VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

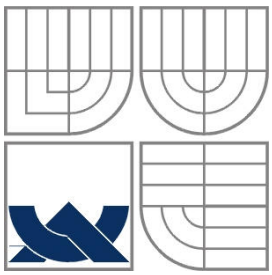
POKROČILÝ EDITOR 3D SCÉNY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

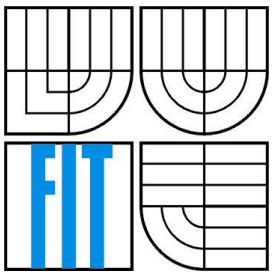
AUTOR PRÁCE
AUTHOR

Bc. MARTIN BIŠOF

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POKROČILÝ EDITOR 3D SCÉNY

ADVANCED EDITOR OF 3D SCENE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN BIŠOF

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV PŘIBYL

BRNO 2009

Abstrakt

Tato práce popisuje důležité etapy při vývoji aplikace na editaci 3D scény. Jde o nástroj, který umožňuje vizualizaci objektů a modelování grafu scény. Úvod dokumentu je věnován teoretickému rozboru, který se zabývá nejprve reprezentací objektů v počítačové grafice s bližším zaměřením na reprezentaci hraniční. Následuje teoretická kapitola transformací a manipulátorů. Návrhová část představí základní problematiku a obeznámí nás s řešením úloh na obecné úrovni. V části implementace je pak detailně popsáno konkrétní řešení jednotlivých etap návrhu. Jedná se o popis pokročilého systému kamer a transformací objektů pomocí grafických manipulátorů. Dále je pojednáno o implementovaných efektech, modifikátorech a o řešení grafického uživatelského rozhraní.

Abstract

This work describes important periods of making application on editing 3D scene. It concerns a program which makes possible visualization of objects and simulation graphic scene. The introduction of this document is dedicated to theorical analysis, which is oriented on objects presentation in computer graphic with detailed orientation on Boundary Representation. The following chapter is about transformations and manipulators. Design part introduces basic problems and apprises with exercise solutions on general level. In the part of the implementation, we find detailed description of actual solution for particular phase of proposal. It states description of advanced camera system and transformation of objects by the help of graphic manipulators. The next chapter discuss implement effects, modifications and solution for graphical user interface.

Klíčová slova

Grafický editor, model, objekt, graf scény, hraniční reprezentace, openscenegraph, transformace, manipulátor, kamera, selekce, grafické uživatelské rozhraní, modifikátor

Keywords

Graphics editor, model, object, scene graph, boundary representation, openscenegraph, transformation, manipulator, kamera, selection, graphical user interface, modifier.

Citace

Bišof Martin: Pokročilý editor 3D scény. Brno, 2009, diplomová práce, FIT VUT v Brně.

Pokročilý editor 3D scény

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jaroslava Přibyla. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Bišof
12.5.2009

Poděkování

Rád bych poděkoval vedoucímu své diplomové práce Ing. Jaroslavu Přibylovi za odbornou pomoc a cenné rady při řešení tohoto projektu.

Obsah

1 Úvod

Problematika editorů virtuálních trojrozměrných scén je řešena již dlouhou řadu let. S rostoucím vývojem počítačové technologie postupovala mílovými kroky v různých formách i počítačová grafika. Konstruktéři i designéři očekávali již od osmdesátých let tvůrčí nástroje k realistické počítačové simulaci, která by byla schopna reflektovat jak realitu, tak jejich osobní vize. Motorem pro rozvoj takových nástrojů byl jistě i marketing, který stojí u zrodu prvních komerčních aplikací tohoto typu. Ty dosahují na tehdejší dobu něčeho nevídaného. Na filmových plátnech, v počítačových hrách či televizních reklamách se objevují propracované modely a realistické efekty. Konečně mají tvůrci k dispozici nástroje k realizaci scén, které by byly běžným způsobem nedosažitelné. Kdybychom se měli vrátit k úplným počátkům počítačových modelů a efektů ve filmu, jmenovali bychom například snímek *Alien*, ve kterém dochází již v roce 1979 k počítačové vizualizaci některých scén podle výtvarníka *H.R.Gigera*.

Vývoj aplikací, které byly schopny zobrazit a modelovat trojrozměrnou scénu se odvíjel v zásadě dvěma směry. Významnou úlohu řešily aplikace zaměřené na konstruktérství (tzv. *CAD* systémy). Ty jsou schopny vytvářet dostatečně přesný popis toho, co a jak se má vyrábět nebo stavět v podobě digitální výkresové dokumentace. V důsledku toho se odvíjí i další významná odvětví, jako je například *CAM*[1] či *FEM*[2]. Dohromady pak dokáží vytvořit komplexní systémy, jež se využívají v oborech, jako je stavitelství, strojírenství, elektrotechnika či například medicína. Druhý směr není v principu příliš odlišný, svým zaměřením se ale dotýká spíše výtvarného umění a zábavní tématiky. Jde o grafické editory, které se využívají především k realistické vizualizaci, animaci či modelování složitých a tvarově náročných těles v prostoru. Jako příklad jmenujme známý nástroj *Autodesk 3ds max* či volně dostupný projekt *Blender*. Objevují se ale i aplikace, které se zaměřují striktně na některá témata, jako například tvorba krajiny a vegetace (*Bryce3D*) nebo animace postav a zvířat (*Poser*).

Vizualizace a design hraje v dnešní době velmi důležitou roli, a proto budou tyto tvůrčí aplikace stále více využívány na poli filmu, herního průmyslu, užitého umění či reklamy. Prostředky vizualizace se stále zdokonalují a s nimi i nástroje k jejich dosažení. Cílem této práce není vyrovnat se svým produktem profesionálním komerčním nástrojům, na jejichž vývoji pracují až stovky zaměstnanců. Hlavním účelem je implementovat vlastní systém, který mnohé typické úlohy řeší vlastní koncepcí. Aplikace má připravit půdu pro implementaci nových nástrojů, které by usnadnily nebo zrychlily některé etapy vizualizace či modelování objektů. V základní fázi by měl program nabídnout především propracovaný systém kamer a moderní návrh grafického uživatelského rozhraní. Je důležité zaměřit se také na řešení obecných problémů, jako je transformace objektů či práce s širokým spektrem souborových formátů. S transformacemi souvisí i významná tématika grafických manipulátorů. Právě ty mohou být nositeli nových možností, jak zefektivnit práci ve vektorovém editoru.

---

[1] CAM (Computer Aided Manufacturing) – Systém přímo generující řídící kód pro obráběcí stroje, CNC.

[2] FEM (Finite Element Method) – Systém pro řešení deformace, napětí, proudění, elasticity materiálu apod.

V rozsahu diplomové práce se jistě nepodaří veškeré vize naplnit. Výstupem praktické části však bude systém, který nemalou část problematiky řeší a především připravuje svou koncepcí půdu pro budoucí realizaci pokročilých nástrojů.

Dokument je strukturován do několika částí. Úvod je věnován teoretickému rozboru, kde je stručně pojednáno o možnostech reprezentace objektů v počítačové grafice, transformacích v prostoru a sytému pro manipulaci s objekty. Tato část byla předmětem semestrálního projektu. Navazuje etapa návrhu, ve které seznámím čtenáře se základní problematikou a zaměřím se na některé typické úlohy vektorových editorů. Nejvíce prostoru je věnováno implementační části, která detailně popisuje konkrétní řešení jednotlivých fází realizace. Při výkladu jsem se snažil text obohatit vlastními ilustracemi, které řešenou problematiku pomohou objasnit. Dosažené výsledky a schopnosti aplikace jsou shrnuty v kapitole 5. Dokument obsahuje i několik příloh. Jedná se o ukázku variability implementovaného prostředí, uživatelský manuál aplikace a řadu demonstračních příkladů. Součástí zadání bylo také vytvořit prezentační plakát a webové stránky, kde bude projekt vystaven pod *open-source* licencí.

2 Teoretický rozbor

Na úvod technické zprávy je pojednáno o teoretických faktech, které přímo nebo i nepřímo souvisí s praktickou částí diplomové práce. Často se přitom opírám o literaturu [Zar98].

Teoretický rozbor je rozdělen do třech základních kapitol. První část pojednává o reprezentaci prostorových objektů v počítačové grafice. Důraz je přitom kladen na metody relevantní pro realizaci implementační části. Navazující kapitolou jsou transformace, kde je vysvětlena matematická podstata dvojrozměrné translace, rotace či změny měřítka a jejich analogie ve 3D. Kromě toho je pojednáno o projekčních transformacích, které jsou pro implementaci také významné. Teoretický rozbor pak uzavírá kapitola, která na transformace přímo navazuje. Jedná se o grafické manipulátory a jejich obecné možnosti.

2.1 3D reprezentace

Úvodní část teoretického rozboru stručně popisuje problematiku reprezentace prostorových objektů v počítačové grafice. Jedná se o několik způsobů, jež se za dobu své působnosti ujaly a do jisté míry mají své zastoupení v různých odvětvích lidské tvorby. Monopolní zastoupení má v dnešní době pravděpodobně reprezentace hraniční (tzv. *B-rep*). Jak již z názvu vyplývá, objekt je zde definován svými hranicemi. Jedná se o reprezentaci pomocí vrcholů, křivek či například množiny trojúhelníků. Za klady můžeme považovat úspornost modelu a možnost modelovat prakticky jakýkoli objekt a tvar. Nevýhodou je naopak pomíjení historie a způsobu vzniku objektu.

Opakem hraniční reprezentace je, z hlediska způsobu vzniku objektu, konstruktivní geometrie. Konstruktivní geometrie těles[3] se zakládá na reprezentaci objektu stromovou strukturou, kde jsou zaznamenány jednotlivé konstrukční kroky. Jedná se o vytváření modelu z jednoduchých geometrických primitiv za pomoci množinových operací a prostorových transformací. Za primitiva můžeme označit kouli, kvádr, válec, kužel, jehlan či toroid. Množinové operace potom zastupuje sjednocení, průnik nebo rozdíl. Nevýhodou může být právě omezená množina přesně popsatelných primitiv a operací nad nimi. Další komplikací může být náročnost na obnovení stromu po každé změně grafu v případě rozsáhlých a komplexních scén. Význam má tato metoda především pro konstruktérské a stavařské obory. Často se potom kvůli zobrazení takto vzniklá geometrie převádí na hraniční reprezentaci.

Za další tématickou oblast můžeme považovat objemovou reprezentaci. Využívá se především v případech, kdy nemáme k dispozici geometrický popis tělesa, ale pouze sadu vzorků v určitém místě povrchu nebo objemu. Jinde může být sada strukturovaná do podoby pravidelných či nepravidelných mřížek. Důležitá je pak opět metodika pro převod do hraniční reprezentace.

V neposlední řadě se můžeme setkat s pojmem procedurální modelování. To je založeno na základě generování objektů podle definovaného algoritmu. Jde o generování ploch, křivek

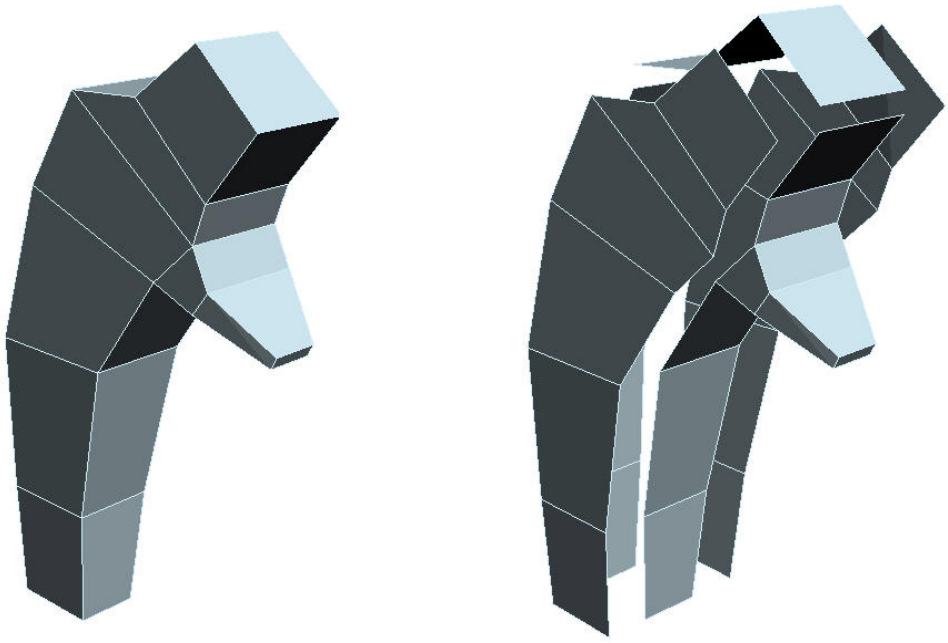
[3] Konstruktivní geometrie těles - CSG z ang. Constructive Solid Geometry

nebo vytváření tvarů pomocí šablon. Jedná se také o metody automatického generování objektů, které se vizuálně či chováním podobají objektům, s nimiž se setkáváme například v přírodě.

Ve stručnosti bych se zmínil o pojmu implicitní plochy. Podstata tkví v modelování objektů pomocí kostry, která je jakoby obalena hmotou podle intenzity pole v každém bodě. Pro zobrazení se pak model převádí například opět na hraniční reprezentaci. Dále se budu výhradně zabývat reprezentací spojenou s praktickou částí této práce, tj. hraniční reprezentací.

2.1.1 B-rep

Teoretickým základem pro hraniční reprezentaci je Eulerova rovnost (viz kap. 2.1.3). Nesporné přednosti metody se týkají především úspory a jednoduchosti reprezentace. Výhody se nabízí ale i z hlediska dalšího zpracování či archivace geometrie. Největším přínosem je však tvarová volnost, kdy hranicí může být například i křivka či plocha NURBS[4]. Zobrazování hraniční reprezentace je pak snadno proveditelné v grafických procesorech.



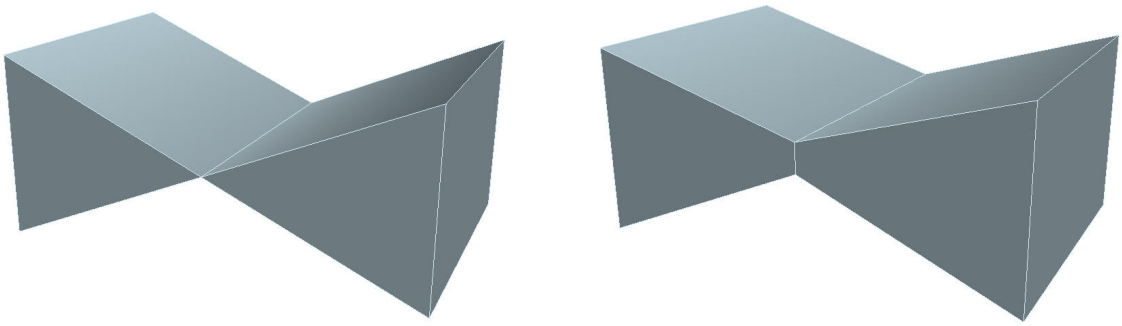
Obrázek 2.1: *Hraniční reprezentace*

Metodu demonstruji na jednoduchém polygonálním modelu (obr.2.1). Model vyobrazený vlevo působí celistvě. Pro názornost jsem pak u tohoto tělesa posunul některé jeho hranice (vpravo). Zjišťujeme, že informace o vnitřních bodech tělesa se neuchovávají (lze je popřípadě odvodit z popisu hranice) a celý model je tedy tvořen hranicemi v podobě křivek a ploch. Tato filosofie však může vést k nechtěným artefaktům – vzniku tzv. nonmanifoldních objektů.

[4] NURBS z ang. non uniform rational B-spline popsáno v [www5]

2.1.2 Manifold objekty

Můžeme říci, že metoda definice tělesa, jak jsme si ji popsali výše, nás zcela jistě neuchrání před možností vymodelovat a popsat objekt, který by nebylo v reálném světě možné vytvořit. Skutečnost, že počítačový popis může určovat i nevyrobitelné těleso, pramení z použití matematické a geometrické abstrakce. Pro názornost uvedu příklad nonmanifoldního (nevyrobitelného) a manifoldního (vyrobitelného) objektu.



Obrázek 2.2: *Nonmanifold a manifold objekt*

Sestrojit v reálném světě těleso, jehož části se budou např. dotýkat pouze v jednom vrcholu, je podobná utopie, jako vyrobit absolutně rovnou plochu. Těchto možností je ale v počítačové grafice nespočet. Na obrázku 2.2 vlevo je příklad nonmanifoldního objektu. Dvě části tělesa se stýkají v místech nekonečně tenké hrany, která inciduje se čtyřmi plochami. Naopak vpravo vidíme objekt, jehož obdobu bude možné reálně zkonstruovat. Podle [zar98] můžeme za manifold z praktického hlediska považovat takové těleso, jehož každá z hran inciduje právě se dvěmi ploškami a jehož hrany neprotínají žádné jiné plochy. Obdobně pak platí, že jeden vrchol nespojuje dvě a více částí tělesa.

2.1.3 Eulerova rovnost

V nemalém zastoupení používaných těles v trojrozměrné grafice jsou mnohostěny[5]. Pro mnohostěn musí platit, že každou z hran sdílí vždy sudý počet stěn. Mnohostěn, který neobsahuje díry a deformací jej lze převést na kouli, se nazývá jednoduchý mnohostěn. Eulerova rovnost zajišťuje, že množina vrcholů, stěn a hran tvoří jednoduchý mnohostěn, jestliže platí, že každá hrana propojuje právě dva vrcholy a stěny se navzájem neprotínají.

$$F + V - E = 2 \tag{2.1}$$

Vzorec udává vztah mezi počtem stěn (F), vrcholů (V) a hran(E) pro jednoduchý mnohostěn. Pro mnohostěny, které jsou prolnuty otvory, platí obecnější Eulerova rovnost, kde jsou přidány další operátory.

[5] Mnohostěn - těleso ohraničené množinou mnohoúhelníků

$$F + V = E + 2\cdot(C - H) + R..$$

(2.2)

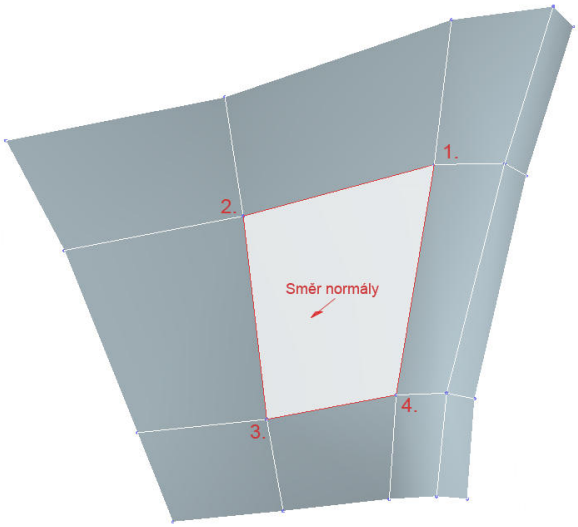
Člen *R* označuje počet vnitřních smyček hran, C počet oblastí nebo samostatných komponent tělesa a H počet otvorů v tělese.

2.1.4 Reprezentace pomocí hran a vrcholů

Existuje několik možností, jak reprezentovat objekt pomocí prvků, jako jsou výše zmíněné vrcholy, hrany či plochy. Nejjednodušší metoda popisu tělesa vychází pouze z prvních dvou uvedených, tedy z vrcholů a hran. Je předem jisté, že takový objekt nebude pokryt žádným pláštěm. Vytvoříme tím efekt tzv. drátový model, u kterého se však může vytvořit problém dvojí interpretace. Ke každé hraně modelu náleží vždy dva vrcholy, jejichž souřadnice jsou vždy striktně definovány. Toto je velmi jednoduchá struktura zabírající minimum paměti.

2.1.5 Reprezentace pomocí triviálních ploch

Drátový model má jistě celou řadu uplatnění, avšak pro realističtější zobrazení je potřeba podat o objektu více informací. Toho lze docílit pokrytím kostry ploškami, které budou simulovat povrch tělesa (tzv. pláty). Plochy mohou být uspořádány do sítě trojúhelníků nebo polygonů. Důležité je však uchovávat vždy informace o počtu a pořadí vrcholů, které plocha sdílí.

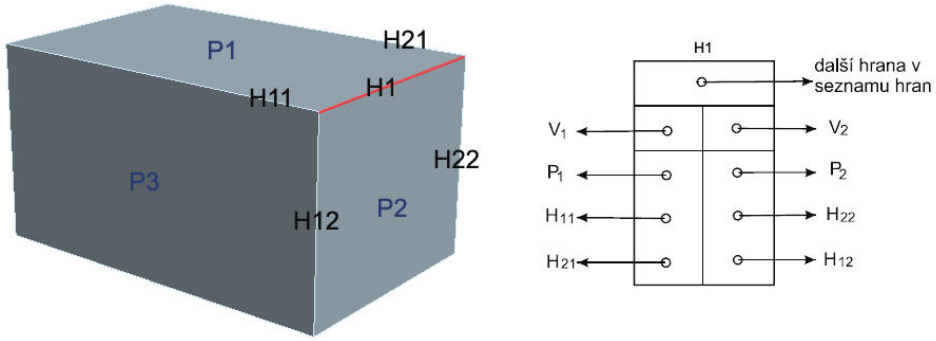


Obrázek 2.3: *Pořadí vrcholů při vytváření plochy*

Ilustrace 2.3 znázorňuje, že je relevantní, s jakým pořadím vrcholů bude plocha vložena. Závisí na tom směr normály, který určuje, bude-li plocha viditelná zvenku či zevnitř objektu. Tímto způsobem tedy zajistíme, že objekt bude viditelný od pozorovatele. Metoda triviálních ploch už prozrazuje o tělese více, není to však stále komplexní přístup, který by říkal například něco o vzájemném sousedství jednotlivých ploch.

2.1.6 Okřídlená hrana

Za opravdu strukturovanou reprezentací, složenou z vrcholů, hran a ploch, lze označit metodu navrženou *B.G. Baumgartem*. Metoda vytváří takovou strukturu, která je vhodná pro popis polygonálního modelu a je určena v základu pouze pro manifold objekty.



Obrázek 2.4: *Okřídlená hrana*

Název si vysloužila díky podobnosti hrany, sousedící vždy se dvěma ploškami. Každá hrana nese informace o sousedních elementech. Konkrétně ukazuje vždy na koncové vrcholy (V1, V2), sousední plochy (P1, P2) a nese informace o dalších čtyřech hranách (H11,H21,H12, H22). V levé polovině se nacházejí hrany sousedící s levou plochou (H11, H21), v pravé související s plochou pravou (H12, H22). Ukazatel *p* má pak hodnotu adresy další hrany ve zřetězeném seznamu. Není pak složité z takovéto struktury určit důležité informace o topologii, jako například všechny sousedící plochy s danou plochou, plošky incidující s danou hranou, vrcholy a hrany dané stěny či plochy stýkající se v daném vrcholu.

Pomocí podobné struktury lze reprezentovat i nonmanifold objekty. Označuje se jako půlhrana a smysl této metody tkví v myšlence rozdělení hrany tělesa na skupiny půlhran spojujících vždy tutéž dvojici vrcholů. Taková půlhrana pak představuje dvojici stěna-hrana.

2.1.7 Reprezentace pomocí bodů

Jak již z názvu vyplývá, půjde o reprezentaci pouze pomocí povrchových bodů. Taková množina bodů může být získána například digitálním snímáním objektu či příslušným algoritmem. Tyto body ve své struktuře nenesou pouze informaci o poloze, ale i směru normálového vektoru, barvě a dalších parametrů týkajících se materiálu tělesa. Taková reprezentace je určena třeba k archivačním účelům reálných architektonických památek, kde je kladen důraz na zachování přesných dat. Paměťové nároky budou pak u detailních objektů mimořádně vysoké.

Při zobrazování této reprezentace však můžeme narazit na problém, že mohou vznikat nechtěné nespojitosti. U modelu složeného z trojúhelníků umíme přesně určit hranice každého z nich, u množiny izolovaných bodů toto nelze. Podle lokální hustoty bodů jsme však schopni určit, jak by měly být jednotlivé body velké, aby se navzájem překrývaly. Metoda není vhodná pro ploché a jednoduché objekty a je nepraktická k aplikaci na ostré hrany. Byly proto vyvinuty hybridní techniky, které kombinují vlastnosti bodů s vlastnostmi trojúhelníků.

2.2 Transformace

Z hlediska tvorby vektorového trojrozměrného editoru je pro nás elementárně významná oblast transformací. Z principu je potřeba odpovědět na otázky, jak budou modely ve scéně umístěny, jakým způsobem budou natočeny, v jakém měřítku se na každý z nich budeme dívat. Transformace můžeme obecně rozdělit do tří skupin.

- Lineární transformace,
- Nelineární transformace,
- Projekční transformace.

První skupinu tvoří lineární transformace. Počítáme do ní například posunutí, rotace a změnu měřítka. Složité operace týkající se deformací a wrapingu označujeme jako nelineární. Zvláštním případem je potom transformace projekční, která se týká převodu trojrozměrných scén na úroveň dvojrozměrného obrazu. Objekty jsou popsány svými souřadnicemi, které jsou vztaženy ke zvolenému souřadnicovému systému. Transformace pak můžeme použít přímo na jednotlivé souřadnice. Objekt potom bude vzhledem k souřadnicovému systému měnit svou polohu. Jinou možností je aplikovat transformace na celý souřadnicový systém, což může být výhodné pro další zpracování objektu, jako je například výpočet objemu apod.

Lineární transformace aplikujeme na všechny body objektu. Bod objektu *P* má v kartézské soustavě pro tři rozměry souřadnice [x, y, z], pro dva rozměry souřadnice [x, y] . Potom transformací tohoto bodu získáme nově bod *P'* o souřadnicích [x', y', z'], obdobně pro dva rozměry [x', y'].

2.2.1 Homogenní souřadnice

Pro práci s transformacemi je výhodný převod na nějakou jednoduchou strukturu, ve které pak nebude problém skládat jednotlivé transformace a dále s nimi pracovat. Homogenní souřadnice jsou využívány tak, aby umožnily vyjádření lineárních transformací ve tvaru jedné matice, což v nehomogenních kartézských souřadnicích není možné.

Homogenní souřadnice bodu P s kartézskými souřadnicemi [x, y, z] zapíšeme ve tvaru [x, y, z, w], kde hodnota *w* představuje váhu bodu a velmi často se volí s hodnotou jedna. Ostatní souřadnice bodu se potom vyjádří následovně:

$$x = \frac{x}{w}, y = \frac{y}{w}, z = \frac{z}{w}, w \neq 0.$$

Ze zápisu vyplývá, že budeme-li mít bod s parametry souřadnic [2,4,6,6] a druhý bod se souřadnicemi [4,8,12,12], půjde stále o ten stejný bod.

Jestliže potom bod *P* s homogenními souřadnicemi [x, y, z, w] transformujeme, dostaneme nový bod *P'* s hodnotami souřadnic [x', y', z', w']. Lineární transformaci bodu *P* budeme reprezentovat pomocí matice *M* o velikosti 4x4 pro tři rozměry. Pro dvourozměrné transformace se potom využívá matice 3x3.

2.2.2 Dvourozměrné transformace

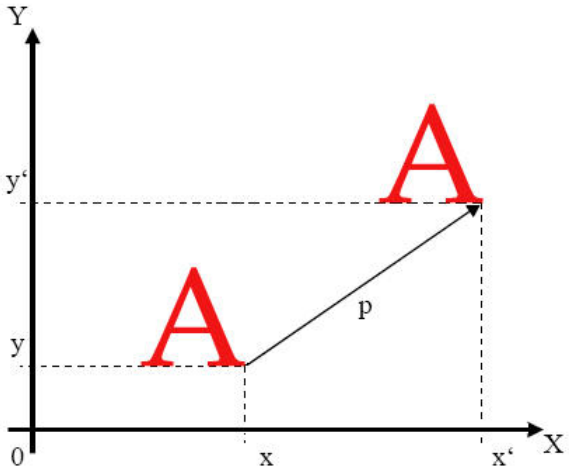
Po uvážení výše zmíněného textu a algebraických pravidel můžeme napsat odpovídající rovnost matic ve tvaru:

$$P' = M \cdot P = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix}. \tag{2.3}$$

Je předem zřejmé, že pro případy trojrozměrných transformací budeme používat obdobný zápis, který bude pouze jakýmsi zobecněním. Nejprve si pro jednoduchost představíme transformace bodu pro dva rozměry a uvedeme si typické případy lineárních transformací, jež se v počítačové grafice používají.

2.2.3 Translace ve 2D

Posun dvourozměrného bodu určíme vektorem posunutí $\vec{p} = (x_t, y_t) = (x' - x, y' - y)$. Matice transformace posunu T má tvar $T(x_t, y_t) = \begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix}$ a inverzní matice k matici T bude ve tvaru $T^{-1}(x_t, y_t) = T(-x_t, -y_t) = \begin{bmatrix} 1 & 0 & -x_t \\ 0 & 1 & -y_t \\ 0 & 0 & 1 \end{bmatrix}$. Případ nazvaný posunutí nebo také translace ve 2D situuje následující obrázek.

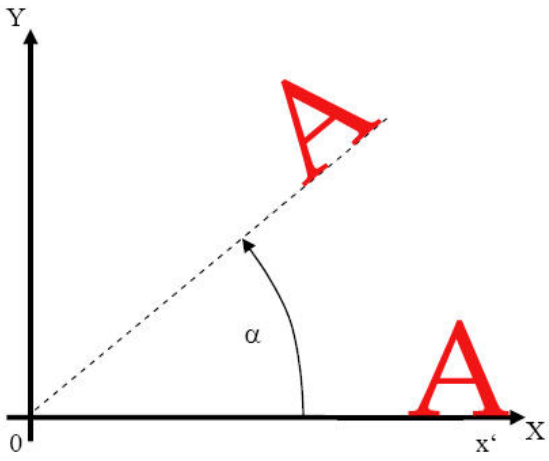


Obrázek 2.6: *Translace*

Po transformaci jsou nově získané souřadnice bodu P' $x' = x + x_t$, $y' = y + y_t$.

2.2.4 Rotace ve 2D

Transformace rotace je označení pro otočení bodu P kolem počátku soustavy souřadnic O=[0,0] o úhel α.



Obrázek 2.7: *Rotace*

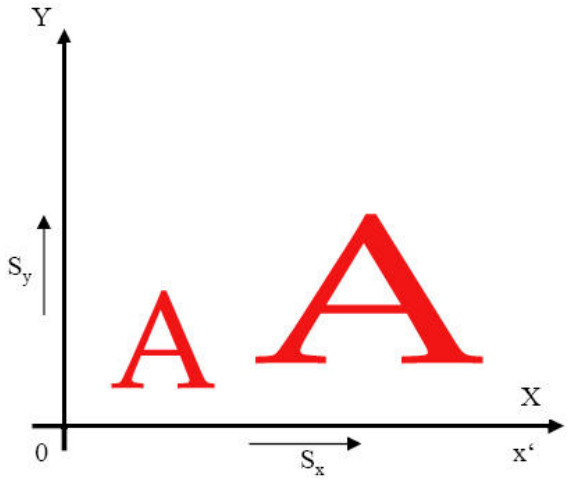
Matice transformace rotace R a její inverzní matice R^{-1} jsou ve tvaru

$$R(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; R^{-1}(\alpha) = R(-\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Po transformaci jsou nově získané souřadnice bodu P' $x' = x \cdot \cos\alpha - y \cdot \sin\alpha$, $y' = x \cdot \sin\alpha + y \cdot \cos\alpha$.

2.2.5 Změna měřítka ve 2D

První dva případy transformací mají relativně jednoduchý charakter v tom směru, že ovlivňují pouze jednu preferovanou veličinu. Naproti tomu, zamyslíme-li se například nad potřebou zvětšovat či zmenšovat objekt, měli bychom vzít v úvahu, že budeme současně ovlivňovat polohu i velikost objektu ve směru souřadnicových os. Koeficient určující míru zvětšení nebo zmenšení je pro horizontální osu S_x a pro vertikální S_y. V intervalu od nuly do jedné dochází ke zmenšování a posunu objektu směrem k počátku souřadnicových os. Naopak hodnoty větší jak jedna objekt zvětší a od počátku se oddálí. Záporné hodnoty budou pak měnit velikost v záporném směru.



Obrázek 2.8: *Změna měřítka*

Matice transformace změny měřítka S a její inverzní matice S^{-1} jsou ve tvaru

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; S^{-1}(s_x, s_y) = S(\frac{1}{s_x}, \frac{1}{s_y}) = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Po transformaci jsou nově získané souřadnice bodu P' $x' = x \cdot s_x$, $y' = y \cdot s_y$.

2.2.6 Zkosení ve 2D

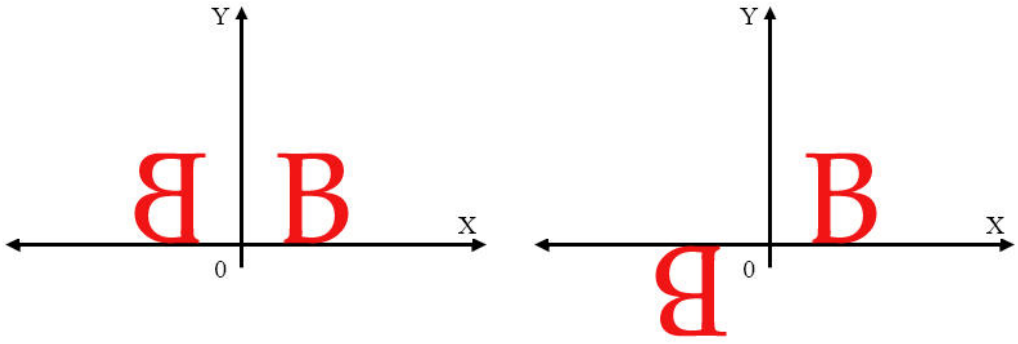
Míru zkosení v matici definujeme jako koeficient sh_x pro horizontální zkosení ve směru osy x a sh_y pro zkosení ve vertikální ose y. Maticové vyjádření je následující:

$$Sh(sh_x, sh_y) = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; Sh^{-1}(sh_x, sh_y) = \begin{bmatrix} 1 & -sh_x & 0 \\ -sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Po transformaci jsou nově získané souřadnice bodu P' $x' = x + sh_x \cdot y$, $y' = sh_y \cdot x + y$.

2.2.7 Zrcadlení ve 2D

Zrcadlení je speciálním případem změny měřítka, kdy absolutní hodnota koeficientu pro změnu je jedna, ostatní členy jsou zachovány. Na následujících obrázcích můžeme sledovat dva typy souměrností. Příklad vlevo prezentuje, jak by mohla vypadat osová souměrnost podle y, stejně tak si jistě dovedeme představit odlišnost transformovaného písmene pro souměrnost podle osy x.



Obrázek 2.9: *Zrcadlení ve 2D (osová a středová souměrnost)*

Klíčové členy S_x a S_y jsou pro osovou souměrnost podle osy y rovny 1 resp. -1. U souměrnosti podle osy x jsou hodnoty $S_x = -1$ a $S_y = 1$. Obrázek vpravo znázorňuje souměrnost středovou. U zrcadlení podle osy jsme převraceli body pouze podle jedné z os, u středové souměrnosti převracíme podle obou. Hodnoty jsou tedy $S_x = -1$ a $S_y = -1$.

2.2.8 Skládání transformací

Práce s transformacemi bude mít skutečně efektivní přínos v případě, že budeme mít možnost jednotlivé transformace kombinovat. Právě z tohoto důvodu jsme si v úvodu kapitoly definovali převod do homogenních souřadnic. Při aplikaci jednotlivých transformací je zřejmě, že bude záležet na pořadí jednotlivých úkonů. Výsledek bude rozdílný, jestliže objekt nejprve posuneme a teprve potom s ním budeme rotovat okolo počátku souřadnicového systému, nebo zda body nejprve otočíme a následně posuneme. Používáme-li zápis $P' = M \cdot P$ (viz vztah 2.3), budeme pozdější transformace přidávat na začátek zleva. Pokud máme tedy nejprve bod P posunout maticí T a následně rotovat maticí R, bude výsledný bod dán vztahem $P' = R \cdot T \cdot P$.

2.2.9 Trojrozměrné transformace

Pro transformace ve třech rozměrech můžeme analogicky využít fakta, jež platí a které jsme si uvedli pro transformace ve 2D. Důvodem, proč jsem nejprve uvedl dvojrozměrné transformace, je snadnost pochopení a vysvětlení na názorných ilustracích. Pro tři rozměry rozšíříme matice o hodnoty osy z, tedy používáme typ matice 4x4. Bude obecně platit následující:

$$P' = M \cdot P = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{14} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}. \tag{2.4}$$

2.2.10 Posun ve 3D

Ve třech rozměrech je posunutí dáno vektorem $\vec{p} = (x_t, y_t, z_t)$. Matice transformace translace T a její inverzní matice T^{-1} jsou ve tvaru

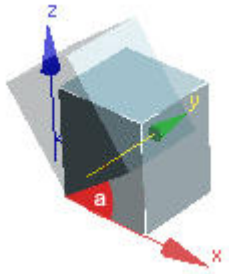
$$T(x_t, y_t, z_{tt}) = \begin{bmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; T^{-1} = T(-x_t, -y_t, -z_{tt}) = \begin{bmatrix} 1 & 0 & 0 & -x_t \\ 0 & 1 & 0 & -y_t \\ 0 & 0 & 1 & -z_t \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

2.2.11 Rotace ve 3D

Transformaci rotace ve 3D o úhel α si pojmeme nejdříve jako rotaci kolem jedné z os. Příslušné vyjádření pro R_x, R_x^{-1}, R_y a R_z je následující:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; R_x^{-1}(\alpha) = R_x(-\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$



Obrázek 2.10: *Ukázka rotace ve 3D o úhel* $a \approx \alpha$

Rotaci kolem obecné osy realizujeme složením několika rotací kolem osy *x*, *y* a *z*. Výsledná osa rotace je dána vektorem a bodem umístění. Nalezení příslušných transformací otočení není triviální. Při řešení volíme bod $P = [P_x, P_y, P_z, 1]$ na ose rotace. Vypočteme jednotkový vektor $\vec{v}$ ve směru osy otáčení. Poté tento bod posuneme do středu souřadnicového systému. Provedeme příslušné rotace a posuneme bod zpět do původních souřadnic. Výsledná matice *M* bude tedy dána vynásobením tří transformačních matic

$$A = T(P_x, P_y, P_z) \cdot R(\vec{v}, \alpha) \cdot T(-P_x, -P_y, -P_z).$$

2.2.12 Zkosení ve 3D

Zkosení se aplikuje na body v trojrozměrné grafice podobně jako u 2D s tím rozdílem, že samotné zkosení není podle jedné osy, ale podle roviny, kterou dvojice os tvoří. Míra zkosení je dána opět členem *Sh*, pro jednotlivé roviny platí následující vyjádření:

$$Sh_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ Sh_x & Sh_y & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; Sh_{xz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ Sh_x & 1 & Sh_z & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; Shy_{yz} = \begin{bmatrix} 1 & Sh_x & Sh_x & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

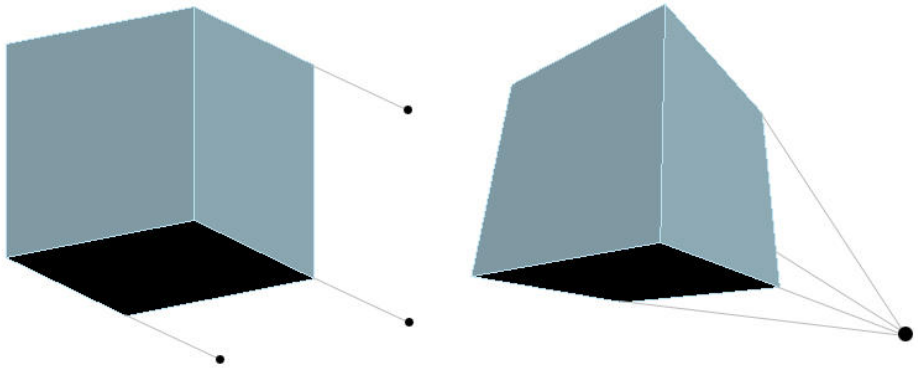
U změny měřítka ve 3D platí stejná pravidla, jako u dvojrozměrné transformace s rozšířením na matici 4x4. Souměrnost je řešena podle rovin, tedy například souměrnost podle roviny *xy* se vytvoří nastavením koeficientů hodnotami $S_x=1$, $S_y=1$, $S_z=-1$.

2.2.13 Projekční transformace

Reprezentace trojrozměrných objektů definovaná v kapitole 2.1 obsahuje informaci o prostoru, kdežto na obrazovku počítače či na plátno jej musíme zobrazit dvojrozměrně a tuto informaci tím ztratit. Tento proces transformace nazýváme projekce nebo také promítání. Existuje několik způsobů projekce, které jsou využívány v určitých oborech. Podle použité metody je zpětně možné určit vzdálenosti, úhly či jiné prostorové vztahy. Podrobný popis projekčních metod lze nalézt v [Zar98].

Z prostorového bodu ve scéně vychází projekční paprsek, který dopadá na plochu (tzv. průmětnu). Průmětnou uvažujeme rovinnou plochu v prostoru. Při dopadu paprsku na tuto plochu vznikají tzv. průměty. Promítání můžeme v zásadě rozdělit na dvě třídy.

- paralelní (rovnoběžné)
- perspektivní (středové)



Obrázek 2.11: *Příklad paralelní projekce vlevo a perspektivní vpravo.*

Paralelní nebo-li rovnoběžná metoda promítání se vyznačuje tím, že její projekční paprsky jsou vyslány rovnoběžně a mají stejný směr. Její typický příklad můžeme vidět na krychli na obr. 2.11 vlevo. Vzniká nerealistický obraz, kde nezáleží na vzdálenosti objektu od kamery, a tedy objekty různě vzdálené se jeví stejně rozměrné. Díky tomu však dosahujeme ve výsledném obrazu rovnoběžnosti u příslušných úseček, což je žádaná vlastnost například u CAD systémů. V praxi se využívá tato projekce především u průmětů do hlavních rovin *xy, xz* nebo *yz*. Jde o nárys, bokorys, půdorys a jejich inverzní variace. Pro nalezení příslušné transformační matice využijeme vztah:

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ w_p \end{bmatrix} = T_{proj} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{2.5}$$

Vektor bodu $P_p = [x_p, y_p, z_p, w_p,]$ označuje výsledné souřadnice původního bodu $[x, y, z,1,]$ na průmětně. T_{proj} je matice, která provádí projekční transformaci bodu. Transformační matice pro hlavní roviny jsou dány následujícím vztahem.

$$T_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, T_{xz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, T_{yz} = \begin{bmatrix} 0 & 0 & 0 & x_0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

U perspektivního nebo také středového promítání vychází paprsky pouze z jednoho bodu. Vytváříme tím realistický výstup, který je bližší vnímání člověka. Při středovém promítání jsou vzdálenější objekty od kamery ve výsledném vyobrazení menší, díky tomu poskytuje perspektivní promítání na rovinné průmětně dobrý prostorový vjem. Transformovaný bod $P_p = [x_p, y_p, z_p, w_p,]$ je dán vztahem

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ w_p \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{z_0} & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{2.6}$$

2.3 Manipulátory

Má-li být práce s transformacemi v aplikaci efektivní a dostupná i pro nezasvěcené jedince, je potřeba uživateli manipulace s objekty zpříjemnit pomocí nějakého rozhraní. Samotná

problematika veškerých výše zmíněných transformací (viz kap. 2.2) zůstane uživateli skryta a k modifikacím dotyčných matic bude docházet právě skrze toto rozhraní. Dobře navržené manipulátory mohou významně zefektivnit a zrychlit práci v prostředí, proto je výhodné se jimi zabývat.

Řešíme-li problematiku návrhu rozhraní pro manipulaci, pokládáme si základní otázku týkající se cílové skupiny, pro kterou bude aplikace určena. V souvislosti s tím klademe důraz na dva rozhodující aspekty:

- přesnost
- uživatelská přívětivost.

V jistých odvětvích využití grafického softwaru bude nezbytně důležité, aby transformace byly naprosto přesné. Když chce konstruktér umístit objekt na osu *x* do vzdálenosti 5,11 od středu souřadnicového systému, musí manipulátor pomocí transformační matice tuto translaci skutečně provést. Jedním z řešení může být zadávání konkrétních hodnot do připravených polí, což požadavek na přesnost určitě splňovat bude. Další možností je realizovat transformace na podobné bázi, ovšem pohybovat se bude objektem o předem stanovený krok v reakci na nějakou událost (např. stisk tlačítka). Zachováme tím přesnost a celkový vykonaný pohyb je potom možné zpětně odvodit podle počtu provedených kroků. Metodě se někdy také říká *offset transformací*, jelikož dochází pouze k přičtení hodnoty k aktuálnímu stavu. Situaci lze řešit i způsobem, kdy bude objektem pohybováno například tažením myši. Přitom budeme dostávat informace o souřadnicích, popřípadě délce uražené dráhy.

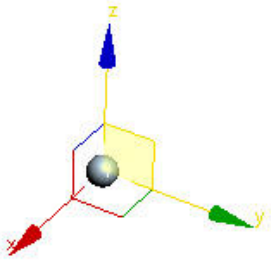
2.3.1 Grafické manipulátory

Pro aplikace jiného typu, kde na přesnosti tolik nezáleží, je spíše rozhodující uživatelská přívětivost. Té lze docílit grafickou úpravou, jež bude vnášet do samotných transformací průhlednost a jednoduchost. Představme si objekt, který chceme transformovat. Účelné by bylo u něj (nejlépe přímo ve středu jeho lokálního souřadnicového systému) zobrazit jednoduché geometrické objekty, které by intuitivně uživatele svým tvarem informovaly o aktuálních možnostech transformace. Tyto objekty by nebyly přímo součástí scény, i když by byly ve scéně zobrazovány. Jejich vzhled by se měnil vždy s typem transformace. Stejně tak by toto bylo možné aplikovat i v rámci skupiny objektů. Transformace by se pak prováděla uchopením a tažením koncového objektu či jiné části manipulátoru. Je přirozeně na nás, jaké schopnosti do manipulátorů implementujeme. Soudobé možnosti manipulátorů si ale můžeme představit na příkladech z praxe.

2.3.2 Manipulátor translace

U manipulátorů, jež mají provádět pohyb, jsou nejdůležitější osy, po kterých bude k transformacím docházet. Tyto osy by měly být reprezentovány úsečkami nejlépe s rozdílnými barvami. Šipky na jejich koncích naznačují směr a možnost posunu. Je také praktické jednotlivé

osy nadepsat, čímž orientaci v prostoru usnadníme. Samotná akce se pak provádí způsobem „*chyť a táhni*“.

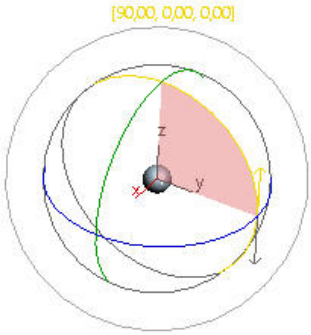


Obrázek 2.12: *Ukázka manipulátoru posunu*

Manipulátor translace (viz obr. 2.12) přímo nabízí možnosti transformovat objekt v rovinách, tedy měnit souřadnice dvou os zároveň. V aplikaci to může být vyřešeno naznačením plošky mezi dvěma dotyčnými osami (zvýrazněno žlutě). Je jasné, že posunutí bude pouze přibližné. Je však možné vylepšit manipulátor o ukazatel hodnoty, jež bude samotný posun uživateli upřesňovat.

2.3.3 Manipulátor rotace

Pro manipulátory otáčení bude tvar objektu takový, aby opět intuitivně naznačoval svou činnost. Rotaci po osách *x, y* a *z* nebudou reprezentovat úsečky, ale kružnice.

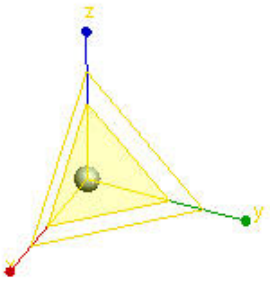


Obrázek 2.13: *Ukázka manipulátoru rotace*

Na obrázku 2.13 můžeme vidět vnější kružnici, pomocí které je možné rotovat kolem pohledového vektoru kamery. Tato kružnice tvoří plochu, která je rovnoběžná s průmětnou kamery. Pro otáčení vzhledem k jednotlivým osám *x, y* a *z* lze využít kružnice, které jsou vždy kolmé k příslušné ose. Nachází se zde ale i několik obohacujících artefaktů, které slouží ke zvýšení názornosti a přesnosti transformace. Jmenujme například barevné označení výseče úhlu, o který se objekt otáčí nebo zvýraznění směru rotace. Významně působí také textová informace o velikosti úhlu aktuálního natočení.

2.3.4 Manipulátor změny měřítka

Pro změnu měřítka objektu bude platit podobná soustava manipulátorů jako pro posun. Opět budou hrát roli příslušné osy, podle kterých budeme chtít zvětšovat či zmenšovat těleso. Takové nerovnoměrné změně se říká neuniformní. Naopak změna měřítka ve všech osách zároveň, jež se provede rovnoměrně na celém tělese, se nazývá uniformní.

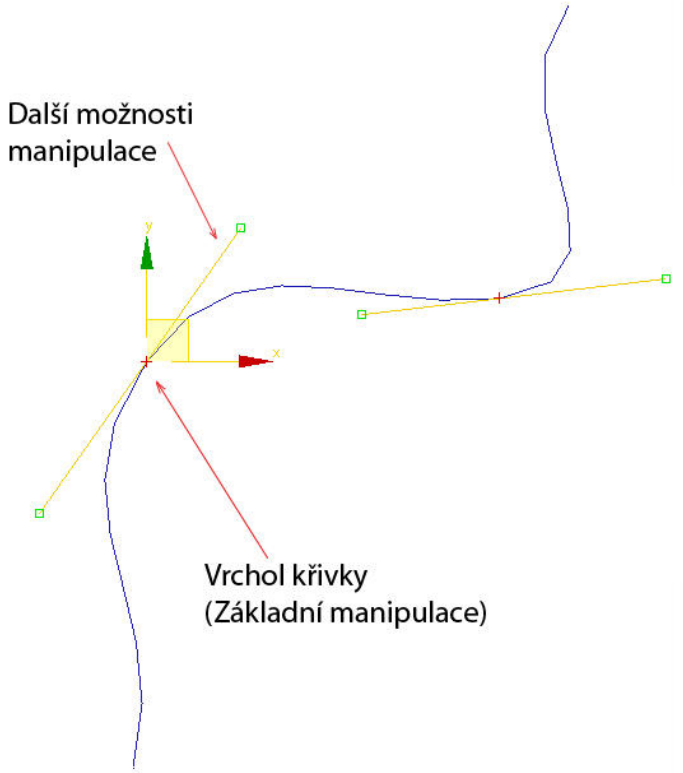


Obrázek 2.14: *Manipulátor změny měřítka*

Opět je zde možná změna měřítka v celé rovině *xy*, *yz* nebo *xz*, jež je naznačena spojením dvou příslušných os úsečkou. Pro změnu velikosti ve všech osách je určen celý střed manipulátoru, tvořící trojúhelník. I zde, stejně jako u rotace, záleží na umístění středu. Jak jsme si řekli v teorii transformací, při změně měřítka může docházet ke zvětšení či zmenšení, současně s tím ale i k posunu v závislosti na umístění středu.

2.3.5 Další možnosti manipulátorů

Popsali jsme si nejzákladnější manipulátory s jejich možnými tvary a schopnostmi. Při návrhu vlastních manipulačních objektů se bude tvůrce přirozeně snažit o jejich maximální použitelnost vzhledem k zaměření jeho aplikace. Můžeme se tedy setkat s manipulátory, jež z části využívají první tři deklarované typy nebo jsou navrženy rozdílně. Popřípadě mohou kombinovat několik druhů v jednom. Nakonec uvedu pár příkladů, kam manipulátory také zasahují. Na obrázku 2.15 můžeme vidět *beziérovu křivku*. Manipulátor obsahuje jednak objekty pro posun celých vrcholů, navíc je vybaven pomocnými body, které určují zakřivení. Pro definici tvaru tohoto druhu křivky je takový manipulátor výhodou, jelikož kombinuje translaci řídících bodů s definicí křivosti v místě transformovaného bodu. Manipulace se stává tedy efektivnější, jelikož není nutné přepínat mezi jednotlivými módy.



Obrázek 2.15: *Beziérova křivka s pomocnými vrcholy*

3 Návrh

Kapitola návrhu se zaměřuje na dvě významné oblasti. První oblast se dotýká uživatelského rozhraní aplikace. Jsou zde vysvětleny současné nároky na ovládání i smysl jednotlivých prvků a oblastí v okně. Stejně tak je obecně vysvětleno typické řešení pohledů ve zobrazovací oblasti. Druhá část pojednává o tvorbě grafu scény. Ten dává do souvislosti jednotlivé elementy, které jsou navrženy pro následnou implementaci.

3.1 Návrh uživatelského rozhraní

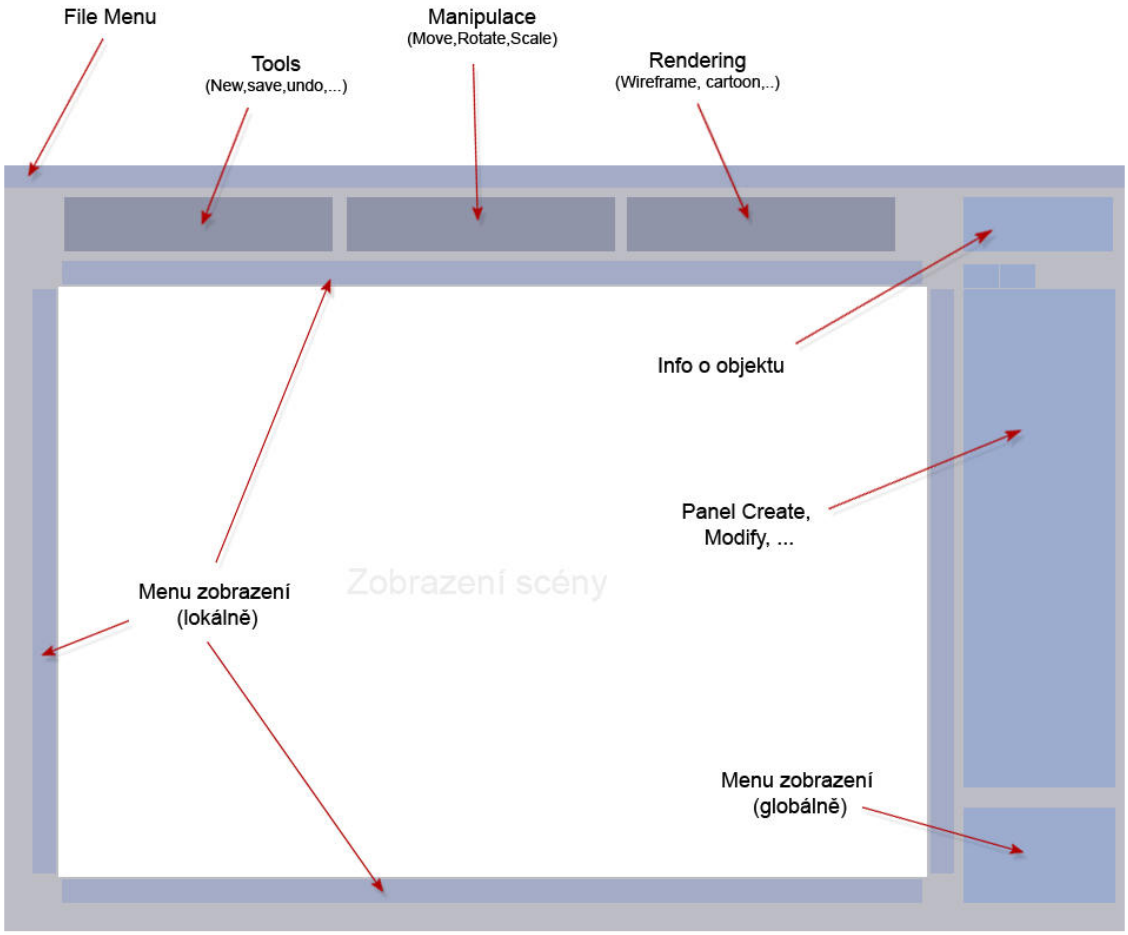
Problematika vektorových 3D editorů je řešena již dlouhou řadu let. Za tuto dobu byly komerční nástroje, jako například *Autodesk 3ds max,* dovedeny téměř k dokonalosti. O tom často svědčí i fakt, že s příchodem nových verzí přibývá stále méně inovací v uživatelském prostředí či jádru aplikace. Proč také měnit něco, co je populární a hojně se s úspěchem využívá již řadu let. Z těchto důvodů bývá velmi výhodné se u některých výrobců softwaru při návrhu uživatelského prostředí inspirovat. Není to dozajista zárukou úspěchu, lze tím ale ušetřit spoustu času při vývoji vlastního produktu.

Navržené uživatelské prostředí je tvořeno grafickým rozhraním, které zprostředkovává interakci uživatele s aplikací. Skládá se obecně z části tvořené zobrazovací oblastí (knihovna *OpenSceneGraph* - prostorová vizualizace, výstup) a části tvořené takzvanými okenními *api* (menu, funkční a ovládací prvky). Aplikace může popřípadě přijímat argumenty příkazové řádky při spuštění programu.

3.1.1 Okenní prostředí

Koncepce navrženého prostředí je volena s ohledem na praktické využití a co možná nejjednodušší obsluhu aplikace. Uživatelské rozhraní je v zásadě rozděleno do pěti základních prostorů.

- Pravé menu
- Horní menu
- Oblast zobrazování scény
- Souborové menu
- Menu kolem zobrazovací oblasti



Obrázek 3.1: *Návrh uživatelského prostředí*

Horní menu se skládá ze třech částí (na obr. 3.1 tmavou barvou). Najdeme zde důležité a často vyhledávané funkce. První blok (*Tools*) se komplexně zaměřuje na ukládání či načítání objektů do scény. To může být typicky formou otevření nové scény, importu souboru nebo přidání objektu do stávajícího grafu. Dále zde najdeme ovladač funkce *Undo/Redo*. Druhý blok je určen k výběru typu manipulace. Toto je bezesporu hojně využívaná část uživatelského prostředí, jelikož manipulace s objekty patří k základním dovednostem vektorového editoru. Třetí blok nabízí podle ilustrace 3.1 kolekci funkcí pro ovládání vykreslení (tzv. *renderingu*).

Tři části má i pravé menu. Majoritní podíl zde má prostřední blok (Panel *Create*), který je určen pro vkládání primitiv do grafu scény. Středový blok je multifunkční a lze v tomto prostoru místo nabízených primitiv zobrazit formou záložek např. panel s nabídkou dostupných modifikátorů či nabízené utility. Případné rozšíření aplikace by bylo vhodné začlenit právě do tohoto bloku formou nové záložky. Horní blok pravého panelu zobrazuje informace o označeném objektu. Pomocí funkcí ze spodního panelu je pak možné globálně ovlivňovat vlastnosti a zobrazení kamer ve scéně.

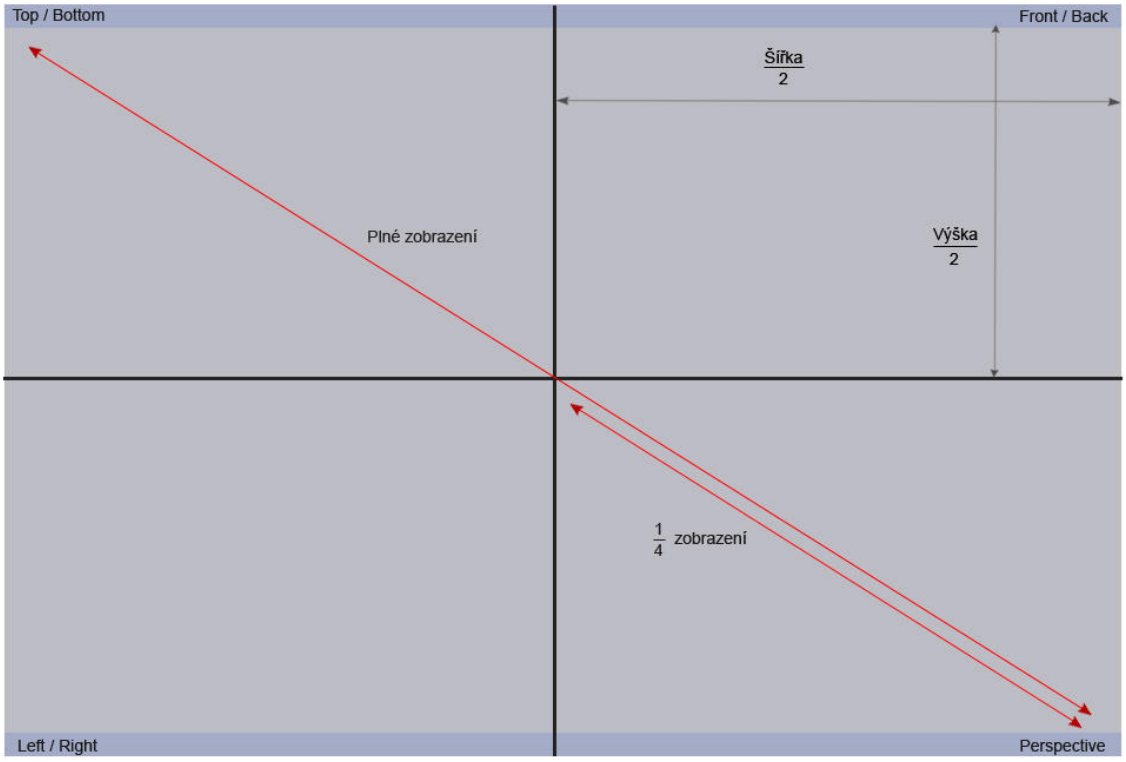
Podobné funkce jako posledně zmiňovaný blok má i nabídka ovládacích prvků kolem zobrazovací oblasti. Jedná se však o funkce vztahující se pouze ke kameře, u které se prvky vyskytují. Dovolují nám ovládat zobrazení kamery či například příslušný výřez zvětšit do celého okna.

Standardem okenních aplikací je tzv. souborové menu, které nacházíme obyčejně v části okna úplně navrchu, v našem případě nad horním menu. Je tvořeno textovými položkami, které mají vazbu na funkce aplikace. Toto menu je vždy viditelné, což můžeme využít v případech, kdy je potřeba skrýt ostatní ovládací panely kvůli úspoře místa pro zobrazení scény. Pomocí souborového menu lze ovládat valnou většinu funkcí aplikace.

3.1.2 Zobrazení scény

Majoritní podíl z prostoru okenní aplikace má oblast, která zprostředkovává výstup, zobrazení scény. Je koncipována tak, aby uživatel dostával interaktivně co nejvíce informací o topologii objektu ve scéně. Zpravidla bývá rozdělena do čtyř základních bloků, jež reprezentují projekci kamer do příslušných rovin. Tuto projekci nazveme pro naše účely zjednodušeně „pohled“.

Nalezneme zde pohledy nárysu (*front*), bokorysu (*left*), půdorysu (*top*) a perspektivní pohled. Jejich rozložení je ilustrováno na obr. 3.2.



Obrázek 3.2: *Volba kamer a umístění pohledů*

Perspektivní pohled (vpravo dole) je určen ke snadnému prohlížení scény, kde je potřeba se flexibilně pohybovat, rotovat kolem objektů. Standardně se volí pro tuto kameru středová

(perspektivní) projekce[6]. Skladbu oken doplňují výše zmíněné pohledy zepředu, zleva a shora. Stejně tak je možné volit jejich obrácené variace. V těchto případech není rotace kolem objektů žádaná a využívá se především posun a přiblížení. Metody projekce jsou zde volitelné s tím, že lze přepínat mezi perspektivním a paralelním promítáním.

Tato skladba pohledů dovoluje uživateli při manipulaci s objekty sledovat současně všechna okna, což mu orientaci v prostoru velmi usnadňuje. Aplikace musí ale umožňovat i zvětšení výřezu pohledu do celého okna, pokud se chce uživatel například zaměřit na detail nebo využít aplikaci k prezentačním účelům.

3.2 Návrh grafu scény

Knihovna *OpenSceneGraph*, která byla navržena pro řešení úlohy, pracuje na principu řazení prvků do tzv. grafu scény. Je to stromová struktura, která není přímo viditelná, obecně však řeší koncepci a fungování celého programu. S podobnou hierarchií se ještě mnohokrát podrobněji setkáme v kapitolách implementace, jelikož definuje vztahy a závislosti jednoho prvku na druhém.

Struktura začíná obvykle u kořene, který se dále dělí do několika podstromů. Na obrázku 3.3 můžeme toto větvení pozorovat. Přímo na kořenovém prvku (*RootNode*) je zavěšeno nastavení pozadí zobrazovací oblasti a tzv. *HUD*[7]. V kapitole 3.2.2 byly definovány pohledy do scény, kdy každý zabírá ve zobrazovací oblasti právě jednu čtvrtinu. *HUD* nám pomůže jednotlivé výřezy vizuálně oddělit. Mimo to slouží i k zobrazování textových informací přímo v oblasti výstupu.

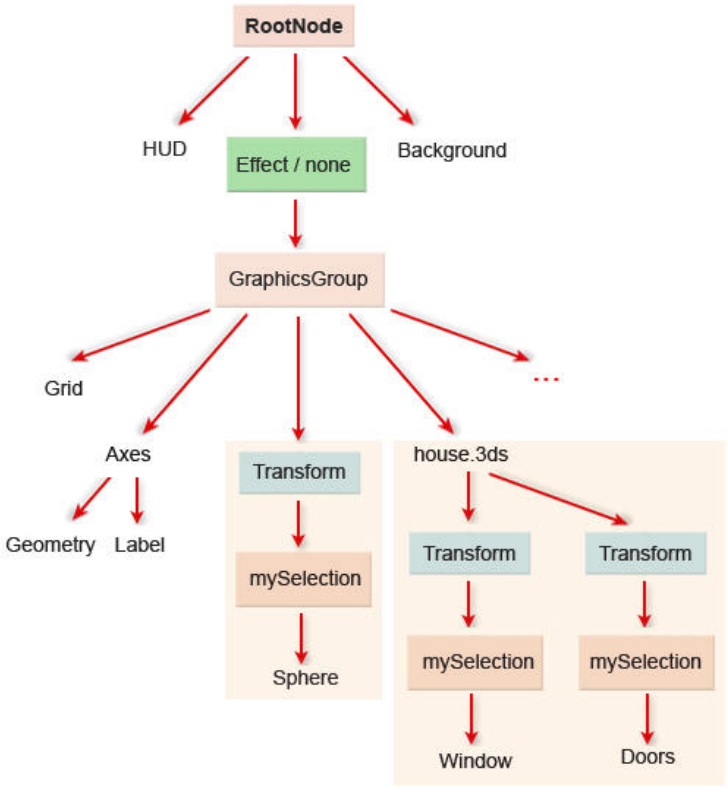
Hlavní částí grafu je větev *Effect/none*, která nese prostřednictvím kontejneru *GraphiscGroup* veškeré objekty scény. Blok *Effect/none* je určen k aplikaci zobrazovacích efektů, jako například drátěný model apod. Jelikož je umístěn hned pod kořenem, znamená to, že efekt ovlivní veškeré potomky tohoto podstromu. Pokud nebude efekt aktivní, nahradí se blok obyčejným kontejnerem (*none*). Objekt *GraphicsGroup* slouží jako uložiště veškerých objektů scény. Pakliže vložíme do scény kouli, jak je naznačeno na obrázku 3.3, dojde k zavěšení tohoto listového prvku nejprve na kontejner *mySelection*. Následuje připojení k transformaci a nakonec napojení k uložišti *GraphicsGroup*. Transformace se vkládá do horní pozice, aby ovlivňovala všechny své potomky v podobě kontejneru *mySelection*. Ten je připraven na stav, kdy je označeno více listových prvků. Podobné řešení je uplatněno i na přidání celé kolekce objektů z externího souboru, jak naznačuje blok vpravo. Načten je celý model domu, který se zavěsí ke kontejneru *GraphicsGroup*. Budeme-li chtít manipulovat s některou jeho částí, vytvoří se opět struktura popisovaná výše a dílčí prvek se zavěsí jako list. Má tedy vlastní transformaci a jeho aktuální pozice je tak zachována. Tuto strukturu nevytváříme pro všechny pod-objekty hned při načtení, ale pouze pokud si přejeme jeho část transformovat při běhu aplikace. Soubor může obsahovat velké množství dílčích těles a nebylo by rozumné, z důvodu náročnosti na přeměnu

[6] Paralelní a perspektivní projekce viz kap. 2.2.13 Projekční transformace

[7] HUD – Z ang. Head up display – zobrazení objektů či textu staticky na display.

grafu scény, přestavbu dělat u všech objektů hned při otevření souboru. Při odstranění konečného objektu je potom důležité odstranit celou jeho náležící větev včetně jeho transformace.

Kromě vkládaných objektů obsahuje kontejner *GraphicsGroup* také statické objekty, které nelze klasickým způsobem odebírat. Jedná se o mříž (tzv. g*rid*) a osy (*axes*) souřadného systému. Mříž vyznačuje počátek světového souřadnicového systému. Osy pak pomáhají určit orientaci v prostoru pomocí barevného značení a popisku (*label*).



Obrázek 3.3: *Návrh grafu scény*

4 Implementace

Implementační část dokumentace pojednává o konkrétním řešení dané problematiky podle návrhových etap. Nejprve je popsán výběr implementačních nástrojů, které dokáží dostát veškerým požadavkům pro tvorbu samotné aplikace. Všechny použité knihovny a nástroje musí být především přenositelné na většinu běžně používaných operačních systémů. Důraz je kladen i na dostatečně kvalitní dokumentaci a budoucnost knihoven. Ty by měly být stále v aktivním vývoji kvůli dalšímu rozšiřování aplikace. Stručně jsou zde pak popsány základní elementy využívané při práci s konkrétní knihovnou.

Kapitoly jsou dále děleny podle základních sekcí návrhu. Podrobně je pojednáno o implementaci grafického uživatelského rozhraní, na které navazuje kapitola o řešení systému kamer. Postupně jsou pak vykládány jednotlivé funkce aplikace až k problematice transformací. To je velmi důležitá část řešení, proto je pro manipulace s objekty věnováno více prostoru. Dále je popsána metoda řešení sytému UNDO/REDO a implementace použitých modifikátorů. Závěrečná kapitola je pak věnována zobrazovacím efektům.

4.1 Jazyk C/C++

Projekt editor 3D scény je koncipován objektově, proto byl pro implementaci praktické části diplomové práce vybrán programovací jazyk *C++* [vir04]. Ten vznikl již v roce 1983 v Bellových Telefonních Laboratořích jako rozšíření jazyka *C*. Volba implementačního jazyka byla dána požadavkem na maximální přenositelnost a rychlost vykonávání programu, čemuž *C++* bez problému vyhovuje.

Samotná implementace programu je napsána v aplikaci *Microsoft Visual Studio 2005*. Ta umožňuje využít široké spektrum nástrojů programovacího softwaru včetně propracovaného *debug* módu a přehledného popisu veškerých zdrojových kódů. Program je přehledně komentován s ohledem na tvorbu kvalitní programové dokumentace, jež je výstupem volně dostupného softwaru *Doxygen*. Nástroj automaticky vytváří *HTML/LaTex* prezentaci, která zobrazuje jednotlivé segmenty kódu s komentáři i provázání implementovaných tříd v grafech.

4.2 Knihovna OpenSceneGraph

OpenSceneGraph (zkr. *OSG*) je vysoce výkonný 3D grafický *Toolkit*, který je vyvíjen pro využití v simulaci, počítačových hrách, ve zobrazování dat a vědeckých vizualizací či například virtuální realitě [www4]. Knihovna je koncipována jako přímá nadstavba nad *OpenGL*[8]. Na rozdíl od jádra *OpenGL* je ale *OSG* kompletně objektově orientován. Svoje uplatnění dokazuje mohutně rostoucím zájmem z řad vývojářů trojrozměrných aplikací. Nesporným kladem je volná dostupnost produktu pro komerční i nekomerční účely a opravdu široká podpora grafických

[8] OpenGL – Softwarové rozhraní ke grafickému hardwaru

souborových formátů . Knihovna je multiplatformní a je podporována prozatím na systémech *Microsoft Windows*, *OSX*, *GNU/Linux*, *IRIX*, *Solaris* a *FreeBSD*.

OpenSceneGraph, jak už z názvu vyplývá, reprezentuje 3D svět jako graf, složený z jednotlivých komponent ve scéně. Tento graf je vždy orientovaný a acyklický. Můžeme si ho představit jako n-nární strom, který má kořen nahoře. Při cestě od vrchu pak procházíme přes slučující prvky, podgrafy a transformace k listům, kde jsou uloženy geometrie jednotlivých elementů scény a jejich nastavení.

4.2.1 Historie a vývoj OSG

OpenSceneGraph byl původně vyvíjen na systému *IRIX*, teprve poté byl portován na *OS Linux*. Následně přišly na řadu další operační systémy. Jmenujme *Microsoft Windows*, *FreeBSD*, *Mac OSX*, *Solaris*, *HP-UX*, *AIX* a dokonce i *PlayStation2*.

Počátky *OSG* se podle [Ost06] datují k roku 1998, kdy zaměstnanec *SGI* (*Silicon Graphics, Inc*.) Don Burns vyvíjel simulátor závěsného kluzáku s použitím tzv. grafu scény (*SG*). Jeho cílem bylo vytvořit systém *SG*, který by byl maximálně dostupný a co možná nejjednodušší na používání. Později začali spolupracovat na vývoji simulátoru další programátoři a *SG* bylo přejmenováno na třípísmennou zkratku OSG, jak ji známe dnes.

Roku 2000 byl přidán *OpenFlight plugin* do modulu *OSG* a další přídavné knihovny byly vyvíjeny rapidním tempem. O tři roky později byla *OSG* knihovna a tzv. *Producer* (původně nazvaný *OSGMP*) použita k poskytování „*multipipe renderingu*“ pro první velký projekt *Magic Earth*.

Nejnovější úpravy pod vedením Roberta Ostfielda se týkají například implementace manipulátorů, jimiž lze transformovat objekty scény na grafické úrovni. Kromě toho se vývoj zaměřuje i na tzv. *Widget* prvky přímo v prostředí, jimiž lze tvořit interaktivní menu. Od verze knihovny 1.9 (2007) je přepracována koncepce závislostí na přidružených knihovnách *OSGProducer* a *OpenThreads* a upravena funkce *Vieweru*. Díky tomu je zcela bezproblémová integrace do *toolkitů* typu *Qt*, *FLTK*, *wxWidgets* či *GTK*. Nutno však podotknout, že od této verze nejsou implementace zpětně kompatibilní.

V současnosti je k dispozici verze 2.8 a lze říci, že knihovna má stále větší podporu díky vzrůstajícímu počtu vývojových týmů pracujících právě s *OpenSceneGraph*. Na jeho vývoji nyní spolupracuje přes 1700 programátorů po celém světě. Do budoucna se počítá i s rozšířením do ostatních programovacích jazyků, jako je např. *Java*, *C#*, *Python*, *Ruby* a další.

4.2.2 Prvky OSG

Pro pozdější popis implementace je dobré alespoň ve stručnosti představit základní elementy a jejich provázání v knihovně *OpenSceneGraph*. Zdrojem je [med06]. Celým programem jsou provázeny tzv. chytré ukazatele `osg::ref_ptr<>`. Spojením elementu s takovou konstrukcí zajistíme automatické snížení referencí na rodiče a dealokaci paměti v případě odstranění. Pakliže by tato konstrukce využívána nebyla, bylo by nezbytné dealokaci činit při ukončení ručně (přirozeně i pro veškeré výjimky, které můžou za běhu programu nastat).

`osg::Node` – Základní třída pro všechny vnitřní uzly stromu. Třída poskytuje rozhraní pro většinu běžných operací s uzly. Lze do ní vkládat geometrii načtených těles. Ze třídy `osg::Node` se dědí třídy `osg::Geode` a `osg::Group`.

`osg::Drawable` – Čistě virtuální třída pro geometrii vykreslovaných objektů. Vše, co je možno zobrazit, je v *OSG* implementováno jako třída děděná ze třídy `osg::Drawable`. Objekty této třídy nemohou být přímo přidány do scény, protože nejsou objekty `osg::Node` nebo jejích potomků. Pokud chceme objekt třídy `osg::Drawable` přidat do scény, musíme ho připojit k objektu `osg::Geode`.

`osg::Geode` - *Geometric Node*. Přestože jde o list, slouží k napojení dalších prvků, které reprezentují viditelné objekty – samotnou geometrii. Objekt této třídy může mít k sobě přidružený objekt třídy `osg::Drawable`. Využíváme ho především k vytváření primitiv či objektů pro *HUD*.

`osg::Geometry` – Třída, která definuje geometrii objektu. Umožňuje u geometrie nastavování vrcholů, normál, barev atd.

`osg::Group` - Slouží jako uzlový kontejner pro uzly `osg::Node`. Obsahuje metody pro správu objektů, jako přidávání potomků nebo odstraňování na něj napojených uzlů. Každá instance této třídy obsahuje svůj seznam instancí třídy `osg::Node`.

`osg::Transform` - Uzel, který reprezentuje transformaci nad objektem popsaným v uzlu `osg::Geode`, popřípadě nad všemi prvky uzlu `osg::Group`. Dědí se od objektu typu `osg::Geode`, může tedy sloužit jako kontejner pro více objektů ve scéně. Všichni potomci jsou potom transformovány tímto uzlem.

`osg::PositonAttitudeTransform` - Uzel, který reprezentuje transformaci nad objektem. Transformační matice lze definovat ručně metodami pro nastavení úhlu rotace, změny měřítka a posunu. Stejně tak zde lze definovat transformační matice přímo.

`osg::Text` - Třída, která umožňuje zobrazování textu. Také se stará o nastavení parametrů zobrazování. Jmenujme velikost písma, umístění, font, ale i nastavení automatického natáčení ke kameře.

4.3 Knihovna QT

QT je robustní knihovna pro vytváření komplexních aplikací s grafickým uživatelským rozhraním [www6]. Vývoj byl zahájen již v roce 1992 norskou firmou *TrollTech*. Nyní je v plném vlastnictví společnosti *Nokia*. Knihovna je primárně navržena pro programovací jazyk *C++*, využít ji lze ale i v prostředí *Java, C#, Python* či například *Ruby*. Využívána je hojně na platformách *Win32*, *UNIX/Linux* i *MacOS*. *QT* je nabízena pod komerční licencí, nebo prostřednictvím *GPL* či *LGPL* pod tzv. *Open Source* licencí.

Knihovna je známa svým pojetím komunikace mezi signály a sloty. Je to řešení oné nekompatibility mezi různými operačními systémy. Využívá tzv. *MOC* (*Meta Object Compiler*), který slouží k před-kompilaci speciálních klíčových slov a k registraci jmen a ukazatelů na sloty se signály nezávisle na platformě. Dále knihovna nabízí možnost návrhu uživatelského prostředí v okenním rozhraní formou aplikace *QT Designer*. Knihovna vyniká především kvalitní dokumentací a rozsahem nabízených funkcí, proto je k implementaci projektu vhodná.

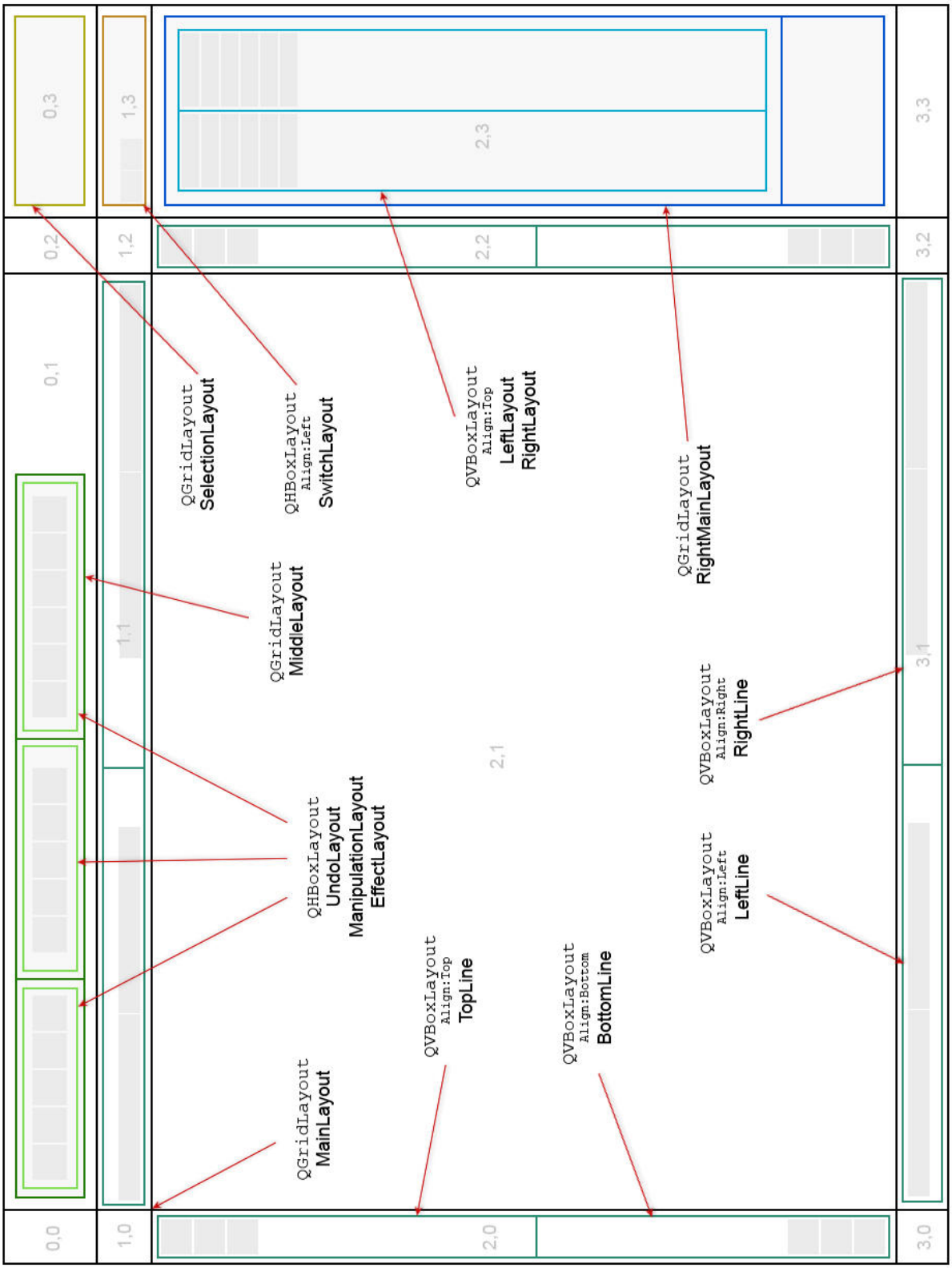
4.4 Grafické uživatelské rozhraní

V kapitole 3.1 jsem definoval základní návrh a logické uspořádání uživatelského prostředí. Při tvorbě koncepce jsem čerpal především z vlastních zkušeností s prostorovými editory. Dovolil jsem si ale navrhnout i poněkud nestandardní topologii ovladačů funkcí kamer, kdy tlačítka obklopují zobrazovací oblast ze všech stran (viz obr. 4.3). Přirozeně jsem si v době návrhu nebyl jistý, jestli moje představy bude možné bez problému realizovat. Zvolil jsem proto k implementaci robustní knihovnu *QT* firmy *Nokia*, která by měla mým požadavkům bohatě dostačovat. Jde o jeden z nejkvalitnějších volně dostupných nástrojů k implementaci okenních aplikací.

Knihovna *QT* nabízí i vlastní prostředí *Designer*, kde je možné si nadefinovat všechny prvky v přehledném okenním prostředí. Já jsem však šel cestou implementace celého rozhraní ručně. Je tak možné mít celý návrh více pod kontrolou a využít nabízené možnosti bez omezení. Naproti tomu za to vývojář zaplatí neúměrně vyšší náročností technickou i časovou. V našem případě obsahuje modul *Widgets*, kde je celé prostředí implementováno, více než šedesát funkcí, které v kódu zahrnují bezmála tři tisíce řádků. Robustnost knihovny se projevuje na místech, kdy zcela zásadní a jednoduchá úloha musí být řešena někdy až zbytečně složitě. Mám tím na mysli například změnu barev prvků, kdy je často nutné definovat palety, štětce apod. aby k takové změně došlo. Samostatnou kapitolou je potom obsluha událostí, jež je řešena způsobem signálů a slotů. Knihovna *QT* si při kompilaci generuje vlastní soubory *moc_název.cpp*, kde dochází právě ke spojení signálů a slotů. Bez těchto souborů obsluha událostí nefunguje, proto bylo zapotřebí složitě dopisovat pravidla kompilace pro preprocessor do souboru *.vcproj* ručně. Obdobný problém nastal u načítání obrázků, kdy je nutné vést jejich kompletní seznam v souboru *.qrc*. Při každé jeho změně pak dochází ke kompilaci automaticky generovaného souboru *qrc_název.cpp*. Ten obrázky ze seznamu obsahuje již v textové podobě a po kompilaci je zapouzdří do spustitelného souboru.

4.4.1 Layout

Hlavní okno aplikace je tvořeno objektem typu `QWidget`. Ten nabízí standardní okenní rozhraní s typickou horní lištou s nástroji minimalizovat, maximalizovat a zavřít. Celé prostředí se potom skládá z tzv. dílčích *Widget*, které jsou uspořádány v určité hierarchii. Tuto hierarchii nám striktně definují objekty typu *Layout*, které nejsou viditelné, ale dovolí nám veškeré prvky uspořádat podle našich představ. Jejich rozmístění je ilustrováno na obrázku 4.1.



Obrázek 4.1: *Schéma rozvržení prostorů (Layout) pro prvky (Widget)*

Objekty typu *Layout* mohou mít různou strukturu. Buď jsou definovány v nejobecnější formě jako mřížka *n*×*n*, kde se potom pohybujeme podobnou indexací jako v poli, nebo je určen pouze směr, ve kterém se vkládané prvky automaticky řadí. Jako hlavní strukturu jsem použil objekt typu `QGridLayout`, který odpovídá dvourozměrné mřížce. V ilustraci i v programu je označen jako *MainLayout*. Tento prostor bude sloužit jako kontejner pro ostatní dílčí prostory. Stejnou strukturu sdílí i podprostor v oblasti (0,1), označený jako *MiddleLayout*. Ten obsahuje další podprostory, které již mají strukturu pouze směrovou (typ `QHBoxLayout`). Teprve do těchto oblastí lze vkládat přímo viditelné prvky. Problematická byla zejména oblast kolem zobrazovacího bloku (2,1), kde bylo návrhem dáno, že se ho ovládací prvky kvůli názornosti přímo dotýkají. Bylo důležité přesně definovat zarovnání a styl prostorů, aby prvky byly současně pohyblivé se změnou velikosti okna.

4.4.2 Widget

Pakliže máme logické uspořádání prostorů implementováno, můžeme do nich vkládat samotné ovládací prvky. Z valné většiny nám postačí skladba tlačítek a viditelného kontejneru (`QGroupBox`), který skupinu ovladačů ohraničí. Jakýkoli viditelný prvek typu tlačítko apod. se do prostoru vkládá knihovní funkcí `addWidget()`. Ta je přetížena různým počtem parametrů podle toho, jestli chceme využít například zarovnání, roztažnost objektu apod. Ovládacím prvkům je přidělena sémantika na základě funkce `connect(Button, SIGNAL(clicked()), this, SLOT(newScene()))`. Signál může být definován celou řadou událostí, v našem případě postačí ošetřit události od kliku myší. Každý ovládací prvek je opatřen krátkou nápovědou, která se rozbalí, pokud nad ním setrváme kurzorem myši. Tento mechanismus obstarává standardní funkce `setToolTip()`. Konstrukce všech prvků aplikace je umístěna ve funkcích `createNázevOblasti()`, které jsou volány konstruktorem třídy *Dialog*. Ve většině případů jde potom o podobné a opakující se definice.

4.4.3 Grafické prostředí

Vizuální podoba aplikace má vždy nemalý vliv na celkový dojem. V důsledku mohou i takové věci, jako je příjemný a ucelený vzhled, rozhodovat o úspěšnosti produktu v prodeji. Ačkoli standardní okna a tlačítka, která tak důvěrně známe, dozajista svůj účel splní, jejich barevná a stylová rozmanitost je již řadu let poněkud omezená. Mnohým pak mohou připadat stále stejně vypadající aplikace trochu fádní.

Pro účely editoru 3D scény jsem navrhl a ilustroval vlastní vzhled aplikace, jehož konečná verze je znázorněna na obrázku 4.2. Výsledný design se skládá z více než osmdesáti originálních prvků, jež byly ručně ilustrovány v rastrovém editoru *Adobe Photoshop*.

Obrázek 4.2: *Realizace GUI*

V původním plánu bylo vytvořit vzhledy dva. Uživatel by pak měl možnost vlastní volby. Návrh a samotná ilustrace veškerých prvků zabrala přibližně pětinu celkového času praktické části, proto nebylo z časových důvodů možné vzhled v jiných barvách zrealizovat. Je třeba ale podotknout, že implementace je připravena na snadnou změnu barev i celého skinu aplikace. Definice tmavého vzhledu je implementována ve funkci `createDarkSkin()` modulu *Widgets*. Dochází zde nejprve k nastavení barev pro paletu, která vyplňuje pozadí a prázdné oblasti. Definovat paletu je nutné i u kontejnerů typu `QGroupBox`. Na obrázku 4.2 můžeme těchto objektů vidět celkem šest. Například kontejner *Tools* v horní oblasti, jednak seskupuje tlačítka stejného zaměření, navíc dotváří pozadí a nese identifikátor tohoto bloku. U tlačítek je nutné striktně nastavit velikost a příslušné parametry (*Flat, AutoFillBackground*), jinak by nebylo možné jejich plochu pokrýt celou. Obrázky jsou načítány pomocí datového typu `QPixmap buttonPix("název.png")`. Přiřazují se potom metodou `setIcon(buttonPix)`. Velmi podobný princip je pak uplatněn na všechny ovládací prvky. Některá tlačítka mají dvě polohy, pak dochází pouze k přepínání jednotlivých pixmap na základě příznaku.

4.4.4 Volba rozlišení

Prvky grafického prostředí mají striktně nastavenou topologii i rozměry, které se nesmí měnit ani se změnou velikosti celého okna. Naopak zobrazovací oblast je obecně definována jako pružná a přizpůsobuje se změnám v šířce i délce. Na rozměrech této oblasti je tedy závislá velikost celého okna.

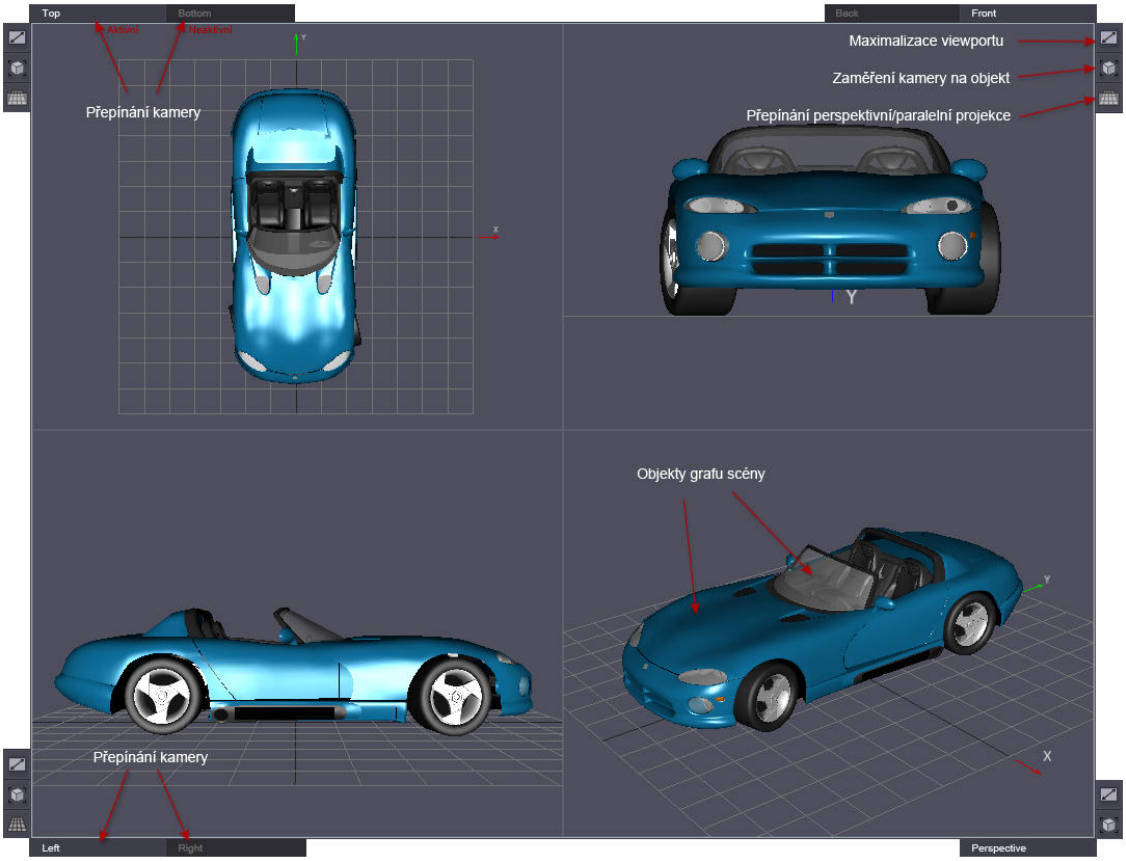
Při startu aplikace je nutné rozhodnout v jakém rozlišení se aplikace spustí, aby nebylo okno příliš velké nebo naopak příliš malé. Nelze rozhodně počítat s tím, že každý má na svém monitoru zvolené rozlišení o šířce 1280 pixelů, navíc když velká část počítačů tvoří notebooky. Řešení je implementováno ve funkci `createViewports()` a spočívá v adaptivním rozhodování. Nejprve zjistíme nastavené rozlišení monitoru v konkrétním operačním systému. Jestliže je v některém směru menší než 1280×1024, zobrazíme vždy okno, které je o 200 pixelů na šířku a o 150 pixelů na výšku menší než aktuální rozlišení displeje. Je-li naopak rozlišení obrazovky větší (např. 1600×1200), zobrazíme okno aplikace vždy v optimální velikosti 1280×960. Měnit velikost okna je potom přirozeně možné, ovšem pouze směrem nahoru, jelikož určené rozlišení je nastaveno zároveň jako minimální.

4.5 Systém kamer

Nedílnou součástí implementace je řešení systému kamer a zobrazení scény obecně. Jak bylo uvedeno v kapitole 3.1.2, do prostředí je zahrnuto celkem 7 pohledů do scény. Jedná se o perspektivní pohled, zobrazení do rovin *Front* (nárys), *Top* (půdorys), *Left* (bokorys) a jejich obrácené varianty *Back*, *Bottom* a *Right*. V implementaci jsou však pouze 4 kamery, které se starají o zobrazení scény. Přepnutí na obrácenou variantu se provádí pouze změnou projekční matice.

Okno, ve kterém zobrazujeme a skládáme výřezy, nazýváme anglicky *viewer*. Potom výřez, který nám zobrazuje průmět do některé z rovin nazýváme anglicky *viewport*. Ke konstrukci kamer a jejich manipulátorů dochází ve třídě `Dialog` ve funkci

`createViewer()`, jež je volána v konstruktoru této třídy. Zde nastavujeme počáteční konfiguraci jednotlivých *viewportů*, jako například topologii a velikost vzhledem k celému oknu. Jako centrální subjekt *vieweru* nám slouží třída `CompositeViewerQT`, jež je děděna od tříd `AdapterWidget` a `osgViewer::CompositeViewer`. `CompositeViewer` nám poskytuje možnosti knihovny *OpenSceneGraph* a třída `AdapterWidget` nám díky objektu `QGLWidget` zpřístupňuje metody knihovny *QT*. Tímto způsobem jsou tedy knihovny provázány.



Obrázek 4.3: *Viewer, skladba viewportů a ovládací prvky kamer*

Co se manipulace s kamerami týče, nabízí knihovna *OpenSceneGraph* vlastní řešení v podobě třídy `osgGA::TrackballManipulator`. Kromě toho je možné využít i jiné podobné třídy. Pro náš případ však nejsou vhodné. Na levé tlačítko myši je standardně namapována akce rotace kolem scény, na pravé tlačítko pak přiblížení. Při stisku obou zároveň se provádí posun v rovině rovnoběžné s průmětnou. Takový manipulátor kamery se nám hodí ve *viewportu perspective*, ale v ostatních oknech je nepoužitelný. Zpravidla nemáme zájem, aby v okně nárysu kamera rotovala kolem scény. Naopak je důležitá akce posunu a přiblížení kamery. Z tohoto důvodu jsem implementoval vlastní třídu `CameraManipulator`, jež umožní jednoznačnou definici akcí tak, jak je vyžadujeme. Nevýhodou je, že jsme nuceni ošetřit množství problémů, které samostatná implementace manipulátoru kamery skýtá.

Třídu `CameraManipulator` využívají kamery *top, left* a *front* způsobem, kdy na levé tlačítko je namapována akce posun a na pravé pak přiblížení. Výpočet aktuální polohy kamery řeší metoda `calcMovement()`, jež sleduje vektor pohybu myši a provádí aritmetické operace s původní (předešlou) maticí. Tím dochází k aktualizaci pohledu v závislosti na pohybu myši.

Na obrázku 4.3 je znázorněno rozložení viewportů, které jsou obklopeny ovládacími prvky. Tyto prvky jsou součástí třídy `Dialog`, avšak volají metody třídy `CameraManipulator`, která zapouzdruje celou práci s kamerami. Nalezneme zde například přepínače určené k maximalizaci či minimalizaci příslušného viewportu. To obstarává metoda třídy `osg::Camera setViewport(osg::Viewport(x,y,width,height)`, kdy parametry `x,y` určují polohu vzhledem k oknu `CompositeViewerQT`. Parametry `width, height` potom udávají velikost příslušného viewportu. Chceme-li tedy výřez maximalizovat, nastavujeme parametry `x,y` do počátku souřadnicového systému a hodnoty `width, height` pak dostanou hodnotu celého výřezu objektu `CompositeViewerQT`. Při každé změně velikosti okna je pak nutné parametry aktualizovat.

4.5.1 Zaměření na objekt

Další nabízenou funkcí je zaměření na scénu. Tento nástroj by neměl v žádném vektorovém 3D editoru chybět. Představme si, že načteme do grafu scény objekt, který je vůči ostatním objektům nesrovnatelně menší nebo naopak větší. Je pak velmi neefektivní, ručním pohybem a přibližováním zacílit na všechny objekty v grafu zároveň. Funkce zaměření (*Fit*) vytvoří ve scéně tzv. *bounding sphere*, jež obklopuje objekty v grafu a nastaví příslušnou kameru tak, aby byly vidět pokud možno všechny objekty grafu. Jinými slovy najde střed *bounding sphere*, na toto místo kameru zaměří a následně ji umístí do vzdálenosti podle velikosti poloměru této koule. Tato metoda nefunguje vždy stoprocentně, pro naše účely však dostačuje.

4.5.2 Ortogonální promítání

Elementární schopností vektorového grafického editoru je možnost využívat ve zobrazování paralelní projekci. V příslušných oknech máme díky tomu nezkreslený geometrický pohled (rovnoběžnost, kolmost), což je předzvěst přehledné a přesné manipulace s objekty ve scéně. Ovládání paralelní projekce nemá v knihovně OpenSceneGraph prozatím podporu, nezbývalo tedy, něž funkce pro ovládání kamer s touto projekcí implementovat ručně.

Nutno podotknout, že zde z principu neexistuje nic jako přiblížení po ose z apod. U paralelní projekce nezáleží na vzdálenosti od kamery, tím pádem se zdají objekty různě vzdálené od kamery stejně rozměrné (viz teoretický úvod kap. 2.2.13). Máme k dispozici metodu třídy `osg::Camera setProjectionMatrixAsOrtho2D(left,right,bottom,top)`.
Pomocí čtyř parametrů *(left, right, bottom, top)* můžeme měnit aktuální výřez okna, tím pádem s ním dle libosti pohybovat. Chceme-li provádět přiblížení, nahrazujeme tyto parametry hodnotami $(\frac{left}{scale}, \frac{right}{scale}, \frac{bottom}{scale}, \frac{top}{scale})$.

Dělitel *scale* potom udává míru přiblížení. Tuto funkcionalitu je nutné propojit s událostmi generovanými myší. Principielně je to pak řešeno podobným způsobem jako u perspektivní projekce.

Z praktické zkušenosti můžu říci, že realizace paralelní projekce není triviální. V praxi jsem se setkal s řadou komplikací, jako například deformace zobrazení z důvodu změny velikosti viewportu (výška, šířka). Aplikace má možnost měnit velikost okna, tím pádem nezachovává statický poměr stran, což vytvářelo nemalé problémy. Další komplikace nastala při přepínání projekce z perspektivní na paralelní, kdy není možné jednoduše najít souvislost mezi vzdáleností kamery od objektu v perspektivní projekci a dělitelem *scale* v projekci paralelní. V důsledku toho při přepnutí projekční metody dochází ke skoku v přiblížení. Abych tento artefakt eliminoval, nastavuji při každém přepnutí projekce kameru do tzv. *home* pozice a tím tyto rozdíly minimalizuji. Nevýhodou je, že uživatel ztrácí původní pohled na scénu.

4.5.3 Adaptace na scénu

Problém nastává i ve chvíli, kdy je načtený model nesrovnatelně větší, než standardní měřítko. V takovém případě se kamera pohybuje, řekněme, v jednotkách, kdežto model je velikosti řádově tisíců jednotek. Zdá se nám tedy, že rychlost pohybu je minimální. Řešení tkví v závislosti rychlosti pohybu kamery na velikosti scény. Velikost scény je, stejně jako v kap. 4.6 (zaměření), odvozena z poloměru tzv. *bounding sphere*, která obklopuje objekty ve scéně. Které objekty budou zahrnuty do *bounding sphere* je zase určeno grafem resp. podgrafem scény. Přírůstek na souřadnicích zaznamenán od událostí myši je vždy násoben poměrnou částí poloměru *bounding sphere*, tím dochází k lineární závislosti rychlosti pohybu kamery na velikosti scény.

4.6 Načítání objektů do aplikace

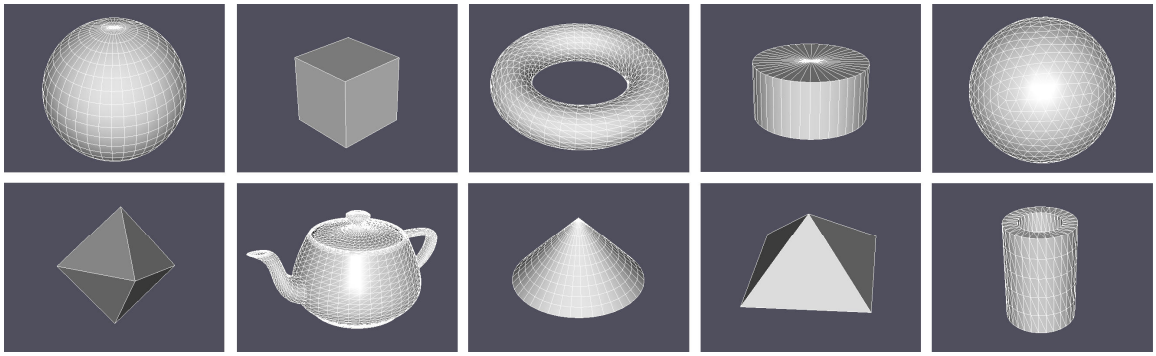
Mezi hlavní schopnosti 3D editoru mimo jiné patří načítání a zobrazování různých grafických formátů. V tomto má knihovna *OpenSceneGraph* velkou výhodu, jelikož díky vestavěnému modulu `osgDB` dokáže pracovat s více než čtyřiceti formáty. Sama potom nabízí dva vlastní souborové formáty, do kterých lze ukládat.

Načítání objektů do scény je implementováno několika způsoby. Ovládací prvky k těmto funkcím jsou umístěny v prvním bloku horního menu společně s tlačítky *Undo/Redo*. Pakliže byla provedena ve stávající scéně změna, aplikace vyzve uživatele k uložení. Změna je definována jako zásah do grafu scény. Pohybujeme-li tedy jenom kamerami nebo měníme-li nastavení, k výzvě nedojde. Volba souboru k otevření je vždy prováděna pomocí standardního dialogového okna `QFileDialog` knihovny *QT*. V případě volby *Open* nabídne dialog otevření souborů pouze s příponami *.osg*, která je volně využitelná k ukládání i načítání. Jako výchozí se potom zobrazí složka *Scenes*, která je umístěna v kořenovém adresáři aplikace. Po výběru souboru je testováno, zda není prázdný a funkcí `resetScene()` dochází k nastavení počátečních hodnot aplikace včetně obnovení grafického uživatelského prostředí. Následuje načtení dat ze souboru knihovní metodou `osgDB::readNodeFile()`, načež je tato geometrie zavěšena na příslušné místo grafu scény.

Další dva způsoby otevření souboru jsou v principu podobné, liší se však nabízenými formáty. U volby *Import* se nabídka rozroste o externí souborové formáty, naopak zde nenajdeme soubory s příponou *.osg*. Škála podporovaných formátů knihovnou *OpenSceneGraph* je skutečně obrovská, já jsem však vybral pouze ty, které bez problému fungují a mají v takové aplikaci smysl. Načítat lze klasické formáty jako *Autodesk .3ds, NewTek Lightwave .lwo*, *Alias|Wavefront .obj* a další. U volby *merge* je nabídka formátů shodná, jako u předešlého případu, obsahuje ale i formát *.osg*. Funkce je určena k přidávání modelů ze souborů do již rozpracované scény. Přirozeně zde tedy nedochází k výzvě k uložení, jelikož se počítá s tím, že uživatel chce pokračovat ve stávající scéně s přidanými objekty.

4.6.1 Primitiva

Další možností, jak přidat objekt do grafu scény, je pomocí menu *create*, které nabízí dostupná primitiva a rozšířené modely (viz obr. 4.4). K tomuto účelu byla vytvořena třída `Primitives`, kde najdeme funkce k vkládání základní geometrie. Některá primitiva můžeme vytvořit za pomocí knihovny *OpenSceneGraph*, jiná je nutné vkládat externě. Do první skupiny počítáme kužel, kvádr, válec či kouli. Jejich proporce jsou pak dány funkcí např. `osg::Sphere(osg::Vec3(Stred), Radius)`. U jiných, jako třeba *torus* je pak potřeba načíst jejich geometrii z externího souboru funkcí `osgDB::readNodeFile()`. Ztrácíme tím tedy možnost definovat jeho parametry.



Obrázek 4.4: *Primitiva a rozšířené modely – Koule, krychle, toroid, válec, geosphere, hedra, konvice, kužel, jehlan, tuba*

4.7 Výběr objektu

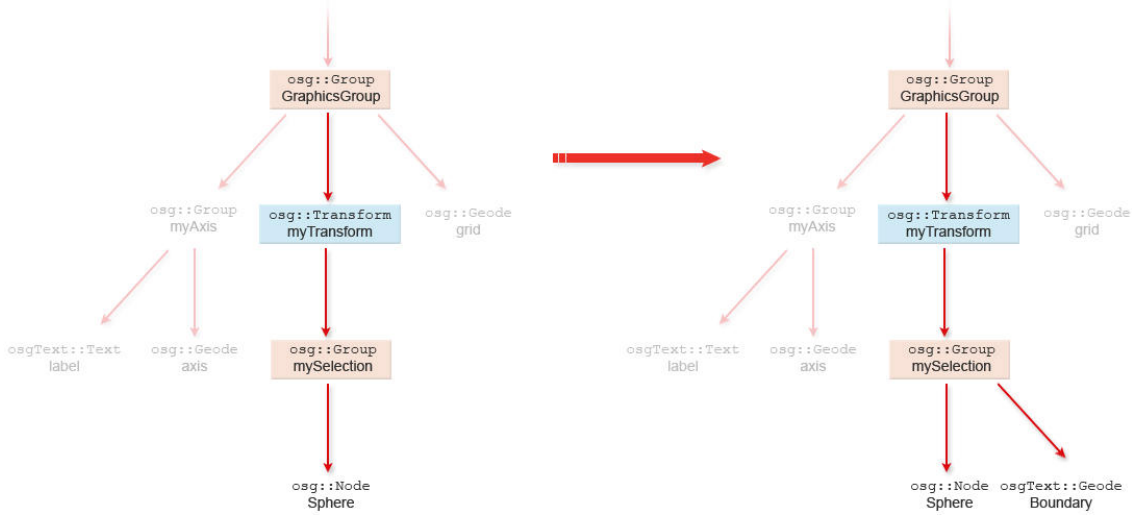
Objekty, které byly do scény jakoukoli formou vloženy, má uživatel možnost zpětně vybírat. Prováděné akce se pak vztahují pouze k tomuto vybranému objektu. Selekce se provede vysláním pomyslného paprsku do scény na místo, kde byla zaznamenána událost kliku od myši. Vytvoříme seznam objektů, které byly protnuty. Seznam je uspořádán v pořadí, kdy nejbližší objekt u kamery je na prvním místě, nejvzdálenější pak na místě posledním. Řešení je implementováno ve funkci `pick()` třídy `PickModeHandler`, jež je součástí modulu *Manipulator.cpp*. Samotnou

intersekci provádí knihovní metoda `computeIntersections(x,y,intersections)`. Výběr požadovaného objektu je pak uskutečněn pohybem v seznamu typu `osgUtil::LineSegmentIntersector::Intersections::iterator`. Ukazatele na protnuté objekty jsou uloženy do příslušných datových struktur a dochází i k zaznamenání jmen těchto objektů. Tyto jsou potom spjaty s výpisem v pravém panelu nahoře, v bloku *Selection*.

4.7.1 Bounding box

Po výběru je třeba objekt nějakým způsobem zvýraznit či vyznačit jeho hranice. Opět jsem se inspiroval komerční aplikací *Autodesk 3ds max*, nicméně způsob zvýraznění je ve většině takových aplikací podobný. Smyslem je vytyčit hranice objektu jednoduchou geometrií tak, aby bylo zřejmé, co je jeho součástí a jak je rozměrný. Současně by přidaná geometrie neměla svojí robustností odvádět pozornost od objektu samotného. Ideálním řešením je obepnutí objektu hranolem (tzv. *Bounding Box*), jež bude mít viditelné pouze rohy. Ty však k definici jeho velikosti bohatě stačí. Ve funkci `showBoundary()` jsem implementoval geometrii takového ohraničení právě na základě struktury `osg::BoundingBox`. Je také velmi názorné, jestliže velikost objektu přímo tuto geometrii ovlivňuje. Na obrázku 4.6 si můžeme všimnout, že úsečky na delších hranách jsou delší než ty na hranách kratších. Délka každé úsečky vedené z rohu je vždy dána jednou čtvrtinou celkové délky hrany.

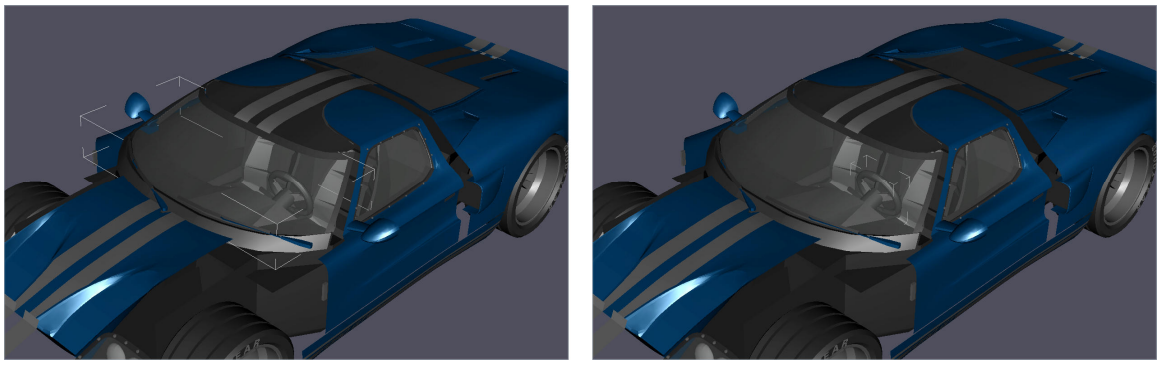
Do grafu scény přidáváme tuto ohraničující geometrii jako dalšího potomka rodiče objektu. V našem případě byla určena hierarchie naznačená na obrázku 4.5.



Obrázek 4.5: *Selekce - přidání ohraničujícího objektu do podgrafu*

Člověk, který se pohybuje v oblasti 3D editorů již jistě častokrát narazil na problém, kdy potřebuje označit ve své aplikaci objekt, který není přímo z úhlu aktivní kamery viditelný.

Uživatel však ví, že objekt je umístěn za jinou geometrií, přes kterou pouze není žádaný objekt vidět. Řešením může být zdlouhavá manipulace s kamerou do takové polohy, kdy bude objekt konečně viditelný (což může být v některých případech nemožné), nebo přizpůsobit takovému požadavku implementaci. Situaci znázorňuje obrázek 4.6. Je zřejmé, že pokud budeme chtít označit přes sklo volant, dojde vždy pouze k selekci čelního okna. Řešením je onen seznam protnutých objektů (viz kap. 4.7), resp. výběr druhého objektu v pořadí. Stane se tak, pakliže uživatel klikne dvakrát na stejné místo. Bez této úpravy by bylo například zmiňované označení volantu více než krkolomné. I takovéto maličkosti dotváří uživatelsky přívětivou aplikaci.



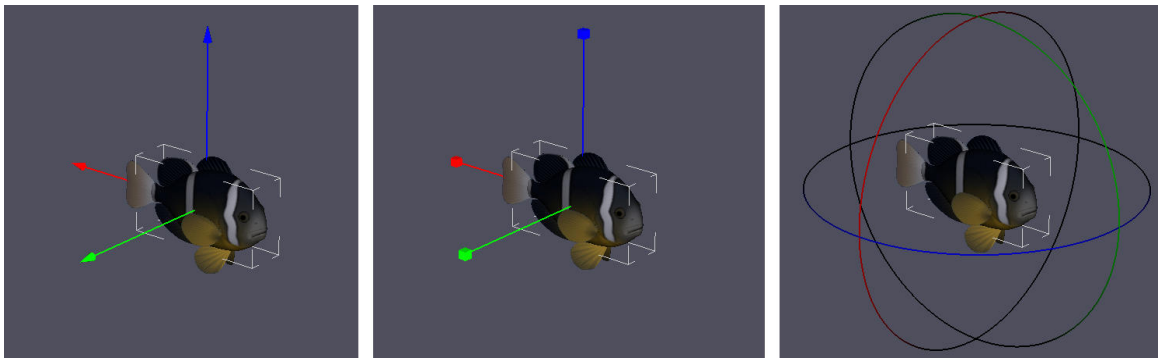
Obrázek 4.6: *Výběr prvního a druhého objektu ze seznamu opětovným klikem*

4.8 Transformace a manipulátory

Manipulace s objekty ve scéně je řešena pomocí grafických manipulátorů. Ty se zobrazí ve středu označeného objektu vždy, když zvolíme některý způsob manipulace formou menu nebo použijeme příslušnou klávesovou zkratku. Implementována je translace, rotace a neuniformní změna měřítka. Knihovna *OpenSceneGraph* nabízí vlastní řešení několika grafických manipulátorů, musím ale podotknout, že zdaleka ne všechny byly v aplikaci použitelné. Jejich vývoj je stále otevřen, proto se u některých vyskytují chyby nebo se u nich potýkáme s nedostatečnou funkčností. Vybrány proto byly pouze nekonfliktní a plně funkční manipulátory vestavěné třídy `osgManipulator`.

Je potřeba říci, že knihovna poskytne pouze grafickou reprezentaci manipulátoru (tzv. *dragger*) s elementárním ošetřením událostí. Ostatní úkoly, jako je umístění, viditelnost, velikost, či obecné řešení grafu scény, je na vývojáři. Veškeré ošetření událostí včetně selekce a transformace je implementováno v modulu *Manipulator.cpp* ve třídě `PickModeHandler`. Přidání grafického manipulátoru do grafu scény se vždy realizuje ve funkci `addDraggerToScene()`. Nejprve zvolíme podle identifikátoru příslušný manipulátor. V metodě `createDragger()` lze vybírat z celkem sedmi typů, jak jsem ale uvedl výše. Využívám pouze tři, které pracují bez komplikací. *Dragger* se skládá vždy z několika primitiv, které svým tvarem naznačují možnosti manipulace (viz obr. 4.7). Ta se obvykle provádí tažením uchopovacích objektů. Model je během manipulace neustále aktualizován tak, aby měl uživatel dostatečný přehled nad prováděnou transformací.

Rotace je realizována pomocí uskupení kružnic, které jsou kolmé k příslušným rovinám v prostoru. Uchopit lze jednak některou z kružnic, potom se provádí rotace vzhledem k příslušné ose, nebo lze otáčet modelem volně, pokud uchopíme manipulátor kdekoli mimo tyto kružnice[9]. Změna měřítka objektu je implementována neuniformně, tedy zvětšovat a zmenšovat model lze vždy po některé ose. Manipulace probíhá, stejně jako u translace, uchopením a tažením koncového objektu barevně naznačených os.



Obrázek 4.7: *Manipulátory – zleva translace, změna měřítka, rotace*

4.8.1 Proporce Manipulátoru

Jedním z problémů při řešení grafických manipulátorů je jejich zobrazovaná velikost. Ze svých zkušeností vím, že *dragger* se statickou velikostí je uživatelsky velmi nepříjemnou záležitostí. Pokud totiž pracujeme na komplexní rozsáhlé scéně, je také pravděpodobné, že se od některých objektů kamerou vzdálíme, abychom se zaměřili na jiné. Když potom chceme manipulovat s těmi vzdálenými, *dragger* se sice zobrazí, ovšem na větší vzdálenost je velmi těžké uchopit kurzorem osu, natož potom správně transformaci odhadnout.

Při řešení tohoto problému jsem se opět inspiroval u firmy *Autodesk*. Pakliže chceme ovlivňovat velikost *draggeru* na základě vzdálenosti, je potřeba dát do souvislosti polohu objektu a polohu kamery. Vzdálenost těchto dvou bodů v trojrozměrném prostoru je podle [Zar98] roven délce vektoru, který je v nich umístěn. Platí tedy

$$d = d(A,B) = \sqrt{(b_0 - a_0)^2 + (b_1 - a_1)^2 + (b_2 - a_2)^2}\,. \tag{4.1}$$

Manipulátor nastavíme do počátku lokálního souřadnicového systému objektu a určíme podle vzorce 4.1 velikost. Ta je přímo úměrná vzdálenosti *d*. Jeho transformační matice je nastavena funkcí

```
setMatrix(osg::Matrix::scale(d,d,d) * osg::Matrix::translate(object->
getBound().center())).
```

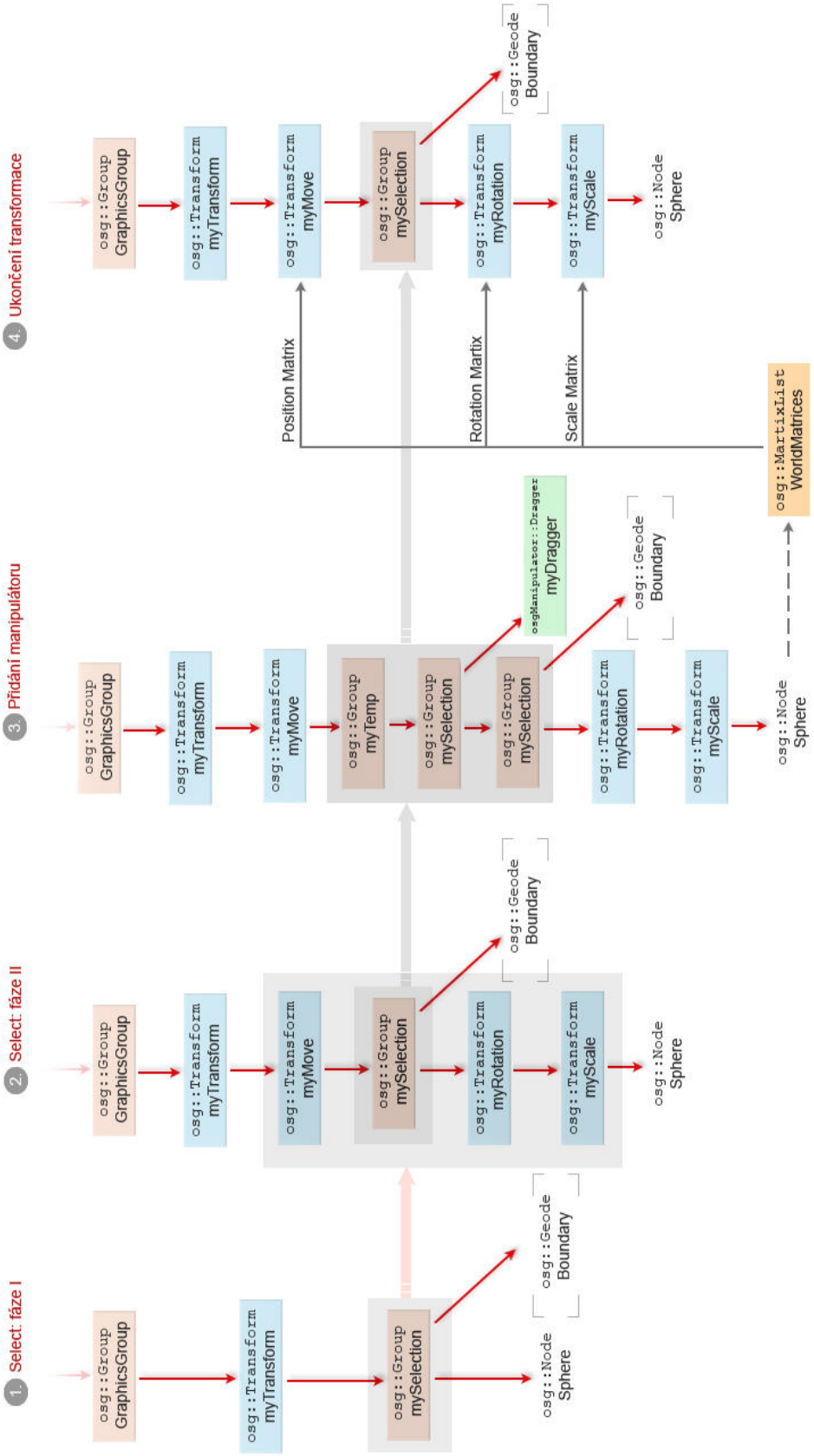
Konečná transformační matice manipulátoru je tedy díky parametru *d* závislá na vzdálenosti kamery od objektu. Skalárním součinem s druhou maticí pak dosáhneme posunu manipulátoru do středu objektu. Je ale potřeba ošetřit spodní hranici velikosti *draggeru*, aby

[9] Myšleno v prostoru pomyslné koule, jejíž hranice vyznačují kružnice.

nebylo možné jej v důsledku velkého přiblížení kamery skrýt za geometrii objektu samotného. To je vyřešeno tak, že manipulátor musí být vždy minimálně stejně velký jako přibližně dvojnásobek velikosti ohraničující koule kolem označeného objektu (tzv. *bounding sphere*). Tím je zaručeno, že koncové objekty budou vždy viditelné. Aktualizace velikosti draggeru se provede vždy po dokončení transformace nebo při změně vzdálenosti kamery od objektu.

4.8.2 Transformace a graf scény

Významným problémem při implementaci transformací je řešení grafu scény. Je třeba se zde řídit určitými zásadami. Jednou ze zásad je, že manipulátory samotné by neměly být v důsledku transformací trvale deformovány. S tím souvisí i fakt, že osy manipulátoru by po ukončení manipulace měly směřovat vždy (i po provedení rotace) rovnoběžně s příslušnými osami světového souřadnicového systému. Takové požadavky vyžadují zamyšlení a není triviální jim dostát. Situace je znázorněna na obrázku 4.8 a 4.9.

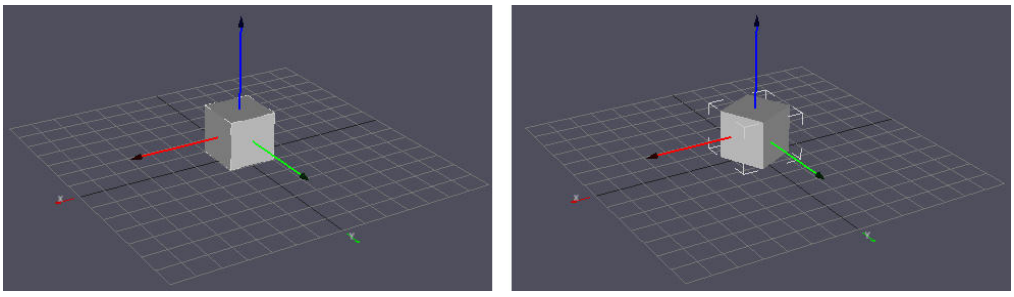


Obrázek 4.8: *Graf řešení transformací a manipulátorů*

Ilustrace demonstruje čtyři kroky, kterými projde každý objekt, se kterým chceme manipulovat. Nejprve dochází k výběru objektu (viz kap. 4.8). Ta má dvě fáze. V první fázi dochází pouze k přidání hranice do grafu scény. Objekt samotný i hranice podléhají stejné transformaci, jelikož jsou rovnocennými potomky objektu *mySelection*. Ve druhé fázi dochází k přestavení grafu scény, kdy kontejner *mySelection* nahrazujeme celým blokem obsahujícím tři nové transformace a starý kontejner *mySelection* mezi nimi. Jde o přípravu na provádění transformací podle zmíněných zásad.

V kroku 3 dochází k přidání manipulátoru do scény. V této fázi se zobrazí příslušný *dragger* ve středu označeného objektu (viz kap. 4.9.1). Můžeme si všimnout, že graf scény je opět přestavěn. Kontejner *mySelection* je nahrazen podgrafem, který už obsahuje samotný *dragger*. Tento podgraf si generuje sama knihovna *OpenSceneGraph* a nezasahujeme do něj. Nacházíme se právě ve fázi, kdy manipulujeme s *draggerem*. V důsledku toho se pohybuje i samotný objekt včetně jeho hranice, jelikož jsou oba potomky kontejneru, který je přímo transformován manipulátorem *myDragger*.

Zásadní okamžik přichází se čtvrtým krokem, kdy stávající manipulace končí v důsledku stisku pravého tlačítka myši nebo výběrem jiného typu manipulace. Jak bylo v úvodu této podkapitoly řečeno, není žádoucí, aby byly manipulátory a hranice objektů transformovány stejně jako objekt samotný. Přesto je potřeba, aby se zobrazovaly na stejném místě jako označený objekt. Struktura transformací, kterou jsme si vytvořili v kroku dvě, nám pomůže toto snadno vyřešit. Oddělili jsme transformace, které chceme aplikovat na objekt a na manipulátor s hranicí. V našem zájmu totiž je, aby se nový *dragger* i hranice zobrazily na aktuální pozici objektu, tudíž transformace translace (*myMove*) by měla ovlivňovat veškeré objektry. Současně chceme, aby transformace rotace (*myRotation*) a změny měřítka (*myScale*) měly dopad pouze na objekt samotný. Proto jsme tyto dvě transformace umístili v grafu až pod kontejner *mySelection*. Je důležité podotknout, že při manipulaci s objektem nedochází ke změně matic v jednotlivých připravených transformacích. Jediným nositelem změny transformace je objekt *myDragger*. Proto je nutné získat výsledné matice koncového objektu, který je ovlivněn všemi transformacemi v tomto grafu. Do kontejneru matic typu `osg::MatrixList` si vložím funkcí `getWorldMatrices()` veškeré aktuální matice pro translaci, rotaci a změnu měřítka. Ty pak aplikuji na příslušná místa ve výsledném grafu scény a ostatní transformace jsou zapomenuty.



Obrázek 4.9: *Řešení transformací – vložený box(vlevo). Po provedení rotace a opětovné aplikaci translace jsou osy stále rovnoběžní se světovým souřadnicovým systémem. Hranice objektu byly přepočítány.*

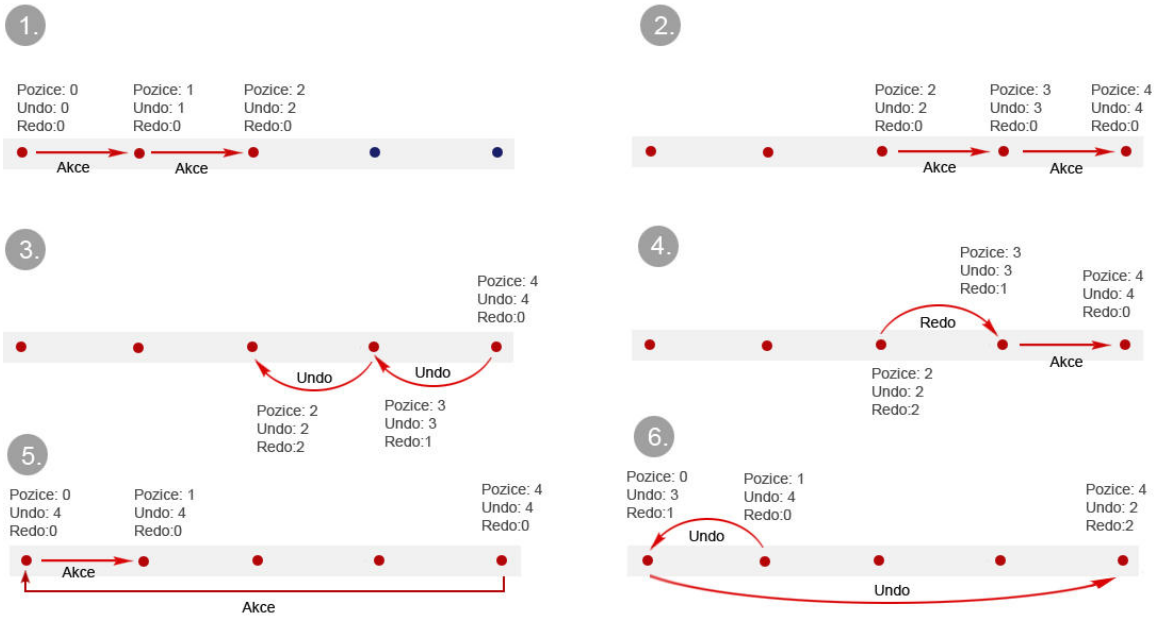
4.9 Systém Undo/Redo

V problematice editorů je funkce kroku zpět či dopředu nepostradatelným nástrojem. Zvláště pak pokud jde o tvůrčí aplikaci, kde nelze jednoduše výsledek svého počínání předvídat.

Je několik možností, jak může takový systém vypadat. V dřívějších dobách existovaly aplikace, kde se historie prováděných akcí neuchovávala, přesto zde podobný systém fungoval. Bylo možné v určité chvíli svou pozici jednou zaznamenat a posléze se k tomuto bodu vrátit. Z dnešního pohledu je to pro pohodlného uživatele velmi omezující a nepraktické. Naproti tomu nabízejí aplikace současnosti možnost cestovat o desítky kroků dopředu či zpět. Některé pak vizuálně poskytují seznam akcí, kde je možné jednoduše zvolit bod, ke kterému se chceme navrátit. Jde zase o zdokonalení, které ušetří čas a námahu se sekvenčním krokováním do požadovaného bodu. Je potřeba se ale zamyslet nad smyslem takového nástroje k dané aplikaci. Pokud je vyvíjen např. jednoduchý textový editor, nemá taková vizuální úprava historie valný smysl. Díky malé rozmanitosti akcí by se zde vyskytovaly převážně položky typu „psaní“ a přívětivosti aplikace by to pravděpodobně nepřidalo. Naproti tomu z praxe mohu říci, že nástroj vizuální historie je velmi chytře využit v rastrovém editoru *Adobe Photoshop*. Ten nabízí nespočet funkcí a variací. Každá akce je pak reprezentována jedním řádkem historie. V takto rozmanité historii je velmi snadné rychle a efektivně se pohybovat, což konkrétně zde ocení drtivá většina uživatelů.

Systém *undo/redo* je implementován i v aplikaci editoru 3D scény. Snažil jsem se, aby nástroj poskytoval dostatek komfortu, na který je uživatel zvyklý. Systém tedy umožňuje standardně pohyb po maximálně třiceti uložených akcích. Tento maximální počet je však možné nastavit a je teoreticky omezen pouze velikostí dostupné paměti. Není ale rozumné volit toto číslo příliš vysoké. Princip řešení systému naznačuje ilustrace 4.10.

Pro názornost jsem zvolil pole pouze o pěti prvcích, z nichž každý představuje právě jeden krok. To může být libovolná transformace, smazání objektu, použití modifikátoru či např. přidání nového primitiva. K orientaci v poli a ošetření jeho mezí je zapotřebí třech proměnných. Jedna informuje o aktuální pozici v poli, další dvě uchovávají hodnoty o počtu možných kroků zpět, resp. dopředu.



Obrázek 4.10: *Princip systému Undo/Redo*

Na počátku je vždy v prvním prvku (pozice 0) uložena počáteční konfigurace. V prvních dvou ilustracích provádíme pokaždé dvě nějaké akce ve scéně. Při každém dokončení akce se nám aktuální stav uloží do následujícího prvku pole. S každou akcí se nám kromě pozice v poli inkrementuje také počet možných kroků zpět. V ilustraci 3 provádíme dvakrát krok zpět, což nám dekrementuje počet možných kroků zpět na hodnotu 2. Naproti tomu dojde k možnosti skočit dopředu, což znázorňuje ilustrace 4. Důležité je uvědomit si, že i když máme počet možných kroků dopředu libovolně velký, jakákoliv nově provedená akce ve scéně toto číslo nuluje, což můžeme vidět ve druhé fázi ilustrace 4.

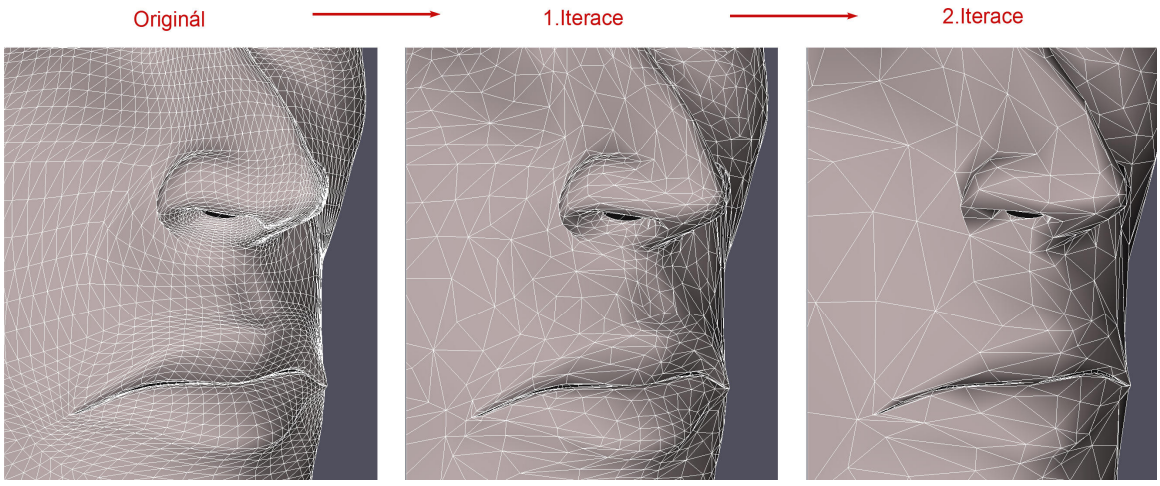
U konečných polí lze očekávat, že za určitý počet kroků dospějeme k jeho mezím (v tomto případě pátý nebo první prvek). Ošetření na konci pole spočívá v tom, že při provedení další akce přeskočí ukazatel pozice na první prvek, hodnotu možných kroků zpět již však neinkrementuje. Maximální počet kroků zpět, resp. dopředu, je tak vždy o jeden menší než je celkový počet prvků pole. Stejný princip je uplatněn, pokud překračujeme meze polí krokem zpět nebo dopředu, což ilustruje bod 6.

Tlačítka ovládání funkce jsou součástí třídy `Dialog`, realizace je pak implementována ve třídě `PickModeHandler` v modulu *Manipulator.cpp*. K ošetření celé problematiky se využívá čtyř metod. Funkce `store()` provádí uložení podgrafu na novou pozici pole po každé provedené změně v geometrii objektu nebo grafu scény. Konfigurace grafu se ukládá do pole ukazatelů typu `osg::ref_ptr<osg::Group>` interní funkcí `clone(osg::CopyOp)`. V reakci na tlačítko *Undo* čí *Redo* je volána metoda `undoStore()` resp. `redoStore()`. V těch dochází především k pohybu v poli ukazatelů a k ošetření mezí pole tak, jak je znázorněno výše. Po každém kroku zpět či dopředu je funkcí `setDefault()` zrušena selekce a popřípadě i aktivní manipulátor. Začínáme-li novou scénu, pak jsou pomocí metody

`resetUndo()` anulovány proměnné možných kroků zpět/dopředu a pozice v poli je nastavena na nultý prvek.

4.10 Modifikátory

Kapitola modifikátorů tvoří velmi významnou roli v problematice 3D editorů. Typicky jsou zde zapouzdřeny skutečné schopnosti editoru jako způsoby editace povrchu či geometrie tělesa nebo možnosti modelování objektů. V aplikaci editoru 3D scény jsem si dovolil zakomponovat jeden demonstrační modifikátor, který provádí změnu detailu geometrie tělesa. Zjednodušuje síť trojúhelníků tak, aby tvar zůstal v rámci mezí zachován, počet trojúhelníků se však při každém aplikování nástroje snižuje. Na obrázku 4.11 je znázorněno použití nástroje na model lidské tváře, který jsem modeloval v aplikaci *Autodesk 3ds max 4* v roce 2003.



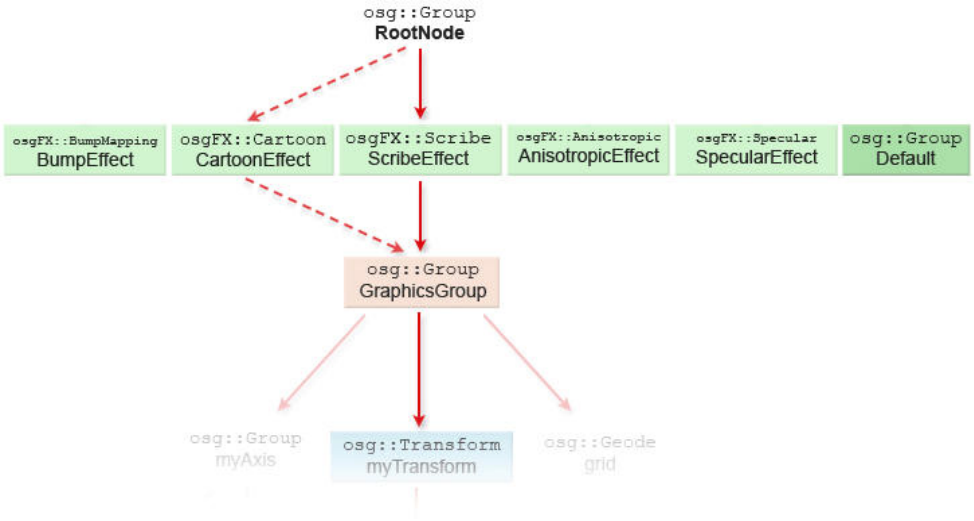
Obrázek 4.11: *Modifikátor Optimalize*

Úpravu geometrie sítě provádí knihovní funkce `osgUtil::Simplifier` která je volána metodou `simplify()` ve třídě `PickModeHandler`. Aplikuje se na podgraf, v našem případě listový prvek typu `osg::ref_ptr<osg::Node>`. Takovým prvkem mohou být jakékoli objekty tvořené sítí trojúhelníků. Výjimkou jsou primitiva vytvořená pouze knihovnou *OpenSceneGraph*, jako například `osg::Box` a podobné. Nástroj je začleněn v aplikaci pouze demonstračně, proto neposkytuje nastavení parametrů a hodnoty pro převod sítě trojúhelníků jsou zvoleny konstatně.

4.11 Efekty

Efekty pro vykreslování objektů mají obecně různý význam. Kromě vizuální podoby scény, které chceme dosáhnout, mohou mít i praktický účel. Častokrát je objekt, na který se chceme zaměřit nebo který chceme manipulovat, skryt za geometrií jiného tělesa. Potom je velmi výhodné mít po ruce nástroj, který zprůhlední veškerou geometrii povrchu objektů a zachová zobrazenou pouze síť trojúhelníků. Toto zobrazení se nazývá *drátový model* (ang. *wireframe*).

Jindy potřebujeme zobrazit povrch objektu společně se sítí trojúhelníků, abychom prezentovali detailnost modelu či jeho geometrickou stavbu (viz obr. 4.5). Efekt nazýváme anglicky *scribe* nebo také *edge facecs*. Pro prezentační účely slouží i zbylé čtyři implementované efekty. Tzv. *cartoon* je určen pro napodobení kreslených objektů pomocí adaptivního prahování. Efekt *anisotropic* simuluje nestejné fyzikální vlastnosti povrchu v různých směrech. Zpodobňujeme tím například tváření některých kovů. *Specular* pak zobrazuje objekty ve scéně standardně, pouze hodnota odlesku materiálu je nastavena na maximální hodnotu. Posledním efektem je tzv. *bump*, který pomocí normálové mapy dosáhne viditelné změny struktury povrchu. Ten se pak zdá plastický. Tato změna je však pouze vizuální. Na geometrii objektu nemá žádný vliv.



Obrázek 4.12: *Graf scény s použitím renderovacích efektů*

Na obrázku 4.12 můžeme vidět řešení grafu scény při použití některého z efektů. Aplikace se provede začleněním prvku (na obr. 4.12 zeleně) kamkoli do grafu. Veškeré jeho potomky potom ovlivní nastavení tohoto prvku. Pro aplikaci editoru 3D scény jsem zvolil umístění prvku hned pod kořenem. Ovlivněny tedy budou veškeré objekty vkládané do grafu scény, jelikož tyto podléhají kontejneru *GraphicsGroup*. K obsluze efektů je implementováno celkem devět funkcí v modulu *Widget.cpp*.

Na počátku dochází k inicializaci veškerých efektů voláním funkce `setupEffects()`. Při startu aplikace je místo efektu začleněn do grafu scény obyčejný kontejner (na obr. 4.12 blok *Default*). Je-li větev *GraphicsGroup* zavěšena za tento prvek, scéna je zobrazována standardně bez použití vizuálního efektu. Jestliže chceme použít některý z efektů, dojde pouze k přepsání ukazatelů na příslušná místa. Efekty na obrázku 4.12 jsou implementovány pomocí knihovní třídy `osgFX`. Jediný efekt, který zde není uveden, je výše zmíněný *drátový model*. Jeho řešení jsem implementoval ručně pomocí kontejneru *Default*. Definoval jsem množinu stavů (tzv. *StateSet*), kde je nastaven způsob vykreslení objektu (*Line, Fill*) podobně jako v api *OpenGL*.

5 Výsledky

Editor 3D scény je okenní aplikace implementovaná v jazyce *C++* s využitím knihovny pro tvorbu grafického uživatelského rozhraní *QT 4.3*. Výstup je realizován prostřednictvím knihovny *OpenSceneGraph 2.7*, jež je přímou nadstavbou nad api *OpenGL*. Aplikace je odladěna v prostředí *Win32*, volba knihoven a implementačního jazyka však zaručuje bezproblémovou portaci na většinu běžně používaných operačních systémů.

Program umožňuje pracovat s minimálně osmi externími grafickými formáty a aktuální graf scény ukládat včetně geometrie do souborového formátu *osg*. Načítání objektů je možné celkem čtyřmi způsoby. První možností je otevřít uloženou scénu ve formátu aplikace (*.osg*). Dále lze importovat externí souborové formáty metodou otevření v nové scéně či přidat geometrii ze souboru do scény stávající (*merge*). Vkládat objekty můžeme také pomocí pravého menu výběrem některého z primitiv či rozšířených těles. V programu lze plně využít systém *undo/redo*, čímž je možné cestovat v historii prováděných akcí na maximálně třiceti krocích. Obecně lze toto limitní číslo nahradit nekonečnem (resp. dostupnou volnou pamětí), je však z pragmatických důvodů omezeno. Každý vložený či načtený objekt lze kurzorem myši vybírat. Taková selekce je vždy vyznačena ohraničující geometrií, která jasně definuje velikost objektu a součásti k němu náležící. Kromě toho je také podána informace o názvu vybraného tělesa. Mezi elementární funkce editoru patří transformace objektů, které lze provádět s jakýmkoli označeným tělesem. Pomocí grafických manipulátorů lze provádět translaci, rotaci a neuniformní změnu měřítka. Vykreslení objektů scény je pak možné obohatit různými vizuálními efekty, mezi které patří například tzv. *drátový model, cartoon* pro simulaci kreslených objektů*, anisotropic* či *bump* efekt. Se zobrazováním souvisí i propracovaný systém čtyř kamer, který umožňuje celkem sedm pohledů do scény. Jedná se o průměty *top, left, front* a jejich obrácené variace *bottom, right* a *back*. V těchto pohledech je možné využít implementovanou ortogonalní projekci. Kromě těchto průmětů nabízí aplikace perspektivní pohled, který je zaměřen spíše na prohlížení objektů a flexibilní pohyb ve scéně. Důraz je kladen na variabilitu prostředí, proto lze veškerá tato okna maximalizovat do plného výřezu na úkor oken ostatních. Kromě toho je možné skrýt panely s nabídkami a roztáhnout tak zobrazovací oblast do maximálních rozměrů. V oblasti modifikátorů je implementován demonstrační nástroj *optimalize*, který dokáže snížit úroveň detailu sítě *mesh* u načtených objektů a tím zjednodušit geometrii modelu.

Implementace je rozčleněna do devíti modulů a celkem patnácti tříd. Zdrojový kód se pak bez generovaných souborů rozkládá na více než sedmi tisících řádcích. Uživatelské prostředí je kompletně navrženo a ilustrováno v rastrovém editoru *Adobe Photoshop CS3* a skládá se z celkem čtyřiaosmdesáti originálních prvků. Okno aplikace je možné roztahovat či maximalizovat dle potřeby, přičemž dochází k přepočítání rozměrů zobrazovací oblasti (výstupu). Z prostředí aplikace lze zobrazit okno s jednoduchou uživatelskou nápovědou. Pro základní a často využívané funkce jsou implementovány klávesové zkratky.

Pro účely prezentace programu jsem vytvořil plakát formátu A3 a webové stránky v jazyce *php* a *css*, kde je možné si stáhnout zdrojové soubory, spustitelnou verzi i programovou dokumentaci. Jsou zde umístěny ukázky z aplikace i kompletní charakteristika programu.

6 Závěr

V závěru své diplomové práce bych rád našel prostor ke zhodnocení dosažených výsledků a problematiky implementace obecně. Realizace zadané úlohy nebyla snadná. Vybral jsem si velmi obsáhlé téma, u kterého je možné zaměřit se vždy pouze na specifickou oblast. V počáteční fázi je však potřeba vytvořit základ, na kterém budou veškeré implementované nástroje a schopnosti editoru postaveny. Právě návrh a realizace takového základu byly smyslem této práce. Implementaci jsem začínal takřka na zelené louce, jelikož bylo možné opřít se pouze o příklady a dokumentace dodávané ke knihovnám *OpenSceneGraph* a *QT*. Měl jsem ale detailní představu o podobě produktu i jeho schopnostech v konečné fázi. Dlouhodobé zkušenosti s podobnými aplikacemi v praxi mi tuto vlastní vizi postupem času samy vytvořily. Nutno ale dodat, že realizace představ je někdy poněkud komplikovaná.

Mezi mé cíle patřilo především komplexní řešení systému kamer v prostředí jako základ prostorového editoru scény. Při návrhu a manipulaci s objekty je nezbytné umožnit uživateli bezproblémovou orientaci v prostoru a poskytnout mu co nejvíce informací o topologii těles. Potom je právě propracovaný kamerový systém klíčem k efektivní manipulaci a editaci objektů. Kapitola kamer se v důsledku stala implementačně velmi náročnou úlohou, jelikož jsem zvolil vlastní realizaci obsluhy většiny pohledů. Stejně tak bylo nutné ručně implementovat paralelní projekci kamer, která je v těchto aplikacích také nezbytná. Dalším cílem bylo navrhnout a realizovat originální grafické uživatelské rozhraní aplikace. Kladl jsem si přitom za cíl vytvořit vlastní prostředí svým vzhledem i celkovou koncepcí. V tomto směru se osobní vize také podařilo naplnit, i když časová náročnost byla mnohem vyšší, než jsem původně předpokládal. Významnou úlohou bylo pak řešení transformací a manipulace s objekty vůbec. I tato fáze implementace přinesla své komplikace a celkově nebyla snadná. Bylo potřeba dbát zejména na zásady při řešení transformace těles v prostoru a na uživatelskou přívětivost manipulátorů jakožto prostředku k základní editaci scény. Kromě toho je v implementaci zahrnuta i řada vlastních vylepšení, které pomohou práci s objekty scény zefektivnit. Jednodušší částí byla pak realizace zobrazovacích efektů a modifikátoru, kdy bylo možné využít dostupné možnosti knihovny *OpenSceneGraph*.

Moje představa o dalším vývoji produktu je velmi konkrétní, avšak pro jednoho člověka je svým rozsahem nerealizovatelná. Svojí koncepcí je aplikace zatím otevřena pro jakékoliv uplatnění, ať už půjde o nástroj zaměřený na konstruování, vizualizaci, animaci nebo třeba medicínské simulace. Základní rozšíření by se ale v každém případě mělo dotknout nejdříve možností manipulátorů, kde jsou velké rezervy. Manipulátory by měly nabízet širší spektrum funkcí, jelikož právě v nich jsou ukryty pokročilé možnosti editace objektů. Systém vkládání primitiv do scény je nutné také inovovat. Před vložením tělesa do prostředí nebo kdykoli po selekci objektu musí být uživateli umožněno definovat zpětně jeho parametry a souřadnice. Největší prostor pro rozšíření se ale nachází v oblasti modifikátorů. Právě zde definujeme pravé schopnosti editoru, ať už jde o nástroje pro modelování složitých tvarů, editaci geometrie nebo například nanášení textur na objekty. Současné 3D editory nabízejí desítky modifikátorů, které zapouzdřují skutečné schopnosti aplikace. Jmenujme například modelování NURBS křivek

a ploch, možnosti optimalizace geometrie nebo třeba simulaci částic či vodní hladiny. V neposlední řadě může být aplikace obohacena o světelné efekty, metody raytracingu a radiosity. Díky takovým nástrojům pak vznikají fotorealistické vizualizace a efekty, které vídáme třeba na filmových plátnech. Jak bylo v úvodu řečeno, oblast editorů 3D scén je velmi rozsáhlá a nabízí nepřeberné možnosti pro vývoj vlastního produktu. Je pak pouze na nás, jakým směrem se navržená aplikace bude v budoucnu ubírat.

Literatura

[Zar98] ŽÁRA, J. – BENEŠ, B. – FELKEL, P.: *Moderní počítačová grafika*. 1. vyd. Praha, Computer press 1998, 448 s., ISBN 80-7226-049-9.

[Vir04] VIRIUS, M.: *Od C k C++*. 2. Vyd., Kopp nakladatelství České Budějovice 2004, 217 s., ISBN 80-7232-110-2.

[Shr05] SHREINER, D. – WOO, M. – NEIDER, J. – DAVIS, T.: *OpenGL Průvodce programátora.* 1. vyd. Brno, Computer Press 2006, 680 s., ISBN 80-251-1275-6.

[Bla08] BLANCHETTE, J. – SUMMERFIELD, M.: *C++ GUI programming with Qt 4*, 2. vyd. Prentice Hall PTR 2008, 560 s, ISBN 978-0131872493.

[Bla04] BLANCHETTE, J. – SUMMERFIELD, M.: *C++ GUI programming with Qt 3*, 1. vyd. Prentice Hall PTR 2004, 464 s, ISBN 978-0131240728.

[Zech93] ZECHER, J.: *Computer Graphics for Cad/cam Systems*. 1. vyd. CRC 1993, 432 s, ISBN 978-0824789527.

[Med06] Tým počítačové grafiky pro medicínu: *OpenSceneGraph tutoriál*. Fakulta informačních technologií, VUT v Brně, 2006.

[Ost06] OSTFIELD, R.: *OpenSceneGraph* (Historie a základní informace), 2006, přístup z internetu: <http://www.cs.umu.se/kurser/TDBD12/VT04/lab/osg/html/introduction.html>

[www1] KRŠEK, P.: *Uživatelské rozhraní v CAD systéměch*. Materiál k přednášce kurzu VIZ, FIT VUT v Brně, 2008, přístup z internetu: <https://www.fit.vutbr.cz/study/courses/VIZ/private/lecture/viz_slide_gui_print.pdf>

[www2] HEROUT, A.: *Reprezentace scén, 3D transformace, Rasterizace*. Materiál k přednášce kurzu PGR, FIT VUT v Brně, 2008, přístup z internetu: <https://www.fit.vutbr.cz/study/courses/PGR/private/lect/PGR-Revision-II.pdf >

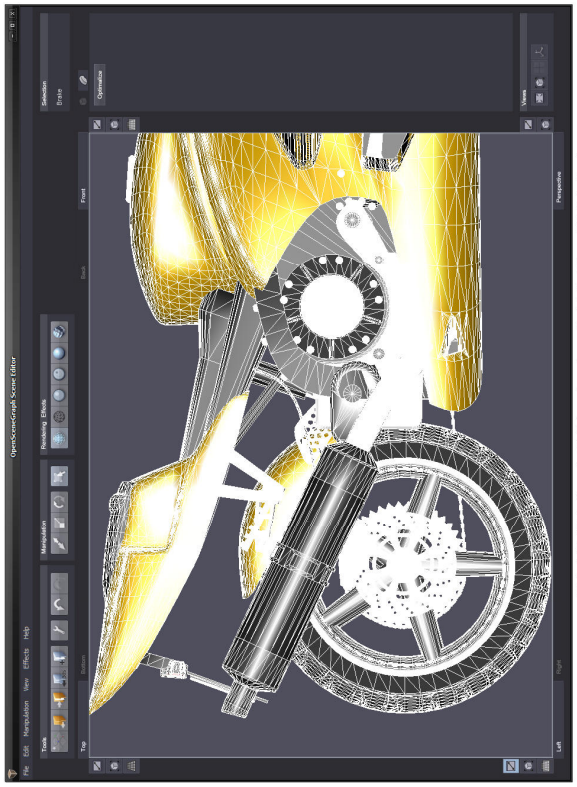
[www4] Oficiální web knihovny *OpenSceneGraph*, dokumentace, přístup z internetu: <http://www.openscenegraph.org>

[www5] Křivky NURBS na serveru root.cz, 3.3. 2006, přístup z internetu: <http://www.root.cz/clanky/krivky-nurbs-1/>

[www6] Oficiální web knihovny *Nokia QT*, přístup z internetu: <http://www.qtsoftware.com/>

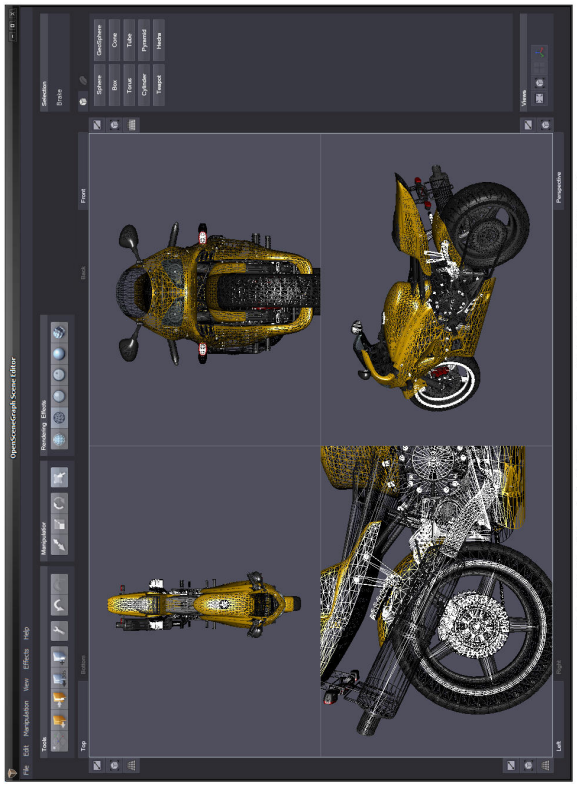
Seznam příloh

Příloha 1. Variabilita GUI



Obrázek 7.2: *Maximalizovaný pohled left se zobrazením panelů nástrojů. Aplikace efektu "edge focus"*

Obrázek 7.4: *Maximalizovaný pohled perspective se skrytými panely nástrojů. Aplikace efektu "cartoon"*

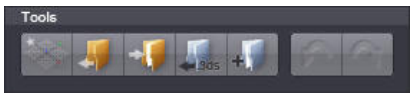


Obrázek 7.1: *Klasické rozvržení pohledů se zobrazením panelů nástrojů. Aplikace efektu "drátěný model"*

Obrázek 7.3: *Klasické rozvržení pohledů se skrytými panely nástrojů pro maximalizaci výstupu. Aplikace efektu "cartoon"*

Příloha 2. Uživatelský manuál

Tools Menu



Funkce tlačítek (zleva):

- **New Scene** – Obnoví veškerá nastavení (menu, kamery) a vloží prázdnou scénu. *Ctrl + N.*
- **Open File** – Otevření souboru formátu *.osg*. Před načtením se dotáže na uložení stávající scény, pokud byla modifikována. *Ctrl + O.*
- **Save File** – Uložení souboru do formátu *.osg*. Automaticky se načte výchozí složka *Scenes*. *Ctrl + S.*
- **Import File** – Import externího formátu do aplikace (*.3ds, .lwo, .obj, .md2, .iv, .geo, .dw*). Před načtením se dotáže na uložení stávající scény, pokud byla modifikována.
- **Merge File** – Přidání souboru do stávající scény. (.osg, *.3ds, .lwo, .obj, .md2, .iv, .geo, .dw*).
- **Undo** – Vrátí krok zpět. Lze provést, pokud je šipka zvýrazněná. Maximálně se lze pohybovat po třiceti provedených akcích. Po vykonání akce je zrušena selekce a odstraněn manipulátor. *Ctrl + Z.*
- **Redo** – Vrátí krok dopředu. Lze provést, pokud je šipka zvýrazněná. Maximálně se lze pohybovat po třiceti provedených akcích. Po vykonání akce je zrušena selekce a odstraněn manipulátor. *Ctrl + R.*

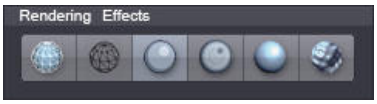
Manipulation Menu

Funkce tlačítek (zleva):

- **Move** – Zobrazí manipulátor posunu. Translace se provádí chycením a tažením koncového objektu manipulátoru (šipka). Musí být označen objekt. Velikost manipulátoru se mění se vzdáleností kamery od objektu. S kamerami nelze během manipulace pohybovat. *Klávesa M.*
- **Scale** – Zobrazí manipulátor změny měřítka. Transformace se provádí chycením a tažením koncového objektu manipulátoru (krychle) vždy v jedné ose. Musí být označen objekt. Velikost manipulátoru se mění se vzdáleností kamery od objektu. S kamerami nelze během manipulace pohybovat. *Klávesa N.*
- **Rotate** – Zobrazí manipulátor rotace. Transformace se provádí chycením a tažením objektu kružnice. Volnou rotaci lze provádět chycením objektu mimo kružnice. Musí být označen objekt. Velikost manipulátoru se mění se vzdáleností kamery od objektu. S kamerami nelze během manipulace pohybovat. *Klávesa B.*

- **Select Object** – Aplikace se přepne do zobrazovacího módu, kde můžeme hýbat s kamerami a označovat libovolné objekty. Výběr se provede klikem na geometrii objektu (vždy pouze jeden). Úspěšný výběr je signalizován bílým ohraničením tvaru boxu kolem tělesa a výpisem jména objektu v poli *Selection*. Manipulaci lze kdykoliv zrušit klikem na ikonu *Select Object* nebo pravým tlačítkem myši.
- **Delete object –** Smaže označený objekt. *Klávesa Delete.*

Rendering Effects Menu



Funkce tlačítek (zleva):

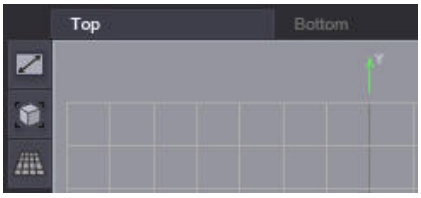
- **Edge Faces** – Efekt zobrazení struktury *mesh* spolu s geometrií ploch. Aplikuje se na všechny objekty ve scéně.
- **Wireframe** – Efekt "Drátěný model" zobrazuje vnitřní strukturu trojúhelníkové sítě. Aplikuje se na všechny objekty ve scéně.
- **Cartoon** – Efekt kresleného zobrazení. Aplikuje se na všechny objekty ve scéně.
- **Anisotropic** – Efekt simuluje nestejné fyzikální vlastnosti povrchu v různých směrech. Například chování některých kovů. Aplikuje se na všechny objekty ve scéně.
- **Specular** – Efekt přidá všem objektům ve scéně maximální odlesky.
- **Bump** – Efekt umožní zobrazit difuzní texturu spolu s texturou normálové mapy. Simuluje hrbolatý povrch podle intenzit v mapované textuře. Aplikuje se na všechny objekty ve scéně.

Views Menu

Funkce tlačítek (zleva):

- **Maximalizme Screen** – Skryje panely menu pro maximalizaci zobrazovací oblasti. *Klávesa F11*.
- **Fit All Views** – Zaměří všechny kamery na scénu tak aby byly vidět, pokud možno, veškeré objekty. Nezáleží na typu projekce u jednotlivých pohledů.
- **Show/Hide Grid –** Skryje/zobrazí mříž, vyznačující počátek souřadnicového systému.
- **Show/Hide Axes –** Skryje/zobrazí osy s popisky, vyznačující orientaci v prostoru.

Camera



Funkce tlačítek (levé menu):

- **Maximalize** – Zvětší náležící výřez na maximální rozměr zobrazovací oblasti. Veškeré panely menu zůstanou viditelné.
- **Fit View –** Zaměří náležící kameru na objekty scény. Nezáleží na aktuální projekci kamery.
- **Orthographic/Perspective –** Přepíná promítání kamery na perspektivní nebo paralelní. Paralelní projekce zachovává rovnoběžnost úseček, nezáleží na vzdálenosti od kamery.

Funkce tlačítek (horní menu):

- **Switch Camera** – Přepnutí pohledu: Seshora/zespoda, zleva/zprava, zepředu/zezadu. Při přepnutí dojde v náležícím okně automaticky k zaměření kamery na objekty scény.

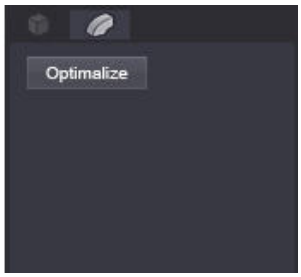
Ovládání kamery *Perspective* pomocí myši:

- **Levé tlačítko** – Rotace kamery kolem scény.
- **Pravé tlačítko** – Přiblížení scény.
- **Obě tlačítka –** Posun kamery v rovině rovnoběžné s průmětnou.

Ovládání kamer *Left, Top, Ftont* pomocí myši:

- **Levé tlačítko** – Posun kamery v rovině pohledu. (Nezáleží na typu projekce)
- **Pravé tlačítko** – Přiblížení scény.

Camera/Modify Menu



Funkce tlačítek (horní menu):

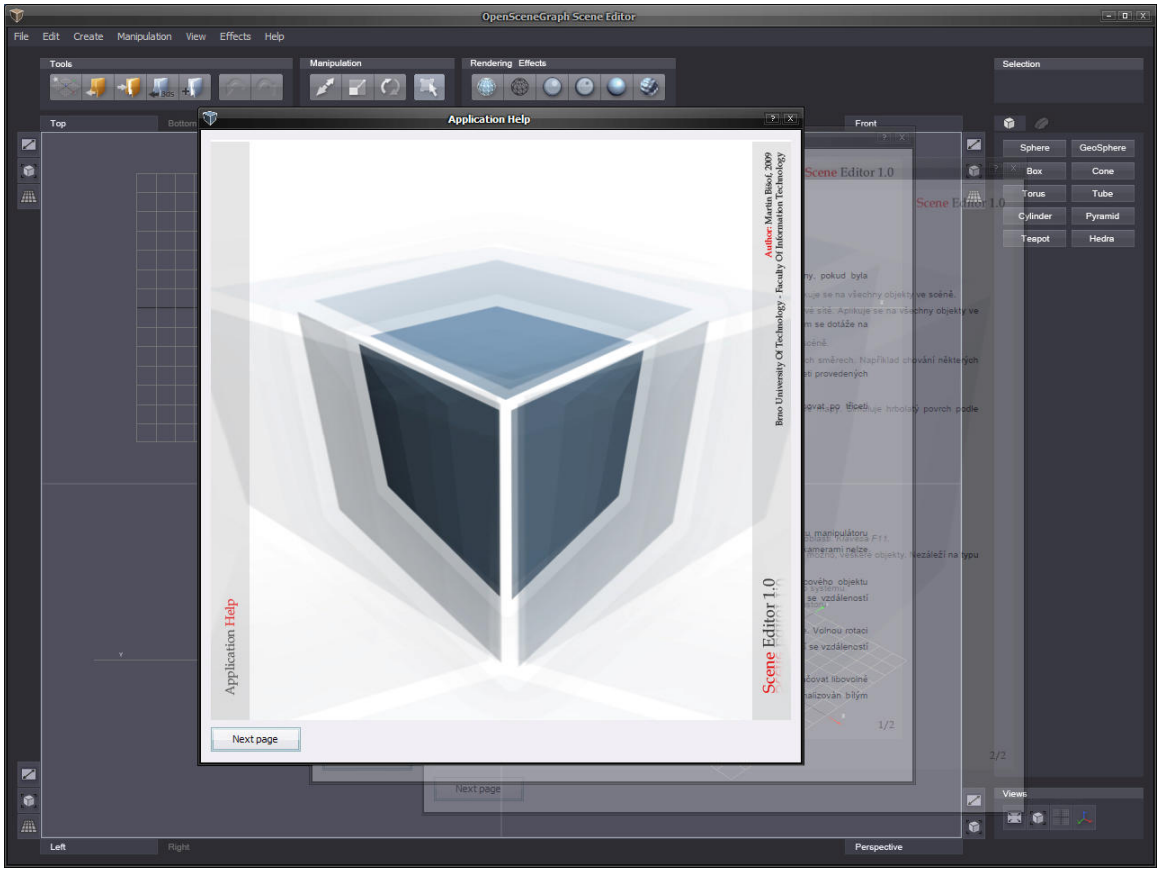
- **Create/Modify** – Přepínání panelů pro vkládání primitiv a nabídky modifikátorů

Funkce tlačítek (kontextové menu):

- **Spere, Box…** – Vložení tělesa do středu scény.
- **Optimalize** – Sníží úroveň detailu trojúhelníkové sítě u označeného objektu. Lze aplikovat opakovaně na veškeré načítané objekty. Výjimkou jsou objekty *Box, Sphere, Cone, Cylinder*. (Vhodná aplikace efektu *Edge Faces* pro vizuální kontrolu modifikace)

Help/About

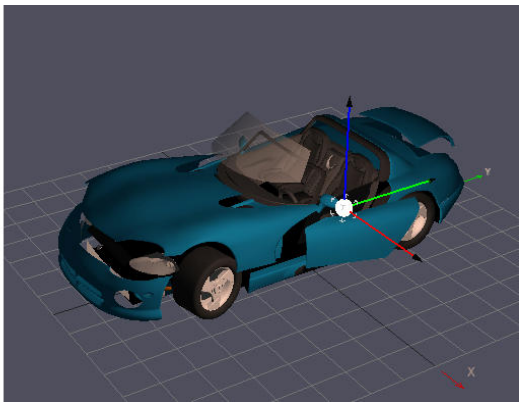
- **Help –** Zobrazí nápovědu aplikace. *Klávesa F1* nebo *Help->Help*
- **Next page –** Přepne na další stranu nápovědy.



Příloha 3. Ukázkové příklady

Ukázka 1. (Transformace a světlo)

- Načtěte soubor viper.osg. *File->Open->viper.osg*
- Provádějte transformace s objekty a posunujte světlem.
- Zkuste označit objekt např. za předním sklem dvojím kliknutím na stejné místo.
- Vyzkoušejte aplikovat modifikátor *Optimalize* na části kapoty. (Doba výpočtu je úměrná složitosti geometrie a hardwarové výbavě – může trvat i několik sekund)

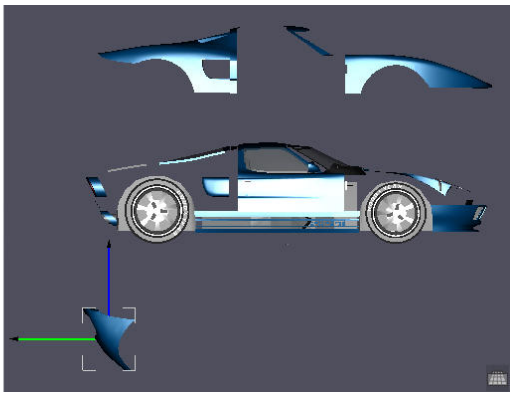


Ukázka 2. (Zaměření na příliš malý objekt)

- Importujte soubor DemeKin0.3ds. *File->Import->DemeKin0.3ds*
- Objekt je příliš malý pro pozorování společně s mřížkou. V pravém spodním panelu (*Views*) klikněte na *Grid* a *Axis*. Tímto skryjete mřížku s popisky os.
- V panelu *Views* zvolte *Fit All Views* a veškeré kamery se vystředí na objekt.
- Podobně můžete experimentovat se soubory *WasureNagusa.3ds* a *Kumanomi0.3ds*.

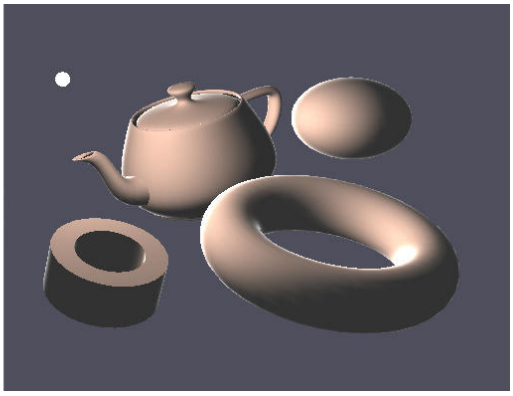
Ukázka 3. (Kamera + příliš velký objekt)

- Importujte soubor Ford GT 40.lwo (je potřeba zvolit v dialogu typ souboru na *.lwo*). *File->Import->Ford GT 40.lwo*
- Objekt je příliš velký pro pozorování, proto v panelu *Views* zvolte *Fit All Views* a veškeré kamery se vystředí na objekt.
- Se zvětšením scény se zvýšila i rychlost pohybu a zoomu kamer. U miniaturních scén (Ukázka 2.) se rychlost naopak sníží.
- Vyzkoušejte posunovat objekty a přepínat ortogonální a perspektivní projekci u kamer. U ortogonální projekce nezáleží na vzdálenosti objektu od kamery a zachovává se rovnoběžnost úseček – přesnost.

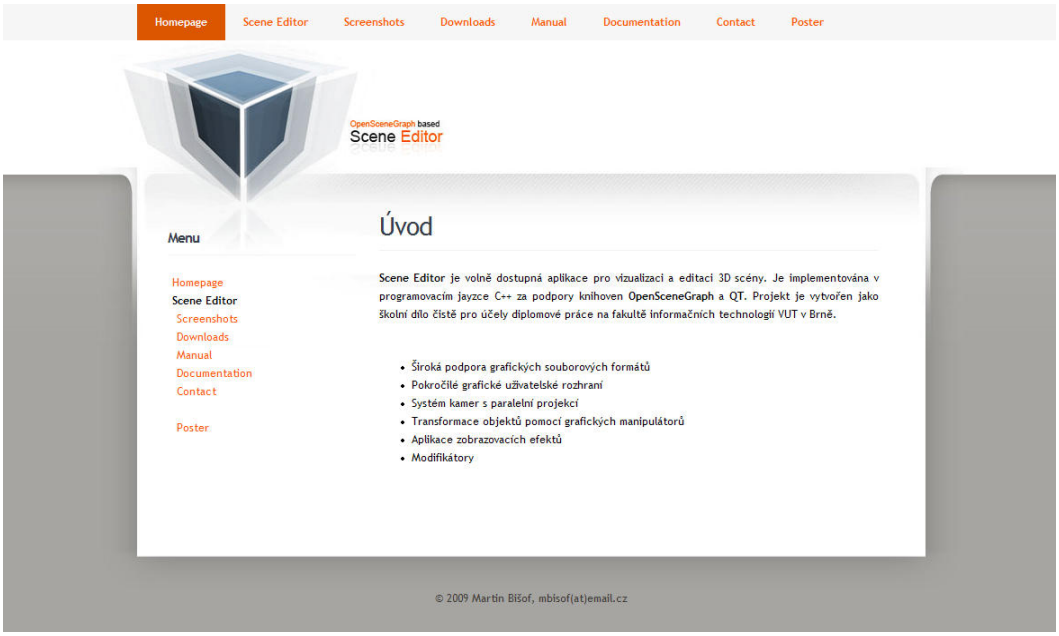


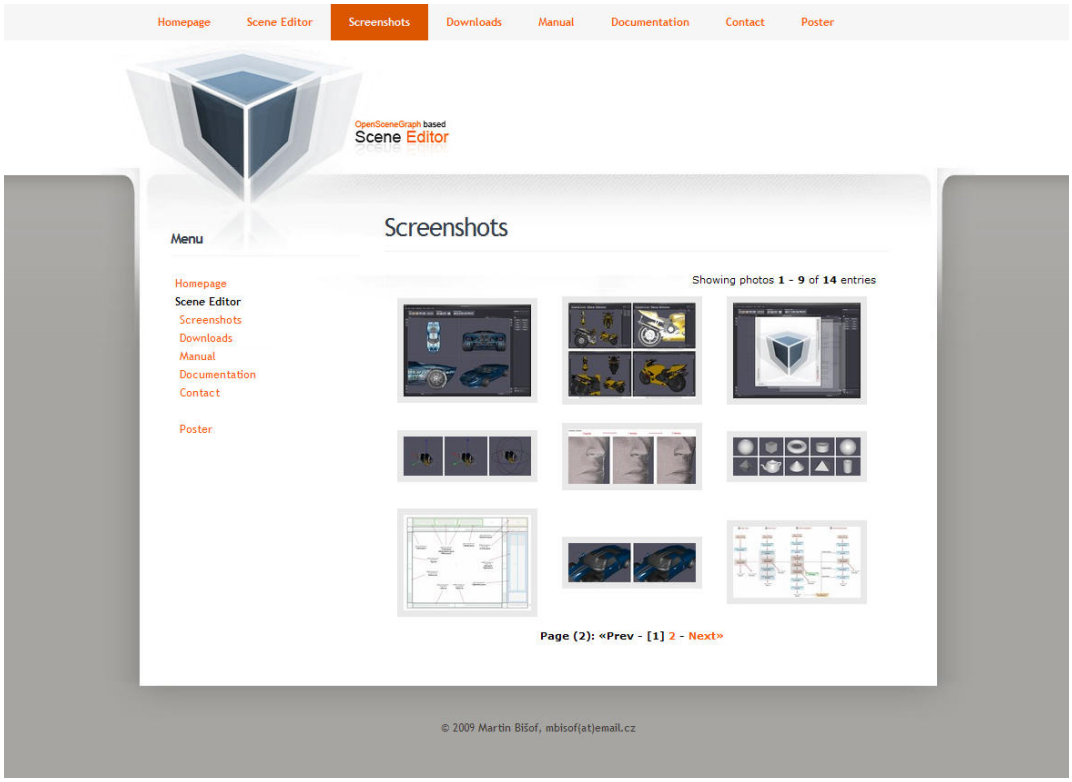
Ukázka 4. (Efekty a primitiva)

- Načtěte si prázdnou scénu. *File->New* nebo *Ctrl + N*
- Vkládejte do scény primitiva a přepínejte zobrazovací efekty v horním panelu *Rendering Effects*.
- Všimněte si, že u použití efektu *caroon* na obyčejnou kouli dochází k nechtěnému artefaktu na severním a jižním pólu kvůli vysokému počtu stýkajících se hran. Objekt *GeoSphere* je strukturován rovnoměrně, proto k vadě nedochází.
- Importujte funkcí *Merge* do stávající scény světlo. *File->Merge. Omni.osg*



Příloha 4. Web prezentace, plakát



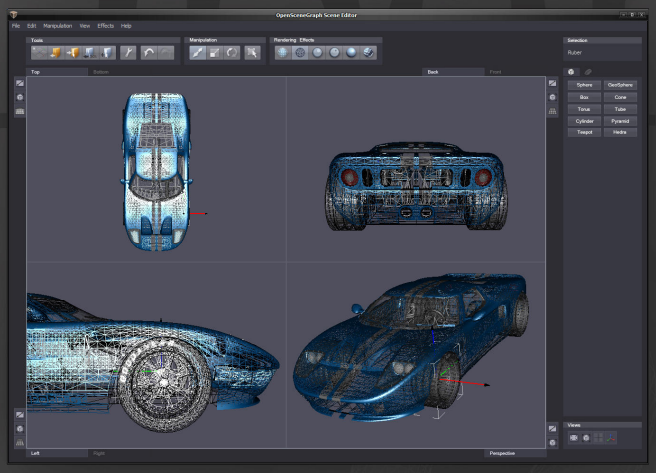


Stránky jsou umístěny na adrese:

- **http://agroprojektjihlava.cz/sceneeditor/**

OpenSceneGraph based
Scene Editor

Autor: Bc. Martin Bišof, 2009, Fakulta informačních technologií, Ústav počítačové grafiky a multimédií, Vysoké učení technické v Brně



Scene Editor 1.0

Aplikace pro vizualizaci 3D objektů a editaci grafu scény
Implementace v C++, knihovna OpenSceneGraph, QT
Website: www.agroprojektjihlava/sceneeditor

Systém kamer

Sedm pohledů do scény pomocí čtyř kamer
Průměty Left/Right, Top/Bottom, Front/Back, Perspective
Otrogonální a perspektivní projekce kamer
Fit to screen - Zaměření kamery na objekty scény
Maximalizace průmětu
Možnost skrýt ovládací panely - maximalizovat výstup
Adaptace zobrazovaní oblasti na velikost okna
Automatická detekce rozlišení displeje při spuštění

Transformace a manipulátory

Selekce objektu - Výpočet a vyznačení jeho hranic + výpis jména
Selekce objektu, který je skryt za jinou geometrií klikem do stejného místa
Move - Grafický manipulátor posunu
Rotate - Grafický manipulátor rotace - rotace podle os nebo libovolně
Scale - Grafický manipulátor neuniformní změny měřítka
Automatická změna velikosti manipulátoru (Autotransform)

Grafické uživatelské rozhraní

Implementace GUI pomocí toolkitu Nokia QT 4.3
Originální design, navržený v Adobe Photoshop
Import externích souborových formátů
(.3ds, .lwo, .obj, .md2, .geo, .iv, .dw, .osg)
Ukládání scény do formátu .osg
Přidávání (Merge) modelů do stávající scény
Nápověda v prostředí aplikace
Klávesové zkratky na často používané funkce

Efekty

Wireframe - drátěný model
Edge Faces - kombinace geometrie + struktura
Cartoon efekt - kreslené objekty
Anisotropic efect
Specular efect
Bump efect

Primitiva a modifikace

Sphere, Geosphere
Box, Pyramid, Hedra
Cylindet, Tube, Cone
Teapot, Torus

Modifikátory, ostatní

Optimalize - Redukuje složitost trojúhelníkové sítě
Grid - vyznačuje počátek světového souřadnicového systému
Axes - Vyznačují směr příslušných os, popisky se natáčejí automaticky ke kameře

Martin Bišof, Fakulta informačních technologií, Ústav počítačové grafiky a multimédií, Vysoké učení technické v Brně