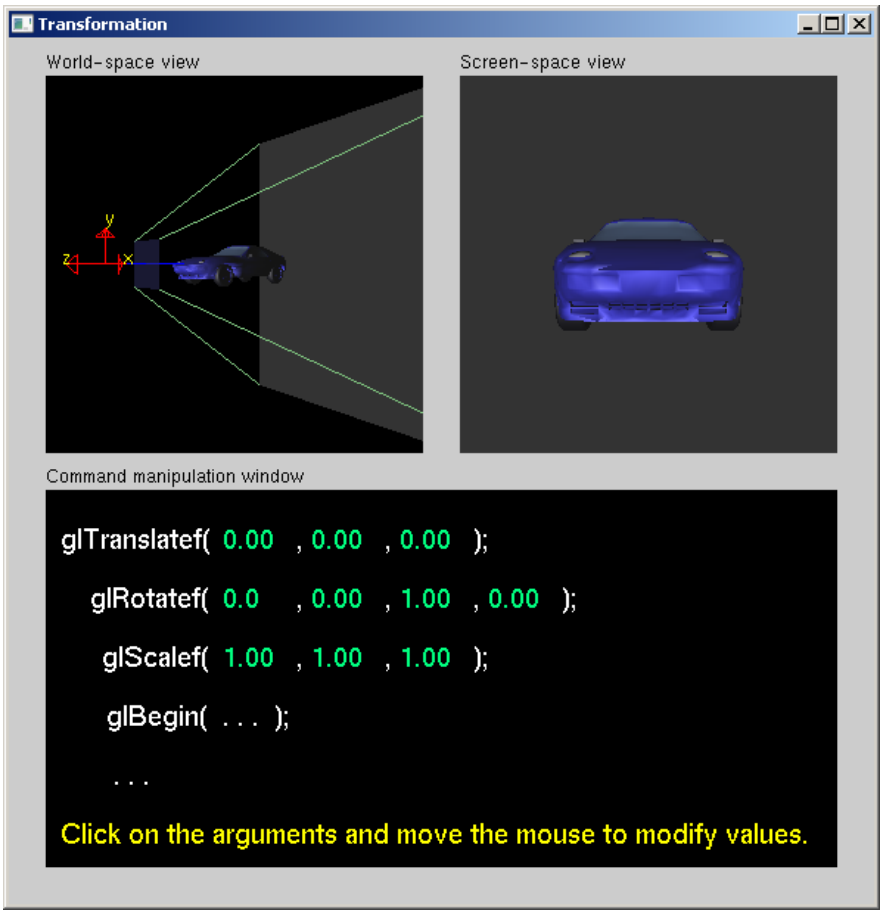
Obrázek 2.2: Song Ho Ahn: OpenGL Projection Matrix

Druhým programem je OpenGL Projection Matrix (obrázek 2.2), který má osvětlit princip projekční matice v OpenGL. Na výběr dostáváme ze dvou druhů projekcí. Perspektivní projekce reprezentované funkcí glFrustum a ortografické funkcí glOrtho. Opět se jedná o implementaci ve staré verzi OpenGL. Ve spodní části můžeme vidět, jak tyto funkce použít v kódu včetně dosazovaných hodnot, které můžeme nastavit. Výsledný efekt lze opět vidět ve dvou náhledech a je zde také vypsána výsledná matice projekce, u které ovšem není objasněn jak význam v celém řetězci transformací, tak význam samotných hodnot v matici. Velkým pozitivem je opět možnost pozorovat scénu v nezávislém náhledu, kde je vidět pohledový jehlan a ořezávací roviny.

zobrazuje matice	umístění v řetězci	moderní OpenGL	3D
			✓

Tabulka 2.2: Song Ho Ahn: OpenGL Projection Matrix

Významným zdrojem výukových aplikací knihovny OpenGL jsou dema Siggraph, které naprogramoval Nate Robins [SIG01]. V těchto aplikacích si uživatel může vyzkoušet několik základní technik a principů používaných v knihovně OpenGL a počítačové grafice obecně, jako jsou například transformace, projekce, techniky texturování, stínování a osvětlení.

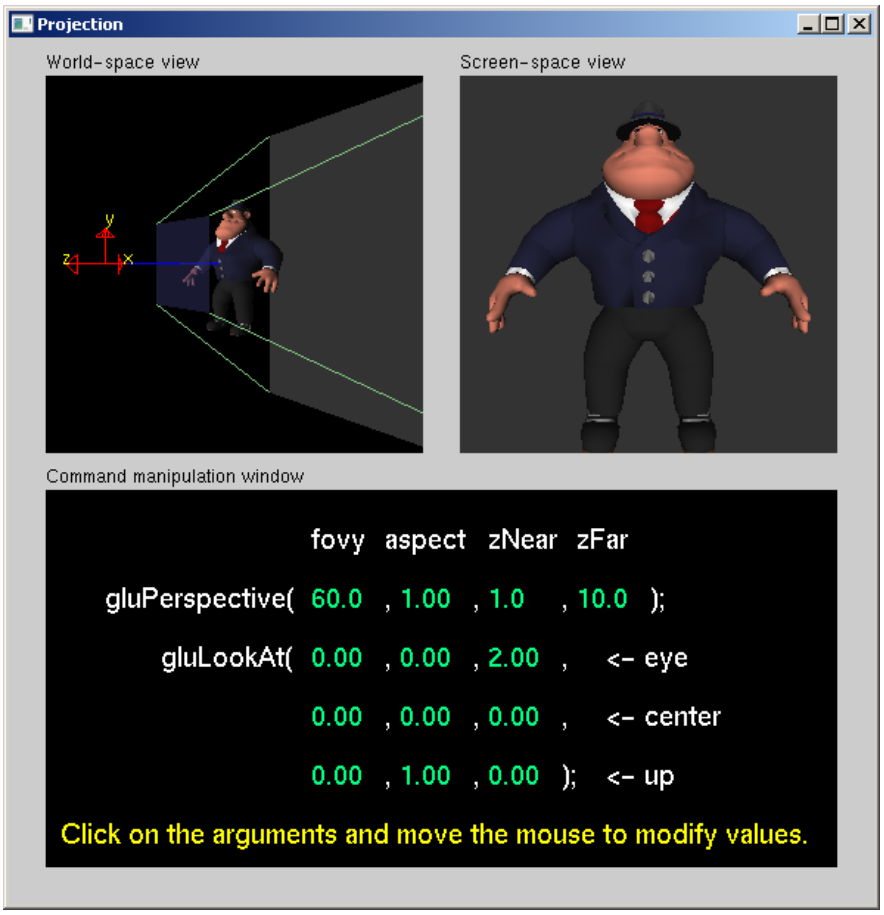


Obrázek 2.3: Siggraph: Transformation

Program Transformation (obrázek 2.3) uživatele seznamuje s modelovou transformací a to nastavením parametrů funkcí glTransltef, glRotatef a glScalef, u kterých ovšem není zobrazen popis jednotlivých parametrů. Výsledek je možné pozorovat jak z pohledu kamery, tak i z jakéhosi globálního náhledu, ve kterém se ovšem namísto modelu pohybuje kamera, což může být trochu matoucí. Pozitivní je, že lze prohodit pořadí translace a rotace a uživatel tak může pozorovat, jaký efekt to má na výslednou transformaci. V tomto směru je ovšem program nedotažený do konce, neboť již nelze zaměnit pořadí u škálování, které je v tomto případě aplikováno vždy na konci a nelze tedy nastavit libovolné pořadí dostupných transformací. Nikde také není zobrazena výsledná transformační matice ani její umístění v celém řetězci transformaci. To je ovšem pochopitelné, vzhledem k tomu, že se opět jedná o ukázku koncipovanou ve staré verzi OpenGL, kde se s modelovou maticí pracovalo způsobem: glMatrixMode(GL_MODELVIEW), glLoadIdentity(), glTranslatef() ... Uživatel tedy nemusel znát umístění modelové transformace, například vzhledem k projekci.

zobrazuje matice	umístění v řetězci	moderní OpenGL	3D
			✓

Tabulka 2.3: Siggraph: Transformation



Obrázek 2.4: Siggraph: Projection

Dalším programem je Projection (obrázek 2.4) demonstrující nastavení projekční transformace a pohledové transformace kamery. Toho lze opět dosáhnout zadáním parametrů do vybraných funkcí. K dispozici je perspektivní projekce glFrustum, gluPerspective a ortogonální projekce glOrtho. Pohledovou transformaci lze nastavit pomocí funkce gluLookAt. Opět je zde možnost pozorovat výsledný efekt z pohledu kamery i z globálního náhledu, ve kterém je vykreslen pohledový jehlan včetně ořezávacích rovin. Nedostatkem je, že v tomto globálním náhledu nelze nastavit úhel pohledu a není úplně jasné, jakým způsobem se vyšetřovaná kamera vlastně pohybuje. Výsledné transformační matice opět nejsou nikde zobrazeny.

zobrazuje matice	umístění v řetězci	moderní OpenGL	3D
			✓

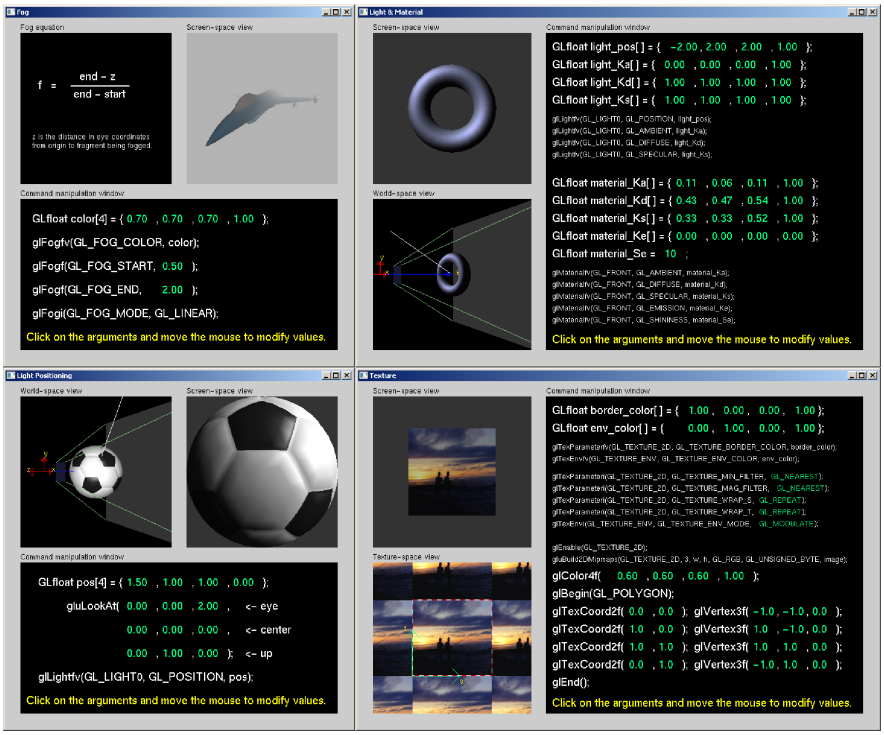
Tabulka 2.4: Siggraph: Projection

Součástí výukových dem Siggraph je i čtveřice aplikací (obrázek 2.5), které se sice nezabývají přímo transformacemi, ale za zmínku stojí, neboť, přestože je vyvíjený program zaměřen na transformace obdobným způsobem jako tyto aplikace, může uživateli přiblížit i jiné ze základních technik, jež jsou v počítačové grafice hojně používány. První aplikací je program Fog (Mlha). Jedná se o techniku hojně využívanou pro omezení dohledu ve virtuální scéně, kdy je barva každého

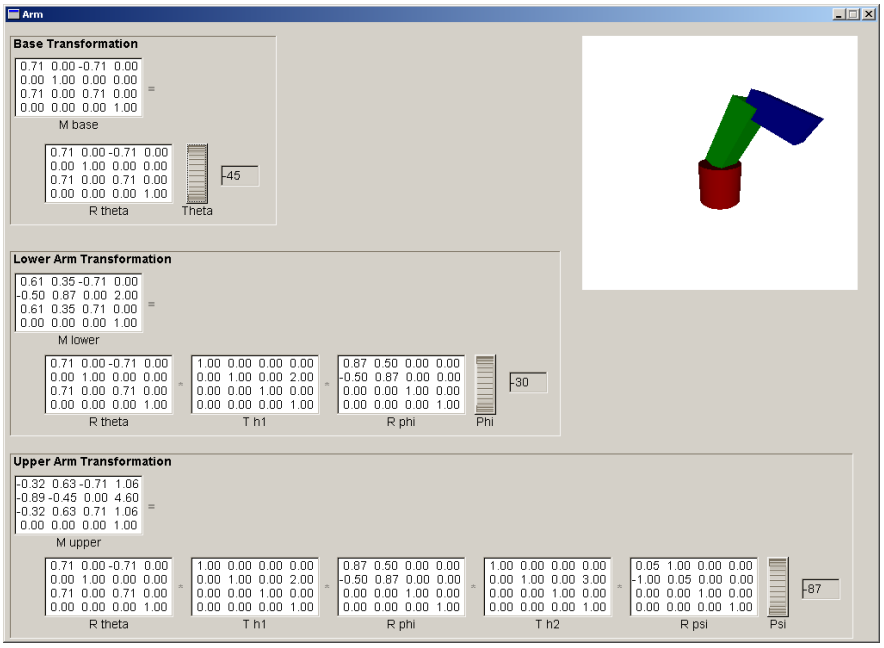
fragmentu lineárně kombinována s barvou mlhy v poměru daném funkcí vzdálenosti od pozorovatele. V aplikaci si uživatel může vybrat ze tří funkcí, lineární, exponenciální a exponenciálně kvadratické, nastavit barvu, začátek a konec mlhy a výsledný efekt pak pozorovat v náhledu. Je zde k nahlédnutí i tvar zvolené funkce a posloupnost příkazů, kterými lze takto definovanou mlhu vytvořit.

Další program s názvem Light & material (Světlo a Materiál) zas demonstruje funkci jednotlivých parametrů tak zvaného Phongovo osvětlovacího modelu v kombinaci s reflektorem. Uživatel může nastavit jednotlivé parametry materiálu a osvětlení a pozorovat výsledný efekt. Další program nazvaný Light Positioning (Umístění světla) pak ukazuje, jakým způsobem se ve starém OpenGL dala nastavit poloha světla. Dnes již však tato ukázka nemá žádnou hodnotu vzhledem k realizaci světel pomocí shaderů.

Poslední program s názvem Texture nás seznamuje s úskalími práce s texturami a texturovacími souřadnicemi v OpenGL.



Obrázek 2.5: Siggraph: V pořadí Fog, Light & Material, Light positioning a Texture



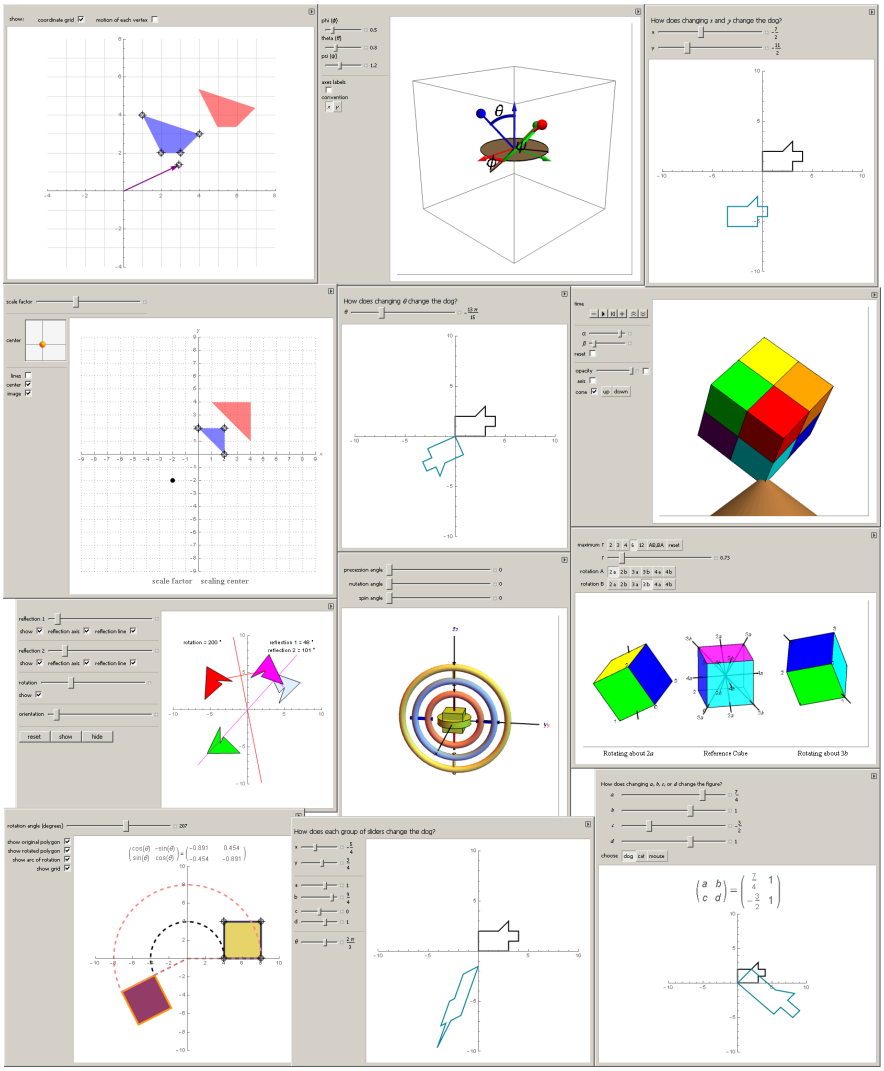
Obrázek 2.6: Arm

Velmi zajímavá je aplikace Arm (obrázek 2.6), kterou lze najít na stránkách University of Washington [CSE03]. Jedná se o jednoduchý program osvětlující hierarchickou posloupnost transformací na modelu, který se skládá ze tří částí. Každá část má pevně definovanou translaci a osu rotace a uživatel může nastavit úhel rotace kolem této osy. Každá část však přebírá transformaci části předchozí a výsledkem je tedy jakési robotické rameno, které lze nastavováním úhlů ovládat. V aplikaci jsou zobrazeny všechny jednotlivé transformační matice a uživatel díky tomu může detailně prostudovat vliv jednotlivých transformací vzhledem k jejich pořadí. Nevýhodou je však nemožnost dosazovat jiné typy transformací a jejich pořadí zaměňovat.

zobrazuje matice	umístění v řetězci	moderní OpenGL	3D
✓			✓

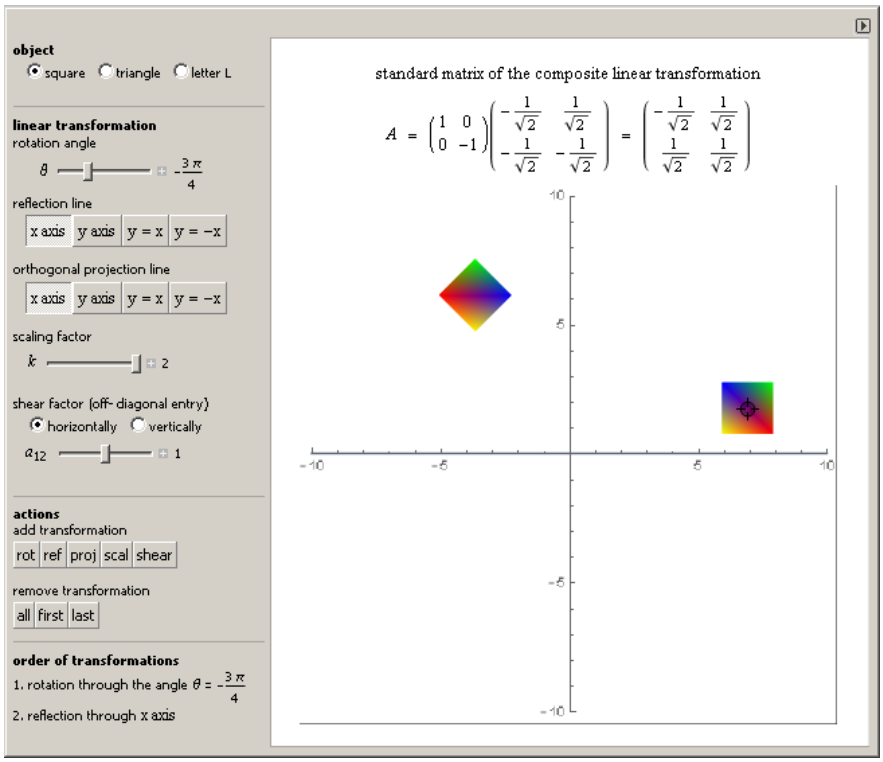
Tabulka 2.5: Arm

Velkým zdrojem výukových aplikací je Wolfram Demonstrations Project [Pro15]. Jedná se o sbírku online aplikací vytvářených komunitou a lze zde nalézt nepřeberné množství interaktivních ukázek ze všech odvětví nejen přírodních věd. Geometrické transformace samozřejmě nejsou výjimkou. Na obrázku 2.7 jsou vidět typické příklady několika těchto aplikací. Vesměs se jedná o demonstraci základních transformací jako je posunutí, rotace, škálování a zkosení. Aplikace nezobrazují transformační matice ani jejich aplikaci a ve většině případů v podstatě nevysvětlují vůbec nic. V případě, že se někdo rozhodne začít studovat geometrické transformace v prostoru pomocí matic, již asi tuší, co se stane, když například posune nějaký objekt o 5 jednotek ve směru osy x nebo ho pootočí o 45°.



Obrázek 2.7: Wolfram: Ukázka aplikací

Světlou výjimkou je však aplikace nazvaná Linear Transformations (Lineární transformace) (obrázek 2.8). Tvůrce programu se nám snaží přiblížit účinky základních transformací jako rotace, škálování a zkosení. Uživatel může nastavit hodnoty zvolených transformací a ty pak přidávat v libovolném pořadí a pozorovat výsledný efekt na testovacím tělese. Jsou zde dokonce k nahlédnutí i matice zadaných transformací včetně výsledku jejich násobení a tedy i výsledné transformace. Nevýhodou však je, že, jak již název napovídá, se jedná pouze o transformace lineární a pouze ve dvou rozměrech. Počet transformací, které lze vložit do výsledného řetězce, je také omezen jen na tři transformace.

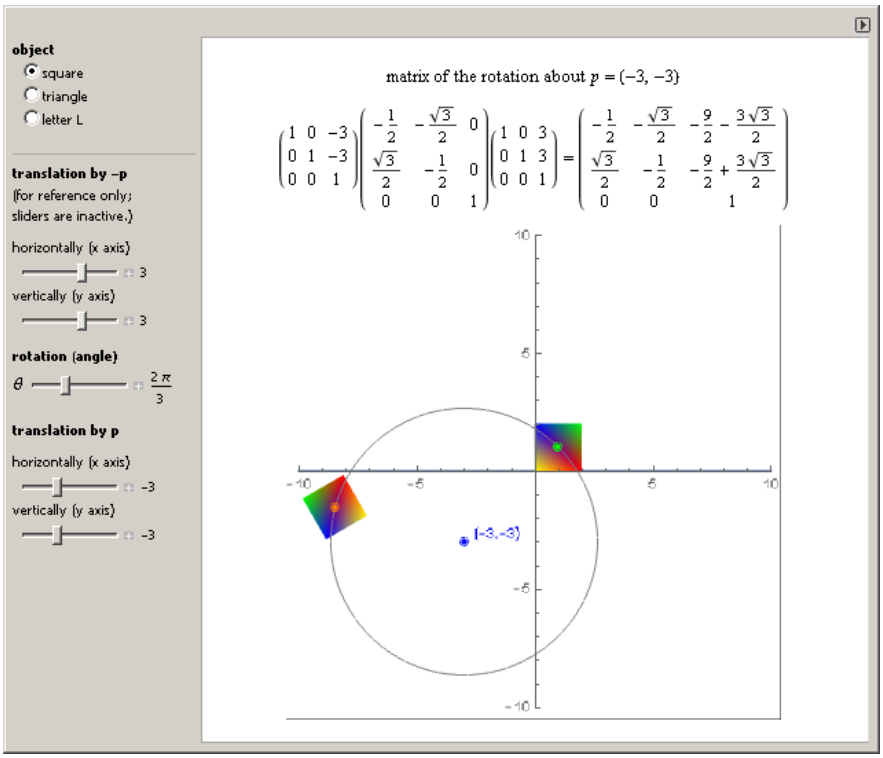


Obrázek 2.8: Wolfram: Linear Transformations

zobrazuje matice	umístění v řetězci	moderní OpenGL	3D
✓			

Tabulka 2.6: Wolfram: Linear Transformations

Za povšimnutí stojí ještě jedna aplikace nazvaná Rotation about a Point in the Plane (Rotace okolo bodu v rovině) (obrázek 2.9). Aplikace popisuje princip rotace tělesa okolo obecného bodu v rovině pomocí afinních transformací. Přesto, že se o tom v samotné aplikaci ani v přiložené dokumentaci nemluví, autor nám přibližuje princip transformací jako přechod mezi různými vztažnými soustavami. Pochopení tohoto principu je však pro osvojení transformací klíčové. Aplikace názorně demonstruje, jak je ukázkové těleso nejprve převedeno do vztažné soustavy bodu, okolo kterého chceme rotovat. Poté je aplikována samotná transformace rotace a nakonec je těleso opět převedeno do vztažné soustavy původní. V aplikaci jsou opět zobrazeny jednotlivé transformační matice, včetně matice výsledné. Je velká škoda, že nám autor aplikace dává k dispozici pouze transformaci rotace, neboť popsaný princip lze aplikovat na libovolnou transformaci. Aplikace popisuje daný problém opět pouze ve dvou rozměrech.



Obrázek 2.9: Wolfram: Rotation about a Point in the Plane

zobrazuje matice	umístění v řetězci	moderní OpenGL	3D
✓			

Tabulka 2.7: Wolfram: Rotation about a Point in the Plane

aplikace	zobrazuje matice	umístění v řetězci	moderní OpenGL	3D
ModelView Matrix	✓			✓
Projection Matrix				✓
Transformation				✓
Projection				✓
Arm	✓			✓
Linear Transf.	✓			
Rot. about a Point	✓			

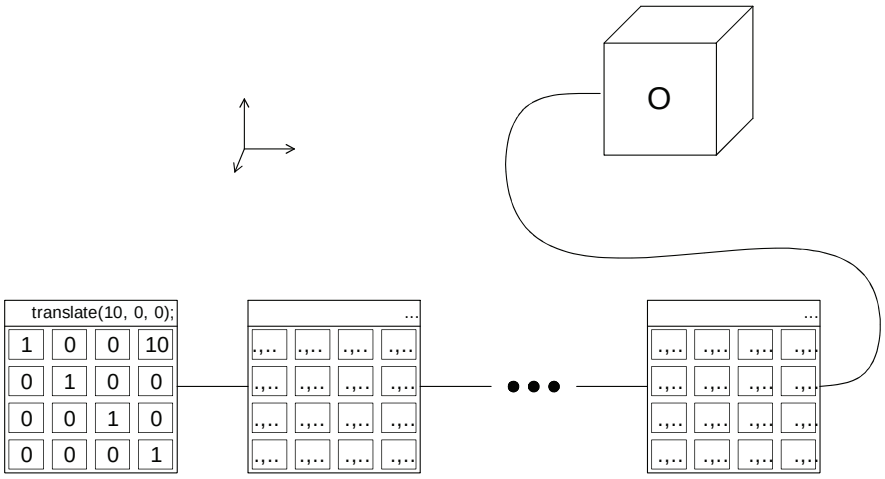
Tabulka 2.8: Aplikace: shrnutí kritérií

Tabulka 2.8 shrnuje výsledky hodnocení všech výše zmíněných aplikací. Ani jedna nesplňuje požadovaná kritéria a to ani zdaleka. Při tvorbě budoucí aplikace však budeme těžit z dobrých nápadů a pozitiv, které tyto aplikace bezesporu mají a též se budeme snažit vyvarovat chyb a nedostatků, které plynou z vysokých nároků a ze specifických požadavků, které na tyto aplikace klademe.

Mezi nejdůležitější vlastnosti vytvářené aplikace musí bezesporu patřit zobrazení všech transformačních matic v pořadí, v jakém jsou aplikovány, jak můžeme vidět v programu Arm 2.6. Musí zde být ovšem možnost tyto matice všemožně přidávat, odebírat a editovat, jako je tomu v programu Linear Transformations (Lineární transformace) (obrázek 2.8). Z aplikací OpenGL ModelViewMatrix (obrázek 2.1) a OpenGL Projection Matrix (obrázek 2.2) od Songho Ho Ahn [Ahn13] a Siggraph tutoriálů od Nate Robinse [SIG01] lze využít hned několik dobrých principů. Za prvé možnost pozorovat scénu ve dvou náhledech. V jednom z pohledu kamery, tak jak bude vypadat výsledná scéna ve skutečné aplikaci. A ve druhém nezávislém pohledu, jehož kameru však bude možno ovládat. To uživateli poskytne dobrý přehled o scéně a transformacích, které v této scéně studuje. Při zkoumání projekčních transformací je také velmi užitečné zobrazení aktuálního pohledového jehlanu. Také je velmi užitečné zobrazení konkrétních funkcí včetně parametrů a dosazených hodnot, které jsou při vytváření transformací použity.

3. Návrh řešení

V této kapitole se budeme věnovat konceptu výsledné aplikace. Základním elementem bude transformační matice 4x4. Uživatel bude mít možnost tyto matice vhodným způsobem vytvářet, editovat a bude mít přístup k hodnotám jejich jednotlivých prvků. Vytvořenou transformaci pak půjde aplikovat na nějaký geometrický objekt a pozorovat její účinek. Cílem je pozorovat chování transformací v případě, že je jich aplikováno několik za sebou.



Obrázek 3.1: Koncept aplikace

Na obrázku 3.1 je znázorněno jednoduché schéma toho, co výsledný program umožňuje, tj. vložit několik transformačních matic, určit jejich pořadí a výsledek aplikovat na nějaký geometrický objekt. Matematicky lze situaci na obrázku popsat následovně.

$$A_1 A_2 \dots A_3 O$$

Kde A_i jsou libovolné transformační matice. Transformační matice A_1 je v tomto konkrétním případě posunutí ve směru osy x o deset jednotek. Matice sloupcových vektorů O o velikosti 4xN představuje geometrický objekt, který je definován N-ticí vrcholů v homogenních souřadnicích.

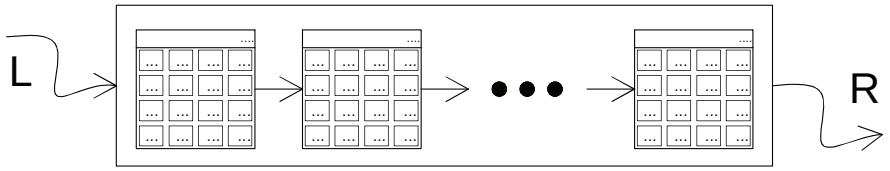
3.1 Transformační Matice

Vzhledem k tomu, že se tento projekt zaměřuje na práci v OpenGL, budeme k tvorbě transformací používat knihovnu GLM. Uživatel bude mít možnost vytvářet všechny možné transformace, které nám tato knihovna poskytuje, a zároveň pozorovat i funkce a příkazy, které jsou přitom používány. Například: Vybereme si z nabídky transformací transformaci posunutí. V dialogovém okně bude znázorněna konkrétní funkce, tedy *glm::translate*(x, y, z). Po vyplnění parametrů dostaneme k dispozici danou matici s jednotlivými hodnotami a v jejím záhlaví bude daná funkce vypsána (jak je vidět na první matici na obrázku 3.1). K takto vytvořené transformaci bude možné připojit geometrický objekt a uživatel bude

moci pozorovat, že se transformovaný objekt nachází na daných souřadnicích. Dále bude zachována i možnost editovat hodnoty dané matice, aby uživatel mohl v reálném čase pozorovat, jaký vliv jednotlivé parametry na výslednou transformaci objektu mají. Editovatelné hodnoty budou dány druhem samotné transformace. V případě translační matice se například budou dát editovat pouze první tři prvky ze čtvrtého sloupce. Uživatel si tak lehce osvojí význam jednotlivých hodnot v transformační matici. Nebude však chybět ani možnost vložit transformační matici zcela obecnou, u které půjdou editovat všechny hodnoty. Je totiž vhodné poskytnout uživateli možnost si transformaci vytvořit zcela sám.

3.2 Řetězec transformací

K pochopení řetězce transformací nestačí pozorovat účinek pouze jedné transformace. Jednotlivé transformace se skládají násobením jejich matic. Násobení matic však není komutativní. Pravděpodobně každý, kdo s danou problematikou začíná, si tedy ihned uvědomí, že na pořadí transformací bude záležet. Typickým příkladem je posun a rotace. Aplikujeme-li nejprve posun a pak rotaci, stane se něco úplně jiného, uděláme-li to naopak. Celá situace se s přidáváním dalších transformací stává více a více komplikovanou a neintuitivní.



Obrázek 3.2: Komponenta řetězce transformací *sequence*

Budeme tedy potřebovat umožnit jednotlivé transformační matice skládat za sebe a editovat jejich pořadí. Kromě toho bude vhodné mít takovýchto řetězců k dispozici více. Tak bychom mohli například vytvořit jeden a umístit do něj translaci a poté rotaci a pak vytvořit druhý a umístit do něj stejné transformace, ale v opačném pořadí. Když k oběma připojíme vlastní objekt ve scéně, můžeme hned velmi názorně pozorovat, jak se jednotlivé transformace liší.

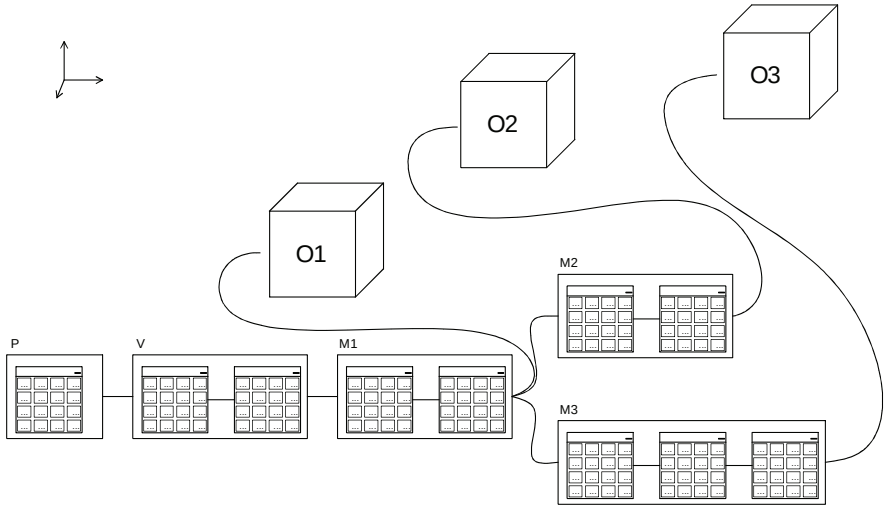
V jiném případě bychom chtěli pozorovat účinek transformací v souřadném systému jiného objektu ve scéně. Typickým příkladem může být robotická ruka, jejíž možnou realizaci můžeme důkladněji prozkoumat v programu Arm (obrázek 2.6). V ukázkovém programu můžeme pozorovat pouze rotace a posuny. Obecně však mohou být jednotlivé transformace komponent v hierarchii zcela libovolné. Tomuto příkladu odpovídá situace, kdy umístíme dva nebo více řetězců transformací do série nebo zcela obecně do stromu.

Na obrázku 3.2 vidíme schéma komponenty řetězce transformací. Sám o sobě bude ukládat posloupnost transformačních matic. Tyto matice bude možno libovolně přidávat, odebírat a prohazovat jejich pořadí. Samotný řetězec reprezentuje transformaci danou součinem jednotlivých matic, které obsahuje. Bude-li řetězec obsahovat transformační matice $T_1, T_2, \ldots, T_n$, pak výsledná transformace, kterou tento řetězec reprezentuje, bude rovna součinu jednotlivých matic $T_1 T_2 \ldots T_n$. Komponenta je také navržena tak, aby se jednotlivé řetězce daly kombinovat s

dalšími. Zprava i zleva lze připojit řetězce jiné. Máme-li řetězec, který reprezentuje transformaci S, zleva k němu připojíme řetězec S_1 a zprava řetězec S_2, celý systém pak reprezentuje transformaci S_1SS_2. Tato komponenta tedy umožňuje rozdělit jednotlivé matice transformací do jakýchsi logických skupin. To má za následek jednodušší a přehlednější práci s nimi, jak uvidíme při vytváření scén v následující kapitole.

3.3 Scéna

Nejjednodušší možná scéna v takto navrženém systému by vypadala asi takto. Obsahovala by jediný řetězec transformací, ve kterém by na prvním místě byla nějaká matice projekce P následovaná maticí pohledu V a libovolnou transformací M. Na výstup tohoto řetězce bychom pak připojili nějaké těleso. Systém nám ovšem dovoluje vytvářet i mnohem komplexnější scény. Příklad takové scény je znázorněn na obrázku 3.3.



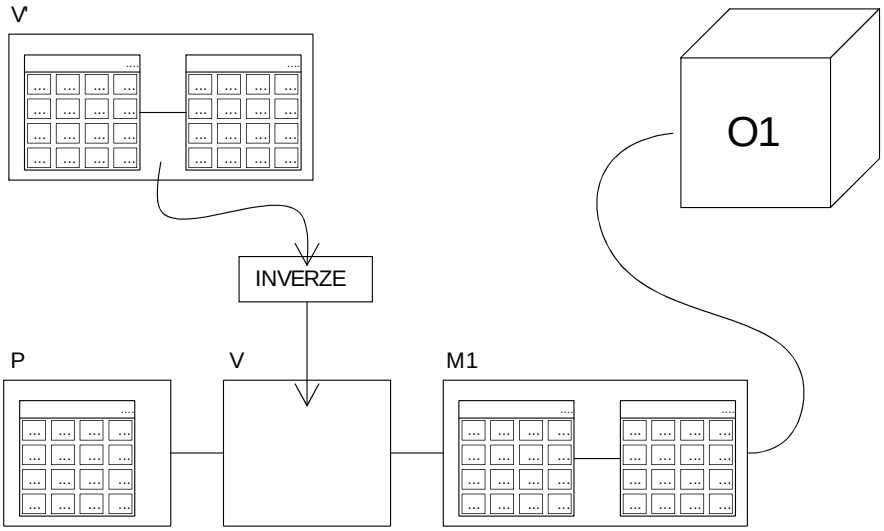
Obrázek 3.3: Scéna se třemi objekty v hierarchii.

Tato scéna je velmi podobná příkladu v programu Arm 2.6. Objekty O_2 a O_3 jsou závislé na transformaci objektu O_1, což půjde velmi názorně pozorovat, začneme-li transformace objektu O_1 měnit. Příklad je doplněn i o nezbytné transformace kamery. Uživatel tak může jasně prostudovat jejich funkci a umístění v celém systému transformací a při editaci jejich hodnot se dostaví požadovaný efekt změny pohledu na studovanou scénu. Řetězce projekční a pohledové transformace (na obrázku 3.3 označen jako P a V) se však zatím svojí funkcionalitou nijak neodlišují.

3.4 Vztažné soustavy

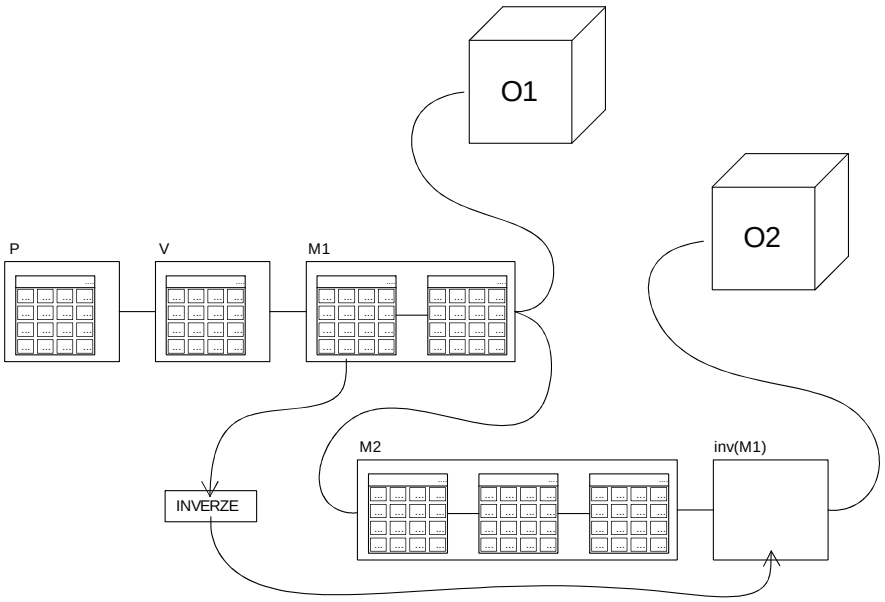
Ačkoliv současný systém dovoluje vytvářet velmi složité struktury transformací, stále nám neumožňuje prostudovat jejich samotnou podstatu. Uživatel musí pochopit význam transformace jako přepočet souřadnic mezi různými vztažnými soustavami. Modelová transformace například převádí souřadnice ze souřadné soustavy modelu do světové soustavy souřadnic. Operaci lze zapsat jako

$x_w = Mx_m$, kde x jsou souřadnice bodu v příslušně soustavě. Samozřejmě bude existovat i transformace z globální soustavy souřadnic do soustavy modelové, tedy $x_m = Wx_w$. Jednoduchou úpravou zjistíme, že matice $W = M^{-1}$, takže $x_m = W^{-1}x_w$. Příklad nabízí hned základní posloupnost projekční, pohledové a modelové transformace PVM. Jsou-li souřadnice v modelové soustavě převedeny do soustavy globální, hned poté jsou zas převedeny do soustavy kamery, tedy $x_v = V'^{-1}Mx_m$, kde transformace V' je inverzí matice pohledové transformace. Je však pouze na uživateli, jestli chce do důsledku chápat samotný matematický princip těchto operací a v případě, že ano, zcela určitě se nevyhne kvalitní literatuře, která se danou problematikou zabývá. Typickým příkladem je rotace kolem obecného bodu, kterou se zabývá program Rotace okolo bodu (obrázek 2.9). Bod na souřadnicích x_w převedeme do souřadné soustavy, ve které chceme transformovat, v tomto případě rotovat, například do soustavy A. Tedy $x_a = A^{-1}x_w$. Poté provedeme samotnou rotaci $x'_a = RA^{-1}x_g$ a na závěr zase souřadnice převedeme do původní soustavy $x'_g = ARA^{-1}x_g$. Místo rotace R můžeme samozřejmě provést i jiné transformace. Stejně tak můžeme zvolit i jiné soustavy souřadnic. Aby se však ve vytvářeném programu dalo těchto transformací docílit, bude nutné přidat operátor inverze.



Obrázek 3.4: Inverze pohledové transformace

Obrázek 3.4 ukazuje realizaci dvou výše zmíněných transformací. Nejprve je na místo pohledové matice kamery dosazena inverze její skutečné transformace. V mnoha praktických aplikacích je vhodné pracovat s kamerou stejně jako s jakýmkoliv jiným objektem ve scéně a ve snaze umístit kameru na požadovanou pozici se nejeden uživatel může divit, proč je jeho kamera někde jinde a ještě k tomu je namířena na opačnou stranu. K pochopení, proč je třeba použít inverzi, je samozřejmě nutné chápat význam jednotlivých souřadných soustav a vztahů mezi nimi.

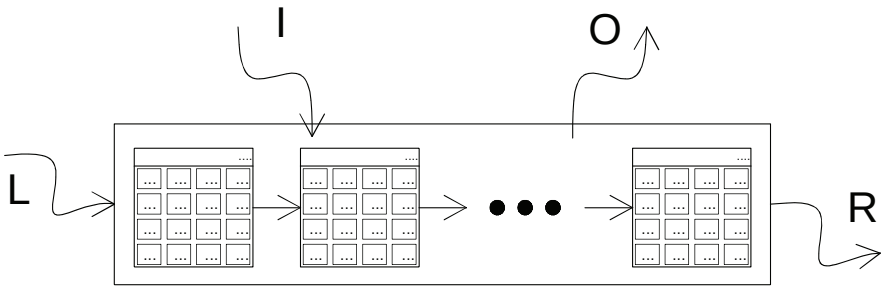


Obrázek 3.5: Příklad použití inverzní transformace ve scéně.

Na obrázku 3.5 vidíme druhý příklad, kdy je objekt O_2 transformován v soustavě objektu O_1 transformací M_2.

Aby tyto operace byly možné, musíme upravit komponentu řetězce transformací (obrázek 3.6). Nyní je řetězec doplněn o výstup O a vstup I. Výstup O je roven součinu obsažených matic. Do tohoto výstupu se však nijak nezapočítávají transformace, které jsou k danému řetězci připojené zprava nebo zleva. Vstup I pak nahrazuje transformace, které by mohly v řetězci být.

Na obrázcích platí konvence, že vstupy I a výstupy O jsou připojeny vždy zespodu nebo shora. Operace násobení jsou zas reprezentovány spoji zleva a zprava. Objekt O_2 je tedy transformován maticí $PV^{-1}M_1M_2M_1^{-1}$. S takto vymyšleným systémem je již možné provádět mnoho operací, které jsou nezbytné pro libovolnou grafickou aplikaci.



Obrázek 3.6: Řetězec transformací doplněný o vstup I a výstup O.

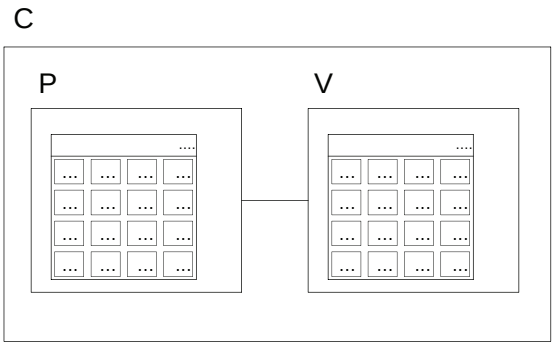
3.5 Geometrické objekty

Jak již bylo zmíněno dříve, aby uživatel mohl pozorovat efekt vytvářených transformací, musí mít možnost je aplikovat na nějaké objekty umístěné ve scéně. Ze začátku bude jistě vhodné zvolit nějaké primitivum jako je například jednotková krychle nebo i jiné ze základních geometrických těles. Pro lepší přehled o transformaci budou k dispozici i tělesa, na kterých se dá jednoznačně určit jejich orientace. Nesmí chybět ani jednoduchý model bází ortonormálního souřadného systému. Bázové vektory však půjde zobrazit pro všechny transformace. Užitečné bude přidat i možnost importovat objekty v některém z běžných formátů jako například Wavefront *.obj. a tím rozšířit možnosti vytváření vlastního obsahu.

3.6 Kamera

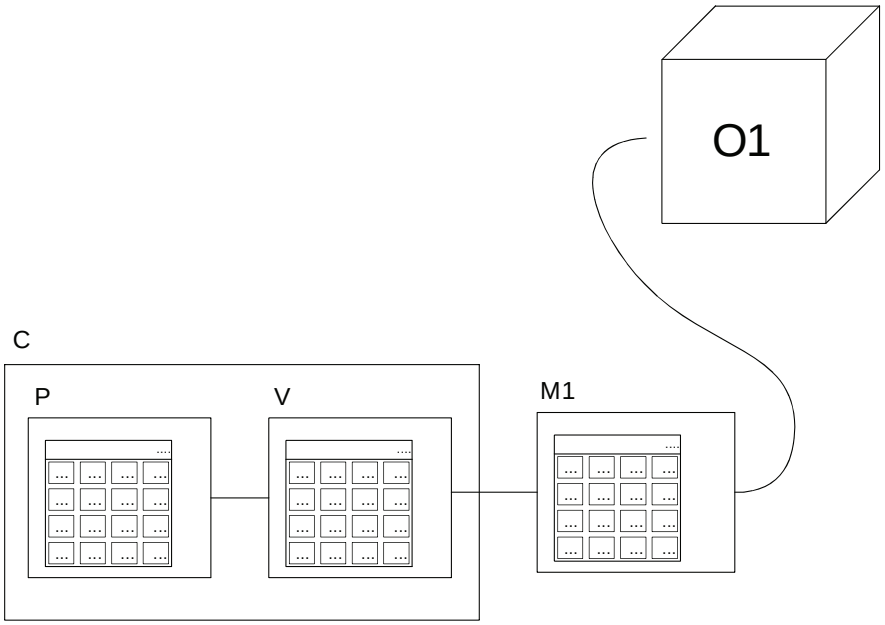
Nejjednodušší možná scéna, ve které můžeme začít pracovat, musí obsahovat projekční a pohledovou transformaci. Uživatel poté začne přidávat objekty a transformovat je. Aby si skutečně mohl prohlédnout efekt jednotlivých transformací, musí mít možnost si scénu prohlížet z různých úhlů a kamerou pohybovat. Toho lze samozřejmě dosáhnout editací jednotlivých parametrů projekční a pohledové transformace.

Takto zvolený způsob ovládání však není příliš praktický. Také je možné, že uživatel bude chtít nejprve prozkoumat základní transformace objektů jako je posun či rotace a nebude mít chuť ani potřebné znalosti k tomu, aby pohyboval kamerou takovýmto způsobem. Proto bude program obsahovat nezávislý náhled na scénu, ve kterém bude možnost kameru ovládat některou z konvenčních interaktivních metod jako například pomocí myši.



Obrázek 3.7: Transformace kamery

Na druhou stranu nastane i situace, kdy se uživatel zaměří právě na transformace spojené s kamerou. Bude chtít například vidět, jak se chovají různé projekční matice. Proto bude mít možnost přidávat kamery vlastní. K tomu bude sloužit nová komponenta (obrázek 3.7), která bude obsahovat projekční a pohledovou transformaci. Konkrétně se skládá ze dvou komponent transformačních řetězců (obrázek 3.6). První je pro projekční transformaci a druhý pro transformaci pohledovou.



Obrázek 3.8: Počáteční scéna

Na obrázku 3.8 vidíme použití nové komponenty kamery v jednoduché scéně, ve které je nyní dobře vidět posloupnost projekční, pohledové a modelové transformace. Uživatel může měnit jak modelovou transformaci geometrického objektu O_1, tak i projekční a pohledovou transformaci kamery C. Scénu bude možné pozorovat jak ze samotné kamery C, tak i z v nezávislém náhledu, ve kterém bude samotná kamera C vidět. Kamera bude v náhledu vykreslena jednoduchým geometrickým objektem, a to včetně jejího pohledového jehlanu. Při editaci příslušné projekční matice bude možné změnu ihned pozorovat na tvaru příslušného pohledového jehlanu.

3.7 Systém výuky

Zbývá zodpovědět otázku, jakým způsobem se uživatel pomocí navrženého programu naučí používat transformace v počítačové grafice. Bez ohledu nato o jakou probíranou látku se jedná, každému vyhovuje jiný proces učení. Někomu například stačí nastudovat dostupné materiály a tím skončit s předpokladem, že již by měl vědět vše potřebné. Jiný si potřebuje dané téma ještě prakticky procvičit, například výpočtem souvisejících příkladů a úloh. Někdo zas začne vypracováním cvičného testu, na kterém zjistí své znalostní nedostatky, které případně doplní.

Vytvářený program by měl především umožnit studovat transformace z praktického hlediska. K tomu budou sloužit tři hlavní přístupy. Prvním je zcela individuální přístup uživatele, při kterém vytváří vlastní scény zaměřené na problematiku, která ho zajímá. Za druhé bude program obsahovat množství výukových scén, navržených pro studium konkrétních problémů, a za třetí bude program obsahovat sadu testovacích úloh, na kterých bude moci uživatel vyzkoušet své znalosti. Každý by tak měl mít možnost zvolit cestu, která mu vyhovuje.

Nyní se blíže podíváme na zmíněné metody výuky. První je individuální přístup uživatele. Vzhledem k velkému množství možností, které systém nabízí , je

možné většinu problémů prostudovat po svém. Při pohledu na jednotlivé obrázky scén je ale jasné, že daní za obecnost bude vyšší složitost ovládání samotného programu. Množství komponent a jejich vstupů a výstupů činí program na první pohled nepřehledným. Aby se uživatel byl schopen něčemu naučit, nemůže program začínat prázdný bez jediné matice a objektu ve scéně. Proto po spuštění programu již scéna bude obsahovat projekční a pohledovou transformaci a objekt transformovaný nějakou základní transformací jako například posunem, jak je vidět na obrázku 3.8. Uživatel si bude moci pohybem kamery prohlédnout jednotkovou krychli a úpravou či přidáváním transformací a dalších objektů začít experimentovat. Snadno tak prostuduje principy základních transformací a jejich funkci a umístění v zobrazovacím řetězci. Prakticky bude moci vyzkoušet jejich chování při aplikaci v různých pořadích. V jiném případě si může rychle vyzkoušet nebo ověřit funkci nějaké transformace, kterou zrovna řeší ve své domácí úloze nebo projektu. Myslím, že už jen touto zcela individuální cestou se, s přihlédnutím k daným možnostem, může uživatel o transformacích mnohému naučit a já osobně považuji tento přístup za hlavní výukový princip vytvářené aplikace.

Program by však měl jít dál a na výuce se podílet i aktivně. Jak bylo řečeno, program bude dále obsahovat sadu předdefinovaných scén, které budou sloužit ke stejnému účelu jako výuková dema popsaná v kapitole Aplikace 2.3. Jednotlivé výukové scény se budou věnovat vybraným tématům. Uživatel tak dostane například za úkol prozkoumat princip projekční matice jako je tomu v aplikaci OpenGL Projection Matrix (obrázek 2.2) nebo nastudovat hierarchii transformací obdobně jako v aplikaci Arm 2.6. Možnosti, které navržený systém poskytuje, umožňují vytvářet scény, které zastanou funkci všech výukových programů, které v kapitole o aplikacích popisuji, a i mnohé další.

Poslední metodou výuky jsou testovací úlohy. Opět se jedná o předdefinovanou scénu, ve které bude muset uživatel splnit zadané požadavky. Uživatel například dostane za úkol postavit celou scénu od začátku s předem definovanými parametry. Například vytvořit kameru na dané pozici s perspektivní projekcí zadanou zorným úhlem a vzdálenostmi ořezávacích rovin a dále do této scény umístit těleso na danou pozici. V jiné úloze by měl za úkol otočit daným tělesem kolem daného bodu a vektoru. V další by dostal za úkol přemístit objekt ve scéně do dané koncové pozice a orientace za použití dostupných transformací. Jednotlivé úlohy bude také možné sdružit do skupiny a vytvořit klasický test.

V případě výukových scén i testovacích úloh se jedná o vytvořenou scénu s doprovodným textem popisujícím smysl a cíle dané úlohy. Jedná se tak o vytvářený obsah programu. V této práci si však nekladu za cíl vytvořit rozsáhlý systém výukových scén a testovacích úloh. Cílem práce je poskytnout nástroj, který tuto tvorbu umožní.

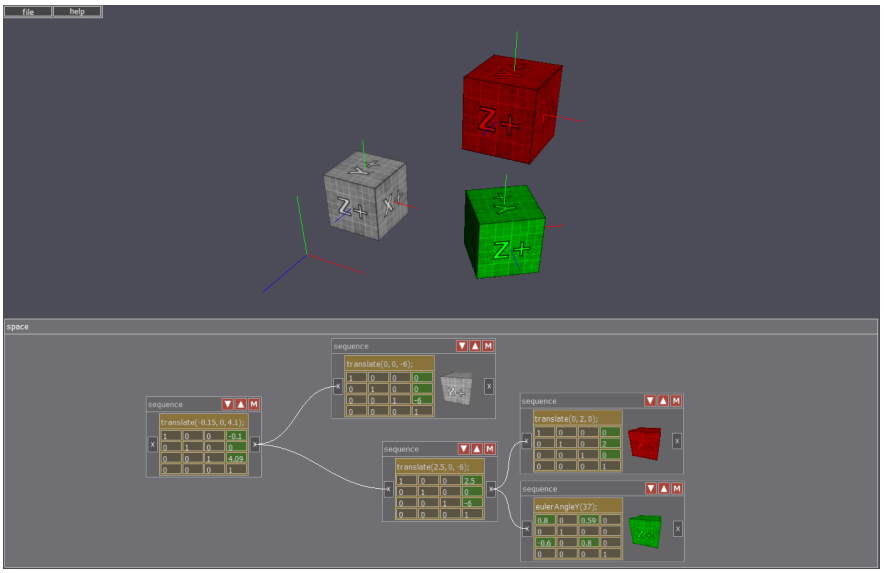
V rámci projektu bude vytvořen především obsah demonstrující možnosti aplikace a pokrývající důležitá témata transformací v počítačové grafice. Vzhledem k rozsahu této problematiky a skutečnosti, že geometrické transformace nacházejí své využití i v mnoha jiných oborech, by ovšem byla škoda neposkytnout možnost vytvářet obsah vlastní. Pro aplikaci tak nalezne využití nejen student, ale i pedagog. Pomocí programu bude možné předvést teoretický výklad na názorné ukázce nebo vytvářet domácí a testové úlohy, ať již je studenti dostanou vypracovat přímo prostřednictvím aplikace, či aplikace poslouží jako nástroj k tvorbě jejich zadání.

4. Implementace

Program je implementován v jazyce C++ v grafickém rozhraní OpenGL. K přístupu k potřebným knihovnám je použita knihovna PGRFrameWork [Fra13]. Jedná se o sadu základních knihoven a funkcí pro práci s OpenGL vytvořenou pro účely předmětu Programování Grafiky (PGR). Z poskytnutých knihoven je v projektu využívána knihovna FreeGLUT, alternativní verze knihovny GLUT (OpenGL Utility Toolkit) poskytující základní funkcionalitu pro práci s OpenGL. Knihovna GLM (OpenGL Mathematics) je používána pro práci s maticemi, vektory, kvaterniony a další matematické operace. Knihovna Assimp (Open Asset Import Library) slouží pro načítání polygonálních modelů ve formátu Wavefront obj. A nakonec knihovna DevIL (Developer's Image Library) slouží k načítání obrázků, které jsou dále používány k texturování objektů.

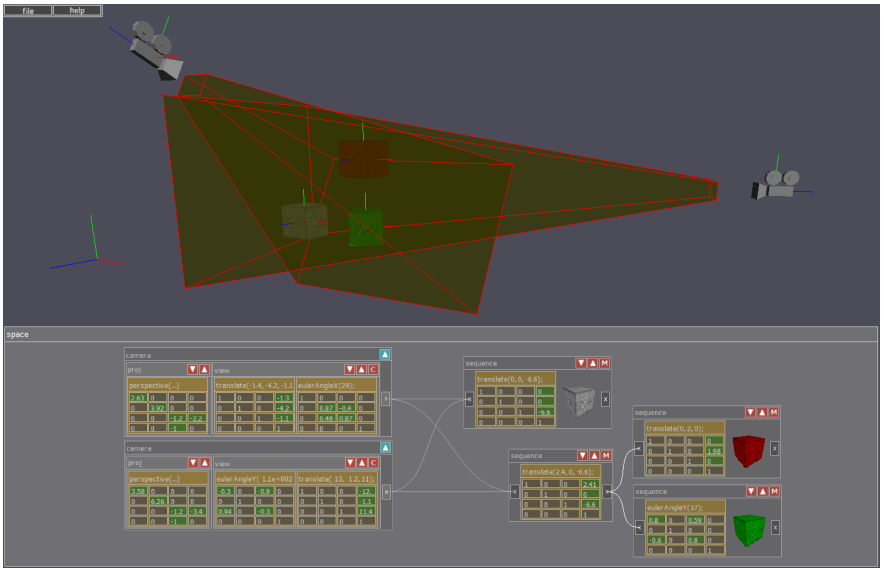
4.1 Koncept programu

Prostředí programu je rozdělené do dvou základních prvků. Prvním prvkem je komponenta *space* (obrázek 4.1 dole) sloužící jako pracovní plocha, na které probíhá proces tvorby samotného obsahu scény. Druhým je globální náhled na scénu z nezávislé kamery (obrázek 4.1 dole), kterou lze ovládat myší. Uživatel tak má dobrý přehled o vytvářených objektech a jejich transformacích.



Obrázek 4.1: Ukázková scéna se třemi objekty strukturované do grafu scény.

Ukázka programu je vidět na obrázku 4.1. Scéna obsahuje tři objekty, jejichž transformace a struktura uspořádání je definována na ploše *space*. Koncept programu je založen na komponentách, které lze specifickým způsobem propojovat a definovat tak různé struktury. Tyto komponenty budeme dále v textu nazývat krabičky. Scéna na obrázku 4.1 ukazuje příklad vícenásobného použití základní krabičky *sequence* (kapitola 4.2.1). Jejich pomocí lze vkládat do scény geometrické objekty, definovat jejich složené modelové transformace a lze je propojovat do stromu grafu scény.



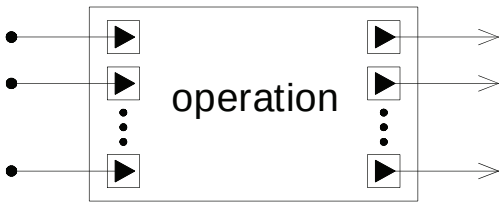
Obrázek 4.2: Ukázka programu. Scéna doplněná o dvě kamery.

Příklad na obrázku 4.2 doplňuje scénu na obrázku 4.1 o dvě kamery. Jejich projekční a pohledové transformace lze editovat příslušnými krabičkami *camera*, které se automaticky připojují ke všem kořenům stromu grafu scény, aby bylo možné pozorovat celou posloupnost projekční, pohledové a modelové transformace. Jednoduchá reprezentace kamer je vidět také v globálním náhledu včetně ořezávacích rovin jejich projekční transformace.

Krabičky *sequence* i *camera* jsou založeny třídě *operator*, jejíž funkcionalita tvoří základ implementace programu. Její pomocí lze v rozhraní vytvářet i matematické operace a detailní popis třídy je obsažen v následující kapitole.

4.2 Operátory

Základním stavebním prvkem v komponentě *space* jsou již zmíněné krabičky, které lze různým způsobem propojovat a které plní rozličné účely. Popsali jsme si krabičku *sequence* transformací, která spravuje posloupnost matic a obsahuje různé vstupy a výstupy. Krabičku kamery, která sjednocuje dvě krabičky *sequence* transformací. Jednu pro projekční a druhou pro pohledovou transformaci. Krabičku inverze, která vytváří inverzní matici z matice, kterou dostává na vstupu. Program dále obsahuje množství dalších podobných krabiček pracujících nejen s maticemi, ale i s vektory, kvaterniony a skaláry, a dokonce i se zcela obecnými informacemi, jako například pouhé předávání ukazatelů. Všechny krabičky jsou v programu realizovány třídou nazvanou Operátor.



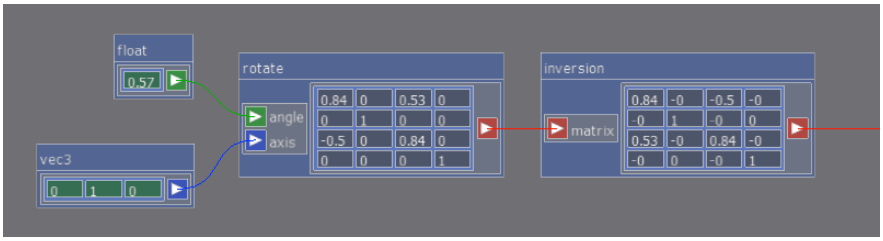
Obrázek 4.3: Schéma obecného operátoru.

	barva	R,G,B
pulse	růžová	1.0, 0.0, 1.0
float	zelená	0.0, 0.7, 0.0
vec3	modrá	0.0, 0.2, 1.0
vec4	hnědá	0.4, 0.3, 0.0
quat	oranžová	1.0, 0.7, 0.0
matrix	červená	1.0, 0.1, 0.0
mulMatrix	bílá	1.0, 1.0, 1.0
screen	tyrkysová	0.0, 1.0, 1.0

Tabulka 4.1: Typy propojení a jejich barva.

Na obrázku 4.3 je vidět schéma obecného operátoru. Operátor může obsahovat libovolné množství vstupů a libovolné množství výstupů (počet je samozřejmě omezen možnostmi zobrazení uživatelského rozhraní), které mohou být nezávisle na sobě jedním z typů obsažených v tabulce 4.1.

Tyto typy slouží k identifikaci jednotlivých vstupů a výstupů tak, aby uživatel mohl propojovat pouze vstupy a výstupy stejného typu. Pro větší přehlednost jsou včetně spojů obarveny různými barvami (tabulka 4.1). Operátor dále obsahuje funkci, která na základě vstupních hodnot přiřadí hodnoty výstupům a proto musí být jednotlivé typy jasně definovány. Typ *pulse* je zcela abstraktní. Slouží pouze k předání informace jinému operátoru, že má vykonat nějakou činnost. Typy *float*, *vector*, *quat* a *matrix* reprezentují příslušné matematické proměnné. Typ *mulMatrix* slouží pro propojování sekvencí transformací v grafu scény a reprezentuje násobení matic. Nakonec typ *screen* předává informace o projekci a pohledové transformaci kamery.



Obrázek 4.4: Ukázka operátorů *float*, *vector*, *rotate* a *inverse*.

Příklady konkrétních operátorů v programu jsou vidět na obrázku 4.4. Operátor *rotate* převádí výstupy operátorů float a vector na rotační matici, kterou dále předává operátor *inversion*. Je vidět, že operátor dokonce nemusí obsahovat žádné vstupy, jako je tomu u operátorů float nebo vector z obrázku 4.4 a slouží tak pouze jako generátor hodnoty. V opačném případě nemusí mít žádné výstupy a bude se jednat o monitor příslušných hodnot na vstupech.

Ke správné činnosti propojených operátorů je nutné vyřešit dvě úlohy. První je předávání informací. Je nutné, aby ve chvíli, kdy jeden operátor změní své hodnoty na výstupech, všechny operátory, které jsou k němu připojeny svými vstupy také aktualizovaly své hodnoty na výstupech a tuto informaci opět předaly dál. Druhou úlohou je kontrola platného propojení operátorů. Prvním stupněm kontroly jsou výše zmíněné typy. Operátory však nesmí být zapojeny ani tako-

vým způsobem, aby výstup nějakého operátoru vedl libovolnou cestou zpět na jeho vstup. Takové zapojení by postrádalo logický význam a navíc by aktualizace hodnot operátorů vedla na nekonečný cyklus. Budeme-li na operátory a jejich propojení nahlížet jako na graf, ve kterém operátory představují vrcholy a propojení orientované hrany ve směru od výstupů ke vstupům, pak přípustné propojení bude vždy vytvářet orientovaný acyklický graf. Obě úlohy můžeme vyřešit vhodným grafovým algoritmem.

```
class Operator {

        Input  * inputs;
        Output * outputs;

        set<Operator*> operators;

        virtual void updateValues() = 0;

        void receive() {

                updateValues();
                spread();
        }

        void spread() {

                for (Operator * op : operators) {
                        op->receive();
                }
        }
}
```

Třída 1: Třída Operator.

Třída 1 ukazuje základní proměnné a metody. Základem třídy Operátor jsou pole vstupů a výstupů *inputs* a *outputs*. Třídy *Input* a *Output* zajišťují logiku propojení a obsahují ukazatele na příslušné vstupy a výstupy, ke kterým jsou aktuálně připojeny, a ukazatel na operátor, který je vlastní. Tímto způsobem je možné traverzovat příslušnou grafovou strukturu operátorů. Přesto však si každý operátor udržuje i množinu ukazatelů bez opakování (set) na všechny operátory, které jsou připojeny k jeho výstupům. Je totiž možné, že nastane případ, kdy více než jeden výstup operátoru bude připojen k operátoru jinému. V příslušných grafech tak mohou existovat vícenásobné hrany, které by zvyšovaly časovou složitost procházecích algoritmů. Jak pole vstupů a výstupů, tak i set operátorů, jsou aktualizovány pouze při změně propojení, takže na časovou složitost nemají negativní vliv.

Pro aktualizace hodnot mezi operátory se používá trojice funkcí *updateValues()*, *receive()* a *spread()*. Funkce *updateValues()* je virtuální funkce, která je definována speciálně pro každý konkrétní operátor, ve které příslušný operátor přepočítá svoje hodnoty podle aktuálních vstupů. Funkce *receive()* tuto funkci zavolá a následně pomocí funkce *spread()* rozšíří informaci o změně svých vstupů připojeným operátorům (v množině *operators*), které opět aktualizují své funkce a posílají signál dál. Tímto způsobem se aktualizace mezi uzly šíří rekurzivně do hloubky a celý proces má lineární časovou složitost.

Grafový algoritmus Algoritmus 1 *isPlugCorrect()* kontroluje platnost grafu z hlediska cyklů a je spuštěn vždy při pokusu o změnu propojení. Parametry *input* a *output* reprezentují příslušný vstup a výstup, který se uživatel snaží propojit

```
bool isPlugCorrect(Input * input, Output * output) {

        if (input->getType() != output->getType()) return false;

        Operator * toFind = input->getOperator();

        Stack<Operator*> stack;
        stack.push_back(output->getOperator());

        while (!stack.isEmpty()) {

                Operator * act = stack.pop();

                if (act == toFind) return false;

                for (Input * inp : act->inputs) {
                        if (inp->isPluged()) {
                                stack.push_back(inp->getOutput()->getOperator());
                        }
                }
        }

        return true;
}
```

Algoritmus 1: Detektor cyklů v grafu operátorů.

a v případě negativní vrácené hodnoty toto propojení nenastane. Algoritmus hledá, zda nějaký vstup operátoru vstupu *output* není propojen nějakou cestou k některému z výstupů operátoru vstupu *input*. V takovém případě by hrana grafu vytvořila cyklus. Jedná se o prohledávaní grafu do šířky pomocí zásobníku a časová složitost je opět lineární.

V programu je realizováno 61 operátorů, které se dají rozdělit do následujících skupin:

- Transformace:
  Skupina operátorů zprostředkovávající dostupné transformační funkce knihovny GLM. Operátory mají stejné vstupy jako odpovídající funkce GLM a danou funkci používají přímo ve své funkci *updateValues()*. Dostupné jsou funkce translace, rotace kolem hlavních souřadnicových os, rotace pomocí osy a úhlu, škálování, ortografická projekce, perspektivní projekce, projekce definovaná funkcí *frustum* a pohledová transformace *lookAt*.

- Generátory:
  Operátory, kterými lze vytvořit vstupní hodnoty matematických typů *float*, *vec3*, *vec4*, *quat*, *matrix*.

- Maticové operace:
  Základní matematické operace nad maticemi zahrnující inverzi, transpozici, determinant, násobení matic, násobení matice vektorem a vektoru maticí atd.

- Vektorové operace:
  Základní matematické operace nad vektory zahrnující vektorový a skalární součin, sčítání a odečítání, násobení skalárem, norma a normalizace atd.

- Operace s kvaterniony:
  Násobení a normalizace.

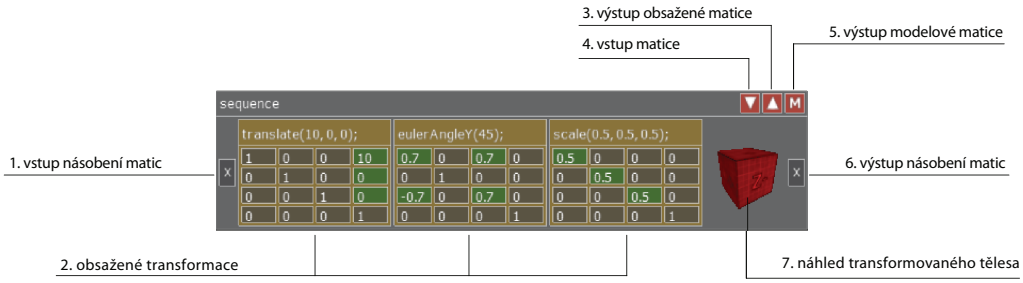
- Operace se skalárními hodnotami:
  Všechny běžné matematické operace s reálnými čísly.

- Konverze:
  Definované konverze mezi výše vypsanými typy. Rozklad matice na lineární a afinní část.

Popsané skupiny operátorů reprezentují základní matematické operace a základní operace pro práci s transformacemi. Dále je v programu implementováno několik specifických operátorů, jejichž funkci popisují následující kapitoly. Jedná se o operátor *sequence* umožnující vytvářet graf scény. Dále o dvojici operátorů *(*camera) a *screen*, které umožňují vytvářet nezávislé pohledy na scénu a studovat tak principy projekčních transformací. Program obsahuje také operátor *cycle*, pomocí kterého lze vytvářet jednoduché animace.

4.2.1 Sekvence transformací

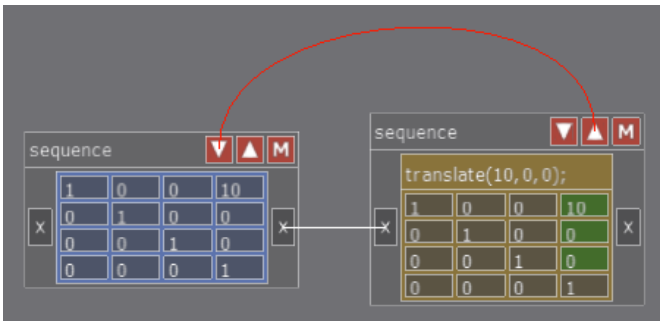
Krabička *sequence* transformací představuje komponentu z kapitoly 3.2, obrázek 3.2 a 3.6 a jeho grafická podoba v programu je vidět na obrázku 4.5.



Obrázek 4.5: Krabička *sequence* obsahující transformaci škálování, rotace a posunu a geometrický objekt

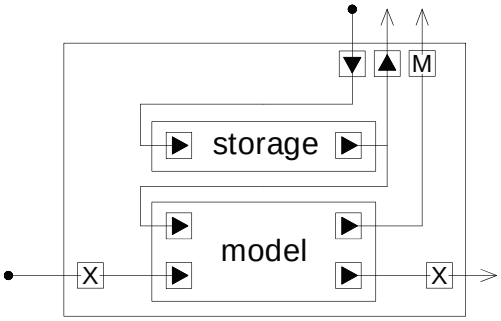
Krabička zastává dvojí funkci (čísla v závorkách v textu níže reprezentují odkazy na popisky v obrázku 4.5). Zaprvé jakéhosi kontejneru na matice transformací (2). Jednotlivé transformace lze libovolně přidávat, odebírat a měnit jejich pořadí. Komponenta pak poskytuje přístup k výsledné matici (3), která vzniká vynásobením matic v ní obsažených. Obsaženou matici lze také definovat na vstupu (4). Druhou funkcí je propojování krabiček do stromových struktur a vytvářet tak graf scény, neboť jednotlivé spoje (vstup (1) a výstup (6)) představují násobení obsažených matic. Krabička také může obsahovat geometrický objekt (7), který je transformován modelovou transformací, která vzniká násobením matic v krabičce a všech matic na cestě do kořene scény. Tato složená matice je zároveň poskytnuta na výstupu M (5).

Jako operátor má krabička dva vstupy a tři výstupy. Vstup (1) a výstup (6) pro násobení matic ve stromové struktuře grafu scény a vstup (4) a výstupy (3) a (5) pracující s obsaženými maticemi. Z hlediska funkcionality operátorů však nyní může nastat problém při zapojování, který demonstruje obrázek 4.6.



Obrázek 4.6: Dvě komponenty *sequence* zapojené cyklicky.

Na obrázku 4.6 jsou vidět dvě krabičky propojené do jednoduchého stromu grafu scény. Obsažená matice krabičky vpravo je navíc předávána na vstup (4) krabičky vlevo, takže jejich obsažené matice jsou nyní stejné. Toto propojení ovšem představuje cyklus, který porušuje podmínky pro orientovaný acyklický graf a kontrolní algoritmus 1 takové propojení zakazuje. Z matematického pohledu však takové propojení nepředstavuje problém, neboť všechny dotčené matice jsou jasně definovány. Tento problém má jednoduché řešení, které nepotřebuje přidávat žádnou další funkcionalitu ani změnu kontrolních mechanizmů.



Obrázek 4.7: Schéma operátorů komponenty *sequence*.

Krabička obsahuje dva pomocné operátory (storage) a (model), jejichž schématické znázornění v rámci komponenty je vidět na obrázku 4.7. Operátor nazvaný *storage* spravuje obsažené matice. Po připojení nějakého operátoru předávajícího matici na jeho vstup jsou obsažené transformace krabičky nahrazeny touto maticí. Výstup vrací buď výslednou matici, která vzniká součinem obsažených matic nebo matici připojenou na vstup v případě, že je připojen. Samotný operátor však s maticemi nedělá žádné operace a slouží pouze k přístupu k obsaženým maticím.

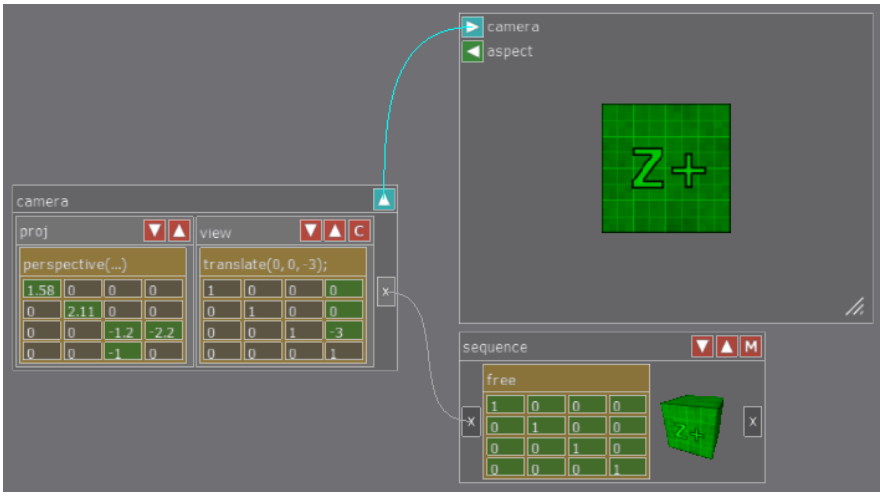
Druhý operátor *model* zajišťuje násobení matic ve stromové struktuře grafu scény. Matici, která je připojena na jeho vstup násobení, vynásobí zprava maticí, která je na výstupu příslušného operátoru *storage* a tuto matici předává na svůj výstup násobení. Operátor tedy obsahuje dva vstupy. Jeden pro matici předchůdce z grafu scény a druhý pro matici, kterou matici z grafu vynásobí. Samotná krabička zajišťuje, že výstup z operátoru *storage* je vždy připojen na příslušný vstup operátoru *model*.

Matice na již zmíněném výstupu pro složenou modelovou matici (5) vznikne násobením všech obsažených matic propojených sekvencí až do kořene stromu

grafu scény. Z programového hlediska se jedná o stejnou výstupní matici jako výstup násobení operátoru *model*. Z logického hlediska se však jedná o dvě odlišné věci, neboť propojení operátorů *model* představují násobení matic, zatímco výstup modelové matice stejně jako vstup a výstup operátoru *storage* představuje ukazatel a předává hodnoty příslušných matic.

4.2.2 Kamera a obrazovka

Program umožňuje uživateli studovat i principy projekčních transformací. K tomu slouží dvojice komponent Kamera a Obrazovka. Příklad použití znázorňuje obrázek 4.8.



Obrázek 4.8: Kamera s připojenou obrazovkou zobrazující objekt ve scéně.

Tato jednoduchá scéna demonstruje základní principy transformací v počítačové grafice. Obsahuje kameru s projekční transformací P, pohledovou transformací V a grafický objekt s modelovou transformací M. Je vidět pořadí násobení matic PVM a výsledný obrázek transformovaného objektu.

Na obrázku 4.8 je vidět komponenta kamery, která sdružuje dvě krabičky *sequence* dohromady. Jedna (na levé straně) slouží pro práci s projekční transformací a druhá s transformací pohledovou. Jejich funkcionalita se liší od normální krabičky *sequence* následujícím způsobem. Nelze je zapojovat do grafu scény. K propojování dochází automaticky tak, že jsou připojeny ke všem kořenovým komponentám grafu. Krabička s projekcí neobsahuje výstup modelové matice M. Krabička s pohledovou transformací tento výstup obsahuje, ale je přejmenován, neboť se nejedná o modelovou transformaci, ale o transformaci kamery známou jako PV nebo C. Tato transformace je rovna násobení matic projekční a pohledové transformace. Obě krabičky *sequence* jsou stále propojeny a z programového hlediska se tento výstup ničím neliší od klasického výstupu pro modelovou transformaci.

Další komponentou je obrazovka, která realizuje vykreslení pohledu z připojené kamery (vstup *(*camera)). Velikost obrazovky lze nastavovat a jako operátor vrací hodnotu poměru stran (výstup *aspect*), která se dá použít při vytváření projekčních transformací.

4.2.3 Cyklus

Operátor *cycle* slouží k vytváření jednoduchých animací objektů ve scéně. Jeho hlavním výstupem je jediná skalární hodnota, která se aktualizuje v reálném čase. Operátor má také tři vstupní hodnoty *from*, *to* a *step*. Výstupní hodnota postupně přechází od *from* k hodnotě *to* s krokem *step*. Výběrem z příslušného menu lze nastavit jednu ze tří metod cyklení.

- ONCE:
  Po dosažení hodnoty *to* cyklus zastaví.
- REPEAT:
  Po dosažení hodnoty *to* cyklus začíná znovu na hodnotě *from*.
- PING-PONG:
  Po dosažení hodnoty *to* se obrátí směr a cyklus běží pozpátku do hodnoty *from*. Poté začíná proces znovu.

Obrázek 4.9: Komponenta cycle.

Na obrázku 4.9 je vidět samotná komponenta operátoru. Hodnoty *from*, *to* a *step* lze získat z výstupů ostatních operátorů nebo zadat ručně pomocí příslušných editačních polí. Připojené vstupy mají vyšší prioritu než zadané hodnoty. K ovládání běhu cyklu slouží ovládací panel, který obsahuje zleva tlačítka play, pause, stop, krok zpět a krok vpřed. Protože cyklus běží v reálném čase, je hodnota kroku běžně nastavena na relativně malou hodnotu. Proto lze hodnotu kroku po stisku tlačítek krok vzad a krok vpřed nastavit zvlášť v editačním poli pod tlačítky.

Operátor obsahuje množství vstupů a výstupů umístěných v liště záhlaví. Jsou zde vstupy a výstupy pro všechna tlačítka ovládacího panelu a dvojice výstupů *start* a *end* signalizující začátek a konec cyklu. Lze jimi propojovat větší množství operátorů *cycle* a vytvářet tak složitější funkcionalitu. Například lze z jednoho operátoru připojit výstup play na libovolné množství dalších operátorů a spouštět tak všechny najednou pomocí jediného tlačítka. Nebo lze výstup *end* jednoho operátoru připojit na vstup *play* operátoru druhého. Po skončení cyklu prvního operátoru se pak rozběhne cyklus operátoru druhého.

4.3 Implementace vlastního operátoru.

Na závěr pojednání o operátorech si popíšeme nutné kroky pro přidání vlastního matematického operátoru do programu. Konkrétně vytvoříme operátor reprezentující funkci signum. Proces zahrnuje tři kroky. Vytvoření vlastní třídy

operatorForm.h:

```
class OperatorFormSignum : public OperatorForm {

public:
   OperatorFormSignum(TransformationSpaceScrollTab * _spaceTab,
                      TabGroup * _tabGroup,
                      CurveTabEngine * _curveTabEngine,
                      float x,
                      float y);

   virtual void updateTransmitterValues(int inputIndex);
   virtual SpaceItem * getCopy();
};
```

operatorForm.cpp:

```
OperatorFormSignum::OperatorFormSignum(TransformationSpaceScrollTab * _spaceTab,
                                       TabGroup * _tabGroup,
                                       CurveTabEngine * _curveTabEngine,
                                       float x,
                                       float y)

   : OperatorForm("Signum", _spaceTab, _curveTabEngine, _tabGroup, "signum", x, y) {

   inputTab = new OperatorFormInputTab(_tabGroup);
   inputTab->inputsLabels.push_back("float");
   inputTab->inputTypes.push_back(OpValueType::FLOAT);
   inputTab->createSubcomponents(this);

   outputTab = new OperatorFormOutputTab(_tabGroup);
   outputTab->functions.push_back(Transmitter(OpValueType::FLOAT));
   outputTab->createSubcomponents(this, false);

   createSubcomponents();
   updateTransmitterValues(0);
}

void OperatorFormSignum::updateTransmitterValues(int inputIndex) {

   if (operatorInputs[0]->isPluged()) {
      float inputValue = operatorInputs[0]->getTransmitter()->getFloat();
      outputTab->functions[0].setValue((inputValue > 0) ? 1.0f : ((inputValue < 0) ? -1.0f : 0.0f));
   } else {
      outputTab->functions[0].setValue(0.0f);
   }
}

SpaceItem * OperatorFormSignum::getCopy() {
   OperatorFormSignum * op = new OperatorFormSignum(spaceTab, tabGroup, curveTabEngine, position.x, position.y);
   return op;
}
```

Obrázek 4.10: Krok 1. Vytvoření třídy v souborech operatorForm.

operátoru, která rozšiřuje základní třídu *OperatorForm* v prostředí uživatelského rozhraní. Přidat vytvořený operátor do nabídky rozhraní a nakonec doplnit potřebnou funkcionalitu do třídy *Reader*, která zajišťuje ukládání a načítání scén.

Novou třídu operátoru *signum* je nutné přidat do souborů *operatorForm* (obrázek 4.10). Důležité je zadefinovat oba textové řetězce v parametrech konstruktoru třídy *OperatorForm*. První ("Signum") definuje identifikátor při ukládání a nahrávání scény a je nutné, aby byl mezi ostatními operátory jedinečný. Druhý ("signum") definuje jeho popis v záhlaví komponenty v rozhraní. Potřebné vstupy a výstupy je nutné přidat do polí *inputTypes* a *functions*. Dostupné enumerační typy jsou *Float*, *Vec3*, *Vec4*, *Quat* a *Matrix*. Pole *inputsLabels* obsahuje slovní popis vstupů. Samotnou operaci je nutné definovat ve funkci *updateTransmitterValues()* namapováním vstupů *operatorInputs[i]* na výstupy *outputTab->functions[i]*. Metoda *isPluged()* poskytuje informaci o tom, zda je příslušný vstup připojen a lze ji využít k upřesnění funkcionality operátoru. V případě, že vstup připojen není, vrací implicitní hodnotu pro typ proměnné vstupu. Parametr *inputIndex*

addOperatorPopup.h:

```
class AddOperatorPopup : public PopupMenu {

public:

  AddOperatorPopup(TabGroup * _tabGroup, float x, float y);

  static void addSignum(Keys::Code button, Tab * sender, glm::vec2 localMouse, bool mouseIn, MouseButtonState & state);
```

addOperatorPopup.cpp:

```
void AddOperatorPopup::addSignum(Keys::Code button, Tab * sender, glm::vec2 localMouse, bool mouseIn, MouseButtonState & state) {

   if (button == Keys::mouseLeft || button == Keys::mouseRight) {

      TransformationSpaceScrollTab * sf = static_cast<TransformationSpaceScrollTab *>(sender->getParent()->data);
      sf->addOperatorForm(new OperatorFormSignum(sf, sf->getTabGroup(), sf->curveTabEngine, 0, 0), false);
   }
}
```

```
AddFloatOperatorPopup::AddFloatOperatorPopup(TabGroup * _tabGroup, float x, float y)
  : PopupMenu("float operator", _tabGroup, x, y, 200) {

  TabPlacer p(9, 1, style->mainOffset, PlaceOrder::COLLUMNS);

  ...

  Button * buttonSignum = new Button("signum", _tabGroup, 0, 0, getClearSize().x, style->editBoxHeight, TabAlign::LEFT_CENTER);
  buttonSignum->onMouseUp = AddOperatorPopup::addSignum;
  p.addTab(buttonSignum, TabAlign::LEFT_TOP);

  ...
```

Obrázek 4.11: Krok 2. Přidání operátoru do vyskakující nabídky.

je roven indexu vstupu, který vyvolal aktualizaci operátoru a lze ho využít při optimalizaci funkce v případě, že některé vstupy neovlivňují všechny výstupy.

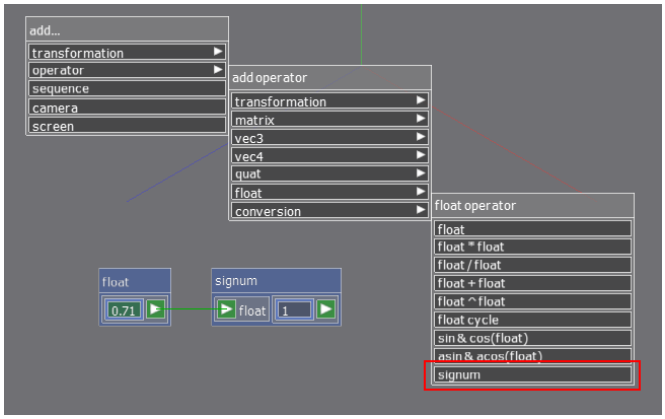
Dalším krokem je přidat operátor do příslušné vyskakovací nabídky operátorů, které jsou definované v souboru *addOperatorPopup* (obrázek 4.11). Nejprve je nutné vytvořit funkci události *addSignum* ve třídě *AddOperatorPopup* a poté přidat tlačítko *Button* v konstruktoru příslušné podnabídky, v tomto případě *AddFloatOperatorPopup*. V konstruktoru je také nutné aktualizovat počet tlačítek v nabídce, který udává první parametr konstruktoru *TabPlacer*.

Posledním krokem je doplnit funkci načítání v souboru *reader.cpp* jak je vidět na obrázku 4.12). Důležité je, aby podmínka *(input == "Signum")* obsahovala klíčové slovo shodné s identifikátorem definovaným v prvním kroku. Výsledný operátor je vidět na obrázku 4.13.

reader.cpp:

```
void Reader::readOperatorForms(ifstream & is, TransformationSpaceScrollTab * parent) {

...

if (input == "Signum") {
  op = new OperatorFormSignum(parent, parent->getTabGroup(), parent->curveTabEngine, 0, 0);
  readedOperators[operatorCounter++] = op;
  parent->addOperatorForm(op, true);
} else

...
```

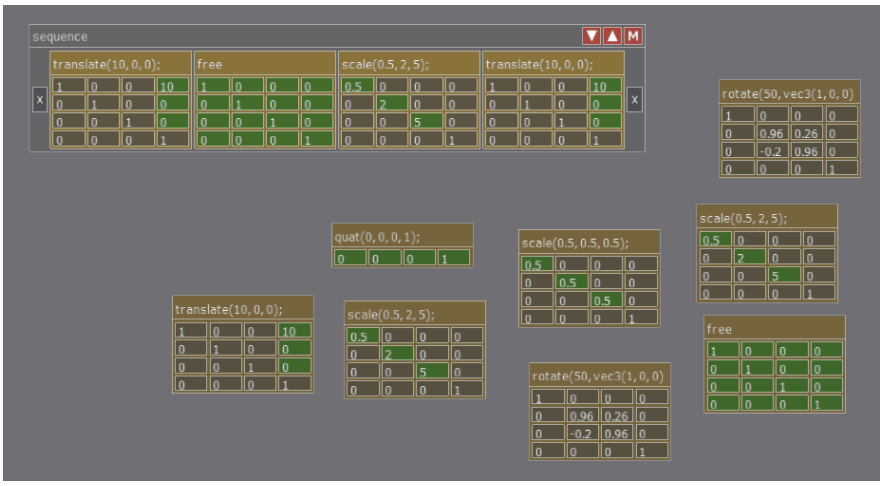
Obrázek 4.12: Krok 3. Doplnění funkce načítání ze souboru.



Obrázek 4.13: Vytvořený operátor *signum*.

4.4 Volné transformace

Volné transformace jsou komponenty reprezentující transformační matice. Tyto komponenty lze libovolně vytvářet a přiřazovat je metodou *drag and drop* do krabičky *sequence*. Spolu s krabičkou vytvářejí nástroj pro tvorbu složených modelových transformací. Obrázek 4.14 ukazuje příklady těchto komponent v programu. V horní části je krabička *sequence* obsahující 4 volné transformace a další jsou umístěné ve zbylé části pracovní plochy komponenty *space*.



Obrázek 4.14: Pracovní plocha obsahující různé komponenty volných transformací.

Program obsahuje 13 typů těchto transformací, kde každý jednotlivý typ reprezentuje dostupnou funkci knihovny GLM nebo obecně známou definici specifické transformace. Pro dané typy navíc příslušné komponenty zajišťují specifickou funkcionalitu. Každý dostupný typ má svůj konstruktor v podobě dialogového okna, které koresponduje s parametry příslušné funkce. Jedinou výjimkou je jednotková matice, která nepotřebuje žádné vstupní parametry.

Jednotlivé hodnoty reprezentované matice lze pak i po vytvoření stále editovat. Aby se zajistila výuková hodnota, editovat lze pouze hodnoty na takových pozicích a na takovém intervalu, který je pro daný typ relevantní. Například pro

transformaci translace lze editovat pouze první tři hodnoty čtvrtého sloupce a uživatel tak hned vidí, která část matice způsobuje posun. Zbylé hodnoty lze však vždy odemknout a opět zamknout. Je zde i možnost resetovat všechny hodnoty na defaultní, které byly zadány při vytvoření a i samotné defaultní hodnoty lze také editovat.

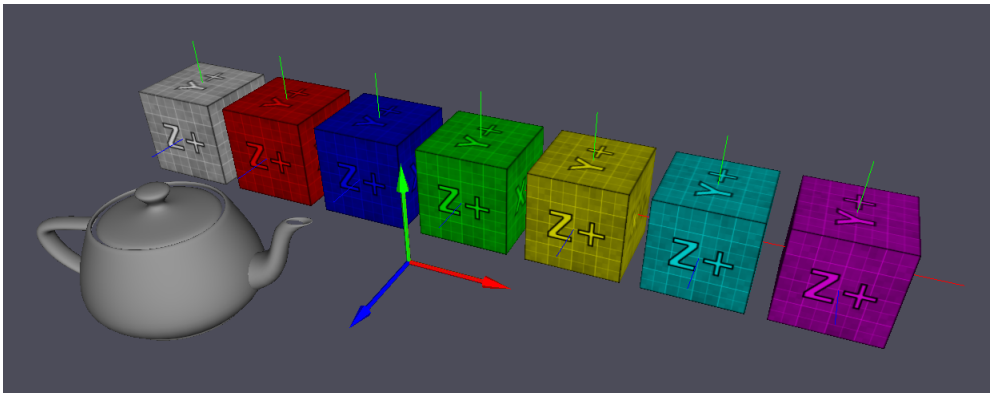
Další vlastnost specifická pro konkrétní typ matice jsou synergie mezi jejími hodnotami. Například u rotace kolem jedné z hlavních souřadnicových os jsou v matici odemknuty pouze příslušné čtyři hodnoty matice k editaci. Při změně jedné z těchto hodnot se ostatní hodnoty dopočítají tak, aby výsledná matice zůstala ortonormální. Tuto funkcionalitu lze opět libovolně vypínat a zapínat.

K dispozici je i úplně obecná matice s výchozí hodnotou identity, na které lze testovat vliv všech dílčích hodnot matice na transformaci a studovat tak i všechny transformace, které nejsou v programu k dispozici explicitně.

Volné matice spolu s komponentou *sequence* poskytují silný nástroj k vytváření komplexních transformací a s jejich pomocí lze snadno studovat problematiku vyplývající z nekomutativity násobení matic.

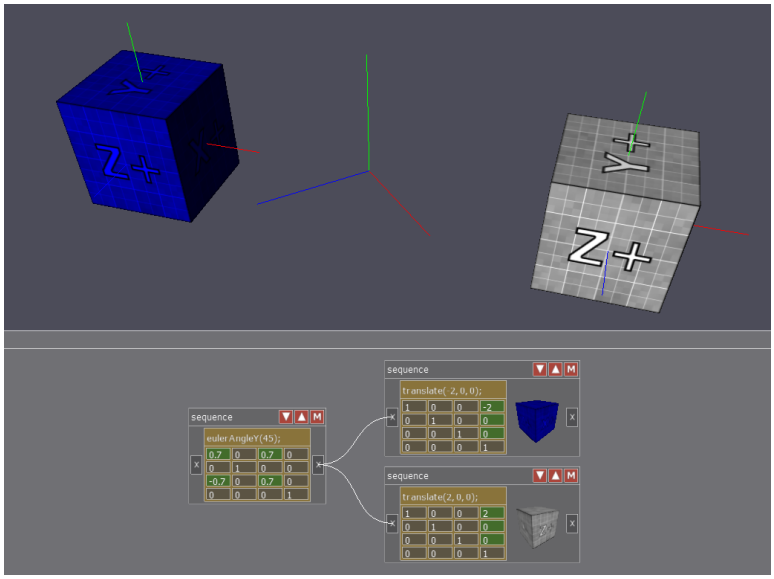
4.5 Grafické Objekty

Aby bylo možné pozorovat účinky vytvářených transformací, program obsahuje sadu geometrických objektů, na které lze tyto transformace aplikovat. Dostupné objekty (7 jednotkových krychliček, čajník a model ortonormálních bázových vektorů) jsou vidět na obrázku 4.15.



Obrázek 4.15: Dostupné grafické objekty v programu.

Základním objektem je model ortonormálních bázových vektorů. Na tomto modelu půjde vždy nejlépe pozorovat efekt aplikovaných transformací. Jednotlivé osy jsou barevně rozlišeny podle známé konvence (osa x červeně, osa y zeleně a osa z modře). Při použití více než jednoho takového objektu však může nastat problém v rozlišení, ke které transformaci náleží. Proto je k dispozici i sada různě barevných jednotkových krychliček. Aby bylo možné rozpoznat jejich natočení, jsou jejich stěny označeny příslušnou osou. Transformace všech objektů jsou navíc automaticky doplněny vykreslením příslušných bázových vektorů.



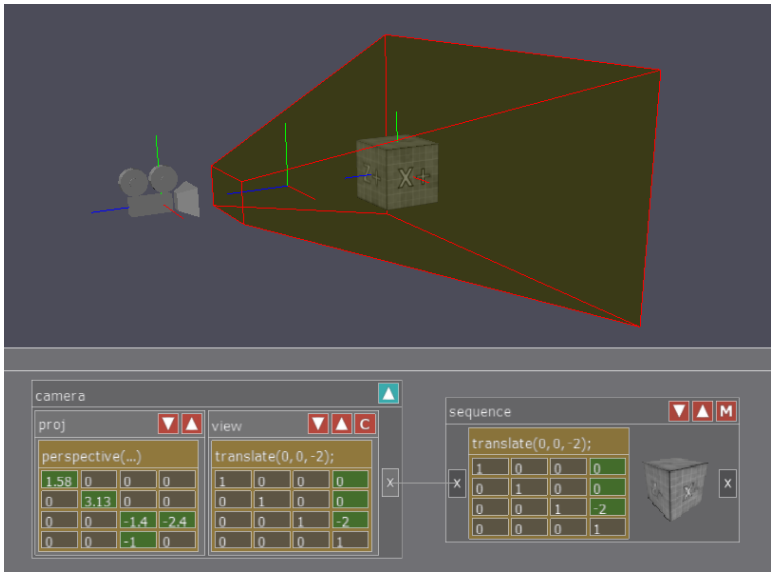
Obrázek 4.16: Dva objekty ve scéně transformované hierarchicky grafem scény.

Jednotlivé objekty lze připojit ke krabičkám *sequence*. Objekt je poté transformován modelovou maticí příslušné krabičky, tedy maticí vzniklou násobením všech matic dílčích krabiček příslušné větve grafu scény až do kořene, což demonstruje obrázek 4.16. K dosažení větší přehlednosti je navíc objekt zobrazen i v příslušné krabičce *sequnce*. Umístění jeho reprezentace v krabičce lze chápat ve smyslu násobení sloupcové matice vektorů souřadnic vrcholů objektu jednotlivými maticemi na cestě do kořene stromu grafu scény.

Celá sada z aplikace dostupných objektů je definována v souborovém systému aplikace. Uživatel tak může bez kompilace programu sám definovat všechny objekty, které budou v rozhraní k dispozici. Program podporuje objekty ve formátu Wavefront obj a v souborovém systému lze dále definovat i použité materiály a textury.

4.5.1 Objekt kamery

Aby bylo možné pochopit funkci projekčních a pohledových transformací je nutné, aby i komponenta kamery měla svou grafickou reprezentaci ve scéně. Ke každé komponentě kamery je tedy automaticky připojen schématický model kamery, který je transformován inverzí její pohledové transformace.

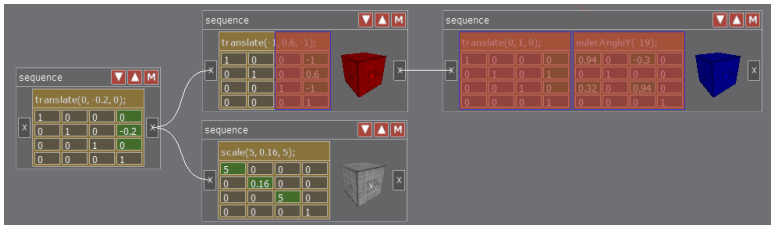


Obrázek 4.17: Grafická reprezentace kamery a její projekční transformace.

Příklad je vidět na obrázku 4.17. Součást grafické reprezentace je i vykreslení pohledového jehlanu včetně blízké a vzdálené ořezávací roviny, jehož tvar je definován příslušnou projekční transformací. Uživatel tak může pozorovat vliv jednotlivých hodnot projekční matice na výslednou projekci.

4.6 Postupná animace transformací

Geometrické objekty ve scéně umožňují pozorovat aplikované transformace pouze ve výsledném tvaru, který může být výsledkem kombinace mnoha dílčích transformací. Během uživatelského testování vznikl návrh na implementaci módu, který umožňuje pozorovat jejich postupný vývoj a lépe tak pochopit princip kombinování transformací. Vstupním parametrem je libovolná komponenta *sequence*, která je součástí grafu scény. Po aktivaci módu může uživatel pohybovat pomyslnou hranicí po celé větvi grafu scény od příslušného kořene do uzlu, který tvoří vybraná komponenta *sequence*. Definuje tak transformaci, která vzniká násobením matic v pořadí od vybraného uzlu grafu až k pomyslené hranici. Hranice se navíc pohybuje s krokem, který dělí transformace obsažené v komponentách *sequence* na příslušné větvi a aktuálně rozdělaná transformace je interpolována.



Obrázek 4.18: Sekvence transformací v módu postupné animace

Obrázek 4.18 zobrazuje příklad grafu scény s aktivním módem postupné animace. Transformace obsažené v částečné transformaci jsou obarveny a scéna ob-

sahuje objekt, který je vzniklou maticí transformován. Lze tak plynule pozorovat celý průběh kombinované transformace.

4.7 Uživatelské rozhraní

V rámci projektu byla vytvořena i jednoduchá knihovna uživatelského rozhraní. Uživatelské rozhraní lze rozdělit na dvě hlavní části. První je samotná knihovna obsahující základní komponenty jako formulář, tlačítko nebo editační pole a struktury řídící jejich aktualizaci a vykreslování a je vytvořena nezávisle na zbytku programu. Druhá část implementuje specializované komponenty rozhraní vytvořené konkrétně pro účely programu. Struktury zajišťující komunikaci mezi rozhraním a logickou částí programu, která zahrnuje struktury potřebné pro správu a vykreslování objektů ve scéně.

4.7.1 Knihovna

Ústřední komponentu knihovny tvoří třída Tab (Třída 2) a je předkem všech ostatních komponent rozhraní. Jedná se o jednoduchou komponentu, která zajišťuje základní interakci a vykreslování. Pomocí ukazatele *parent* a pole ukazatelů *tabs* jsou všechny komponenty uspořádány do stromové struktury.

```
class Tab {

private:

        Tab * parent;
        vector<Tab *> tabs;

        bool visible;
        bool ordered;
        bool clickable;

        vec2 position;
        vec2 size;

        void * data;

        virtual Tab* mouseDown(Keys::Code button, vec2 mouse, vec2 locMouse);
        virtual Tab* mouseUp(Keys::Code button, vec2 mouse, vec2 locMouse);
        virtual Tab* mouseOver(vec2 mouse, vec2 locMouse, vec2 mouseDelta);

        bool pointIn(locMouse);

public:

        void (*onMouseDown)(Keys::Code button, Tab * sender, vec2 localMouse);
        void (*onMouseUp)(Keys::Code button, Tab * sender, vec2 localMouse, bool mouseIn);
        void (*onMouseOver)(Tab * sender, vec2 localMouse, vec2 mouseDelta);

        virtual void update();
        virtual void draw();
};
```

Třída 2: Třída Tab.

Stromové uspořádání umožňuje rekurzivní volání klíčových funkcí *draw*, *update* a funkcí událostí *mouseDown*, *mouseUp* a *mouseOver*. Zmíněné funkce jsou virtuální a odvozené třídy mohou podle potřeby modifikovat jejich funkcionalitu.

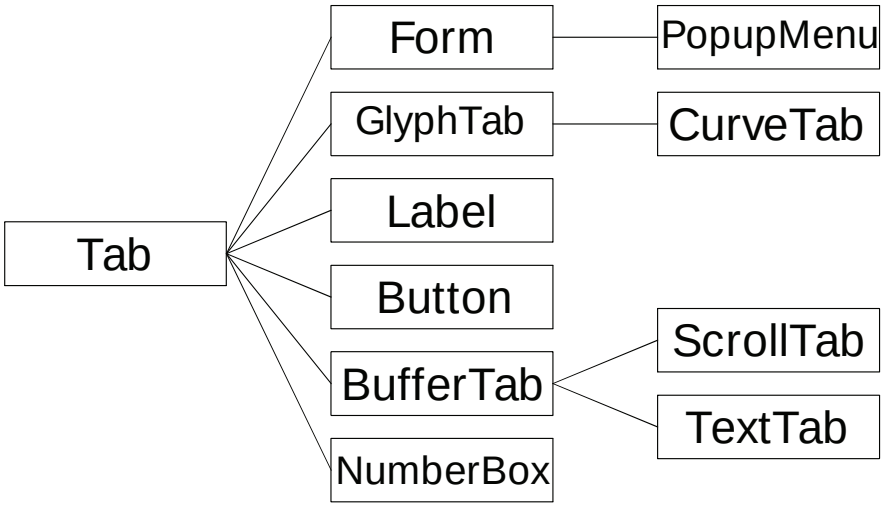
Funkce *draw* zajišťujíce vykreslování komponenty. Funkce *update* provádí aktualizaci komponenty, kterou je nutné vykonávat v každém logickém kroku. Většina komponent ovšem tuto funkci nechává prázdnou a nevykonává její volání ani pro své potomky a její používání je v programu spíše výjimkou. Příkladem použití je rotace objektu v náhledu komponenty *sequence*, pokud má nějaký geometrický objekt připojený.

```
Tab* Tab::mouseDown(Keys::Code button, vec2 mouse, vec2 locMouse) {

        if (!clickable || !visible) return NULL;

        if (pointIn(locMouse)) {

                if (ordered && parent != NULL) parent->setActiveTab(this);

                Tab * t = NULL;
                for (unsigned int i=0; i<tabs.size(); i++) {
                        t = tabs[i]->mouseDown(button, mouse, locMouse - tabs[i]->position);
                        if (t != NULL) return t;
                }

                if (onMouseDown != NULL) onMouseDown(button, this, locMouse);

                return this;
        }

        return NULL;
}
```

Algoritmus 2: Funkce události stisku tlačítka na myši.

Uživatelská interakce s komponentami je zajištěna funkcemi *mouseDown*, *mouseUp* a *mouseOver*, které jsou volány v případě stisku nebo uvolnění tlačítka myši a v případě, že je ukazatel myši v prostoru komponenty. Funkce *mouseDown* je ukázána v Algoritmu 2. Funkce *mouseUp* a *mouseOver*, ale i *draw* a *update* jsou koncipovány velmi podobně. Vstupními parametry funkce je kód tlačítka myši *button*, poloha kursoru myši v globální souřadné soustavě rozhraní a lokální poloha kurzoru myši v souřadné soustavě příslušné komponenty. V Algoritmu 2 je vidět, že pokud je komponenta neviditelná nebo není interaktivní, algoritmus končí a nevolá funkci ani na své potomky. Stejný konec nastane i v případě, že ve chvíli události není ukazatel myši v prostoru komponenty. V opačném případě je funkce volána pro všechny potomky a její volání tak odpovídá rekurzivnímu prohledávání do hloubky. Funkce zároveň vrací ukazatel na komponentu, která reaguje na událost, což definuje přiřazený ukazatel na funkci příslušné události, v případě funkce *mouseDown* (Algoritmus 2) ukazatel *onMouseDown*. Tímto způsobem jsou definovány všechny události specializovaných komponent uživatelského rozhraní programu. Návratová hodnota je používána v některých komplexních komponentách rozhraní a zároveň je použita pro detekci ukončení volání a návrat z rekurze. Přiřazená funkce ukazateli příslušné události tak proběhne pouze pro komponentu, která je ve struktuře komponenty nejvíce na vrchu.

Komponenta také implementuje funkcionalitu spojenou s pořadím vykreslování, které funguje na principu malířova algoritmu. Komponenty jsou vykreslovány v obráceném pořadí, odpovídajícímu jejich seřazení v poli dílčích komponent *tabs*. Funkce *setActiveTab* při volání události *mouseDown* zařadí komponentu na začátek a ostatní komponenty, které byly zařazeny přední, posune. Pole dílčích komponent je tak během interakce postupně řazeno.



Obrázek 4.19: Hierarchie komponent Knihovny uživatelského rozhraní.

V rámci knihovny je vytvořeno 10 základních komponent. Všechny rozšiřují předka třídy *Tab* a jejich přehled a struktura dědění je vidět na obrázku 4.19.

Popis komponent knihovny uživatelského rozhraní:

- Form:
  Přidává popis (komponenta *Label*) a oddělující čáru záhlaví.
- PopupMenu:
  Obsahuje funkcionalitu spojenou s hierarchickou strukturou vyskakovacích nabídek a událostmi jejich zobrazení.
- GlyphTab:
  Komponenta vykreslovaná s obrázkem, který je definován regionem a texturou.
- CurveTab:
  Komponenta používána pro propojování krabiček rozhraní. Přidává událost, která je volána při propojení komponent a také poskytuje ukazatel na funkci, která kontroluje korektnost propojení. Pro vykreslování křivek požívá Fergusonovu kubiku. Každá komponenta zároveň obsahuje obrázek upřesňující její funkci.
- Label:
  Komponenta obsahující jednořádkový text.
- Button:
  Tlačítko s textem.
- BufferTab:
  Před samotným vykreslením komponenta nejprve svůj obsah vykreslí do objektu FrameBufferObject knihovny OpenGL. Toho je využito ke snížení složitosti vykreslování obsáhlých komponent a u komponent, které po grafické stránce manipulují se svým obsahem. Jedna bufferovaná komponenta může obsahovat další bufferované komponenty a ke korektnímu připojovaní a odpojování objektů FrameBufferObject je použit zásobník.

- ScrollTab:
  Komponenta umožňuje přibližování, oddalování a posouvání svého obsahu modifikací projekční matice rozhraní.

- TextTab:
  Komponenta vykreslující víceřádkový text.

- NumberBox:
  Komponenta vykresluje referenční číselnou hodnotu a umožňuje její editaci. Editovat hodnotu je možné klávesovým vstupem a po kliknutí také tahem myši.

5. Výukové scény

Hlavním cílem programu je poskytnout uživateli nástroj, který mu umožní svou vlastní cestou studovat a pochopit principy geometrických transformací v prostoru. Skrze vytvořené rozhraní může vytvářet libovolné transformace, které lze popsat maticemi 4x4 a kvaterniony a provádět s nimi dostupné matematické operace. Podle potřeby lze vytvořené transformace aplikovat na geometrická tělesa a pozorovat tak jejich účinek. Parametry jednotlivých transformací lze kdykoliv libovolně nastavovat a tyto změny pozorovat na transformovaných tělesech v reálném čase. Lze tak snadno a názorně studovat principy geometrických transformací individuální cestou.

V rámci projektu je však v programu dále vytvořena série výukových scén, které si kladou za cíl seznámit uživatele s vybranými úlohami a základními transformačními funkcemi, které se běžně používají při práci s knihovnou OpenGL a počítačovou grafikou obecně. Skrze vytvořené scény uživatel postupně prostuduje principy sestavení matic modelové transformace, problematiku spojenou s kombinováním většího počtu transformací, princip grafu scény a význam vztažných soustav, které transformace reprezentují. Dále prostuduje transformace spojené kamerou. Připravené scény demonstrují funkcionalitu standardních projekčních transformací a význam pohledové transformace.

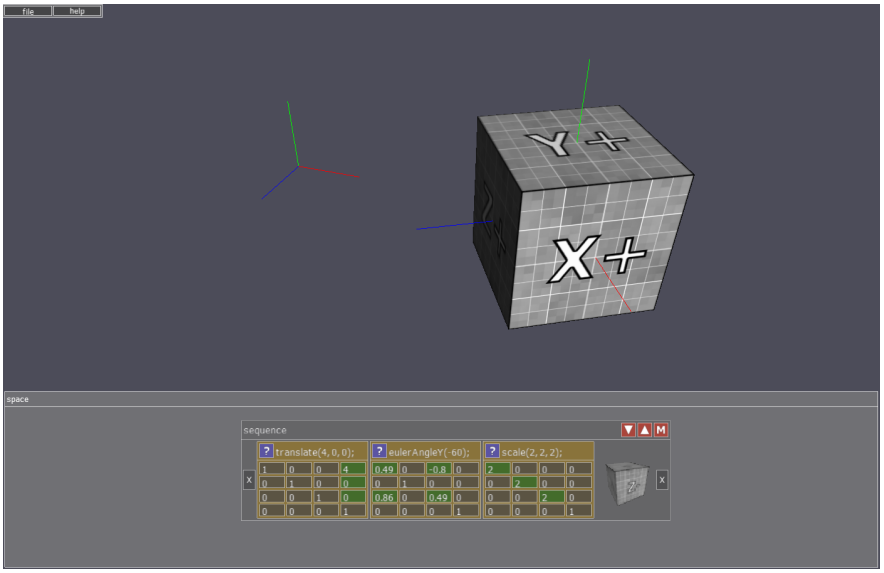
Každá výuková scéna navíc obsahuje úvodní popis, který má uživatele informovat o dané problematice a jednotlivých cílech vybrané scény. Tímto způsobem je uživatel vyzván ke konkrétním úkonům, které by měl v dané scéně vyzkoušet. Popis také obsahuje seznam otázek, na které by měl být, po pochopení vybrané problematiky, schopen odpovědět. Vybrané komponenty jsou navíc doplněny popisem, který vysvětluje jejich význam. Vytvořená série výukových scén tak vytváří základní kurz transformací v počítačové grafice.

Všechny výukové scény jsou vytvořeny v samotném rozhraní aplikace a k jejich tvorbě či úpravám není zapotřebí kompilace. Zkušený pedagog tak může aplikaci využít k tvorbě vlastní výukové sady scén či doplnit tu stávající o vlastní scény dle úvahy. Ve výukových scénách je zároveň stále dostupný veškerý zbylý obsah aplikace. Student tak může výukovou scénu začít libovolným způsobem editovat, přidávat nové transformace a komponenty a studovat tak další témata spojená s nastolenou problematikou zcela po svém.

Následující kapitoly stručně popisují jednotlivé výukové scény.

5.1 Modelová transformace

Modelová transformace typicky definuje pozici, orientaci a škálování objektů ve scéně a je základním typem transformace. Proto je tato scéna zařazena hned na úvod výukových scén. Scéna neobsahuje žádné kamery ani jiný nadbytečný obsah, který by odváděl pozornost od tématu a zbytečně snižoval přehlednost úvodní výukové scény. Scéna je vidět na obrázku 5.1.



Obrázek 5.1: Výuková scéna modelová transformace.

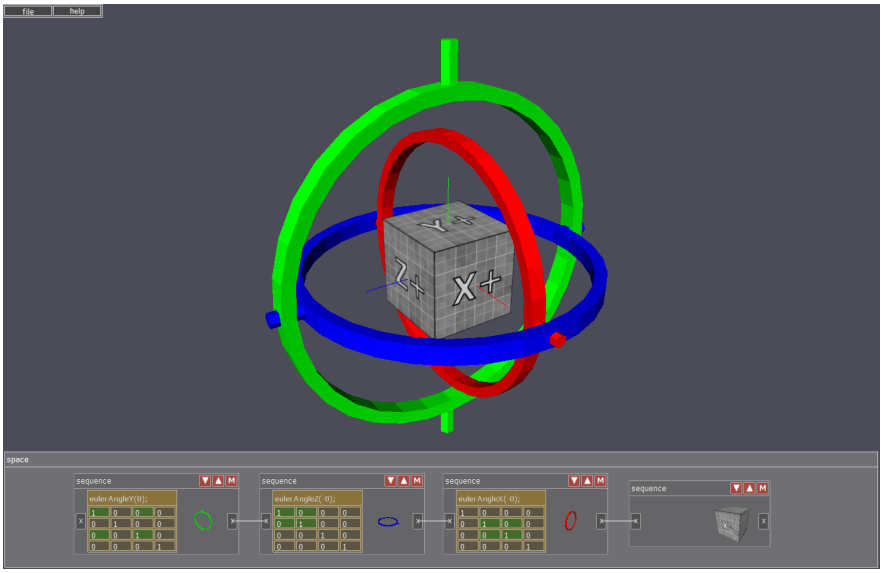
Obsahuje jedinou krabičku *sequence* s přiřazeným geometrickým tělesem krychle. Komponenta obsahuje tři základní transformace: škálování, rotaci a posun. Přednastavené transformace ani jejich pořadí není vybráno náhodou. Jedná se totiž o známý způsob popisu transformace těles používaný například v prostředí VRML. Jedině při aplikaci transformací v tomto pořadí bude opravdu těleso na pozici, která je definován v transformaci posunutí.

Uživatel může zaměnit pořadí transformací v sekvenci a pozorovat odlišný výsledek. Dále může studovat funkcionalitu jednotlivých transformací editací jejich parametrů, transformace odebírat a přidávat jiné. Přidáním jediné matice identity, která má editovatelné všechny hodnoty, může studovat vliv všech parametrů transformace popsané maticí 4x4.

Scéna Modelová transformace simuluje aplikaci Wolfram: Linear Transformations (obrázek 2.8), ve které je omezený počet dostupných transformací i počet transformací, které lze aplikovat zároveň. Scéna navíc popisuje transformace ve 3D a také se neomezuje pouze na lineární transformace.

5.2 Eulerovy úhly

Eulerovy úhly jsou běžně používány pro reprezentaci orientace objektu. Jejich používání však sebou nese několik úskalí, a pro člověka, který s transformacemi v počítačové grafice teprve začíná, může být jejich používání opravdu matoucí. Jedná se však o absolutní základ popisovaný v mnoha tutoriálech a publikacích, a z toho důvodu se nejeden začínající programátor setká s Eulerovými úhly jako první možností, jak reprezentovat rotaci. Problematika spočívá ve volbě pořadí aplikace rotací kolem souřadných os a navíc dochází i k nepříjemnému problému splynutí os, kdy nejpravější rotace přestane pracovat a splyne s jinou osou.

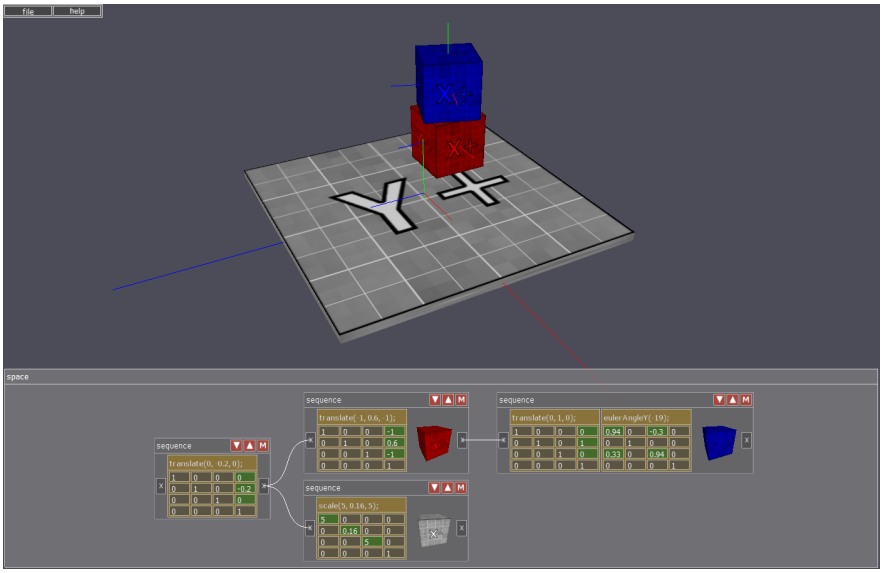


Obrázek 5.2: Výuková scéna Eulerovy úhly.

Výuková scéna je vidět na obrázku 5.2. Obsahuje testovací těleso, které je postupně transformováno rotacemi kolem všech tří hlavních souřadných os. Scéna je doplněna tříosým gimbalem, jak tomu bývá v mnohých ukázkách demonstrujících tuto úlohu. Ve scéně je však přehledně vidět, jakými transformacemi a v jakém pořadí jsou jednotlivé transformace aplikovány na testovací těleso a dokonce i na jednotlivé osy gimbalu a celý výpočet tak není schován za pouhé tři vstupní hodnoty úhlů. Uživatel tak může názorně studovat nejen princip Eulerových úhlů, ale i problematiku posloupnosti transformací. Celá soustava může být dále transformována a uživatel tak může vyzkoušet, jakým způsobem korektně spojit takto definovanou rotaci například s posunem nebo jinými transformacemi. Ve scéně lze také velmi dobře demonstrovat příčiny vzniku problému splynutí os, konkrétně otočením tělesa o 90 stupňů okolo osy z.

5.3 Graf scény

Další výuková scéna se zabývá standardním popisem scény pomocí grafu, obecně nazývaným graf scény. Existuje více způsobů reprezentace objektů a transformací v grafu. Princip je ovšem vždy stejný. Jednotlivé geometrické objekty ve scéně jsou hierarchicky strukturovány za účelem sdílení transformací. Výuková scéna je vidět na obrázku 5.3.

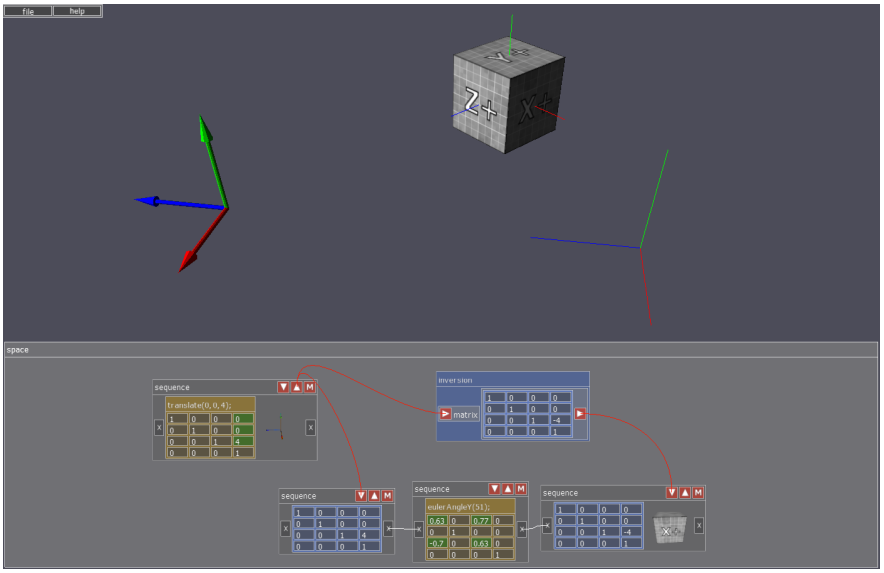


Obrázek 5.3: Výuková scéna graf scény.

Scéna obsahuje tři objekty seskupené do stromové struktury a uživatel může snadno vyzkoušet, že jednotlivé transformace ovlivňují všechny objekty, které se nachází na společné větvi směrem od kořene stromu doprava. Dále může přidávat nové uzly a objekty do scény a libovolně je transformovat. Pomocí dostupných výstupů a vstupů krabičky *sequece* je navíc možné i studovat složitější variantu grafu scény, který je tvořen orientovaným acyklickým grafem a kopírovat tak transformaci nějakého uzlu, nebo dokonce samotnou modelovou transformaci vzniklou násobením matic od kořene, do jiného uzlu grafu.

5.4 Rotace okolo bodu

Výuková scéna rotace okolo bodu demonstruje známou úlohu geometrických transformací. Scéna, kterou můžeme vidět na obrázku 5.4, názorně ukazuje, jakým způsobem lze otočit geometrickým objektem, v tomto případě krychličkou, kolem definovaného bodu v prostoru, který je znázorněn bázovými souřadnicemi.



Obrázek 5.4: Výuková scéna rotace okolo bodu.

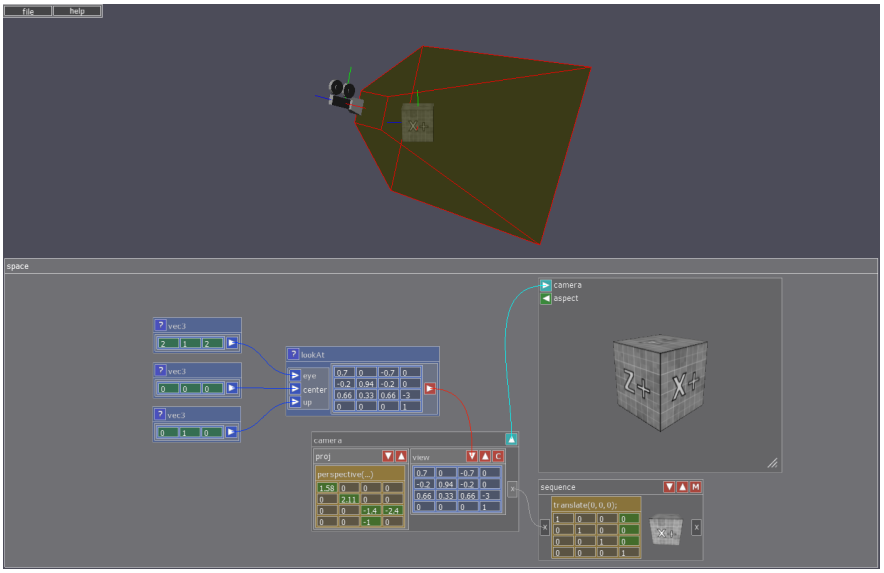
Transformaci lze matematicky popsat výrazem TRT^{-1}, kde T je matice translace bodu, kolem kterého chceme tělesem otáčet a R je samotná matice rotace, což je ve scéně velmi názorně ukázáno.

Velmi důležité je ovšem pochopit tuto transformaci vzhledem k významu vztažných souřadných soustav, které jednotlivé matice reprezentují. Geometrické těleso je nejprve transformováno do lokální souřadné soustavy bodu, kolem kterého je otáčeno. Po aplikaci samotné rotace je transformované zpět do globální souřadné soustavy. Stejnou úlohu demonstruje aplikace Wolfram: Rotation about a Point in the Plane (obrázek 2.9). Nyní je ovšem možné nahradit rotaci za jinou transformaci nebo dokonce nahradit posun a definovat tak jinou vztažnou soustavu, ve které je těleso transformováno.

Scéna tak ukazuje, jak je možné obecně transformovat z globální souřadné soustavy do lokální souřadné soustavy vyjádřené invertibilní maticí a naopak z lokální souřadné soustavy do globální. Celý princip lze snadno přesunout výše a nahradit globální souřadnou soustavou soustavou jinou. Tento základní princip transformací lze v programu snadno a názorně pozorovat, konkrétně stačí připojit oba kořeny k nové komponentě *sequence* a doplnit libovolnou transformaci.

5.5 LookAt

Scéna LookAt seznamuje uživatele s funkcí *lookAt*, která slouží k vytvoření pohledové transformace kamery.



Obrázek 5.5: Výuková scéna pohledové transformace lookAt.

Scéna zobrazená na obrázku 5.5 obsahuje jeden objekt a kameru. Kamera obsahuje projekční transformaci a její pohledová transformace je vytvořena pomocí operátoru *lookAt*. Uživatel tak může zkoušet nastavovat jednotlivé parametry vstupních vektorů a ihned pozorovat tvar výsledné matice pohledové transformace i její vliv na umístění a orientaci kamery ve scéně a stejně tak lze pozorovat i aktuální pohled z této kamery. Zároveň však je zachována možnost editovat projekční i modelovou transformaci a je názorně vidět posloupnost transformací PVM.

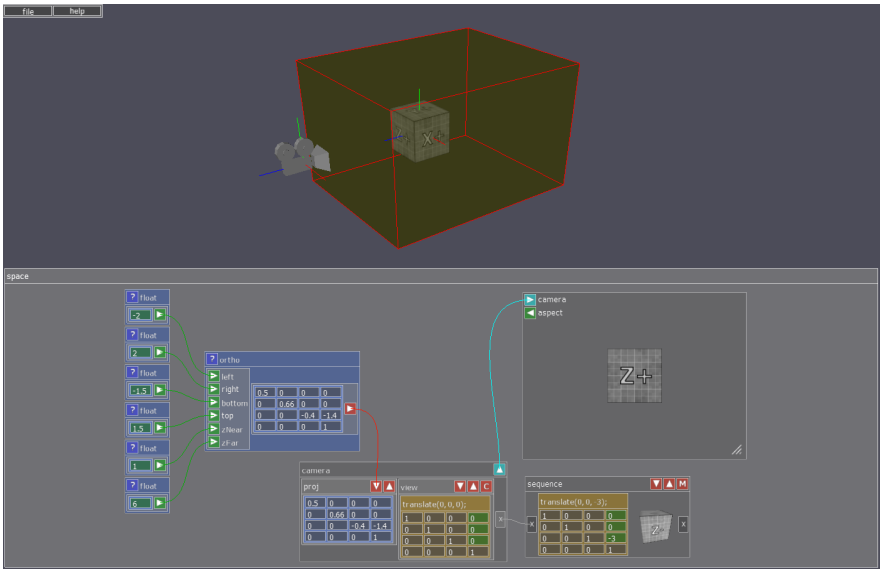
Výuková scéna LookAt ukazuje, že lze v programu snadno simulovat výukové aplikace Song Ho Ahn: OpenGL ModelView Matrix (obrázek 2.1) a Siggraph: Projection (obrázek 2.4). Navíc lze pomocí dostupných funkcí vytvořit scénu, ve které například kamera sleduje daný objekt nebo ze vstupních vektorů funkce *lookAt* spočítat výslednou pohledovou matici manuálně nebo od funkce *lookAt* úplně upustit a vytvořit pohledovou transformaci pomocí dostupných oprátorů a volných transformací stejně jako u modelové transformace.

5.6 Projekční transformace

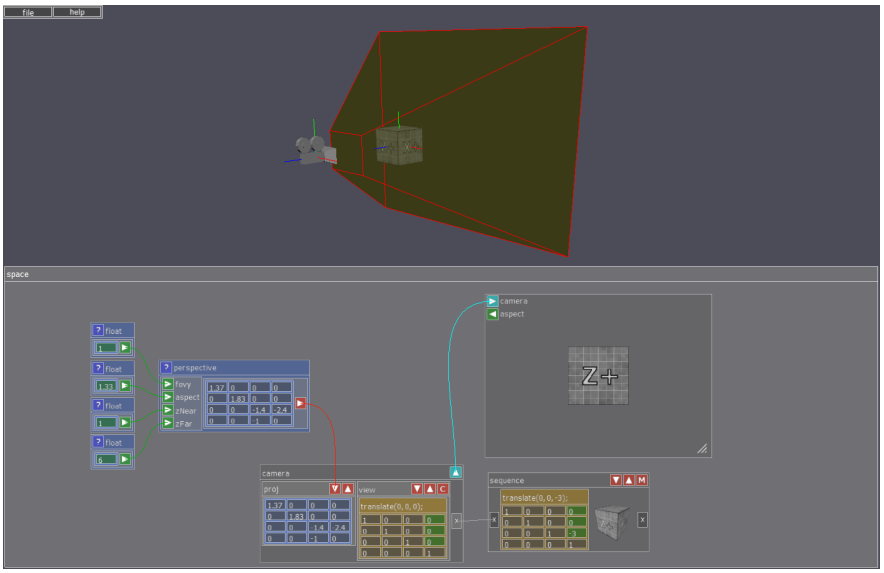
Následující scény demonstrují základní funkce pro vytváření projekčních transformací *ortho()*, *perspective()* a *frustum*. Do komponenty *sequence* projekce kamery lze ovšem dosadit libovolné matice 4x4 a studovat tak i další typy projekčních transformací.

5.6.1 Ortho, perspective a frustum

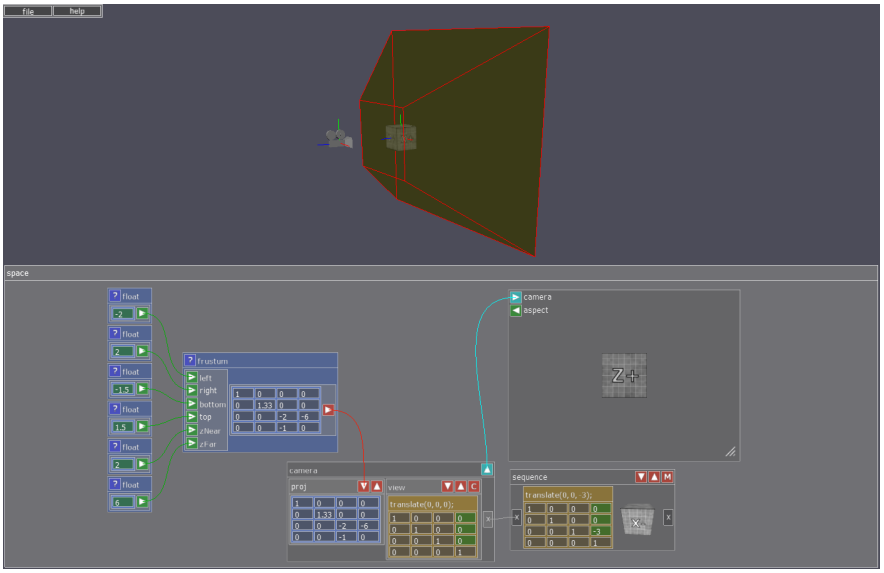
Výuková scéna Ortografická projekce (obrázek 5.6) demonstruje základní funkci vytvářející projekční transformaci rovnoběžného promítání *ortho*. Scéna Perspektivní projekce (obrázek 5.7) demonstruje funkcionalitu funkce *(*perspective) a scéna Perspektivní projekce frustum (obrázek 5.8) předvádí alternativní funkci *frustum* k funkci *perspective* vytvářející perspektivní projekční transformace.



Obrázek 5.6: Výuková scéna ortografická projekce.



Obrázek 5.7: Výuková scéna perspektivní projekce.



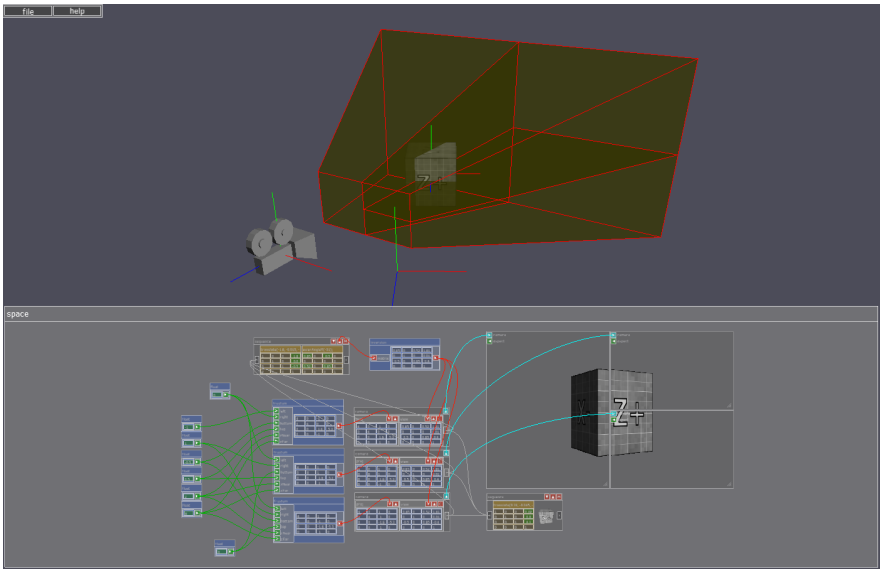
Obrázek 5.8: Výuková scéna perspektivní projekce frustum.

Všechny tři scény jsou koncipovány stejným způsobem. Matice projekční transformace je vytvořena příslušným operátorem. Jednotlivé matice lze vytvořit i pomocí komponent volných matic, ale operátory umožňují snadnou editaci jednotlivých parametrů transformačních funkcí jako například polohy ořezávacích rovin. Změny parametrů lze okamžitě pozorovat na tvaru pohledového jehlanu v globálním náhledu i na tvaru výsledné matice projekce. Scény obsahují i geometrický objekt, jehož zobrazení vznikající příslušnou projekcí lze pozorovat na připojené krabičce obrazovky a jehož modelovou transformaci lze nadále editovat.

Scény simulují výukové programy Song Ho Ahn: OpenGL Projection Matrix 2.2 a Siggraph: Projection 2.4. Navíc lze nastavovat i pohledovou a modelovou transformaci a všechny matice jednotlivých transformací jsou vidět stejně jako posloupnost transformací PVM.

5.6.2 Projekce na více monitorů

Výuková scéna Projekce na více monitorů (obrázek 5.9) demonstruje nejen možnosti funkce *frustum*, ale i samotného programu. Scéna názorně předvádí, jakým způsobem lze vytvořit projekci na více monitorů zároveň. Ve scéně jsou tři monitory různých velikostí a každý z nich má svojí kameru se specifickou projekční transformací. Šest parametrů funkce *frustum* udávajících tvar pohledového jehlanu je sdíleno jednotlivými funkcemi a polohy ořezávacích rovin tak lze nastavovat pro všechny tři kamery najednou. Další dva parametry nastavují polohy ořezávacích rovin ve středu jehlanu. Scéna lze vyřešit i sofistikovanějším způsobem, který uvažuje poměry stran (*aspect*) jednotlivých monitorů, ale výsledná scéna by byla kvůli množství operátorů nepřehledná.

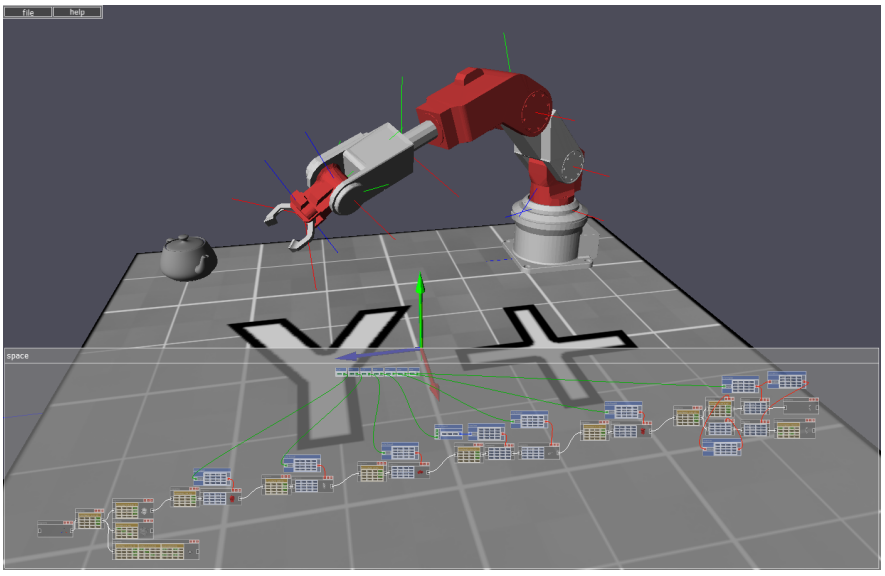


Obrázek 5.9: Výuková scéna perspektivní projekce frustum na více monitorů.

Všechny tři kamery sdílí společnou pohledovou transformaci a lze tak ovládat jejich orientaci zároveň. Uskupení kamer se tak chová jako by šlo o kameru jedinou. Transformace kamer je vytvořena rotací a posunem a dá se dále modifikovat stejným způsobem jako modelová transformace. Ve scéně je vidět, že aby bylo možné transformaci korektně použít jako pohledovou transformaci kamery, je nutné vytvořenou matici invertovat.

5.7 Arm

Scéna Arm (obrázek 5.10) inspirovaná programem Arm (obrázek 2.6) ukazuje praktické využití hierarchie transformací a grafu scény. Zároveň demonstruje možnosti programu, využití matematických operátorů a možnosti použití vlastního obsahu.

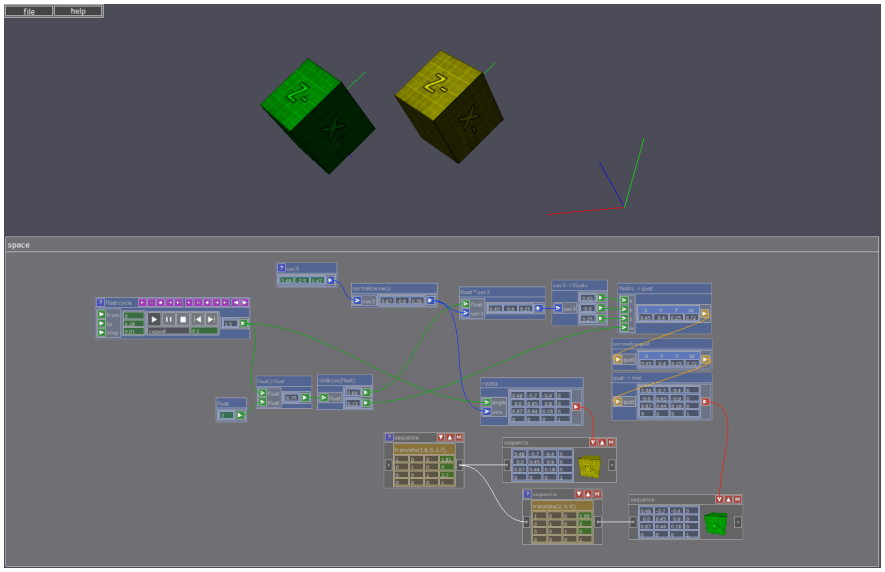


Obrázek 5.10: Scéna Arm.

Scéna obsahuje mechanické rameno, které má sedm stupňů volnosti zahrnující rotace i posunutí. Pro každý stupeň volnosti scéna obsahuje jeden ovládací prvek nastavující příslušný úhel nebo posunutí. Jednotlivé díly ramena jsou strukturovány do grafu scény a příklad zároveň ukazuje možnosti využití acyklického orientovaného grafu pro její reprezentaci. Je tak možné jediným parametrem ovládat více prvků zároveň. Příklad také ukazuje možnosti využití inverze a transpozice při vytváření transformací. Scéna ovšem může na první pohled působit složitě a proto je k dispozici i zjednodušená verze využívající pouze jednoduchý graf scény reprezentovaný stromem.

5.8 Kvaternion

Scéna Kvaternion (obrázek 5.10) vznikla jako výsledek uživatelského testování a názorně ověřuje definici kvaternionu zadáním osy a úhlu.



Obrázek 5.11: Scéna Kvaternion.

6. Testování a budoucí práce

V průběhu vývoje byl program pravidelně konzultován s pedagogy a s kolegy z řad studentů. Ranná verze programu byla dokonce použita vedoucím práce při výuce. Všechny aspekty programu po stránce funkcionální i stránce uživatelského rozhraní tak prošly proměnou a také vznikla celá řada nápadů a návrhů, které by výsledná aplikace mohla zahrnovat. Nejpodstatnější úpravou je vytvoření systému matematických operátorů, které nakonec tvoří základní princip celé aplikace.

Dokončený program včetně vytvořených výukových scén byl nakonec testován na úzké skupině tří studentů, jejichž znalosti problematiky transformací byly na velmi rozdílné úrovni. První student, který se v problematice orientuje velmi dobře, vyjádřil k aplikaci velice kladný postoj a ocenil i možnosti využití aplikace při samotné výuce. V průběhu testu také vzniklo několik návrhů, které jsou popsány dále v této kapitole. Druhý student, který s problematikou teprve začíná, vyjádřil o aplikaci hluboký zájem. Třetí student, který se v problematice transformací neorientoval vůbec, využil situace a nechal si problematiku vysvětlit. Program se ukázal jako opravdu neocenitelný nástroj při výkladu transformací.

Projekt by si jistě zasloužil testování ve větším měřítku. Možností by bylo nechat program vyzkoušet větší skupinou osob ve více kolech s průběžnými testy a závěrečným dotazníkem. Takové testování se ovšem nevešlo do časového období, které je na zhotovení diplomové práce k dispozici a jako takové by bylo nad rámec zadání práce. Bylo by také nutné mít použitelnou verzi programu dlouho dopředu. Práce na samotném programu však tvořila podstatnou část této práce. Další možností by bylo poskytnout program veřejnosti na internetu a potřebná data získat z reakcí a připomínek uživatelů. V tomto případě je však časová náročnost sběru dat ještě vyšší. Testování programu na internetu je součástí budoucí práce a věřím, že tak finální verze ještě projde mnohými úpravami.

Následující odstavce popisují úzký výběr úprav programu, které byly iniciovány na základě testování. První skupina obsahuje návrhy, které byly v programu již implementovány. Druhá skupina pak návrhy, které se staly součástí budoucí práce na projektu.

Návrhy, které jsou v programu již implementovány:

- Matematické operátory:
  Koncept matematických operátorů v omezené míře obsahoval již původní návrh programu. V průběhu vývoje se však stal zásadní součástí. Nyní lze jejich pomocí provádět základní matematické operace nejen s maticemi, ale i s vektory, kvaterniony a skaláry a program tak získal nebývalé možnosti. Operátory však zároveň vnáší do programu jistou dualitu. Hlavním konceptem původního návrhu byl graf scény tvořený krabičkami *sequence*. Po rozšíření je ovšem možné graf vytvořit i pomocí samotných matematických operátorů. Součástí budoucí práce se tak stává návrh řešení, ve kterém by byly oba koncepty sjednoceny.

- Výběr komponent:
  Možnost označit a tím vybrat skupinu komponent a provádět s ní operace jako s celkem. V programu je implementován výběr kliknutím a tahem.

Výběr pracuje ve dvou režimech. Tahem doleva jsou vybrány komponenty celé obsažené v regionu výběru. Tahem doprava jsou vybrány komponenty, které jsou v kolizi s regionem výběru. Jednotlivé komponenty lze také do výběru přidávat a odebírat kliknutím se stisknutou klávesou výběru. Vybranou skupinu komponent lze posouvat, smazat a kopírovat. Při vytvoření kopie výběru se kopírují i spoje, které spojují komponenty obsažené ve výběru. Implementace výběru značně zjednodušila manipulaci s komponentami a aranžování celé struktury komponent, které je klíčové pro přehlednost funkcionality scény.

- Barvy propojovacích komponent a spojů:
  Vstupy a výstupy, stejně jako čáry propojení, mají specifickou barvu v závislosti na typu přenášené informace. Různé barvy výrazně zvýšily přehlednost propojení krabiček.

- Editace hodnot kliknutím a tahem:
  Vstupní hodnoty v editačních polích lze editovat metodou kliknutím a tahem. Klasická metoda zadáním číselné hodnoty na klávesnici, která je v programu také implementována, nedovoluje pozorovat efekt postupné změny referenční hodnoty. Nyní lze myší hodnotu plynule ovládat a lépe tak pozorovat její vliv na výslednou transformaci.

- Náhled geometrického objektu v komponentě *sequence*:
  Vizuální propojení komponenty a geometrického objektu ve scéně, který je transformací komponenty transformován, se stalo předmětem mnoha diskuzí a vzniklo několik návrhů řešení. Návrhy zahrnovaly propojení objektu s komponentou křivkou stejným způsobem, jako je tomu u operátorů, a různé metody zvýrazňování objektu i komponenty výběrem myší. Propojení je nakonec vyřešeno vykreslením objektu do samotné komponenty. Toto řešení poskytuje informaci o propojení nezávisle na poloze kurzoru myši a nesnižuje přehlednost scény dodatečnými spoji. Umístění objektu v komponentě navíc naznačuje posloupnost násobení transformačních matic se souřadnicemi vrcholů geometrické reprezentace objektu.

- Výstup modelové matice komponenty *sequence*:
  Komponenta obsahuje výstup modelové matice, která vzniká násobením dílčích matic větve grafu scény. Modelovou transformaci tak lze použít na vstupu operátorů. Příkladem může být použití modelové matice k vytvoření pohledové transformace kamery a tímto způsobem navázat kameru na objekt ve scéně. Zároveň se zobecnila možnost tvorby grafu scény reprezentovaného orientovaným acyklickým grafem.

- Obrazovka jako operátor:
  V původním návrhu obrazovka tvořila nezávislou komponentu, která se automaticky vytvořila pro každou komponentu kamery ve scéně. Toto řešení ovšem postrádalo vizuální propojení obrazovky s komponentou kamery. Obrazovky také zastiňovaly globální náhled na scénu, zvláště když jich scéna obsahovala větší množství. Obrazovky jsou nyní součástí komponenty *space* a lze s nimi pracovat stejně jako s jinými operátory. Jako operátor navíc

poskytují výstup obsahující informaci o poměru stran, kterou lze využít při konstrukci projekčních transformací.

- Mód postupné animace transformace:
  Mód je popsán v kapitole 4. Vliv vytvářených transformací lze pozorovat na transformovaných geometrických objektech v globálním náhledu. Náhled ovšem poskytuje pouze informaci o výsledné transformaci, která může být vytvořena složením mnoha transformací dílčích. Mód postupné animace poskytuje možnost pozorovat jejich postupný vývoj a lépe tak pochopit problematiku spojenou s kombinací více transformací.

- Defaultní hodnoty volných matic:
  Při vytvoření si komponenta volné matice uloží své zadané hodnoty. Uživatel může měnit hodnoty v matici a studovat jejich vliv na výsledek. Nyní má možnost vrátit matici do původního tvaru před editací. Defaultní hodnoty lze také přenastavit dle aktuálního stavu nebo opětovným nastavením hodnot v dialogovém okně příslušné transformace.

- Cyklus:
  Specifický operátor umožnující vytvářet jednoduché animace popsaný v kapitole 4.

- Kopírování komponent:
  Komponenty lze kopírovat kliknutím se stisknutou klávesou kopírování. Tvorba scén je vzhledem ke konceptu programu náročná. Každou komponentu lze přidat pomocí vyskakující nabídky a možnost kopírovat existující komponenty proces tvorby značně urychluje.

- Popis scény a komponent:
  Scény a komponenty mohou obsahovat textový popis a je tak možné dát uživateli nápovědu a popsat jejich účel a funkci.

Návrhy, které jsou obsahem budoucí práce:

- Záznam scény:
  Možnost sejmutí obrazovky programu a záznamu videa, což by usnadnilo prezentaci vybraných problémů při výuce.

- Režimy obrazovky:
  Komponenta obrazovky by mohla zobrazovat scénu v různých módech odpovídajících funkcionalitě OpenGL. Například by bylo možné zapínat a vypínat test hloubky *GL_DEPTH_TEST* a test *GL_CULL_FACE* a nastavovat jeho módy *GL_FRONT* a *GL_BACK*. Dále by bylo možné přepínat mezi zobrazením bufferu barev a bufferu hloubky.

- Zvýraznění spojů:
  Při ukázání na komponentu by se její spoje kreslily tlustou čárou. Tímto způsobem by bylo možné u složitějších případů lépe pochopit strukturu zapojení komponent.

- Maticová sonda:
  Možnost zobrazení detailních informací o maticích transformací, které by zahrnovaly zhodnocení singularity, rigidnost, zachovávání velikosti apod.

- Cykly:
  Návrh vytvořit operátory cyklu pro ostatní proměnné. Fungovaly by podobně jako operátor cyklus pro *float*, ale pracovaly by s vektorem, kvaternionem a maticí.

- Automatické aranžování grafu komponent:
  Přidat možnost automatického rozložení komponent na pracovní ploše komponenty *space*. Vzhledem k rozdílnému tvaru a způsobu zapojení ovšem není možné použít některého z běžných algoritmů pro výpočet rozložení grafu.

- Parser kódu:
  Možnost popsat scénu skriptem v textovém editoru a ten pak importovat do programu. Vytvářet komplexní scény v programu může být poněkud pracné a určitě by se našel uživatel, který by tuto možnost uvítal. Bylo nutné implementovat i algoritmus pro automatické rozložení grafu komponent. V opačném případě by bylo nutné polohy krabiček definovat přímo v kódu, to by ovšem bylo velmi nepraktické.

- Generátor kódu scény:
  Možnost exportovat předpis scény a použitých příkazů knihovny GLM tak, aby ho šlo jednoduše použít k popisu scény v jiném projektu.

- Fokus mód komponenty *space*:
  Umožnit prohlížet a pracovat s komponentami grafu některou z technik nelineárního přiblížení jako je například rybí oko.

- Trackball operátor:
  Operátor vytvářející rotační matici na principu trackballu. Rotační matice lze editovat pouze editací číselných hodnot a trackball by poskytl intuitivnější možnost zadávání rotací.

- Operátor s informacemi o vstupu myši:
  Operátor, který by obsahoval výstupy s polohou a změnou polohy kurzoru myši.

- Paměti komponent:
  Komponenty by obsahovaly několik paměťových stavů. Do nich by šlo uložit a opět vyvolat stav příslušné komponenty.

- Slučování operátorů:
  Bylo by možné skupinu operátorů sloučit do jediné komponenty, která by obsahovala vstupy a výstupy směřující ven ze skupiny. Bylo by tak možné definovat komplexní funkce a zjednodušit tak práci s rozhraním. V některých případech by bylo možné zjednodušit a zpřehlednit komplexní scénu.

- Programovatelné operátory:
  Možnost vytvářet vlastní operátory popsané skriptem.

7. Diskuse

Koncept tvorby transformací a scény s použitím dvou typů prvků: stromových komponent *sequence* a operátorů obsahuje jistou dualitu. Neboť transformace i graf scény, který lze vytvořit pomocí komponent *sequence* a volných matic, lze vytvořit i pomocí matematických operátorů. Otázka tedy spočívá v tom, zda-li by nebylo možné oba systémy sjednotit a vytvořit tak přehlednější rozhraní. Z programátorského hlediska jsou všechny komponenty založeny na stejném principu a proto není problém strukturu komponent upravit a vytvořit tak vhodnější koncept celé aplikace. Jak by měl takový koncept vypadat je určitě vhodným tématem pro další diskusi.

Důležitým aspektem práce s aplikací je použitá knihovna uživatelského rozhraní. Knihovna je implementována v rámci projektu a je navržena konkrétně pro potřeby programu. Tvorba knihovny uživatelského rozhraní je ovšem náročný úkol a je diskutabilní, zda by nebylo vhodnější využít některé z existujících profesionálně vytvořených knihoven. Knihoven uživatelského rozhraní použitelných v prostředí knihovny OpenGL není mnoho. Jedná se buď o jednoduché knihovny, které neposkytují dostatečnou funkcionalitu pro potřeby aplikace nebo naopak o rozsáhlé knihovny, implementující i funkcionalitu, která je pro potřeby aplikace naprosto zbytečná. Čas potřebný k nastudování a použití těchto knihoven i k vytvoření obsahu, který knihovny explicitně neposkytují, je navíc srovnatelný s časem potřebným na vytvoření jednoduchého rozhraní, které je v projektu použito. Vytvořené rozhraní může v některých aspektech, jako například editace textu, působit poněkud těžkopádně.

Prostor pro vylepšení poskytuje i sada použitých geometrických objektů, které jsou použity k pozorování vlivu vytvářených transformací. V programu jsou použity jednotkové krychličky, které umožňují dobře pozorovat vliv posunu a škálování. Jejich jednotková velikost také zvyšuje chápání poměrů vzdáleností v prostoru scény. Nevýhodou je ovšem jejich symetrie podél všech tří souřadných os - lze těžko rozpoznat jejich orientaci, kterou lze interpretovat šesti různými způsoby. Tento problém je vyřešen textovým popisem jejich stran. Informace o orientaci je tak ale rozdělena do dvou typů vizuální informace. Lze diskutovat, zda by nebylo vhodnější použít jiné asymetrické geometrické těleso. Nevýhodou asymetrických těles je ovšem horší interpretace jejich polohy i velikosti a i chápaní orientace je také sporné, neboť je nutné znát použitou konvenci.

Prostor pro diskuzi vytváří otázka použití transformací geometrických objektu v programu. Běžným způsobem je ukládat a aktualizovat pozici a orientaci geometrického objektu kontinuálně v každém logickém kroku programu. Aplikace však v každém kroku stále popisuje informaci o celém procesu vzniku transformací z příslušných vstupních hodnot a neobsahuje mechanismus, který by dovoloval aktuální transformaci uložit a aktualizovat její stav inkrementálně, což je běžným způsobem při tvorbě grafických aplikací.

S příchodem nové generace grafických karet a programovatelných shaderovacích programů v moderních verzích knihovny OpenGL je nyní nutné, aby uživatel plně rozuměl posloupnosti a funkci projekční, pohledové a modelové transformace. Důležité funkce zobrazovacího řetězce, které jsou podstatné pro pochopení principu transformací v počítačové grafice, jsou však stále skryté v implicitní

funkcionalitě knihovny OpenGl. Jednou z nich je princip perspektivního dělení a druhou transformace viewportu. Program tyto mechanizmy vůbec nezahrnuje a je otázka, jakým způsobem by je bylo možné začlenit do současného konceptu.

Testovací scény byly součástí původního návrhu projektu. Program měl obsahovat sérii scén zadávajících uživateli úkol k vyřešení i možnost tyto scény vytvářet. Během vývoje se však výrazně rozrostly možnosti vytváření scén a s tím se zkomplikovala problematika vyhodnocování správnosti řešení testovací úlohy. Existuje totiž více způsobů, jakými lze v programu vyřešit možná zadání a není zřejmá metodika kontroly správného výsledku. Jednou z možností by bylo porovnávat tvary výsledných transformací s referenčními hodnotami. Správné transformace lze však dosáhnout i naprosto nevhodným způsobem. Druhou možností je vyhodnotit správnost výsledného grafu komponent. To může být vzhledem k počtu možností řešení, které je navíc závislé na dané úloze, velmi komplikované. Implementace konceptu testovacích úloh by zahrnovala tvorbu komplikované heuristiky zahrnující vlivy obou výše popsaných kontrolních mechanizmů a její principy jsou určitě předmětem diskuze.

8. Závěr

V rámci projektu byla navržena a implementována aplikace na výuku transformací v počítačové grafice. Program umožňuje vytvářet komplexní scény složené z geometrických objektů a kamer a rozličnými způsoby vytvářet a editovat použité modelové, projekční a pohledové transformace. Důraz je kladen na zobrazení tvarů parametrů a posloupností použitých transformačních matic, tak aby bylo možné vždy pozorovat a pochopit jejich transformační funkci na geometrické objekty ve scéně.

Program poskytuje nástroj umožňující studovat transformace individuální cestou, což je hlavním výukovým konceptem projektu. V rozhraní aplikace byla vytvořena série výukových scén zaměřujících se na vybrané úlohy transformací v počítačové grafice. Program lze úspěšně využít i při samotné výuce. Učitel může rozhraní spolu s poskytnutými výukovými scénami použít k demonstraci vykládané látky a v případě potřeby vytvářet vlastní scény i obsah programu.

V rámci projektu byl proveden výzkum dostupných zdrojů výuky geometrických transformací v počítačové grafice. Zkoumané zdroje zahrnují publikace, internetové tutoriály a počítačové programy. Textové zdroje neposkytují interaktivnost, která je při výuce transformací potřebná. Internetové tutoriály popisují problematiku jen velmi okrajově a pro studium jsou naprosto nedostačující. Dostupné programy, kterých není mnoho, nesplňují požadavky, které jsou kladeny na výukovou aplikaci v tomto projektu. Znalosti získané při výzkumu zdrojů však byly využity při návrhu výsledné výukové aplikace, která je výsledkem tohoto projektu. Aplikace odstraňuje nedostatky existujících výukových aplikací, které jsou jen úzce specializované na jeden typ transformací, jsou zastaralé nebo popisují problém pouze ve dvou dimenzích.

Program je implementován v knihovně OpenGL a koncepčně vychází z knihovny GLM. Program lze úspěšně využít při studiu i výuce transformací. Problematika transformací je ovšem velmi komplexní a k pochopení celé její matematické podstaty je stejně nutné nastudovat některou z dostupných publikací. Aplikace však může proces učení značně ulehčit.

Literatura

[Aca15] Khan Academy. *Khan Academy: Geometry transformations.* https://www.khanacademy.org/math/geometry/transformations/, 2015.

[Ahn13] Song Ho Ahn. *OpenGL transformation.* http://www.songho.ca/opengl/index.html, 2013. 9.12.2015.

[Ala13] Marco Alamia. *World, view and projection transformation matrices.* http://www.codinglabs.net/article_world_view_projection_matrix.aspx, 2013. 11.12.2015.

[CSE03] CSE. *CSE 457 - Lecture demos.* http://courses.cs.washington.edu/courses/cse457/03au/localdemos457.html, 2003. 9.12.2015.

[Dal12] Tom Dalling. *Modern OpenGL 03 - Matrices, depth buffering, animation.* http://www.tomdalling.com/blog/modern-opengl/03-matrices-depth-buffering-animation/, 2012. 13.12.2015.

[Fra13] PGR Framework. *PGR framework documentation.* https://cent.felk.cvut.cz/courses/PGR/framework/doc/index.html, 2013. 07.04.2016.

[Gro10] Joe Groff. *3D Transformation and projection.* http://duriansoftware.com/joe/An-intro-to-modern-OpenGL.-Chapter-3:-3D-transformation-and-projection.html, 2010. 13.12.2015.

[HC12] Chua Hock-Chuan. *3D Graphics with OpenGL.* https://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html, 2012. 13.12.2015.

[Jr.11] Richard S. Wright Jr. *OpenGL SuperBible: Comprehensive tutorial and reference.* Addison-Wesley Professional, fifth edition, 2011.

[Kin13] Jamie King. *Model view projection matrices.* https://www.youtube.com/watch?v=-tonZsbHty8#aid=P8Xs_WR4zIw, 2013. 13.12.2015.

[Len11] Eric Lengyel. *Mathematics for 3D game programming and computer graphics.* Cengage Learning PTR, third edition, 2011.

[Ol07] Petr Olšák. *Lineární algebra.* Petr Olšák, 2007.

[ot11] opengl tutorial.org. *OpenGL tutorial.* http://www.opengl-tutorial.org/, 2011. 11.12.2015.

[Ove15] Alexander Overvoorde. *OpenGL transformations.* https://open.gl/transformations, 2015. 11.12.2015.

[Par02] Ian Parberry. *3D Math primer for graphics and Game development.* Jones & Bartlett Learning, first edition, 2002.

[Pro13] Solarian Programmer. *OpenGL 101: Matrices - projection, view, model.* https://solarianprogrammer.com/2013/05/22/opengl-101-matrices-projection-view-model/, 2013. 11.12.2015.

[Pro15] Wolfram Demonstrations Project. *Wolfram demonstrations project.* http://demonstrations.wolfram.com/, 2015. 13.12.2015.

[Rat15] Egon Rath. *Basic 3D Math: Matrices.* http://www.matrix44.net/cms/notes/opengl-3d-graphics/basic-3d-math-matrices, 2015. 13.12.2015.

[SIG01] SIGGRAPH. *SIGGRAPH 2001 An interactive introduction to OpenGL programming.* https://www.opengl.org/archives/resources/code/samples/s2001/, 2001. 9.12.2015.

[vO11] Jeremiah van Oosten. *Understanding the view matrix.* http://www.3dgep.com/understanding-the-view-matrix/, 2011. 13.12.2015.

[Zah03] Miloš Zahradník, Luboš Motl. *Pěstujeme lineární algebru.* Karolinium, 2003.

Přílohy

A. Manuál k programu

Manuál je rozdělen do dvou částí. První vysvětluje ovládání programu a obsahuje popis obsažených komponent rozhraní. Druhá popisuje strukturu souborového systému a práci s ním.

A.1 Rozhraní a Ovládání

V celém rozhraní platí konvence: Kliknutím pravým tlačítkem myši na komponentu se rozbalí vyskakovací nabídka, která obsahuje dodatečné možnosti komponenty. Kliknutím levým tlačítkem myši do záhlaví komponenty a táhnutím lze s komponentou pohybovat (pokud to komponenta umožňuje).

Pracovní prostor je rozdělen na dvě hlavní části. Náhled a pracovní plochu. V náhledu jsou zobrazeny vytvořené geometrické objekty a kamery. Pracovní plocha slouží k editaci obsahu scény. Rozmístění komponent je vidět na obrázku A.1.

Obrázek A.1: Prostředí programu a popis základních komponent.

A.1.1 Náhled

Slouží k náhledu na geometrické objekty ve scéně. Jediným interaktivním prvkem je kamera, jejíž ovládání je popsané v tabulce A.1.

klávesa	funkce
pravé tlačítko myši	rotace kamery
prostřední tlačítko myši	posun kamery
scroll myši	přibližování / oddalování kamery

Tabulka A.1: Ovládání náhledu.

A.1.2 Pracovní plocha

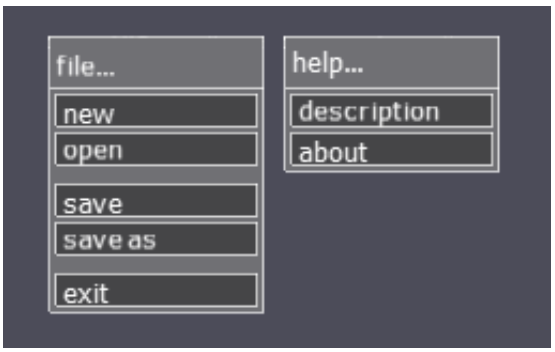
Pracovní plocha slouží k tvorbě, editaci a rozmísťování modulů reprezentujících reálná čísla, vektory, kvaterniony a transformační matice a operace a k jejich propojování do složitější struktury. Moduly nazýváme v textu zkráceně slovem krabičky. Ovládání je popsané v tabulce A.2. Lze ovládat posun a přiblížení celého pracovního prostoru a výběr krabiček. Výběr pracuje ve dvou režimech, lišících se směrem tažení myši se stisknutým levým tlačítkem myši. Při tažení doleva jsou po uvolnění myši vybrány krabičky, které zasahují nějakou svojí částí do výběrového regionu. Při tažení doprava jsou vybrány jen ty krabičky, které jsou ve výběrovém regionu obsaženy celé.

klávesa	funkce
levé tlačítko myši	region výběru
levé tlačítko myši (na krabičku) + levý shift	přidání / odebrání do výběru
levé tlačítko myši (na krabičku) + levý ctrl	vytvoření kopie krabičky
pravé tlačítko myši	nabídka přidání krabičky
prostřední tlačítko myši	posun
scroll myši	přibližování

Tabulka A.2: Ovládání pracovní plochy.

A.1.3 Hlavní menu

Obsahuje dvě nabídky *file* a *help* (obrázek A.2).



Obrázek A.2: Nabídky hlavního menu.

- *new*: Resetuje stav programu.
- *open*: Vyvolá dialogové okno načtení scény ze souboru.
- *save*: Uloží scénu do aktuálního souboru. Pokud zatím scéna do žádného souboru uložena nebyla, vyvolá dialogové okno uložení do souboru. Pokud aktuální soubor existuje, cesta k němu je uvedena v záhlaví hlavního okna aplikace.
- *save as*: Vyvolá dialogové okno uložení do souboru.

- *exit*: Ukončí aplikaci.
- *description*: Otevře okno s popisem scény.
- *about*: Otevře okno s verzí programu.

A.1.4 Popis scény

Okno obsahuje textový popis scény. Kliknutím pravým tlačítkem myši do textového pole lze vyvolat editační okno a zadat text. Pokud scéna obsahuje textový popis, okno popisu se po načtení scény objeví. Tlačítkem *ok* je okno zavřeno a lze jej znovu otevřít v hlavním menu.

A.1.5 Práce s krabičkami

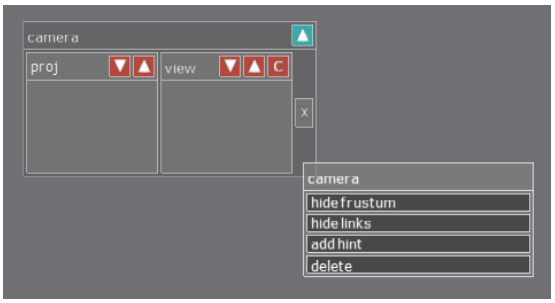
Krabičku lze přidat výběrem z nabídky přidání krabiček na pracovní ploše nebo vytvořením kopie existující krabičky. Se všemi lze pohybovat po pracovní ploše. Různé krabičky mají své vyskakující nabídky popsané níže. Popis funkcí jednotlivých krabiček je obsažen v kapitole Implementace 4.

Nabídka krabičky *sequence*

Obrázek A.3: Nabídka krabičky *sequence*.

- *bind object*: Z příslušné nabídky lze krabičce přidat geometrický objekt. Poté se funkce změní na *unbind object*, která objekt krabičce odebere.
- *add hint*: Přidá ikonu nápovědy do záhlaví krabičky. Kliknutím levým tlačítkem myši na ikonu je nápověda zobrazena. Pravým tlačítkem myši je vyvoláno editační okno, ve kterém lze zadat text nápovědy. V případě, že krabička již nápovědu obsahuje, se funkce změní na *remove hint*, která nápovědu odebere (dojde k jejímu úplnému smazání).
- *start tracking*: Přepne rozhraní do módu postupné animace transformací. Mód lze ovládat šipkami na klávesnici a ukončit klávesou *Esc*.
- *delete*: Smaže krabičku.

Nabídka krabičky *camera*



Obrázek A.4: Nabídka krabičky *camera*.

- *hide frustum* / *show frustum*: Schová / zobrazí vykreslení pohledového jehlanu v náhledu.
- *hide links* / *show links*: Schová / zobrazí spoje s kořenovými krabičkami *sequence*.
- *hide links* / *show links*: Schová / zobrazí spoje s kořenovými krabičkami *sequence*.
- *add hint* / *remove hint*: viz nabídka krabičky *sequence*.
- *delete*: Smaže krabičku.

Nabídka volné matice

Obrázek A.5: Nabídka volné matice.

- *edit*: Otevře dialogové okno. Lze tak zadat nové defaultní hodnoty.
- *set as default*: Nastaví aktuální hodnoty jako defaultní.
- *reset*: Nastaví hodnoty na defaultní.
- *unlock* / *lock*: Odemkne / zamkne ostatní hodnoty k editaci.

- *disable / enable synergies*: Zakáže / povolí závislosti mezi hodnotami při editaci.
- *add hint / remove hint*: viz nabídka krabičky *sequence*.
- *delete*: Smaže krabičku.

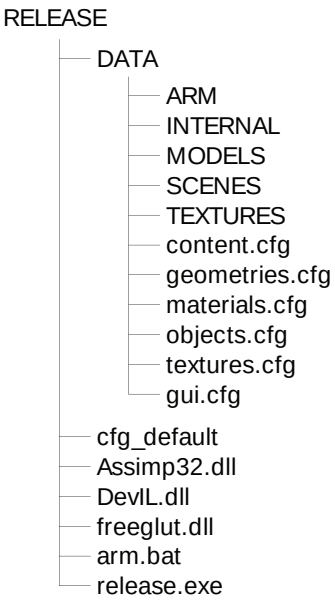
Matematické operátory mají také svou nabídku, která obsahuje *add hint / remove hint* a *delete*.

Propojování krabiček

Krabičky se propojují kliknutím levým tlačítkem myši na vstup / výstup, přesunutím kursoru myši na cílený vstup / výstup a následným uvolněním tlačítka myši. Rozpojit krabičky lze kliknutím na vstup, který chceme odpojit a následným tahem do prázdného prostoru.

A.2 Souborový systém

Na obrázku A.6 je vidět struktura souborů programu. Pro správný chod aplikace je nutné zachovat: Obsah a umístění složky *internal*, dynamické knihovny (*.dll) a soubor *cfg_ default*, který ale může být podle potřeby modifikován. Program lze spustit souborem *release.exe*. Struktura a názvy ostatních složek a souborů jsou definovány v konfiguračních souborech a lze je v případě potřeby změnit. Vytvořené scény jsou uložené ve složce *scenes*.



Obrázek A.6: Soubory programu.

A.2.1 Konfigurační soubory

Všechny konfigurační soubory jsou textové a všechny dostupné parametry jsou popsány v jednotlivých souborech. Všechny cesty k souborům definované v konfiguračních souborech musí být relativní ke kořenové složce programu.

- *cfg_ default*: Hlavní konfigurační soubor. Definuje základní parametry programu jako například velikost okna aplikace. Program může být spuštěn s parametrem cesty k jinému konfiguračnímu souboru. V opačném případě je použit soubor *cfg_ default*. Dále definuje cestu ke konfiguračním souborům *content.cfg* a *gui.cfg*.
- *gui.cfg*: Definuje grafické nastavení uživatelského rozhraní.
- *content.cfg*: Definuje cesty k souborům *geometries.cfg*, *materials.cfg*, *objects.cfg*, *textures.cfg*.
- *geometries.cfg*: Definuje množinu použitých modelů ve formátu (*.obj).
- *materials.cfg*: Definuje množinu dostupných materiálů.
- *textures.cfg*: Definuje množinu dostupných textur.
- *objects.cfg*: Definuje množinu dostupných geometrických objektů a jejich strukturu. Geometrický objekt je v souboru definován názvem, modelem (*.obj), materiálem a texturou. Struktura v souboru definuje strukturu nabídky geometrických objektů v nabídce krabičky *sequence*.

A.2.2 Mód Arm

Scéna Arm (kapitola 5.7) přidává do základního programu nový obsah, který je celý uložen ve složce *arm*. Složka obsahuje modely ve formátu (*.obj), modifikovaný hlavní konfigurační soubor *cfg_ arm* a modifikované konfigurační soubory *armContent.cfg*, *armGeometries.cfg*, *armMaterials.cfg* a *armObjects.cfg*. Hlavní konfigurační soubor je modifikován následujícím způsobem:

```
SOUBOR: cfg_arm

...

// content
CONTENT_FILE   data/arm/armContent.cfg
LOAD_SCENE     data/arm/scene/arm.scn

...
```

Parametr *CONTENT_ FILE* definuje cestu k novému konfiguračnímu *armContent.cfg*. Nepovinný parametr *LOAD_ SCENE* definuje cestu k souboru scény, která se má otevřít ihned po spuštění aplikace. Soubor *armContent.cfg* je definován následovně:

```
SOUBOR: armContent.cfg

// textures
TEXTURES_CFG    data/textures.cfg

// materials
```

```
MATERIALS_CFG   data/materials.cfg
MATERIALS_CFG   data/arm/armMaterials.cfg

// geometries
GEOMETRIES_CFG  data/geometries.cfg
GEOMETRIES_CFG  data/arm/armGeometries.cfg

// objects
OBJECTS_CFG     data/objects.cfg
OBJECTS_CFG     data/arm/armObjects.cfg
```

Jsou přidány nové cesty ke konfiguračním souborům materiálů, geometrií a objektů. Použitých konfiguračních souborů jednoho typu může tedy být i víc. Mód Arm tak zachovává i základní obsah programu.

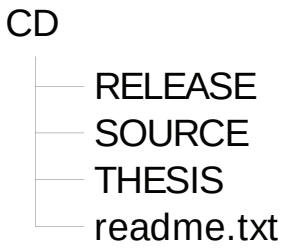
Spustit program v módu Arm je možné spuštěním souboru *release.exe* s parametrem cesty ke konfiguračnímu souboru *cfg_ arm*. K tomu je možné použít přiložený dávkový soubor *arm.bat*.

A.3 Instalace

Program není potřeba instalovat. Stačí zkopírovat obsah složky *release*. Pokud je požadován i arm, pak je nutné okopírovat i soubor (arm.bat) a podadresář *arm*.

B. Obsah přiloženého CD

Přiložené CD obsahuje složku *release* s kompilovanou verzí programu (obrázek A.6). Obsahem složky jsou i vytvořené výukové scény. Dále složku *source*, která obsahuje zdrojové soubory projektu a složku *thesis*, která obsahuje pdf soubor s textem této práce. Podrobný popis obsahu je v textovém souboru *readme.txt*.



Obrázek B.1: Soubory na přiloženém CD.