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

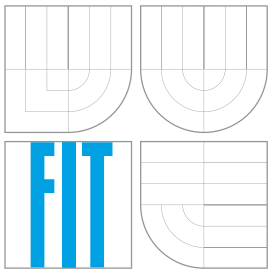
INTERAKTIVNÍ MANIPULACE S 3D OBJEKTY VE VIRTUÁLNÍM PROSTORU S VYUŽITÍM 3D SKENERU MICROSCRIBE – 3D MYŠ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAN BĚLÍN
AUTHOR

BRNO 2007

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

INTERAKTIVNÍ MANIPULACE S 3D OBJEKTY VE VIRTUÁLNÍM PROSTORU S VYUŽITÍM 3D SKENERU MICROSCRIBE – 3D MYŠ

INTERACTIVE MANIPULATION WITH 3D OBJECTS IN VIRTUAL SPACE USING 3D DIGITIZER MICROSCRIBE – 3D MOUSE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN BĚLÍN
AUTHOR

VEDOUCÍ PRÁCE — Ing. PŘEMYSL KRŠEK, Ph.D.
SUPERVISOR

BRNO 2007

Abstrakt

Tato bakalářská práce se zabývá zpracováním manipulátoru ke skeneru MicroScribe, který se v současné době nachází na Ústavu počítačové grafiky Fakulty informačních technologií VUT v Brně. Obsahem této práce je nejprve obeznámit čtenáře s teorií maticí, trojrozměrných transformací a potom vysvětlit způsob komunikace se skenerem MicroScribe. V závěrečné části práce se čtenář dozví způsob návrhu a implementace samotného manipulátoru skeneru.

Klíčová slova

Matice, trojrozměrné transformace, manipulátor, skener MicroScribe

Abstract

This bachelor's thesis is concerned with making manipulator for digitizer MicroScribe, that can be in present found at Department of computer graphics and multimedia at Faculty of information technology at Brno University of technology. Reader becomes acquainted with theory of matrixes and three-dimensional transformations at the beginning and then the communication with digitizer MicroScribe is explained. At last parts of this thesis is reader introduced into concept and implementation of manipulator itself.

Keywords

Matrixes, three-dimensional transformations, manipulator, digitizer MicroScribe

Citace

Jan Bělín: Interaktivní manipulace s 3D objekty
ve virtuálním prostoru s využitím
3D skeneru MicroScribe – 3D myš, bakalářská práce, Brno, FIT VUT v Brně, 2007

Interaktivní manipulace s 3D objekty ve virtuálním prostoru s využitím 3D skeneru MicroScribe – 3D myš

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Přemysla Krška, Ph.D.. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Bělín
15. 5. 2007

Poděkování

Děkuji Ing. Přemyslu Krškovi, Ph.D. za ochotu a za cenné rady při vytváření manipulátoru.

Zadání bakalářské práce/5309/2006/xbelin00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Bělín Jan**
Obor: Informační technologie
Téma: **Interaktivní manipulace s 3D objekty ve virtuálním prostoru s využitím 3D skeneru MicroScribe - 3D mys**
Kategorie: Počítačová grafika

Pokyny:

1. Stručně prostudujte problematiku práce s 3D objekty ve virtuálním 3D prostoru prostřednictvím 3D transformací
2. Prostudujte dodané API pro práci s 3D skenerem MicroScribe (www.immersion.com/digitizer), který je k dispozici na UPGM
3. Navrhněte programový systém pro interaktivní manipulace s 3D objekty ve virtuálním prostoru s využitím 3D skeneru MicroScribe
4. Implementujte navržený programový systém formou DEMO aplikace
5. Zhodnoťte dosažené výsledky a stanovte další vývoj projektu

Literatura:

1. Žara J., Beneš B., Felkel P.: Moderní počítačová grafika. 1. vyd. Praha, Computer press 1998, 448 s., ISBN 80-7226-049-9

Při obhajobě semestrální části projektu je požadováno:

- Splňte první tři body zadání

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kršek Přemysl, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Jan Bělín**
Id studenta: 84440
Bytem: Pomněnková 504, 763 14 Zlín
Narozen: 13. 09. 1984, Zlín
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   bakalářská práce

Název VŠKP: Interaktivní manipulace s 3D objekty ve virtuálním prostoru s využitím 3D skeneru MicroScribe - 3D mys
Vedoucí/školitel VŠKP: Kršek Přemysl, Ing., Ph.D.
Ústav: Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Obsah

Kapitola 1

Úvod

V moderních počítačových aplikacích se s počítačovou grafikou setkává každý uživatel i programátor. Grafické zpracování uživatelského rozhraní je důležitou součástí aplikací a proto se klade velký důraz na formu tohoto rozhraní. Tvůrci aplikací se snaží, aby rozhraní bylo pro uživatele nejintuitivnější a nejpřehlednější. Tento přístup je ještě umocněn v aplikacích pracujících s trojrozměrným virtuálním prostorem. S virtuálním prostorem se dnes pracuje stále více.

Trojrozměrný přístup zobrazení objektů se používá nejenom v zábavním průmyslu (především počítačové hry), ale uplatňuje se stále častěji například v medicíně, strojírenství, stavebnictví nebo v meteorologii. Důvodem k přechodu na trojrozměrnou reprezentaci objektů je hlavně názornost. S názornosti bohužel souvisí i technická náročnost při tvorbě aplikací. Teoretické základy trojrozměrné grafiky tvoří zobecnění matematických postupů z dvourozměrné grafiky a na těchto matematických postupech potom staví technické vybavení (hardware).

Jestliže se vyvíjí programové zázemí (software) pro virtuální prostor, souběžně s ním se vyvíjí a vytváří nové technické vybavení. Jedním z moderních kusů hardwaru je bezpochyby skener MicroScribe G2X, který je používán na Fakultě informačních technologií VUT v Brně. Tento skener je zkonstruován pro využití v různých odvětvích lidského počínání, které pracuje s virtuálním prostorem. Tato práce obsahuje popis tvorby programového spojení mezi aplikací a skenerem MicroScribe. Díky tomuto spojení bude potom možné využít skener při práci s trojrozměrnými objekty. Vytvářené programové spojení je nazváno manipulátorem. Pomocí manipulace je možné modifikovat vlastnosti objektu v reálném čase a tím názorně pracovat s objekty.

V první kapitole této technické zprávy se nejdříve seznámíte s matematickou teorií pro reprezentaci a manipulaci trojrozměrných objektů. V teoretické části budete uvedeni do problematiky matic, maticových operací a základních kamenů grafických systémů. Obsaženy jsou základy trojrozměrných transformací, které využívají pro svoji reprezentaci právě matice a úvod do vytváření grafu scény objektů trojrozměrných systémů.

Další částí zprávy je popis návrhu komunikace se skenerem MicroScribe. Uvedeny jsou zde standardní struktury a funkce knihovny skeneru i způsoby komunikace se skenerem. Druhou část této kapitoly tvoří návrh samotného manipulátoru, který byl cílem této práce. Tato část obsahuje stručný popis tříd pro komunikaci se skenerem, jejich datovou část a metody s krátkými popisy funkčnosti.

Následuje kapitola v níž je obecně popsána implementace knihovny manipulátoru a nástin tvorby ukázkové aplikace.

Obsah této technické zprávy navazuje na semestrální projekt. Tato práce rozvíjí teoret-

ické základy trojrozměrných transformací a ukazuje jejich aplikaci v praktickém příkladu tvorby knihovny manipulátoru pro skener MicroScribe.

Kapitola 2

Teoretický rozbor

2.1 Matice

Teorie matic [3] tvoří úvod a základ lineární algebry[1]. V matematice se matice společně s determinanty [7] aplikují při řešení soustav lineárních rovnic. V počítačové grafice se matice hojně využívají při vyjádření transformací (dále část 2.2), protože toto uspořádaní podporují i moderní grafické procesory.

Matice je možno definovat [7] následovně:
Matice A $= (a_{ij})$ typu m/n nad množinou $X \neq \emptyset$ je schéma složené z $m \cdot n$ prvků množiny X zapsaných do m řádků a n sloupců. Přesněji matice $\mathbf{A}$ typu m/n nad X je zobrazení množiny $\{1, \ldots, m\} \times \{1, \ldots, n\}$ do množiny X.

Množina X bývá často číselná, zj. $X \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}\}$. Prvky matice mohou být ale i komplikovanější objekty, například algebraické výrazy, nebo funkce.

Jednoduše je možno definovat matice jako schématické uspořádání objektů – prvků matice (nebo také elementů matice) do m řádků a n sloupců. Takové matice potom označujeme jako matice typu $m \times n$ nebo m/n.

Matici typu $m \times n$ zapisujeme ve tvaru

$$\mathbf{A} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix},$$

nebo krátce ve tvaru $\mathbf{A} = (a_{ij})$, kde i je řádkový index a j je sloupcový index. Pro názornou ukázku je zde matice

$$\mathbf{A} = \begin{pmatrix} 1 & 7 & 3 \\ 2 & 6 & 4 \end{pmatrix}.$$

Matice $\mathbf{A}$ je typu 2×3 nad množinou $\mathbb{N}$. Platí například, že prvek $a_{23} = 4$, protože tento prvek leží ve druhém řádku a třetím sloupci matice $\mathbf{A}$.

2.1.1 Názvosloví matic

Aby nedocházelo k omylům v termínech používaných při práci s maticemi.

- Je-li $m = n$, nazývá se matice **čtvercová**.

[1] Lineární algebra je odvětvím matematiky, která se zabývá vektory, vektorovými prostory, soustavami lineárních rovnic a lineárními transformacemi [2]

- V obecném případě $m \neq n$ je matice **obdélníková**.
- Množina všech prvků se stejným řádkovým a sloupcovým indexem se nazývá **hlavní diagonála** matice.
- **Nulová matice O** je matice, jejíž všechny prvky jsou nuly.
- **Jednotková matice E** je čtvercová matice, jejíž prvky mimo hlavní diagonálu jsou nuly a prvky na hlavní diagonále jsou rovny jedné.
- Matice **A** se nazývá **trojúhelníková** matice, přesněji **dolní trojúhelníková**, pokud pro libovolné dva indexy i, j platí $i > j \Rightarrow a_{ij} = 0$. Dolní trojúhelníková matice má nuly pod hlavní diagonálou.
- Analogicky je definována **horní trojúhelníková matice**, která má nuly nad hlavní diagonálou.
- Dvě matice $\mathbf{A}, \mathbf{B}$ se rovnají, když mají stejný typ a pro libovolné indexy i, j platí $a_{ij} = b_{ij}$. Pak píšeme $\mathbf{A} = \mathbf{B}$.

2.1.2 Operace s maticemi

Operace nad maticemi jsou velmi jednoduché.

Násobení matice číslem – každou matici $\mathbf{A}$ typu $m \times n$ lze vynásobit prvkem $c \in X$. Výsledkem $c\mathbf{A}$ je matice $\mathbf{C} = (c_{ij})$ typu $m \times n$, kde

$$c_{ij} = c \cdot a_{ij}. \tag{2.1}$$

Sčítaní matic – matice $\mathbf{A}, \mathbf{B}$ lze sečíst, když mají stejný typ $m \times n$. Pak výsledek $\mathbf{A} + \mathbf{B}$ je matice $\mathbf{C} = (c_{ij})$ typu $m \times n$, kde

$$c_{ij} = a_{ij} + b_{ij}. \tag{2.2}$$

Odečítání matic – odečítání matice $\mathbf{A}, \mathbf{B}$ lze pak definovat pomocí vztahů 2.2 a 2.1.

$$\mathbf{A} - \mathbf{B} = \mathbf{A} + (-1)\mathbf{B} \tag{2.3}$$

Násobení matic – pro násobení matic platí komplikovanější vztahy. Přesně dvě matice $\mathbf{A}, \mathbf{B}$ lze vynásobit v tomto pořadí, tj. vytvořit součin $\mathbf{A} \cdot \mathbf{B}$, když typy matic na sebe navazují v následujícím smyslu: pokud typ $\mathbf{A}$ je $m \times k$, typ $\mathbf{B}$ je $k \times n$, pak typ $\mathbf{A} \cdot \mathbf{B}$ je $m \times n$. Výsledkem násobení je tedy matice $\mathbf{C} = c_{ij}$ typu $m \times n$, přičemž platí

$$c_{ij} = \sum_{s=1}^{k} a_{is} b_{sj}. \tag{2.4}$$

Prvek ležící v i. řádku a j. sloupci výsledné matice tedy získáme tak, že procházíme i. řádek v matici $\mathbf{A}$ a jeho prvky postupně násobíme prvky ležícími v j. sloupci matice $\mathbf{B}$ a vytvořené součiny sečteme.

Transponování matice – libovolnou matici $\mathbf{A} = a_{ij}$ typu $m \times n$ lze transponovat. Výsledkem transpozice je matice $\mathbf{A}^T = aji$ typu $n \times m$.

Inverzní matice – nechť $\mathbf{A} = a_{ij}$ je matice typu $n \times n$. Čtvercová matice $\mathbf{B}$ typu $n \times n$ se nazývá **inverzní** k matici $\mathbf{A}$, když

$$\mathbf{A} \cdot \mathbf{B} = \mathbf{B} \cdot \mathbf{A} = \mathbf{E}. \tag{2.5}$$

Jestliže je matice $\mathbf{B}$ inverzní maticí k matici $\mathbf{A}$, pak se tato inverzní matice značí jako matice $\mathbf{A}^{-1}$.

2.2 Transformace

Následující část je převzata z knihy Moderní počítačová grafika [9].

Geometrické transformace jsou jedněmi z nejčastěji používaných operací v počítačové grafice. Transformace je možno rozdělit na lineární a nelineární. Mezi lineární patří otáčení, posunutí, změna měřítka, zkosení a operace vzniklé jejich skládáním. S nelineárními transformacemi se v počítačové grafice setkáváme při složitějších změnách tvaru grafických objektů, např. deformace prostorových modelů nebo warping obrazu. Zvláštní transformací je potom projekce, která převádí objekty z vícerozměrného prostoru do prostoru o méně rozměrech. Nejčastěji se setkáváme s projekcí trojrozměrné scény do roviny obrazu.

Objekty jsou popsány svými souřadnicemi, které jsou vztaženy ke zvolenému souřadnicovému systému. Geometrické transformace mohou být aplikovány na jednotlivé souřadnice objektu, který tak mění svou polohu. Další možností je podrobit transformaci souřadnicový systém. To obyčejně činíme za účelem získání výhodnější reprezentace objektu pro jeho další zpracování.

Dále budeme pracovat s bodem P, který má kartézské souřadnice $[X,\ Y,\ Z]$ ve třech rozměrech. Transformací bodu P získáme bod P' o souřadnicích $[X',\ Y',\ Z']$. Transformací objektu budeme rozumět aplikaci transformace na všechny body, ze kterých se objekt skládá nebo, pokud to transformace a současně reprezentace objektu umožňují, aplikaci transformace na parametry, které objekt jednoznačně určují. Například posunutí koule reprezentované středem a poloměrem nebudeme řešit transformací každého povrchového bodu, stačí pouze transformovat středový bod.

Pro zjednodušení výpočtů transformací se s výhodou používá reprezentace pomocí homogenních souřadnic. Tato reprezentace se používá z několika důvodů. Homogenní souřadnice umožňují vyjádření nejčastěji používaných lineárních transformací pomocí jedné matice, což v nehomogenních kartézských souřadnicích není možné. Skládání transformací se v tomto kontextu realizuje jako násobení matic, inverzní transformace je reprezentována inverzní maticí, atd.

Uspořádaná čtveřice čísel $[x,\ y,\ z,\ w]$ představuje homogenní souřadnice bodu P s kartézskými souřadnicemi $[X,\ Y,\ Z]$ ve třech rozměrech, platí-li:

$$X = \frac{x}{w},\ Y = \frac{y}{w},\ Z = \frac{z}{w},\ w \neq 0$$

Bod P je svými homogenními souřadnicemi určen jednoznačně. Souřadnici w se také nazývá váhou bodu. Často se volí $w = 1$, potom jsou homogenní souřadnice bodu $[X,\ Y,\ Z,\ 1]$. Homogenní souřadnice transformovaného bodu P' s kartézskými souřadnicemi $[X',\ Y',\ Z']$ budeme označovat $[x',\ y',\ z',\ w']$. Rozdíl dvou bodů $A = [a_0,\ a_0,\ a_2,\ 1]$ a $B = [b_0,\ b_1,\ b_2,\ 1]$ určí vektor $\vec{p} = (a_0 - b_0,\ a_1 - b_1,\ a_2 - b_2,\ 0)$, sečtením bodu a vektoru dostaneme bod.

Obecnou matici typu 4×4 reprezentující lineární transformaci bodu $P = [x,\ y,\ z,\ w]$ na bod $P' = [x',\ y',\ z',\ w']$ budeme označovat $\mathbf{A}$, její speciální případy pak podle druhu

transformace, např. **T** (translace), **R** (rotace). Transformaci souřadnic zapíšeme

$$P' = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \mathbf{A}_{4\times4} \cdot P = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

2.3 Trojrozměrné geometrické transformace

Lineární transformace v prostoru jsou zobecněním rovinných transformací. V počítačové grafice se pro transformace používají matice typu 4×4.

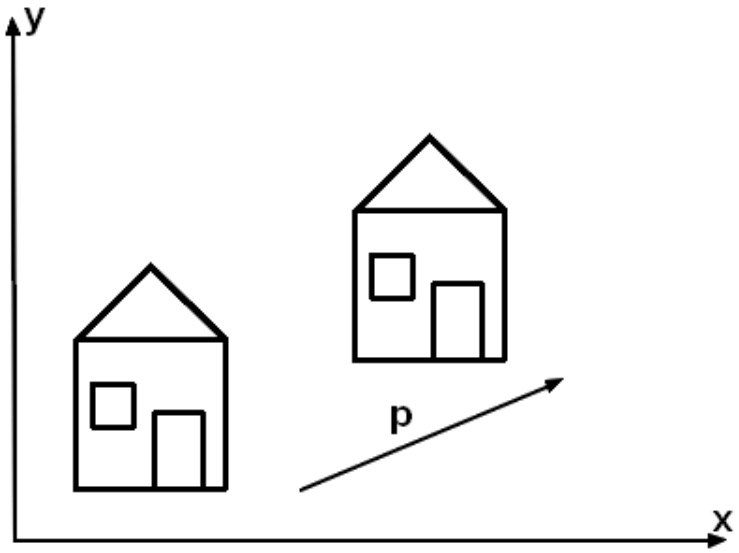
2.3.1 Posunutí

Posunutí je určeno vektorem posunutí $\vec{p} = (X_t,\ Y_t,\ Z_t)$. Posunutí je znázorněno na obrázku 2.1. Transformační matice posunutí **T** má tvar

$$\mathbf{T} = \mathbf{T}(X_t,\ Y_t,\ Z_t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ X_t & Y_t & Z_t & 1 \end{bmatrix}$$

a inverzní matice $\mathbf{T}^{-1}$

$$\mathbf{T}^{-1} = \mathbf{T}(-X_t,\ -Y_t,\ -Z_t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -X_t & -Y_t & -Z_t & 1 \end{bmatrix}$$



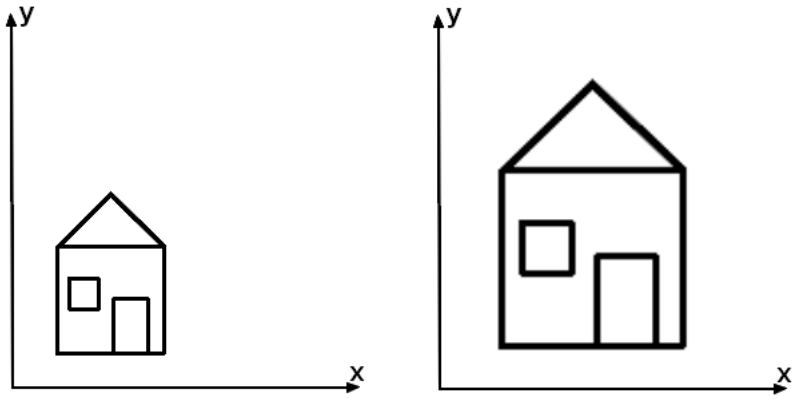
Obrázek 2.1: Posunutí podle vektoru $\vec{p}$

2.3.2 Změna měřítka

Změnu měřítka (scale) v prostoru popisují matice:

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

v níž koeficienty $s_x \neq 0,\ s_y \neq 0$ a $s_z \neq 0$ určují změnu ve směru příslušné souřadnicové osy. Pomocí měřítkových koeficientů můžeme realizovat některou z transformací souměrnosti v prostoru (středovou souměrnost, souměrnost podle roviny a osovou souměrnost). Např. souměrnost podle roviny xy bude realizována pomocí koeficientů $s_x = 1,\ s_y = 1,\ s_z = -1$.



Obrázek 2.2: Změna měřítka s koeficienty $s_x = 1{,}5,\ s_y = 2$

2.3.3 Zkosení

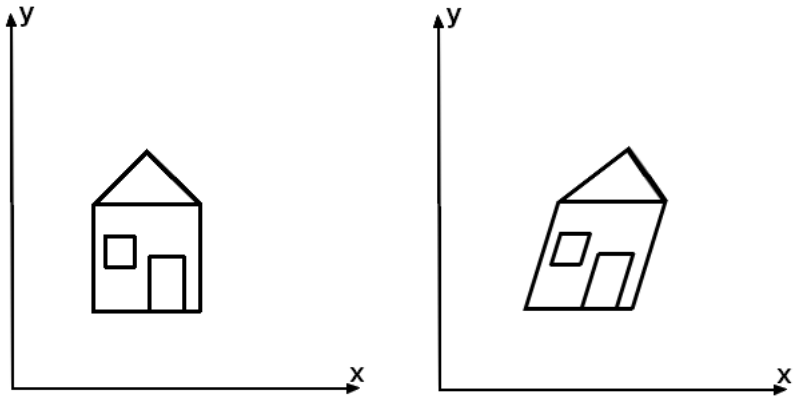
Operaci zkosení (shear) ve třech rozměrech můžeme rozdělit na tři případy zkosení ve směru jednotlivých rovin xy, xz, yz. Ve všech třech případech určují koeficienty sh_x, sh_y a sh_z míru zkosení v odpovídajícím směru. Matice jednotlivých transformací zkosení:

$$\mathbf{Sh}_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ Sh_x & Sh_y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{Sh}_{xz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ Sh_x & 1 & Sh_z & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{Sh}_{yz} = \begin{bmatrix} 1 & Sh_y & Sh_z & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.3.4 Otáčení

Otáčení ve třech rozměrech může být jedním z podpřípadů otáčení kolem jednotlivých souřadnicových os. Matice $\mathbf{R}_x$ reprezentuje otáčení kolem osy x o úhel α a odpovídající inverzní matice:

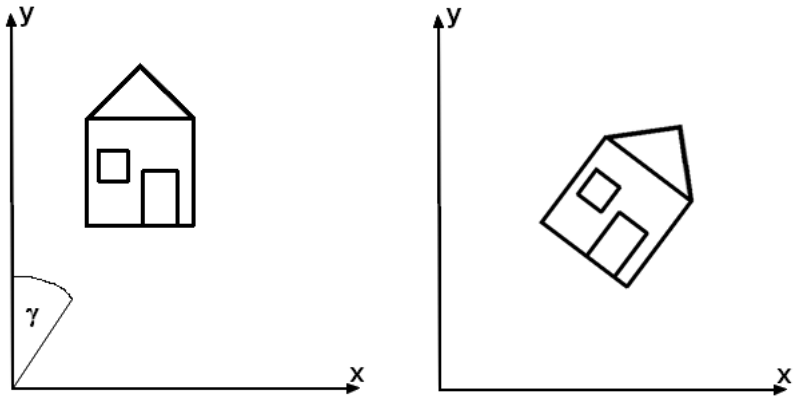
$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{R}_x^{-1}(\alpha) = \mathbf{R}_x(-\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha &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



Obrázek 2.3: Zkosení ve směru osy x

Odpovídajícím způsobem jsou sestaveny matice pro otáčení kolem osy y a z:

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{R}_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Obrázek 2.4: Otočení objektu kolem osy z

2.3.5 Otáčení kolem obecné osy

Otáčení kolem obecné osy v prostoru lze realizovat složením několika dílčích transformací kolem os x, y, z, nalezení příslušných transformací (úhlů otočení) však není jednoduché. Lze však využít Rodriguesovy formule, která předvádí rotační úlohu na promítaní a skládání několika vektorů. Výchozí situace je znázorněna na obrázku 2.5. Předpokládejme, že osa otáčení je určena počátkem souřadnicového O systému a jednotkovým vektorem $\vec{a}$. Polohový vektor $\vec{x}$ transformovaného bodu X je kolem osy této osy otočen o úhel α a výsledný polohový vektor je označen $\vec{x}'$. Za těchto předpokladů lze rotaci popsat vztahem

$$\vec{x}' = \cos\alpha \cdot \vec{x} + (1 - \cos\alpha)(\vec{a}.\vec{x})\vec{a} + \sin\alpha(\vec{a} \times \vec{x}).$$

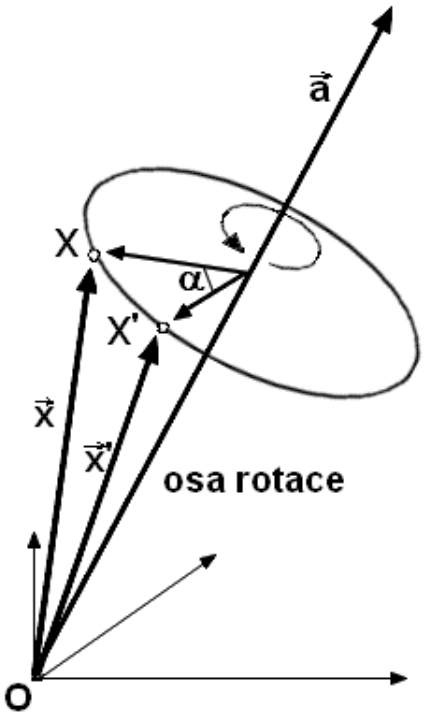
Rotaci bodu X lze přepsat do maticového tvaru s využitím skalárního a vektorového součinu, matice $\mathbf{I}$ je jednotková matice (identita).

$$X' = \mathbf{R}(\vec{a}, \alpha).X.$$

$$\mathbf{R}(\vec{a}, \alpha) = \cos\alpha.\mathbf{I} + (1 - \cos\alpha).\begin{bmatrix} a_x^2 & a_x a_y & a_x a_z & 0 \\ a_x a_y & a_y^2 & a_y a_z & 0 \\ a_x a_z & a_y a_z & a_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \sin\alpha.\begin{bmatrix} 0 & -a_z & a_y & 0 \\ a_z & 0 & -a_x & 0 \\ -a_y & a_x & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Zvolíme-li například osu rotace shodnou s osou x, tj. $\vec{a} = [1,\ 0,\ 0]$, po dosazení do předešlého vztahu dostaneme matici rotace kolem osy x. Obecnou rotaci řešíme obdobně, jako u obecné rotace v rovině, tj. na ose rotace zvolíme bod $P = [P_x,\ P_y,\ P_z,\ 1]$, vypočteme vektor ve směru osy $\vec{a}$, zvolený bod posuneme do počátku souřadnicového systému, otočíme transformované objekty o daný úhel a zpětným posunutím vrátíme výsledek do výchozí pozice. Transformaci matice $\mathbf{A}$ bude složena ze tří základních transformací

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



Obrázek 2.5: Rotace kolem obecné osy

2.3.6 Kvaterniony

Pro reprezentaci rotací nejsou matice vždy nejvýhodnější. Jednak obsahují nadbytečné údaje (devět čísel místo tří hodnot úhlů natočení) a hlavně jsou obtížně interpolovatelné.

Přechod z jedné obecné polohy do jiné pomocí interpolovaného otáčení ve třech směrech nelze pomocí matic jednoduše vyřešit. Kvaterniony byly navrženy irským matematikem W. R. Hamiltonem v 19. století jako analogie komplexních čísel v prostoru. Praktický význam teorie kvaternionů byl rozpoznán nejen v kvantové mechanice, ale i při řešení animačních úloh v počítačové grafice. Podrobnější matematické vlastnosti kvaternionů lze nalézt v [9] nebo v práci [8].

Kvaternion $\mathbf{q}$ je reprezentován čtveřicí $\mathbf{q} = w + xi + yj + zk$, kde $w,\ x,\ y,\ z$ jsou reálná čísla a $i,\ j,\ k$ jsou kvaternionové jednotky (i odpovídá komplexní jednotce). Kvaternion *sdružený* ke kvaternionu $\mathbf{q}$ definujeme jako $\mathbf{q}^* = w - xi - yj - zk$. Velikost kvaternionu $\mathbf{q}$ je $|\mathbf{q}| = \sqrt{w^2 + x^2 + y^2 + z^2}$. Kvaternion jehož velikost je jedna nazýváme *jednotkový* kvaternion. Pro jednotkový kvaternion $\mathbf{q}$ platí $\mathbf{q}^*\mathbf{q} = \mathbf{q}\mathbf{q}^* = 1$.

Kvaternion si můžeme také představit jako dvojici složenou ze skalární (s) a vektorové (v) části. Tento pohled vede k jednoduché notaci

$$\mathbf{q} = (s, \vec{v}).$$

Libovolnou rotaci lze popsat úhlem α a jednotkovým vektorem $\vec{a} = (a_0,\ a_1,\ a_2)$, který reprezentuje osu otáčení. Taková rotace odpovídá jednotkovému kvaternionu

$$\mathbf{q} = \cos(\alpha/2) + a_0 \sin(\alpha/2)i + a_1 \sin(\alpha/2)j + a_2 \sin(\alpha/2)k.$$

Tento zápis se obvykle zkracuje na

$$\mathbf{q} = \cos(\alpha/2) + \mathbf{a} \sin(\alpha/2), \tag{2.6}$$

přičemž $\mathbf{a} = (0,\ \vec{a})$ chápeme jako jednotkový kvaternion se skalární částí $s = 0$, tj. $\mathbf{a} = a_0 i + a_1 j + a_2 k$. Přiřazení jednotkového kvaternionu k rotaci není jednoznačné, neboť $-\mathbf{q}$ odpovídá téže rotaci jako $\mathbf{q}$. Kvaternion $1+0i+0j+0k$ představuje identitu (rotaci s nulovým úhlem). Sdružený kvaternion $\mathbf{q}^*$ reprezentuje inverzní rotaci.

Vektor $\vec{v} = (v_0,\ v_1,\ v_2)$ můžeme chápat jako kvaternion $\mathbf{v}$ s nulovou reálnou částí, tedy $\mathbf{v} = v_0 i + v_1 j + v_2 k$. Otočení vektoru $\vec{v}$ jednotkovým kvaternion $\mathbf{q} = \cos(\alpha/2) + \mathbf{a}\sin(\alpha/2)$ kolem osy $\mathbf{a}$ o úhel α spočítáme pomocí kvaternionového násobení

$$\mathbf{v}' = \mathbf{q}\mathbf{v}\mathbf{q}^*. \tag{2.7}$$

Platí, že reálná část kvaternionu $\mathbf{v}'$ vyjde vždy nulová, tedy $\mathbf{v}' = v'_0 i + v'_1 j + v'_2 k$. To nás opravňuje tvrdit, že vektor $(v'_0,\ v'_1,\ v'_2)$ představuje rotaci vektoru $\vec{v}$ zadanou kvaternionem $\mathbf{q}$.

Vzorec (2.7) má důležitý důsledek pro skládání rotací. Mějme rotaci reprezentovanou jednotkovým kvaternionem $\mathbf{r}$ a otočme jím vektor $\vec{v'}$ reprezentovaný kvaternionem $\mathbf{v}'$. Výsledek otáčení lze napsat jako

$$\mathbf{v}'' = \mathbf{r}\mathbf{v}'\mathbf{r}^* = \mathbf{r}\mathbf{q}\mathbf{v}\mathbf{q}^*\mathbf{r}^*. \tag{2.8}$$

Vidíme že je to totéž, jako kdybychom vektor $\vec{v}$ otočili pomocí jednotkového kvaternionu $\mathbf{rq}$. Můžeme tedy shrnout: složení rotací odpovídá násobení kvaternionů.

Výpočet otočení vektoru podle rovnice (2.7) je pomalejší, než násobení vektoru rotační maticí. Potřebujeme tedy převod mezi kvaterniony a rotačními maticemi. Je důležitý i proto, že některé knihovny a grafický hardware používají rotace v maticovém tvaru. Převod jednotkového kvaternionu $\mathbf{q} = w + xi + yj + zk$ na rotační matici $\mathbf{R}$ je snadný:

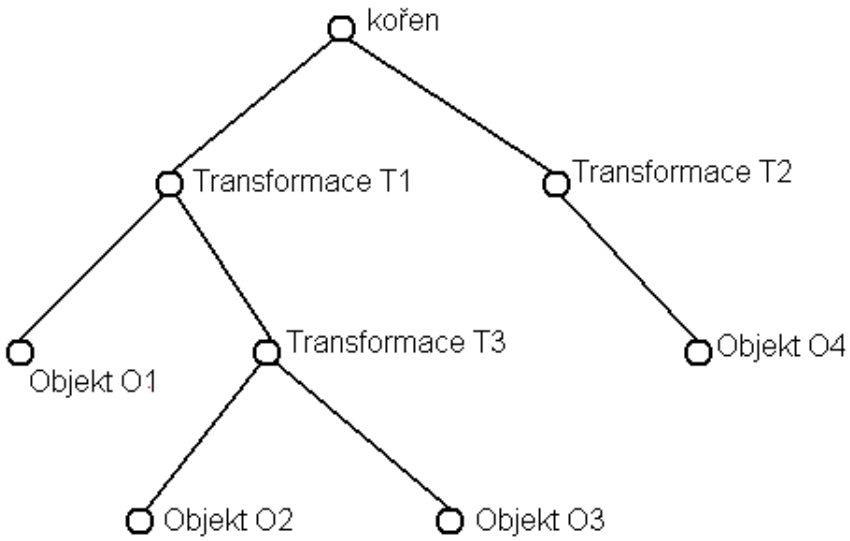
$$\mathbf{R}(\mathbf{q}) = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2wz & 2xz + 2wy & 0 \\ 2xy + 2wz & 1 - 2x^2 - 2z^2 & 2yz - 2wx & 0 \\ 2xz - 2wy & 2yz + 2wx & 1 - 2x^2 - 2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

2.4 Graf scény

Scénou nazýváme množinu prostorových objektů doplněnou dalšími informacemi potřebnými pro jejich zobrazení. Přestože se zdá, že vytvoření scény je poměrně jednoduchým závěrečným krokem po předchozím vymodelování individuálních objektů, je tvorbu prostorové scény možno chápat jako samostatnou úlohu. Zatímco systémy pro modelování těles zpracovávají geometrická data definující tvar jednotlivých objektů, systémy pro tvorbu scén přidávají k objektům transformace (2.2) do jejich cílové polohy, určují informace potřebné pro zobrazování (světla, kamery) a především umožňují do scény opakovaně vkládat stejně nebo podobně vypadající objekty (instance). Scéna obvykle obsahuje:

- nezobrazované objekty – kamery, osvětlení scény
- zobrazované objekty – jejich geometrie, barevné vlastnosti, textury
- prvky definující logickou strukturu scény – definice skupin a jejich instancí
- transformace – definované hierarchicky kvůli snadnější manipulaci s objekty

Více detailů o jednotlivých položkách scény je možno nalézt v knize [9].



Obrázek 2.6: Schéma jednoduchého grafu scény

Tělesa a další zobrazované objekty je vhodné uspořádat do datové struktury, která umožňuje seskupovat logicky k sobě patřící části, efektivně je transformovat a jejich instance vkládat úsporným způsobem do prostoru scény. Tato struktura se obecně nazývá *graf scény* (obrázek 2.6).

Graf scény je n-ární strom, tj. takový graf, v němž lze pro každý uzel nalézt právě jednoho předchůdce. Výjimku tvoří kořen stromu, který stojí na nejvyšší úrovni. Graf scény může obsahovat i několik stromů, tzv. les. Graf scény není ve všech systémech definován stejným způsobem, odlišnosti jsou v typech uzlů, v pravidlech pro stavbu stromu i pro interpretaci dat ve stromu uložených. Dále jsou uvedeny principiální vlastnosti grafů scény bez ohledu na systém.

Důležitou vlastností stromu je schopnost vyjádřit vztahy mezi uzly. Jedním z těchto vztahů je *dědičnost*. Umožňuje v jednom místě stromu definovat vlastnost, která bude platná pro řadu dalších uzlů. Rozsah platnosti je dán vzájemnou polohou uzlů v rámci stromu. Lze například stanovit, že vlastnost definovaná v uzlu je platná pro všechny následníky tohoto uzlu. Pravidla pro dědění mohou být být definovány různými způsoby.

V rámci této části se soustředíme pouze na význam transformací v grafu scény. Je zřejmé, že každé těleso může být pevně umístěno do své cílové pozice, tj. veškeré souřadnice tělesa mohou být předem transformovány pomocí některé z trojrozměrných transformací (viz část 2.3). Pro manipulaci se scénou je však mnohem výhodnější ponechat těleso v jejich základních polohách (lokálních souřadnicových systémech) a potřebné transformace zapsat do grafu scény, například v podobě transformační matice. Hierarchické uspořádaní scény umožní transformace skládat a změnou jedné transformace ovlivnit celý podstrom.

Na obrázku (2.6) grafu scény jsou některé uzly označeny transformace Tx. Názorný příklad skládání transformací v grafu scény si uvedeme právě pro toto schéma. Objekt $O1$ bude ovlivněn pouze transformací $T1$. Objekty $O2$ a $O3$ budou ovlivněny složením transformací $T1.T3$ a objekt $O4$ bude opět ovlivněn pouze transformací $T2$. Jestliže ovšem změníme transformaci $T1$, ovlivníme výsledné transformace pro objekty $O1$, $O2$ a $O3$.

Kapitola 3

Návrh manipulátoru

3.1 Skener MicroScribe G2X

Obrázek 3.1: Skener MicroScribe G2X na Fakultě informačních technologií

V současné době společnost Immersion [5] vyrábí několik verzí skeneru G2. Skenery MicroScribe jsou určeny pro digitalizaci a měření trojrozměrných objektů. Skener MicroScribe lze nazvat 3D myší, ale již na první pohled se skener liší nejenom svou konstrukcí, ale i svými technickými možnostmi snímání prostoru. Narozdíl od klasické myši, která snímá svoji polohy pouze ve dvou směrech (osy x a y), umožňuje skener MicroScribe získávat polohu snímacího hrotu ve směru tří os a navíc je možné získat i orientaci (natočení) hrotu (stylus). Takové parametry skeneru dovolují získávat přesné souřadnice a úhly pro manipulaci s objekty v grafických prostředích, kde se pracuje s trojrozměrnými modely.

Společnost Immersion vyrábí několik řad skeneru MicroScribe. Skenery s označením G2 jsou řadou s nejmenší přesností snímání určené pro digitalizaci a tvorbu objektů nebo právě pro grafické aplikace. Řada MX je potom určena již pro profesionální použití ve strojírenství, kde se očekává velmi vysoká přesnost snímání. Nejnovější modelovou řadou je

potom verze MicroScribe X, která je vysoce profesionálním nástrojem v oboru digitalizace a měřeni objektů.

Ke skeneru MicroScribe G2X se dodává software ve formě knihovny `ArmDll32`, která umožňuje využít funkce v implementaci programu v jazyce C/C++ a komunikovat se skenerem a získávat data. Získaná data lze dále použít podle typu a zaměření programu.

3.1.1 Datové struktury

Knihovna skeneru `ArmDll32` obsahuje dvě základní datové struktury, díky kterým je možné získávat souřadnice snímacího hrotu. Obě struktury obsahují tři složky typu `float`. Podrobnější významy jednotlivých složek jsou popsány v následujícím výčtu:

`length_3D` – pomocí této struktury je možné získat souřadnice polohy snímacího hrotu skeneru. Přibližné rozložení souřadnicového systému skeneru je znázorněno na obrázku 3.2.
Složky struktury:

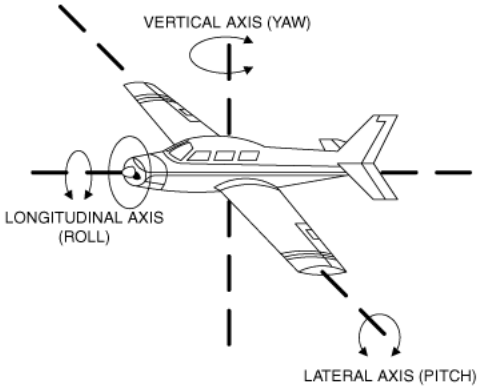
`.x`	souřadnice polohy snímacího hrotu ve směru osy x
`.y`	souřadnice polohy snímacího hrotu ve směru osy y
`.z`	souřadnice polohy snímacího hrotu ve směru osy z

Obrázek 3.2: Orientace os při snímání skenerem

`angle_3D` – struktura obsahuje natočení snímacího hrotu. Skener MicroScribe používá pro popis orientace snímacího hrotu Tait-Bryanovy úhly [4], často také značené jako Roll–Pitch–Yaw rozložení. Tait-Bryanovy úhly se používají například v letectví, kde se

takto značí rotace letounu (obrázek 3.3). Rozložení Tait-Bryanových úhlů odpovídá klasickému rozložení Eulerových úhlů, tj. úhly otočení kolem souřadných os x, y a z. Složky struktury:

`.x`	úhel otočení kolem osy x, který odpovídá úhlu Roll
`.y`	úhel otočení kolem osy y, který odpovídá úhlu Pitch
`.z`	úhel otočení kolem osy z, který odpovídá úhlu Yaw



Obrázek 3.3: Orientace úhlů Roll–Pitch–Yaw

Knihovna `ArmDll32` má však ještě jednu důležitou datovou strukturu `arm_rec`. Tato struktura je uživateli skryta a uživatel s ní přímo nepracuje. Struktura `arm_rec` je vnitřní datovou kostrou knihovny, která si do ní ukládá všechny potřebné informace o poloze a orientaci snímacího hrotu, data vztahující se k nastavení spojení a skeneru a vnitřní proměnné knihovny. Uživatel získává všechny souřadnice právě z této struktury skrze funkce knihovny.

3.1.2 Funkce

Knihovna `ArmDll32` obsahuje funkce pro navázání, průběh a ukončení komunikace se skenerem. Pro více informací o ostatních funkcích a možnostech knihovny skeneru hledejte v uživatelském manuálu [6]. Následující funkce byly použity při implementaci manipulátoru.

`ArmStart(HWND hWnd)` – připraví knihovnu `ArmDll32` ke komunikaci. Funkce připraví vlákno pro čtení dat ze skeneru. Parametr `hWnd` je Window handler okna, pro které bude skener posílat data skrze Windows zprávu `ARM_MESSAGE`. Pokud se parametr rovná `NULL`, skener nebude posílat zprávy a komunikace se skenerem bude probíhat dotazováním (více část 3.2).

`ArmConnect(int port, long baud)` – detekuje a nastaví spojení se skener. Nejprve se vždy kontrolují USB porty a hledá se spojení se skenerem. Pokud se oba parametry rovnají nule, kontrolují se postupně všechny sériové porty a jestliže je skener nalezen, pokusí se funkce připojit na frekvenci 115 200 baudů. Uživatel však může přímo zadat číslo sériového portu a frekvenci spojení.

`ArmSetUpdate(int type)` – určuje pomocí parametru `type`, které souřadnice je nutné při spojení se skenerem aktualizovat. V implementaci manipulátoru je použita hodnota `ARM_FULL`, která znamená aktualizaci polohy i orientace.

`ArmGetTipPosition(length_3D *position)` – získá souřadnice polohy snímacího hrotu do struktury `position` typu `length_3D`.

`ArmGetTipOrientation(angle_3D *orientation)` – získá souřadnice natočení snímacího hrotu do struktury `orientation` typu `angle_3D`.

`ArmGetButtonsState(DWORD *button)` – získá bitovou mapu stavu tlačítek skeneru. Stav tlačítek je zapsán do hodnoty `button` typu `DWORD` nebo `unsigned long`. Posloupnost tlačítek je následující: bit 0 = tlačítko 1 a bit 1 = tlačítko 2.

`ArmDisconnect()` – ukončí spojení s porty, tudíž se skenerem.

`ArmEnd()` – má stejnou funkčnost jako předchozí `ArmDisconnect()`, ale navíc i ukončí vnitřní vlákno knihovny `ArmDll32`. Tato funkce je doporučeným způsobem, jak ukončit komunikaci a práci se skenerem.

3.2 Způsoby komunikace

Software skeneru MicroScribe dovoluje komunikovat s programovým okolím dvěma způsoby. Jak již bylo naznačeno v popisu funkce `ArmStart()`, je možné komunikovat se skenerem MicroScribe skrze zprávy systému Windows nebo dotazováním, tzv. pollingem.

Polling

Prvním způsobem komunikace je polling. Tato varianta znamená, že uživatel zajistí, aby se uvnitř programu, nejlépe periodicky, volala funkce, která získá potřebné data od knihovny skeneru. Získaná data potom zpracuje a upotřebí v dalších operacích.

Windows zprávy

Komunikace skrze Windows zprávy narozdíl od pollingu nepotřebuje vnitřní funkci, která se periodicky volá, ale musí do systému Windows zaregistrovat zprávy, na které bude reagovat obslužná funkce. Zprávy přicházejí od skeneru pouze v okamžiku, když se změní některý z atributů polohy a orientace skeneru. Obslužná funkce v reakci na zprávu skeneru provede naprogramované operace. Windows zprávy se dají považovat za zautomatizovanou verzi pollingu.

3.3 Knihovna manipulátoru

Manipulátor pro skener MicroScribe je navržen jako knihovna `Arm`, která obsahuje dvě třídy. Pomocí těchto tříd je obsluhována správa komunikace se skenerem a zpracování získaných dat do formy transformačních matic. Pro lepší přehled o jednotlivých třídách a jejich metodách nahlédněte do dokumentace (část B).

3.3.1 Třída `Matrix`

Třída `Matrix` je určena pro uložení transformační matice a obsahuje metody poskytující základní matematické prostředky potřebné při práci s transformačními maticemi.

Proměnné třídy `Matrix`

`data[16]` – pole hodnot typu `float`, které reprezentují matici. Matice je uložena formou jednorozměrného pole, ale pomocí metody `*operator []` je možno pracovat s tímto polem jako s dvourozměrným.

Metody třídy `Matrix`

`Matrix()` – konstruktor třídy. Naplní pole `data` hodnotami jednotkové matice.

`*operator [] (int row)` – umožňuje přístup do pole přes indexy. Pomocí této metody se s polem `data[]` pracuje jako s dvourozměrným polem.

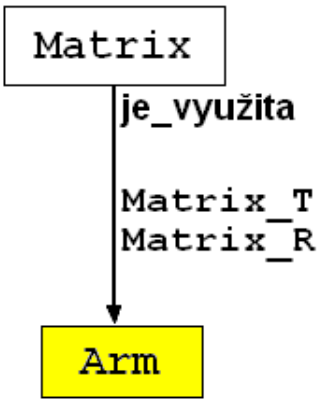
`Clear()` – přepíše hodnoty pole `data[]` opět na hodnoty jednotková matice.

`Print()` – dovoluje vypsat obsah proměnných pole. Výpis je formátován do tvaru matice typu 4×4. Jednotlivé hodnoty jsou vypsány s přesností na dvě desetinná místa.

`Mult(Matrix &m)` – provede vynásobení proměnných hodnotami matice `m`. Prakticky se provede vynásobení interní matice v poli `data[]` maticí `m` a výsledek se uloží zpět do interní matice (pole `data[]`).

3.3.2 Třída `Arm`

Třída `Arm` je implementovaným manipulátorem pro skener MicroScribe. Tato třída obsahuje metody pro navázání komunikace, získávání a zpracování dat, výstup zpracovaných dat a ukončení komunikace se skenerem.



Obrázek 3.4: Znázornění vztahu mezi třídami `Matrix` a `Arm`

Proměnné třídy `Arm`

`Matrix Matrix_T` – Objekt pro uložení vytvořené transformační matice posunutí (translace)

`Matrix Matrix_R` – Objekt pro uložení vytvořené transformační matice otočení (rotace)

`length_3D length` – Struktura pro ukládání souřadnic pozice snímacího hrotu

`angle_3D angle` – Struktura pro ukládání souřadnic orientace snímacího hrotu

`Axis version` – údaj výčtového typu Axis reprezentující pořadí os

Veřejné metody třídy `Arm`

`Arm()` – konstruktor třídy. Nastaví výchozí hodnotu proměnné `version` na hodnotu XYZ.

`int ArmInit(char *string)` – navazuje komunikaci se skenerem. Parametr `string` určuje pořadí os vůči osám skeneru. `String` musí být ukazatelem do tří prvkového pole znaků se znaky `x, y, z`.

`void ArmClearPosition()` – nastaví transformační matici posunutí na jednotkovou matici.

`void ArmClearOrientation()` – nastaví transformační matici otočení na jednotkovou matici.

`void ArmPrintMatrixPosition()` – vytiskne transformační matici posunutí.

`void ArmPrintMatrixOrientation()` – vytiskne transformační matici otočení.

`int ArmInfo()` – vytiskne informace o skeneru. Název a model skeneru, sériové číslo a verzi ovladačů.

`Matrix ArmMatrixPosition()` – vrátí hodnoty transformační matici posunutí do objektu typu `Matrix`.

`Matrix ArmMatrixOrientation()` – vrátí hodnoty transformační matici otočení do objektu typu `Matrix`.

`int ArmPositionStart()` – uloží současnou pozici snímacího hrotu do struktury `length`.

`int ArmPositionUpdate()` – získá současnou pozici snímacího hrotu. Provede rozdíl nové a uložené pozice hrotu. Tyto rozdíly souřadnic předá jako parametry funkci `PositionMatrix` v pořadí podle hodnoty proměnné `version`.

`int ArmOrientationStart()` – uloží současnou orientaci snímacího hrotu do struktury `angle`.

`int ArmOrientationUpdate()` – získá současnou orientaci hrotu. Provede rozdíl nové a uložené orientace snímacího hrotu. Tyto rozdíly souřadnic předá jako parametry funkcím `OrientationMatrix()` v pořadí podle hodnoty proměnné `version`.

`int ArmButton1()` – vrací stav pravého tlačítka skeneru. Hodnota 0 znamená nestisknuté tlačítko a hodnota 1 značí stisknutý stav.

`int ArmButton2()` – vrací stav levého tlačítka skeneru. Hodnota 0 znamená nestisknuté tlačítko a hodnota 1 značí stisknutý stav.

`void ArmOver()` – ukončí komunikaci se skenerem.

Chráněné metody třídy `Arm`

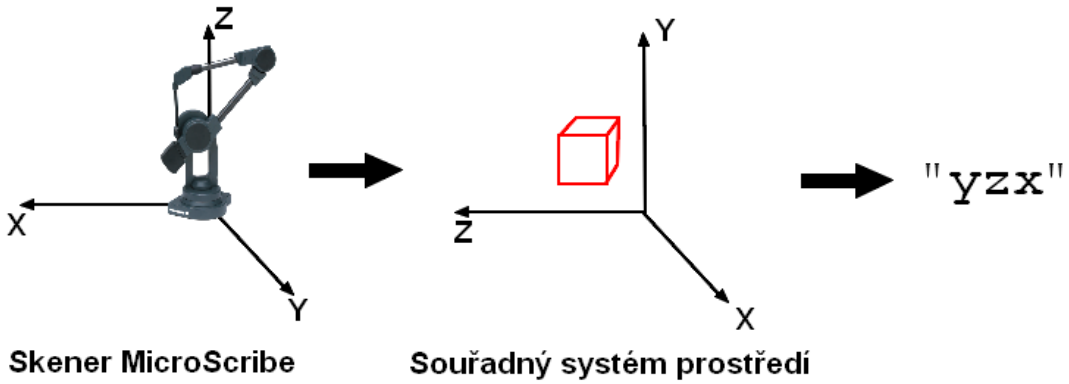
`void PositionMatrix(float par1, float par2, float par3)` – vytvoří z předaných parametrů transformační matici posunutí, kterou vynásobí matici `Matrix_T`.

`void OrientationMatrixX(float par)` – vytvoří z předaného parametru transformační matici otočení kolem osy x, kterou vynásobí matici `Matrix_R`.

`void OrientationMatrixY(float par)` – vytvoří z předaného parametru transformační matici otočení kolem osy y, kterou vynásobí matici `Matrix_R`.

`void OrientationMatrixZ(float par)` – vytvoří z předaného parametru transformační matici otočení kolem osy z, kterou vynásobí matici `Matrix_R`.

Při použití metody `ArmInit()` je nutné předat textový řetězec, který představuje pořadí os souřadnicového systému. Toto pořadí ovlivňuje tvorbu transformačních matic a tudíž i manipulaci s objektem v použitém grafickém systému. Je nutné dobře si představit souřadný systém grafického prostředí a podle tohoto systému přiřadit pořadí souřadnic získávaných ze skeneru.



Obrázek 3.5: Získání pořadí os souřadného systému

Jak znázorněno na obrázku 3.5 souřadný systém skeneru se nemění, ale souřadný systém grafického prostředí nemusí mít a většinou nemá stejně orientovaný souřadný systém. V případě neshody je tedy nutné upravit právě pomocí parametru funkce `ArmInit()` přizpůsobit práci se souřadnicemi. Pokud jsou tedy osy x, y, z grafického prostředí orientovány vůči osám skeneru stejně jako na obrázku 3.5, musíme si uvědomit, které souřadnice skeneru budeme potřebovat v dané ose grafického prostředí. Pořadí grafických os je vždy uvažováno do pořadí x, y, z. V ose x tedy použijeme souřadnice osy y skeneru, v ose y souřadnice osy z skeneru a nakonec v ose z budeme počítat se souřadnicemi osy x skeneru MicroScribe. Tímto logickým postupem získáváme textové pořadí os pro parametr funkce `ArmInit()` ve tvaru `yzx`.

Kapitola 4

Implementace

Manipulátor pro skener MicroScribe je knihovna `Arm` implementovaná v jazyce C/C++. K vytvoření knihovny manipulátoru jsem použil překladač MinGW a společností Immersion dodávanou knihovnu `ArmDll32` skeneru MicroScribe.

4.1 Manipulátor

Při implementaci manipulátoru jsem použil standardní prostředky jazyka C/C++ pro vytvoření tříd `Arm` a `Matrix`.

Třída `Matrix` je převzata ze cvičení předmětu Základy počítačové grafiky. Tato třída byla upravena o některé nové metody, aby její použití bylo intuitivní a efektivní v rámci třídy `Arm`.

Třída `Arm` je objektem zpracovávající data pro práci s trojrozměrnými objekty pomocí skeneru MicroScribe. Při komunikaci se skenerem využívá třída `Arm` funkce a struktury knihovny skeneru `ArmDll32`, blíže popsané v části 3.1.

Při implementaci byly použity funkce `printf()` a `fprintf()` standardní knihovny `stdio.h` pro výpis upozornění a hodnot, dále funkce knihovny `string.h` pro práci s řetězci a matematické funkce `cos()` a `sin()` knihovny `math.h` pro výpočet hodnot úhlů při vytváření rotačních transformačních matic.

Datovým výstupem třídy `Arm` jsou dvě transformační matice `Matrix_T` a `Matrix_R`. Jejich obsah je možné získat do objektů třídy `Matrix` skrze metody `ArmMatrixPosition()` a `ArmMatrixOrientation()`.

Příklad jednoduchého získání a vypsání matice posunutí od skeneru:

```
Arm Skener;
Matrix help_mat;

Skener.ArmInit(‘‘xzy’’);

Skener.ArmPositionStart();

Skener.ArmPositionUpdate();

help_mat = Skener.ArmMatrixPosition();

help_mat.Print();
```

4.2 Ukázková aplikace

Názornou ukázkou použití knihovny `Arm` pro manipulaci s objekty je ukázková (také DEMO) aplikace. Pro grafické zobrazení byla použita knihovna `GLUT` [1]. DEMO aplikace je upravenou verzí programu na procvičení trojrozměrných transformací ze cvičení předmětu Základy počítačové grafiky z roku 2006.

Nejdůležitější částí ukázkové aplikace je funkce `TimerFunction(int value)`, která realizuje hlavní komunikaci se skenerem a umožňuje vytváření transformačních maticí. Zde je zjednodušený kód funkce:

```
int down;

if(Skener.ArmButton2()){
  if(down == 0){
    Skener.ArmPositionStart();
    down = 1;
  }
  Skener.ArmPositionUpdate();
}
else if(down != 2) down = 0;

if(Skener.ArmButton1()){
  if(down == 0){
    Skener.ArmOrientationStart();
    down = 2;
  }
  Skener.ArmOrientationUpdate();
}
else if(down != 1) down = 0;
```

Funkce `TimerFunction()` je periodicky volána jako obsluha funkce `glutTimerFunc()` knihovny `GLUT`. Funkce `TimerFunction()` kontroluje pomocí metod objektu `Skener` třídy `Arm` (viz. část 3.3) stav tlačítek. Stisk tlačítka 2 provádí vytváření matice posunutí[1] a stisk tlačítka 1 potom vytváří matice otočení[2]. Matice jsou následně získány do objektu třídy `Matrix` ve funkci `DrawScene()` a jsou použity při transformaci vykreslovaného trojrozměrného objektu.

[1] Více část 2.1
[2] Více část 2.3.4

Kapitola 5

Závěr

Tato bakalářská práce se zabývala tvorbou manipulátoru pro skener MicroScribe, který byl jakožto konečný produkt návrhu implementován a použit při programování ukázkové aplikace.

V teoretickém rozboru byly ukázány a vysvětleny matematické základy pro zpracování transformací objektů. Nejdříve byly objasněny principy práce s maticemi, které jsou základním stavebním kamenem pro tvorbu transformačních matic. Tvoření jednotlivých transformačních matic, které mohou ovlivnit atributy trojrozměrných objektů, následovalo a ukázalo prosté šablony pro zpracování lineárních transformací.

V kapitole návrhu se čtenář seznámil s produktem společnosti Immersion skenerem MicroScribe. Byly obecně popsány jednotlivé datové struktury a funkce knihovny `ArmDll32` skeneru a nastíněny možnosti komunikace. V závěru byl ukázán návrh knihovny manipulátoru, dvě třídy, které obstarávají komunikaci i zpracování dat od skeneru MicroScribe.

Knihovna `Arm` je praktickou realizací manipulátoru pro skener MicroScribe G2X, který je možno použít v grafických systémech při manipulaci s objekty. Nejlépe pak ve spojení s grafickou knihovnou `GLUT`, která byla testovacím prostředím při tvorbě.

Další vývoj manipulátoru pro skener MicroScribe by se měl zaměřit na komunikaci pomocí zpráv systému Windows. Při tvorbě této varianty je možné použít například knihovnu MFC (Microsoft Foundation Class). Následujícím vývojovým krokem v tvorbě manipulátorů by mohl být manipulátor pro 3D haptický skener, který je k dispozici na Ústavu počítačové grafiky a multimédií.

Literatura

[1] *GLUT and OpenGl utility libraries.* [online]. [cit. 2007–04–20].
URL: `http://www.opengl.org/resources/libraries/glut/`.

[2] *Lineární algebra.* [online]. Poslední modifikace 22. 4. 2007. [cit. 2007–05–02].
URL: `http://cs.wikipedia.org/wiki/Line%C3%A1rn%C3%AD_algebra`.

[3] *Matice.* [online]. Poslední modifikace 28. 4. 2007. [cit. 2007-05-02].
URL: `http://cs.wikipedia.org/wiki/Matice`.

[4] *Tait-Bryan angles.* [online]. Poslední modifikace 26 march 2007. [cit. 2007–05–02].
URL: `http://en.wikipedia.org/wiki/Tait-Bryan_angles`.

[5] Immersion Corporation. *3D Digitizing, Measuremment and Inspection.* [online].
Poslední modifiace 2007. [cit. 2007-05–02].
URL: `http://www.immersion.com/digitizer/`.

[6] Immersion Corporation. *MicroScribe ARMDLL32 API 2.0 User Reference*, 2002.

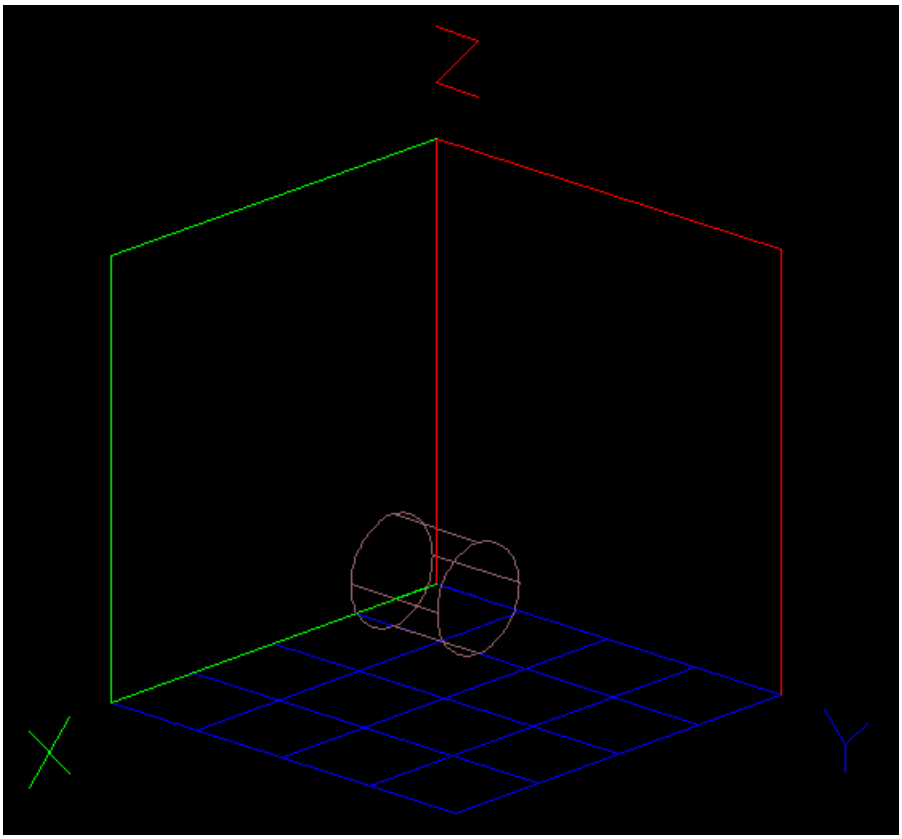
[7] Klaška J. *Matematika I: Matice a determinanty.* [online]. Poslední modifikace 13. 7. 2006. [cit. 2007–04–30]. URL: `http://mathonline.fme.vutbr.cz/`.

[8] Mocek T. *Efektivní manipulace s objekty ve 3D.* diplomová práce. FIT VUT v Brně. 2006.

[9] Žára J., Beneš B., Sochor J. *Moderní počítačová grafika.* Computer Press, 2004.
ISBN 80-251-0454-0.

Dodatek A

Manuál ukázkové aplikace



Obrázek A.1: Okno ukázkové aplikace po spuštění

Na obrázku A.1 je zachycen vzhled obsahu okna ukázkové aplikace po spuštění programu. Jsou zde znázorněny tři osy a roviny, které by měly odpovídat svým označením a rozložením odpovídat souřadným osám skeneru MicroScribe (obrázek 3.2). Po spuštění aplikace je navázáno spojení se skenerem a je možné manipulovat s objektem. Ovládání aplikace skrze klávesy je následující:

R – resetování pozice objektu. Po stisku klávesy **R** se objekt navrátí do výchozí pozice, středu souřadného systému. V rámci kódu aplikace tento úkon znamená nastavení

transformačních matic na jednotkovou matici.

A – přepnutí zobrazení do paralelní projekce.

S – přepnutí zobrazení do perspektivní projekce.

Esc, Q, X – vypnutí aplikace. Provede i ukončení spojení se skenerem.

F – přepnutí do celoobrazového zobrazení.

W – navrácení z celoobrazového zobrazení.

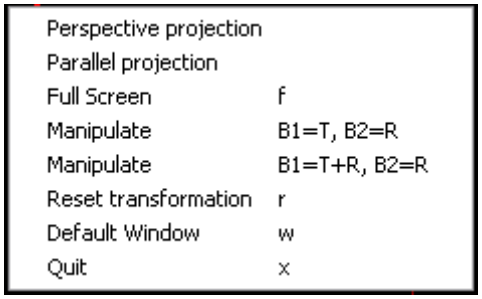
1 – přepnutí na výchozí způsob manipulace. V tomto režimu levé tlačítko skeneru slouží pro posouvání objektu a pravé tlačítko pro otáčení objektu kolem jeho středu. Zjednodušený kód takové manipulace (získání transformačních matic) je uveden v části implementace (4.2).

2 – přepnutí na druhý způsob manipulace s objektem. Levým tlačítkem skeneru je možné s objektem pohybovat volně (posunovat i otáčet) a pravé tlačítko slouží opět pouze k otáčení objektu kolem jeho středu.

Demo aplikaci je možné ovládat i pomocí tlačítek myši:

Levé – při stisknutí je možné rotovat se scénou kolem znázorněné osy Z.

Pravé – po stisku se objeví menu (obrázek A.2), kterým je možné ovládat aplikaci stejně jako při použití výše popsaných kláves.



Obrázek A.2: Menu ukázkové aplikace

Dodatek B

Dokumentace knihovny `Arm`

Obsah

Kapitola 1

Arm Rejstřík tříd

1.1 Arm Seznam tříd

Následující seznam obsahuje především identifikace tříd, ale nacházejí se zde i další netriviální prvky, jako jsou struktury (struct), unie (union) a rozhraní (interface). V seznamu jsou uvedeny jejich stručné popisy:

Kapitola 2

Arm Rejstřík souborů

2.1 Arm Seznam souborů

Zde naleznete seznam všech dokumentovaných souborů se stručnými popisy:

Kapitola 3

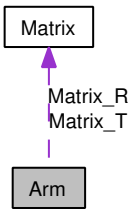
Arm Dokumentace tříd

3.1 Dokumentace třídy Arm

Třída **Arm** (s. 5).

`#include <Arm.h>`

Diagram tříd pro Arm:



Veřejné metody

- **Arm** ()

  Konstruktor.

- void **ArmClearPosition** ()

  Nastaví jednotkovou matici do objektu Matrix_ T.

- void **ArmClearOrientation** ()

  Nastaví jednotkouvou matici do objektu Matrix_ R.

- void **ArmPrintMatrixPosition** ()

  Tisk matice Matrix_ T.

- void **ArmPrintMatrixOrientation** ()

  Tisk matice Matrix_ R.

- int **ArmInit** (char *string)

  Nastaví se spojení se skener MicroScribe.

- int **ArmInfo** ()

  Tisk informací o skeneru.

- **Matrix ArmMatrixPosition** ()

  Předání matice Matrix_T.

- **Matrix ArmMatrixOrientation** ()

  Předání matice Matrix_R.

- int **ArmPositionStart** ()

  Uložení pozice hrotu.

- int **ArmPositionUpdate** ()

  Aktualizace pozice hrotu a vytvoření transformační matice.

- int **ArmOrientationStart** ()

  Uložení orientace hrotu.

- int **ArmOrientationUpdate** ()

  Aktualizace orientace hrotu a vytvoření transformační matice.

- int **ArmButton1** ()

  Stav tlačítka 1.

- int **ArmButton2** ()

  Stav tlačítka 2.

- void **ArmOver** ()

  Ukončení komunikace se skenerem.

Chráněné metody

- void **PositionMatrix** (float par1, float par2, float par3)

  Vytvoření transformační matice posunutí.

- void **OrientationMatrixX** (float par)

  Vytvoření transformační matice rotace kolem osy X.

- void **OrientationMatrixY** (float par)

  Vytvoření transformační matice otočení kolem osy Y.

- void **OrientationMatrixZ** (float par)

  Vytvoření transformační matice rotace kolem osy Z.

3.1.1 Detailní popis

Třída **Arm** (s. 5).

Třída **Arm** (s. 5) je určena pro komunikaci se skenerem MicroScribe. Tato třída pomocí svých metod komunikuje se skenerem, získává data a získaná data zpracovává do formy transformačních matic posunutí a rotace.

3.1.2 Dokumentace konstruktoru a destruktoru

3.1.2.1 Arm::Arm ()

Konstruktor.

Nastavení výchozí hodnoty proměnné version

Provede se inicializace proměnné version na hodnotu XYZ.

3.1.3 Dokumentace k metodám

3.1.3.1 void Arm::ArmClearPosition ()

Nastaví jednotkovou matici do objektu Matrix_T.

Skrze metodu Clear() třídy **Matrix** (s. 13) jsou upraveny hodnoty matice Matrix_T.

3.1.3.2 void Arm::ArmClearOrientation ()

Nastaví jednotkouvou matici do objektu Matrix_R.

Pomocí metody Clear() třídy **Matrix** (s. 13) jsou upraveny hodnoty matice Matrix_R.

3.1.3.3 void Arm::ArmPrintMatrixPosition ()

Tisk matice Matrix_T.

Vytisknutí hodnot matice Matrix_T do tvaru matice typu 4x4 s přesností na dvě desetninná místa.

Za použití metody Print() třídy **Matrix** (s. 13) je vypsán obsah matice Matrix_T.

3.1.3.4 void Arm::ArmPrintMatrixOrientation ()

Tisk matice Matrix_R.

Vytisknutí hodnot matice Matrix_R do tvaru matice typu 4x4 s přesnotí na dvě desetinná místa.

Metoda Print() provede vypsání hodnot matice Matrix_R.

3.1.3.5 int Arm::ArmInit (char * *string*)

Nastaví se spojení se skener MicroScribe.

Inicializace komunikace se skenerem formou Polling.

Parametry:

string je ukazatelem na tří znakový řetězec obsahující znaky x, y, z.

Nejprve se zkontroluje délka předaného řetezce string. Jestliže řetězec nemá délku 3, je vypsáno upozornění a navrácena chybová hodnota.

Následuje kontrola obsahu řetězce string. Podle prvního znaku řetězce se rozpoznání dělí na tři podpřípady. Řetězec může obsahovat pouze tři různé znaky a to znaky x, y a z. Podle složení znaků se do proměnné version přiřadí jedna z hodnot XYZ aú ZYX. V případě výskytu jiného znaku nebo opakování znaku je vypsáno upozornění a navrácena chybová hodnota.

Funkcí ArmStart() je připraveno navázání spojení se skenerem.

Obsluhy chybových stavů jsou nastaveny na hodnotu NULL.

Návážeme spojení se skenerem přes funkci ArmConnect().

Nastaví se aktualizace souřadnic polohy i otočení snímacího hrotu.

Získá se výchozí polohy hrotu skrze funkci **ArmPositionStart()** (s. 9).

Dále se uloží orientace hrotu pomocí funkce **ArmOrientationStart()** (s. 9).

Návratová hodnota:

-1 => nastala chyba při komunikaci se skenerem a spojení bylo ukončeno.
0 => vše proběho v pořádku.

3.1.3.6 int Arm::ArmInfo ()

Tisk informací o skeneru.

Zobrazí název a model skeneru, seriové číslo a verzi ovladače.

Návratová hodnota:

-1 => nastala chyba při komunikaci se skenerem a spojení bylo ukončeno.
0 => vše proběho v pořádku.

3.1.3.7 Matrix Arm::ArmMatrixPosition ()

Předání matice Matrix_T.

Vrácení matice Matrix_T do objektu třídy **Matrix** (s. 13).

Do objektu třídy **Matrix** (s. 13) je zkopírován obsah objektu Matrix_T.

3.1.3.8 Matrix Arm::ArmMatrixOrientation ()

Předání matice Matrix_R.

Vrácení matice Matrix_R do objektu třídy **Matrix** (s. 13)

Do objektu třídy **Matrix** (s. 13) je zkopírován obsah objektu Matrix_R.

3.1.3.9 int Arm::ArmPositionStart ()

Uložení pozice hrotu.

Získání a uložení nových souřadnic polohy snímacího hrotu do proměnné length.

Jsou získány souřadnice polohy hrotu do proměnné length.

Návratová hodnota:

-1 => nastala chyba při komunikaci se skenerem a spojení bylo ukončeno.
0 => vše proběho v pořádku.

3.1.3.10 int Arm::ArmPositionUpdate ()

Aktualizace pozice hrotu a vytvoření transformační matice.

Nejprve se získájí současné souřadnice polohy snímacího hrotu a provede se rozdíl s uloženými souřadnicemi. Rozdíl hodnot je použit pro vytvoření transformační matice posunutí.

Nejdříve se získájí současné souřadnice polohy snímacího hrotu do pomocné struktury pom.

Dále se provede výpočet rozdílů jednotlivých souřadnic a jejich uložení do

pomocných proměnných typu float

Získané rozdíly jsou podle hodnoty proměnné version předána funkci **PositionMatrix()** (s. 11) jako parametry posunutí v daných osách. Funkce **PositionMatrix()** (s. 11) vytvoří transformační matice posunutí.

Získaná hodnota polohy je uložena do proměnné length.

Návratová hodnota:

-1 => nastala chyba při komunikaci se skenerem a spojení bylo ukončeno.
0 => vše proběho v pořádku.

3.1.3.11 int Arm::ArmOrientationStart ()

Uložení orientace hrotu.

Získání a uložení nových souřadnic orientace snímacího hrotu do proměnné angle.

Skrze funkci ArmGetTipOrientation() jsou získány souřadnice orientace hrotu do struktury angle.

Návratová hodnota:

-1 => nastala chyba při komunikaci se skenerem a spojení bylo ukončeno.
0 => vše proběho v pořádku.

3.1.3.12 int Arm::ArmOrientationUpdate ()

Aktualizace orientace hrotu a vytvoření transformační matice.

Nejprve se získájí současné souřadnice orientace snímacího hrotu a provede se rozdíl s uloženými souřadnicemi. Rozdíl hodnot je použit pro vytvoření transformační matice rotace.

Nejprve se získají současné souřadnice orientace snímacího hrotu do pomocné

struktury pom.

Následuje určení rozdílu souřadnic otočení do pomocných proměnných.

Podle hodnoty proměnné version se určí pořadí otáčení kolem os. Volány jsou funkce Orientation-Matrix_() pro vytvoření transformačních matic. Jako parametry jsou předávány rozdíly souřadnic.

Na závěr se přepíší souřadnice orientace nově získanými.

Návratová hodnota:

-1 => nastala chyba při komunikaci se skenerem a spojení bylo ukončeno.
0 => vše proběho v pořádku.

3.1.3.13 int Arm::ArmButton1 ()

Stav tlačítka 1.

Podle vrácené hodnoty funkce se určí stav tlačítka 1. Hodnota 0 znamená, že tlačítko je stále ve výchozí pozici, tzn. nestisknuté. Hodnota 1 značí stisknutí a hodnota -1 znamená problém při komunikaci se skenerem a ukončení spojení.

Funkcí ArmGetButtonsState() je do pomocné proměnné ButState získán bitový

popis stavu tlačítek skeneru.

Pokud je potvrzeno stlačení tlačítka je přepsáná hodnota proměnné ButDown.

Nakonec je navrácena hodnota proměnné ButDown.

Návratová hodnota:

-1 -> nastala chyba při komunikaci se skenerem a spojení bylo ukončeno.
ButDown = 0 -> tlačítko nestisknuto.
ButDoen = 1 -> tlačítko stisknuto.

3.1.3.14 int Arm::ArmButton2 ()

Stav tlačítka 2.

Podle vrácené hodnoty funkce se určí stav tlačítka 2. Hodnota 0 znamená, že tlačítko je stále ve výchozí pozici, tzn. nestisknuté. Hodnota 1 značí stisknutí a hodnota -1 znamená problém při komunikaci se skenerem a ukončení spojení.

Funkcí ArmGetButtonsState() je do pomocné proměnné ButState získán bitový

popis stavu tlačítek skeneru.

Pokud je potvrzeno stlačení tlačítka je přepsáná hodnota proměnné ButDown.

Nakonec je navrácena hodnota proměnné ButDown.

Návratová hodnota:

-1 -> nastala chyba při komunikaci se skenerem a spojení bylo ukončeno.
ButDown = 0 -> tlačítko nestisknuto.
ButDown = 1 -> tlačítko stisknuto.

3.1.3.15 void Arm::ArmOver ()

Ukončení komunikace se skenerem.

Je ukončeno spojení se skenerem.

Ukončení spojení se skenerem funkcemi ArmDisconnect() a ArmEnd().

3.1.3.16 void Arm::PositionMatrix (float *par1*, float *par2*, float *par3*) [protected]

Vytvoření transformační matice posunutí.

Vytvoří translační matici a vynásobí s hlavní maticí posunutí Matrix_T.

Parametry:

par1 značí posunutí v ose x.
par2 značí posunutí v ose y.
par3 značí posunutí v ose z.

Do příslušných prvků pomocné matice h_mat jsou přiřazeny posuny.

Vytvořená transformační matice posunutí je vynásobena s hlavní matici Matrix_T.

3.1.3.17 void Arm::OrientationMatrixX (float *par*) [protected]

Vytvoření transformační matice rotace kolem osy X.

Vytvoří rotační matici a vynásobí jí hlavní matici rotace Matrix_R.

Parametry:

par je úhel otočení podle osy x.

Předaný parametr otočení je přepočítán do radiánů a zpracován funkcemi cos() a sin().

Výsledné hodnoty jsou uloženy v pomocných proměnných cosalpha a sinalpha.

Proměnné cosalpha a sinalpha jsou přiřazeny na svá místa v pomocné matici h_mat.

Vytvořená matice rotace je vynásobena s hlavní maticí otočení Matrix_R.

3.1.3.18 void Arm::OrientationMatrixY (float *par*) [protected]

Vytvoření transformační matice otočení kolem osy Y.

Vytvoří rotační matici a vynásobí jí hlavní matici rotace Matrix_R.

Parametry:

par je úhel otočení kolem osy y.

Předaný parametr otočení je přepočítán do radiánů a zpracován funkcemi cos() a sin().

Výsledné hodnoty jsou uloženy v pomocných proměnných cosalpha a sinalpha.

Proměnné cosalpha a sinalpha jsou přiřazeny na svá místa v pomocné matici h_mat.

Vytvořená matice rotace je vynásobena s hlavní maticí otočení Matrix_R.

3.1.3.19 void Arm::OrientationMatrixZ (float *par*) [protected]

Vytvoření transformační matice rotace kolem osy Z.

Vytvoří matici otočení a vynásobí jí hlavní matici rotace Matrix_R.

Parametry:

par je úhel otočení v ose z.

Předaný parametr otočení je přepočítán do radiánů a zpracován funkcemi cos() a sin().

Výsledné hodnoty jsou uloženy v pomocných proměnných cosalpha a sinalpha.

Proměnné cosalpha a sinalpha jsou přiřazeny na svá místa v pomocné matici h_mat.

Vytvořená matice rotace je vynásobena s hlavní maticí otočení Matrix_R.

Dokumentace pro tuto třídu byla generována z následujících souborů:

- D:/BP/ArmFit/**Arm.h**
- D:/BP/ArmFit/**Arm.cpp**

3.2 Dokumentace třídy Matrix

Třída **Matrix** (s. 13).

`#include <Arm.h>`

Veřejné metody

- **Matrix** ()

  Konstruktor třídy.

- float * **operator[]** (int row)

  Metoda [] s jediným parametrem row.

- void **Print** ()

  Výpis dat.

- void **Clear** ()

  "Vyčištění" pole data[]

- void **Mult** (**Matrix** &m)

  *Vynásobení pole data[] obsahem objektu třídy **Matrix** (s. 13).*

3.2.1 Detailní popis

Třída **Matrix** (s. 13).

Třída **Matrix** (s. 13) obsahuje data pro uložení matice typu 4x4. Dále obsahuje operaci pro práci s maticí uloženou v datové části. Operace s maticí je omezena pouze na vynásobení uložených dat jinou maticí.

3.2.2 Dokumentace konstruktoru a destruktoru

3.2.2.1 Matrix::Matrix ()

Konstruktor třídy.

Konstruktor naplní data hodnotami jednotkové matice .

Pomocí dvou for cyklů se přistupuje k poli proměnných objektu **Matrix** (s. 13)

Do hlavní diagonály matice jsou nastaveny hodnoty 1.0. Je vytvořena jednotková matice.

3.2.3 Dokumentace k metodám

3.2.3.1 float * Matrix::operator[] (int *row*)

Metoda [] s jediným parametrem row.

Vrací ukazatel do pole dat, aby bylo možné pracovat s daty jako s dvourozměrným polem hodnot (maticí). Index řádku je předán jako parametr row.

Parametry:

row hodnota typu int.

Vrací se ukazatel vždy na první prvek v poli data[] v řádku row.

3.2.3.2 void Matrix::Print ()

Výpis dat.

Tato metoda provede výpis pole data[16] na standardní výstup. Formátování výpisu je přízpůsobeno tvaru matice 4x4. Hodnoty pole jsou vypisovány s přesností na dvě desetnná místa.

Výpis pole data[] pomocí dvou for cyklů do tvaru matic. Za výpisem je vypsán ještě jeden prázdný řádek.

3.2.3.3 void Matrix::Clear ()

"Vyčištění" pole data[]

Je nastavena znovu jednotková matice do datové části třídy.

Stejně jako v případě konstruktoru jsou hodnoty pole data[] změněny na hodnoty jendotkové matice pomoci dvou for cyklů.

3.2.3.4 void Matrix::Mult (Matrix & *m*)

Vynásobení pole data[] obsahem objektu třídy **Matrix** (s. 13).

Je provedeno vynásobení matice uložené v poli data[] maticí v objektu m. Výsledek násobení je uložen zpět do pole data[] toho objektu.

Parametry:

m je objekt typu **Matrix** (s. 13)

Do pomocného pole pom[] je zkopírováno interní pole data[].

Opět přes dva for cykly přepisujeme stávající hodnoty pole data[] a přiřazujeme nové hodnoty získané podle teorie násobení matic. Násobíme hodnoty v řádcích objektu m a hodnoty sloupcových prvků pomocného pole pom.

Dokumentace pro tuto třídu byla generována z následujících souborů:

- D:/BP/ArmFit/**Arm.h**
- D:/BP/ArmFit/**Arm.cpp**

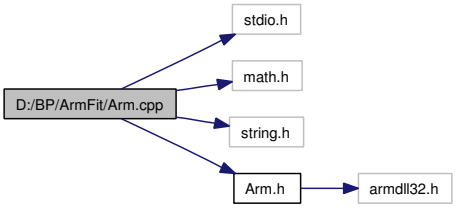
Kapitola 4

Arm Dokumentace souborů

4.1 Dokumentace souboru D:/BP/ArmFit/Arm.cpp

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "Arm.h"
```

Graf závislostí na vkládaných souborech pro Arm.cpp:



Definice maker

- #define **PI** 3.1416

  Definice pomocné konstanty PI (Eulerova čísla).

4.1.1 Detailní popis

Zdrojový soubor pro hlavičkový soubor **Arm.h** (s. 16). V tomto souboru jsou obsaženy zdrojové kódy všech metod tříd **Matrix** (s. 13) a **Arm** (s. 5). Metody využívají funkce knihovny ArmDll32 skeneru MicroScribe pro komunikaci.

4.2 Dokumentace souboru D:/BP/ArmFit/Arm.h

```
#include "armdll32.h"
```

Graf závislostí na vkládaných souborech pro Arm.h:

Následující graf ukazuje, které soubory přímo nebo nepřímo vkládají tento soubor:

Třídy

- class **Matrix**

  *Třída **Matrix** (s. 13).*

- class **Arm**

  *Třída **Arm** (s. 5).*

Výčty

- enum **Axis**

  Výčtový typ Axis pořadí os.

4.2.1 Detailní popis

Soubor **Arm.h** (s. 16) obsahuje deklarace tříd **Matrix** (s. 13) a **Arm** (s. 5) a jejich metod. Dále obsahuje definici výčtového typu Axis pro použití při uložení pořadí os souřadného systému.

4.2.2 Dokumentace výčtových typů

4.2.2.1 enum Axis

Výčtový typ Axis pořadí os.

Podle hodnoty proměnné je určeno pořadí os při vytváření transformací.