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

PREZENTAČNÍ SOFTWARE MATEMATIKY PRO SŠ

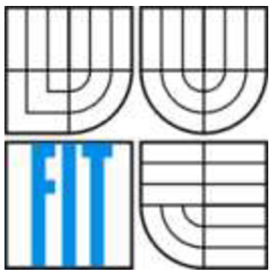
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ŠTEFAN SZABO

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

PREZENTAČNÍ SOFTWARE MATEMATIKY PRO SŠ

PRESENTATION SOFTWARE FOR MATHEMATICS ON HIGH SCHOOLS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
ŠTEFAN SZABO

VEDOUCÍ PRÁCE
SUPERVISOR
ING. VÍT ŠTANCL

BRNO 2009

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2008/2009

Zadání bakalářské práce

Řešitel: **Szabo Štefan**
Obor: Informační technologie
Téma: **Prezentační software matematiky pro SŠ**
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte možnosti zobrazování základních matematických jevů.
2. Prostudujte dostupné materiály a seznamte se se základy modelování a simulace, pedagogiky a didaktiky.
3. Vyberte vhodnou metodu a navrhněte systém pro názorné vyučování základních vybraných matematických jevů.
4. Implementujte část systému formou demo aplikace.
5. Diskutujte možnosti budoucího vývoje. Zvažte další pokračování v rámci diplomové práce.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění prvních tří bodů 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 nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Štancl Vít, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2008
Datum odevzdání: 20. května 2009

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce slouží jako prezentační software matematiky pro střední školy. Její cílem je zjednodušit, zefektívnit, nebo jinak zdokonalit výučbu za využití reprezentace jednolitvých matematických okruhů prostředníctvím počítačové grafiky. Aplikace je navržena tak, aby bylo možné jednoduše přidávat okruhy formou zásuvných modulů.

Abstract

This work serves as a mathematics presentation software for high schools. Its aim is to simplify, effect or otherwise improve the process of learning, by using computer graphics representation of individual math topics. Aplication is designed in such a manner so it can be easily extended by adding new modules.

Klíčová slova

OpenGL, matematika, vyuka, prezentace, objem, povrch, rez, animace.

Keywords

OpenGL, mathematics, teaching, presentation, volume, area, slice, animation.

Citace

Szabo Štefan: Prezentační software matematiky pro SŠ, bakalářská práce, Brno, FIT VUT v Brně, 2009

Prezentační software matematiky pro SŠ

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením ing. Víta Štancla.
Další informace jsem čerpal z internetových zdrojů, odborných publikací a pramenů.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Štefan Szabo
19.05.2009

Poděkování

Touto cestou bych se chtěl poděkovat svému vedoucímu, panu Štanclovi, za odbornou pomoc a čas věnovaný mé práce. Dále bych rád poděkoval všem, kdo mě učili a pomohli mi získat vědomosti k vytvoření této práce. V neposlední řadě bych chtěl také poděkovat svým rodičům, za jejich podporu při studiu, bez které bych tyto řádky jistě nepsal.

Obsah

1 Úvod

Táto práca vznikla ako "Prezentačný software matematiky pre SŠ". Pod týmto slovným spojením rozumieme aplikáciu pre podporu výučby spomínaného predmetu na stredných školách, skrátene výukový program. Hlavným cieľom takýchto programov je zjednodušiť, zefektívniť alebo inak vzpružiť výučby predmetov za pomoci moderných výukových prostriedkov.

Náplňou tejto práce je dosiahnuť vyššie uvedené ciele, za pomoci rôznych grafických vizualizácií , reprezentácií rôznych matematických javov prostredníctvom počítačovej grafiky.

Nasledujúca kapitola sa zaoberá analýzou danej problematiky, je venovaná výukovým programom, ich vlastnostiam, požiadavkám, takisto aj základom pedagogiky, didaktiky, modelovaniu a simuláciám.

Tretia časť obsahuje návrh riešenia. Táto kapitola obsahuje zhrnutie problému a vytýčenie programových, ako aj didaktických požiadaviek, ktoré od aplikácie očakávame.

Štvrtá kapitola je časťou teoretickou. Ako už názov napovedá, rozoberá sa tu teória potrebná k implemtácii aplikácie a k programovaniu počítačovej grafiky.

Šiesta, predposledná kapitola, je venovaná implementácii. Sú tu rozpísané použité technológie, nástroje a knihovne, ktoré som pri implementácii aplikácie využil. Záverečná časť tejto kapitoly je venovaná samotnej implementácii, využitiu teórie v programovaní aplikácie.

Posledná, záverečná kapitola, zhodnocuje dosiahnuté výsledky v porovnaní s vytýčenými cieľmi a takisto obsahuje návrch možných rozšírení, vylepšení aplikácie v rámci ďalšieho vývoja.

2 Výukový program

Pod pojmom výukový program rozumieme software, ktorý je navrhnutý k výukovým účelom a je schopný plniť aspoň jednu z didaktických funkcií (podrobnejšie v podkapitole 2.1).

Prostredie výukového programu by malo byť prehľadné, intuitívne, jednoduché a užívateľ by sa v ňom mal vedieť jednoducho orientovať. Mnohé výukové programy sa dajú použiť priamo pri prezenčnom vyučovaní (prostredníctvom počítačov, dataprojektorov, atď.) alebo slúžia pre samoštúdium vo voľnom čase.

Cieľom týchto nástrojov nie je, a nikdy nebolo nahradiť v komplexnej rovine úlohu vyučujúceho. Žiadny software totiž nie je schopný nahradiť osobnosť vyštudovaného pedagóga. Je však úplne normálne, ak sa rozhodne zapojiť tieto prvky do svojho výkladu, z dôvodu či už precvičovania danej látky, testovania alebo zefektívnenia a optimalizácie vzdelávacieho procesu.

Úlohou je teda vytvoriť software, ktorý by vyučujúcim matematiky slúžil ako podporný prvok vo výučbe, reprezentáciou rôznych matematických problematík využitím počítačovej grafiky. Samotným zapojením technických prvkov a vymožeností do výučby sa zaoberajú vedy pedagogika a didaktika.

Z hľadiska informatického však nemôžme zabudnúť ani na fakt, že jednotlivé demonštrácie musia byť aj reálnym spracovaním danej skutočnosti, to znamená, že musíme dodržiavať základné princípy modelovania a simulácie.

2.1 Základy pedagogiky a didaktiky

Pedagogika je jednak spoločenská disciplína, náuka o výchove a vzdelávaní, a na druhej strane je to aj k tomu prislúchajúca prax. Opiera sa o výsledky jak spoločenských, tak aj prírodných vied. Predmetom skúmania pedagogiky je *vzdelávací proces,* jeho výsledky ako aj efekty vzdelávania.

Súčasťou pedagogiky je ďalšia veda - *didaktika*. Zaoberá sa teóriou vyučovania, jeho podstatou, cieľom a obsahom ako aj samotným vyučovacím procesom, zásadách, formách a metódach uplatňovaných vo vyučovaní.

Didaktické prostriedky, materiálne i nemateriálne, slúžia ako podporné prostriedky k dosahovaniu výchovne vzdelávacích cieľov. Okrem iného sa sem zahrňujú aj technické výukové prostriedky, ktoré majú nasledovné funkcie:

- pomáhajú študentom k osvojeniu spoločensky nutných poznatkov a morálnych postojov na úrovni modernej komunikácie
- prispievajú k zvýšeniu účinnosti vyučovacieho a učebného procesu ich riadením
- motivujú študentov, naväzujú vnútorný vzťah študentov k poznaniu, k vede, k technike, ako aj k samotnému učeniu
- aktivizujú študentov a podnecujú ich k tvorivým činnostiam

V. Rambousek v publikácii Technické výukové prostředky [8] zdôrazňuje:
"TVP umožňujú multisenzoriálne vnímanie a viackanálovú komunikáciu, celkové zkvalitnenie procesu predávania informácií a stále širšie možnosti objektivizácie a racionalizácie bezprostredného riadenia, kontroly a hodnotenia učebných činností študentov."

Podľa Tollingerovej objektivizácia živej práce učiteľa modernými prostriedkami didaktickej techniky preukazuje 3 efekty:

- umožňuje z činnosti učiteľa i študentov *odstrániť mechanické*, neplodné *operácie*
- vytvára predpoklady, aby sa človek mohol podľa potreby *učiť vysoko efektívne*
- dáva možnosť chrániť najlepšie pedagogické pôsobenie v multimediálnych konzervách, čím sa *učenie stáva nezávislé na dobe a podmienkach priebehu pedagogického pôsobenia*

Po zhrnutí by sme mohli vymedziť hlavné didaktické funkcie na:

1. motivácia
2. expozícia učiva
3. upevnenie osvojených vedomostí a znalostí
4. kontrola získaných vedomostí a znalostí.

Didaktické prostriedky (výukové programy) sa delia z mnohých hľadísk podľa určitých vlastností.

Interaktivita je vlastnosť, ktorá je z didaktického hľadiska veľmi dôležitá. Spočíva v tom, že užívateľ môže aktívne ovplyvňovať beh programu a nielen pasívne vstrebávať vzdelávací obsah. Hovoríme teda o vzájomnej komunikácií medzi užívateľom a programom, tým pádom sa študent viac zapojuje do procesu učenia, je vyššie motivovaný, a proces výuky naberá samozrejme vyšších pedagogických hodnôt.

Schopnosť programov poskytovať *spätnú väzbu* je tiež významné kritérium z hľadiska didaktického, ako aj z hľadiska programátorského. Študent tak má možnosť uistiť sa, že jeho zmýšlanie, predstavivosť, prípadne výpočty naberajú správne dimenzie. Z psychologického hľadiska je to taktiež významný faktor pre danú osobnosť.

Z pohľadu možností vnímania rozdeľujeme výukové prostriedky na *vizuálne* a *audiovizuálne.* Ako názvy napovedajú, jedná sa o pôsobenie programu na študujúceho prostredníctvom vizuálnych, alebo aj paralelným zapojením auditívnych vnemov.

2.2 Základy modelovania a simulácie

Pred vytváraním jednotlivých grafických prezentácií si musíme uvedomiť, že budeme zobrazovať presne popísané, definované matematické javy, tzn. že budeme vytvárať ich *modely*, *modelovať* ich.

Model je napodobnenina systému iným systémom. V našom prípade to bude napodobnenina určitých matematických javov, ich zákonitostí, priebehov grafickým zobrazením, vizualizáciou. Model musí napodobňovať všetky, pre naše účely podstatné, vlastnosti daného systému. Príkladom môže byť napríklad 3D model valca, kde si musím pri modelovaní dávať pozor na dodržanie jeho matematických / geometrických vlasností (podstavou je kruh, plášť je obdĺžnik o rozmeroch $2\pi r.v$).

Modelovanie je proces vytvárania modelov systémov na základe našich znalostí. Tento proces je obecne dosť náročný a zvyčajne vyžaduje znalosti z viacerých oborov. Obvykle nasleduje simuláciou, čo je experimentovanie s vytvoreným modelom za účelom získavania nových znalostí o systéme. Toto som využil napríklad pri simulácii rozkladu telies na sieť za účelom demonštrácie objemu a povrchu telies. Pri modelovaní vychádzame z informácií o systéme, ktoré sú dostupné. Model, ktorý vznikne reprezentuje formalizované znalosti o modelovanom systéme z hladiska, ktoré chceme skúmať. Modely obvykle pokrývajú len tú časť znalostí, ktoré sú pre daný účel potrebné (nezaujíma nás napr. hmotnosť telesa pri procese skúmania jeho povrchu). Aplikácia bude pracovať s *demonštračnými modelmi* daných matematických javov, operácií.

3 Návrh riešenia

Obsahom tejto práce je návrh a implementácia aplikácie, ktorá, ako som už vyššie uviedol, bude slúžiť pri vyučovaní matematiky na stredných školách ako podporný prvok.

Na začiatku je potrebné vytýčiť si určité ciele, stanoviť požiadavky, ktoré by mal program spĺňať.

3.1 Implementačný návrh

Po zvážení didaktických zásad a vzhľadom k využitiu aplikácie je potrebné prehodnotiť a stanoviť si určité kritéria, ktorými by sa mal výukový program riadiť.

Mal by byť plne kompatibilný s operačnými systémami MS Windows, bez akýchkoľvek dodatočných inštalácií, nastavovaní apod. Najideálnejším riešením by bola situácia, ak by užívateľovi stačilo len nakopírovať program do ľubovoľnej zložky v počítači, a následne by mohol program využívať. Takto by sa zamedzilo zbytočne stratenému času venovanému rôznym inštaláciám a konfiguráciám ako u iných programov, čo by naopak zvýšilo kvalitu a čas venovaný samotnému výukovému procesu.

Z hľadiska tvorby užívateľských rozhraní by aplikácia mala mať čo najjednoduchšie ovládanie, pričom by nemala chýbať interaktivita a spätná väzba k užívateľovi. Po jej spustení by sa mohlo objaviť nejaké hlavné, riadiace okno, ktorého dominantou by boli ovládacie prvky napojené na prácu s hlavnými didaktickými prvkami programu – matematickými demami.

Z hľadiska funkcionality bude teda hlavné okno slúžiť napr. na spúšťanie samotných matematických diem, ich pridávanie a odstraňovanie. Z toho vyplýva, že by bolo vhodné ubrať sa cestou implementácie formou zásuvných modulov.

3.2 Matematický návrh

Z didaktického hľadiska sa jedná o polytematický výukový program s predmetovým zameraním *Matematika*. Keďže sa jedná o modulárny program jednotlivé matematické javy sa budú pridávať dodatočne. Demo aplikácia by mohla prezentovať matematické javy a zákonitosti ako napríklad:

- základné matematické vzťahy a zákonitosti
- vzťahy rôznych matematických javov (napr. funkcií, množín, atď.)
- manipulovanie s 2D alebo 3D objektmi ako napr. rovina, kocka, valec, apod., za účelom skúmania ich vlastností a matematických zakonitostí
- vzájomné polohy 2D objektov
- vzájomné polohy 2D a 3D objektov
- pridávanie, upravovanie a následne zobrazovanie definícií a vzorcov prostredníctvom textov alebo obrazových formátov
- spojenie geometrie s algebrou – dopočítavanie vlastností telies ako napr. objem, povrch, atď.

3.3 Zhrnutie

Po zhrnutí je teda cieľom vytvoriť program, ktorý bude spĺňať nasledovné kritériá (zhodnotím ich v záverečnej kapitole):

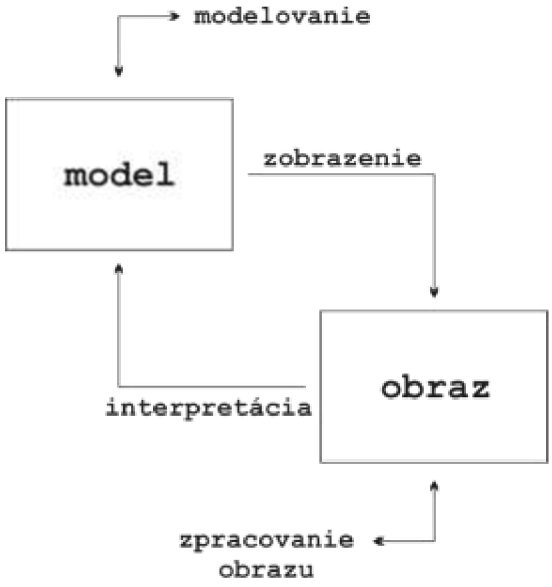
- kompatibilita so systémami MS Windows
- jednoduchá inštalácia
- určenie najmä pre vyučujúcich
- jednoduché, intuitívne ovládanie
- interaktivita
- spätná väzba
- vizuálna aplikácia, tzn. bez audiovizuálneho doprovodu, ten bude nahradený výkladom pedagóga
- možnosť rozširovania formou zásuvných modulov
- možnosť úprav v prezentáciách

4 Teoretická časť

Nasledujúce podkapitoly obsahujú teóriu, ktorú som využil pri tvorbe tejto bakalárskej práce. V jednotlivých častiach sa môže stať, že sa objavia presné definície, citácie alebo vzorce, ktoré som čerpal z literárnych prameňov uvedených v závere práce.

4.1 Počítačová grafika

Počítačová grafika je súčasťou oboru Informatika, ktorá sa zaoberá interpretáciou alebo tvorbou grafickej obrazovej informácie. K dispozícii máme dané matematické javy, z ktorých je potrebné postupnosťou krokov vytvoriť demonštračné *modely*. Tieto modely však následne musíme vedieť aj previezť na grafickú obrazovú informáciu a zároveň ich budeme musieť aj zobraziť (Obr. 4.1).



Obr. 4.1: Schéma počítačovej grafiky

V porovnaní s inými popismi, či už textovými alebo zvukovými, je pre nás grafické vyjadrenie informácie najviac informačne obsiahnuté. Ľudský mozog je schopný veľmi rýchlo a detajlne analyzovať a interpretovať grafickú informáciu. Takisto musíme podotknúť, že dochádza pri tomto popise v porovnaní s ostatnými, už spomenutými, k najmenšej možnej dezinterpretácie.

Počítačová grafika sa delí z rôznych hladísk. Z hľadiska programátora existujú dva základné prístupy k počítačovej grafike:

- *vektorová* grafika
- *rastrová* grafika

U vektorovej grafiky je spôsob popisu spracovávanej a zobrazovanej informácie vo forme skupiny *základných vektorových entít* (úsečky, krivky, kružnice, atď.) analyticky. To znamená, že ukladá presné geometrické dáta, napríklad súradnice dvoch bodov úsečky.

Pričom u rastrovej grafiky je tento popis vo forme *rastrovej matice* (2D / 3D) diskrétne. Môžeme si ju predstaviť ako pravidelnú sieť pixelov.

Jeden *pixel* (z angl. pixel element) je základná jednotka rastrovej grafiky. Predstavuje jeden svietiaci bod na našom monitore a z hľadiska počítačovej grafiky nesie špecifické informácie, napríklad o jase, farbe, priehľadnosti bodu, prípadne kombinácie týchto hodnôt.

Rasterizácia je proces prevodu vektorových entít na ich zodpovedajúce zobrazenia vo forme rastrovej grafiky. Budem ju využívať pre potreby zobrazovania. Pre všetky druhy 2D i 3D objektov je ľahko riešiteľná.

Obr.4.2: Rastrová a vektorová grafika

Z pohľadu programátora existuje mnoho spôsobov, ako docieliť požadované výsledky. Či už sa jedná o programovanie základnej aplikácie alebo už finálnych grafických interpretácií.

V zásade, programovanie grafiky nie je ani až tak o programovaní. Je to takmer výlučne o matematike. Miestami zaberá oveľa viac času premýšľanie nad tým, čo sa snažíme docieliť a ako to docieliť, ako samotné písanie kódu búšením klávies. Vo všeobecnosti, ak niekto rád programuje, k tomu si obľubuje aj matematiku, je preňho programovanie grafiky ako stvorené. Pri programovaní jednoduchých aplikácií si vystačíme so základnými matematickými operáciami ako sčitavanie, delenie, násobenie a pod. Postupne sa k nim pridajú od základov geometrie, goniometrických funkcií cez analytickú geometriu, až k maticiam, ktoré sú okrem iného základom transformácií v rovine či priestore. Práve tieto, vyššie uvedené, zohrávajú najväčšiu úlohu v tejto práci a v nasledovných podkapitolách sa postupne prepracujem od základov matematiky až po jej využitie pri implementácií modulov.

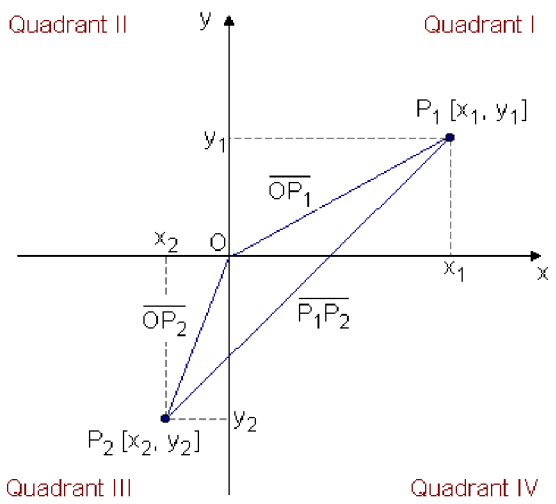
4.2 Súradnicové sústavy

Keď chceme definovať multi-dimenzionálne priestory, používame súradnicové sústavy. Existuje mnoho druhov súradnicových systémov ako napr. Polárny súradnicový systém, Sférický súradnicový systém alebo Karteziánska súradnicová sústava, ktorú budem využívať aj ja.

Karteziánska súradnicová sústava pozostáva z jedného alebo viacerých rozmerov, ktoré sú navzájom kolmé.

4.2.1 2D Karteziánska súradnicová sústava

Dvojdimenzionálna Karteziánska súradnicová sústava je definovaná dvoma *súradnicovými osami*, ktoré zvierajú pravý uhol, vytvárajúc rovinu, rozdelenú na 4 quadranty (Obr. 4.3). Horizontálna os je označovaná x a vertikálna y. Priesečník osí sa nazýva *počiatok súradnicovej sústavy* a je označovaný O. K definovaniu bodu P v 2D súradnicovom systéme zadávame x-ovú súradnicu nasledovanú y-ovou v tvare $[x, y]$.



Obr. 4.3: 2D Karteziánska súradnicová sústava

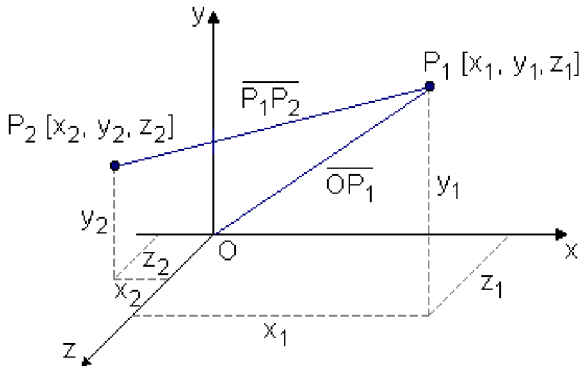
Vzdialenosť určitého boda *P* od počiatku súradnicovej sśutavy sa určuje pomocou vzťahov v *Pytagorovej vete* (4.1). Vzdialenosť dvoch bodov umiestnených v rovine sa určuje rovnakým spôsobom (4.2).

$$\overline{OP} = \sqrt{x^2 + y^2} \tag{4.1}$$

$$d = \overline{P1P2} = \sqrt{(x_1 + x_2)^2 + (y_1 + y_2)^2} \tag{4.2}$$

4.2.2 3D Karteziánska súradnicová sústava

So štandardným 2D súradnicovým systémom sa už určite väčšina z nás stretla. Pre naše programové účely ale budeme potrebovať trojdimenzionálny systém súradníc (Obr.4.4). Pridáme teda jednu dimenziu, označujeme ju z. V praxi to vyzerá tak, že osi x a y tvoria rovinu, príkladom môže byť naša obrazovka. V 3D pridávame každému bodu tejto roviny tretí rozmer, hĺbku, čiže niečo vstupuje do obrazovky a naopak, niečo z nej potom vystupuje. Vzťahy pre výpočet vzdialenosti bodu od počiatku a vzdialenosti dvoch bodov v trojrozmernej karteziánskej súradnicovej sústave sú podobné ako v 2D (vzťah 4.3, resp. 4.4).



Obr. 4.4: 3D Karteziánska súradnicová sústava

$$\overline{OP} = \sqrt{x^2 + y^2 + z^2} \tag{4.3}$$

$$d = \overline{P1P2} = \sqrt{(x_1 + x_2)^2 + (y_1 + y_2)^2 + (z_1 + z_2)^2} \tag{4.4}$$

Karteziánske súradnicové sústavy sú teda prostriedky, pomocou ktorých každému bodu v rovine / priestore vieme jednoznačne priradiť usporiadanú dvojicu / trojicu reálnych čísiel, ktoré voláme súradnice daného bodu. Skutočnosť, že nami zvolený bod P má súradnice x_1, y_1, z_1, zapisujeme $P[x_1, y_1, z_1]$ alebo $P = [x_1, y_1, z_1]$. 3D Karteziánska súradnicová sústava je nevyhnutným základom pri programovaní počítačovej grafiky.

4.3 Analytická geometria

Analytická geometria je oblasť matematiky, v ktorej sa študujú geometrické útvary pomocou ich analytických vyjadrení. Pomocou zvolenej súradnicovej sústavy vieme každý základný geometrický útvar vyjadriť jednoznačne v tvare istej rovnice, prípadne nerovnice.

4.3.1 Bod, priamka, vektor

Bod môže byť reprezentovaný ako n-tica reálnych čísiel, kde n udáva rozmer priestoru, v ktorom sa nachádza, takisto ako udáva aj počet jeho súradníc. Napríklad v dvojrozmernom priestore je bod daný dvoma súradnicami, v trojrozmernom je bod určený troma súradnicami. Z programovacieho hľadiska môžme bod definovať ako štruktúru s troma informáciami, pričom každá z nich nesie hodnotu zodpovedajúcej súradnice.

Priamka je geometrický útvar, ktorý si môžme predstaviť ako nekonečne dlhú a dokonale rovnú čiaru. Priamku môžeme zadať viacerými spôsobmi, my budeme využívať *smernicovú rovnicu* priamky:

$$y = kx + q, \tag{4.5}$$

kde k, q sú reálne čísla. K sa nazýva smernica priamky a rovná sa tangensu uhlu priamky, zvierajúcej s kladným smerom osi x. Smernicou priamky vyjadrujeme relativne zmeny premennej y pri zmene premennej x. Číslo q je y-ová súradnica priesečníka priamky s osou y.

Vektor je geometrický objekt, ktorý je určený dĺžkou, smerom a orientáciou. Môžeme si ho predstaviť ako orientovanú úsečku, tzn. úsečku, na ktorej je vyznačený začiatočný a koncový bod. Pritom nesmieme zabudnúť, že dve rôzne orientované úsečky, ktoré majú zhodnú dĺžku (tj. veľkosť), smer aj orientáciu, predstavujú ten istý vektor, ide len o dve rôzne umiestnenia toho istého vektora.

Súradnice vektora sú súradnice jeho koncového bodu v takom umiestnení vektora, v ktorom je začiatočný bod totožný s počiatkom súradnicovej sústavy.

Polohovým vektorom bodu A chápeme vektor $A - O$.

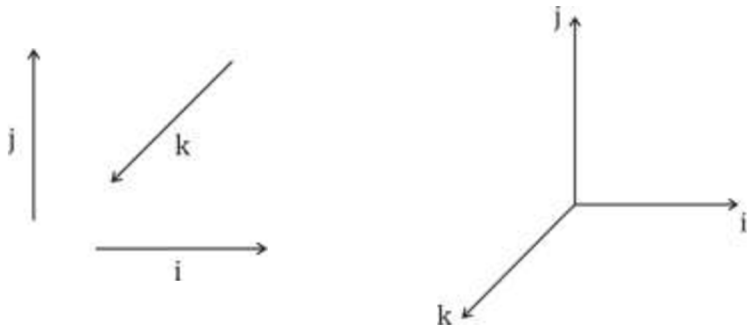
Dĺžka vektora v je vzdialenosť jeho začiatočného a koncového bodu a označujeme ju $||v||$.

$$\|B - A\| = d(A, B) = \sqrt{(a_1 + b_1)^2 + (a_2 + b_2)^2 + (a_3 + b_3)^2} \tag{4.6}$$

Nulový vektor je (jediný) vektor, ktorého dĺžka je *0*. Všetky súradnice nulového vektora sú rovné nule.

Jednotkovým vektorom je každý vektor, ktorého dĺžka je rovná jednej. Jednotkový vektor orientovaný v kladnom smere osi *x* označujeme *i*, jednotkový vektor orientovaný v kladnom smere

osi *y* označujeme *j*, jednotkový vektor orientovaný v kladnom smere osi *z* označujeme *k*. Vektory *i, j, k* (Obr.4.5a) tvoria *bázu* trojrozmerného priestoru (Obr. 4.5b).



a) b)

Obr. 4.5: Jednotkové vektory tvoriace bázu trojrozmerného priestoru

4.3.2 Základné operácie nad vektormi

Násobenie vektorov reálnym číslom:

Skalárny násobok vektora *v* číslom c je vektor $c \cdot v$, pričom:

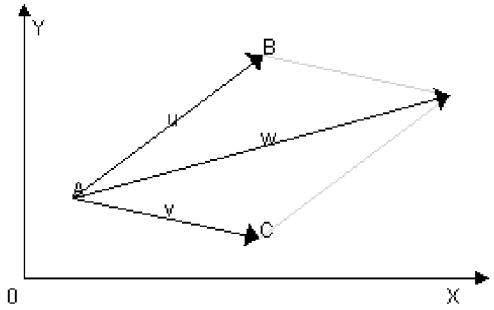
1. dĺžka vektora $c \cdot v$ je $|c|$ násobkom dĺžky vektora v
2. obidva vektory majú rovnaký smer
3. ak:
   - $c > 0$, tak v a $c \cdot v$ majú zhodnú orientáciu
   - $c < 0$, tak v a $c \cdot v$ majú opačnú orientáciu
   - $c = 0$, tak $c \cdot v = 0$

$$c \cdot v = [cv_1, cv_2, cv_3] \tag{4.7}$$

Vektor $(-1) \cdot v$ voláme *vektorom opačným* k vektoru v a označujeme ho – v.

$$-v = [-v_1, -v_2, -v_3] \tag{4.8}$$

Súčet vektorov $\boldsymbol{u}$ a $\boldsymbol{v}$ je vektor $\boldsymbol{u} + \boldsymbol{v}$, ktorý môžeme znázorniť v súradnicovej sústave ako uhlopriečku v rovnobežniku so stranami tvorenými vektormi $\boldsymbol{u}$ a $\boldsymbol{v}$, pričom jeho orientácia je znázornená na obrázku 4.6.



Obr. 4.6: Súčet vektorov

Rozdiel vektorov u a v je vektor

$$u - v = u + (-v) \tag{4.9}$$

Ak máme dva vektory u a v, tak výraz $cu + dv$,kde c a d sú ľubovoľné reálne čísla, nazývame *lineárna kombinácia vektorov* u a v.

Podobne, ak máme tri vektory u, v a w, tak výraz $cu + de + ew$, kde c, d a e sú ľubovoľné reálne čísla, nazývame *lineárna kombinácia vektorov* u, v a w.

Čísla c,d a e v lineárnych kombináciách voláme *koeficienty* kombinácie. Pre každú konkrétnu hodnotu koeficientov dostávame konkrétny vektor.

Skalárny súčin vektorov u a v je číslo $u \cdot v = \|u\|\|v\| \cos(u, v)$. Z toho nám vyplýva:

- $u \cdot v > 0$ práve vtedy, ak uhol vektorov u a v je ostrý
- $u \cdot v < 0$práve vtedy, ak uhol vektorov u a v je tupý
- $u \cdot v = 0$ práve vtedy, ak uhol vektorov u a v je pravý

Smerový vektor priamky p je každý vektor rovnobežný s priamkou p.
Normálový vektor priamky p je každý vektor kolmý na priamku p.
Smerový vektor roviny α je každý vektor rovnobežný s rovinou α.
Normálový vektor roviny α je každý vektor kolmý na rovinu α.

4.3.3 Rovina

Rovina (angl. plane) patrí k základným geometrickým útvarom. Je to plocha určená troma bodmi alebo priamkou a jedným bodom ležiacim mimo priamky, poprípade dvoma netotožnými rovnobežnými priamkami. V matematike rovina predstavuje dvojrozmerný geometrický útvar, ktorý môžeme algebraicky zadefinovať ako *množinu izomorfných bodov s dvojrozmerným lineárnym priestorom.*

Normálová rovnica roviny určenú bodom X_0a normálovým vektorom n:

$$(X - X_0) \cdot n = 0. \tag{4.10}$$

Ak máme súradnice normálového vektora $n = [a, b, c]$ a zadaného bodu $X[x_0, y_0, z_0]$, dosadením do (4.10) dostávame $a(x - x_0) + b(y - y_0) + c(z - z_0) = 0$ a po úprave si vyjadríme všeobecnú rovnicu roviny (4.11):

$$ax + by + cz + d = 0, \tag{4.11}$$

kde a, b, c, d sú reálne čísla, $n = [a, b, c]$ je normálový vektor roviny a číslo $-d$ je rovné jeho skalárnemu súčinu s polohovým vektorom ľubovoľného bodu roviny. Teraz už jednoducho vieme zistiť vzdialenosť bodu $X[x_0, y_0, z_0]$ od roviny α určenej vyššie uvedenou rovnicou dosadením do vzorca:

$$d(X_0, \alpha) = \frac{|ax_0 + by_0 + cz_0 + d|}{\sqrt{a^2 + b^2 + c^2}} \tag{4.12}$$

4.3.4 Vzájomné polohy

Priesečníkom geometrických útvarov rozumieme bod, ktorý majú tieto útvary spoločný.

Vzájomné polohy dvoch priamok: $p(A,u)$, $q(B,v)$, kde u, v sú smerové vektory priamok alebo normálové v rovine:

- $v = ku$ a $p \cap q = \emptyset$ rovnobežné rôzne
- $v = ku$ a $p \cap q \neq \emptyset$ rovnobežné totožné
- $v \neq ku$ a $p \cap q = \{P\}$ rôznobežné

V priestore môže ešte nastať prípad: $v \neq ku$ a $p \cap q = \emptyset$. V tomto pipade sú priamky mimobežné a priamky týmpádom neležia v jednej rovine.

Vzájomná poloha dvoch rovín $\alpha: ax + by + cz + d = 0$ a $\beta: ex + fy + gz + h = 0$, kde $n = [a,b,c]$ a $m = [e,f,g]$ sú normálové vektory rovín α, β.

- $\exists k \in R, k \neq 0: n = km$ a $d = kh$ roviny sú totožné
- $\exists k \in R, k \neq 0: n = km$ a $d \neq kh$ roviny sú rovnobežné a rôzne
- $\forall k \in R, k \neq 0: n \neq km$ roviny sú rôznobežné

Vzájomná poloha priamky $p(A,s)$ a *roviny* $\alpha: ax + by + cz + d = 0$, kde $n = [a,b,c]$ je normálový vektor roviny a s je smerový vektor priamky.

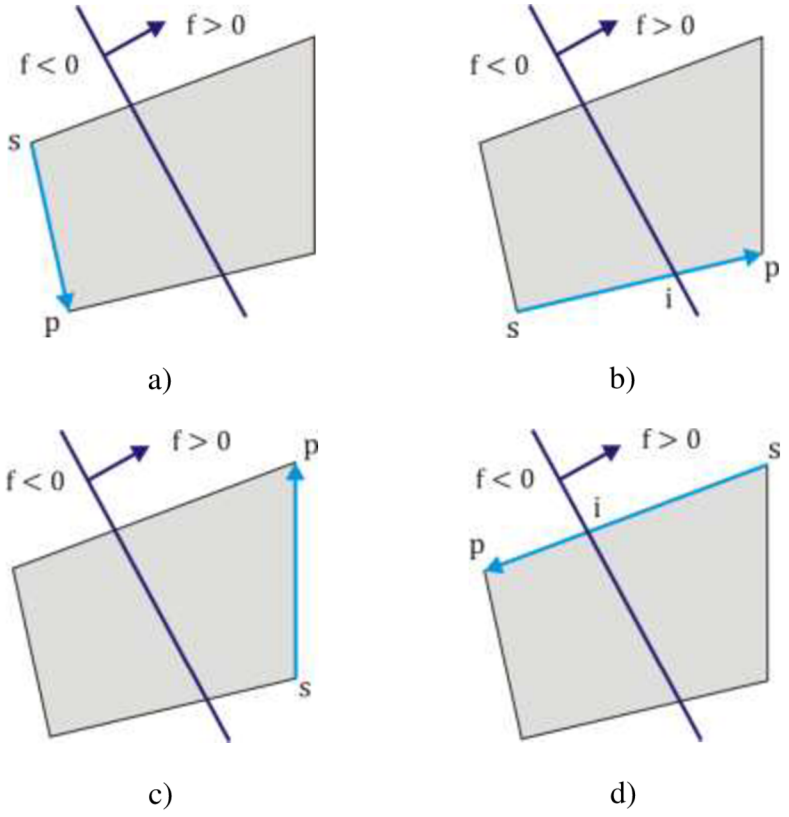
- $s \cdot n = 0$ a $p \cap \alpha \neq 0$ priamka p leží v rovine α
- $s \cdot n = 0$ a $p \cap \alpha = 0$ priamka p je rovnobežná s rovinou α
- $s \cdot n \neq 0$ priamka p je rôznobežná s rovinou α

Ak je priamka rôznobežná s rovinou, a poznáme ich smernice, resp. normály, jedoduchými spôsobmi už dokážeme dopočítať súradnice ich priesečníka.

4.3.5 Delenie polygónu rovinou

Delenie polygónu rovinou záleží v prvom rade od toho, na ktorej strane roviny sa polygón nachádza. Tomuto sa hovorí predný ($f < 0$) / zadný ($f > 0$) test (z angl. front-back test, Obr.4.7). Tento test je vykonávaný na základe testovania každého bodu polygónu voči rovine. Ak všetky body ležia na jednej strane roviny, tak je celý polygón na tejto strane a tým pádom nemusí byť rozdelovaný. Ak sú body rozmiestnené na oboch stranách, v tom prípade je polygón rozdelovaný do dvoch alebo viacerých kusov.

Základný algoritmus spočíva v prejdení pozdĺž všetkými vrcholmi polygónu a nájdu sa tie vrcholy, ktoré sú na daných stranách roviny f. Priesečníky týchto hrán a roviny sú vypočítané a tieto body sa použijú ako nové vrcholy pre novovzniknuté polygóny. Môžu nastať 4 prípady (Obr. 4.7).



Obr. 4.7: Delenie polygónu rovinou

		predný výstup	zadný výstup
a)	$f(s) < 0, f(p) < 0$	$\emptyset$	p
b)	$f(s) < 0, f(p) > 0$	p, i	i
c)	$f(s) > 0, f(p) > 0$	p	$\emptyset$
d)	$f(s) > 0, f(p) < 0$	i	p, i

4.3.6 Matice

Schému $m \cdot n$ čísiel $\boldsymbol{a_{11}, \ldots, a_{1n}, a_{21}, \ldots, a_{2n}, \ldots, a_{m1}, \ldots, a_{mn}}$ zapísanú v tvare

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & a_{(m-1)n} \\ a_{m1} & \cdots & a_{m(n-1)} & a_{mn} \end{pmatrix} \tag{4.13}$$

nazývame maticou typu $m/\,n$.
N-ticu $(a_{k1} \ldots a_{kn})$ $(1 \leq k \leq m)$ nazývame k-*ty* riadok matice A,
m-ticu $(a_{1i} \ldots a_{mi})$ $(1 \leq i \leq n)$ nazývame i-ty stĺpec matice A.
Číslo a_{ki} $(1 \leq k \leq m, 1 \leq I \leq n)$ nazývame *prvok* na (k, i)-tej pozícii v matici A.

Matice typu $1/\,n$ nazývame *horizontálnymi* alebo *riadkovými vektormi*. Značíme ich malými písmenami $a = |\,a1\ a2 \ldots an\,|$.

Diagonálnymi maticami nazývame matice, pre ktoré platí:

$$a_{ik} = 0, \text{pre } i \neq k \tag{4.14}$$

$$a_{ik} \neq 0, \text{pre } i = k \tag{4.15}$$

$$A = \begin{pmatrix} a_{00} \neq 0 & 0 & 0 & 0 \\ 0 & a_{11} \neq 0 & 0 & 0 \\ 0 & 0 & a_{22} \neq 0 & 0 \\ 0 & 0 & 0 & a_{33} \neq 0 \end{pmatrix} \tag{4.16}$$

Jednotkovou maticou nazývame diagonálnu maticu I, v ktorej sa všetky diagonálne prvky rovnajú 1:

$$a_{ik} = 0, \text{pre } i \neq k \tag{4.17}$$

$$a_{ik} \neq 1, \text{pre } i = k \tag{4.18}$$

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.19}$$

Determinant štvrocovej matice rádu n nazývame súčet všetkých súčinov n prvkov tejto matice takých, kde v žiadnom z uvedených súčinov sa nevyskytujú dva prvky z toho istého riadku ani stĺpca. Každý súčin pritom násobíme číslami r a s, kde r predstavuje znamienko príslušného poradia prvých indexov a s znamienko príslušného poradia druhých indexov.

Inverzná matica k danej matici je taká, po ktorej vynásobení s pôvodnou maticou dostávame jednotkovú maticu. Inverznú maticu k matici *A* značíme A^{-1}, dopočítava sa pomocou adjungovanej matice.

$$A \cdot A^{-1} = A^{-1} \cdot A = 1 \tag{4.20}$$

Ak v matici A zameníme stĺpce s riadkami, dostaneme *transponovanú maticu* A^T, ktorú možno definovať rovnicou $a_{xy}^T = a_{yx}$. Rovnako aj pri implementácii, jednoducho vymeníme stĺpce za riadky: $A^T[x][y] = A[y][x]$.

Násobenie matíc skalárom je vynásobenie každého prvku matice reálnym číslom z definičného oboru.

$$A \times c = \begin{pmatrix} 1 & -2 & 3 \\ 4 & 5 & 6 \\ 7 & -8 & 9 \end{pmatrix} \times (-1) = \begin{pmatrix} -1 & 2 & -3 \\ -4 & -5 & -6 \\ -7 & 8 & -9 \end{pmatrix} \tag{4.21}$$

Násobenie matíc môže fungovať len vtedy, ak je počet stĺpcov ľavej matice rovnaký ako počet riadkov pravej matice. Ak *A* je *m*-krát-*n* matica a *B* je *n*-krát-*m* matica, tak ich *maticový produkt AB* má rozmery *m*-krát-*n* (*m* počet riadkov (ako v prvej matici) -krát- *n* počet stĺpcov (ako v druhej matici)).

$$A \times B = \begin{pmatrix} 2 & 6 \\ 4 & 7 \\ 3 & 1 \end{pmatrix} \times \begin{pmatrix} 0 & 2 & 1 \\ 1 & 3 & 1 \end{pmatrix} = \begin{pmatrix} 2 \cdot 0 + 6 \cdot 1 & 2 \cdot 2 + 6 \cdot 3 & 2 \cdot 1 + 6 \cdot 1 \\ 4 \cdot 0 + 7 \cdot 1 & 4 \cdot 2 + 7 \cdot 3 & 4 \cdot 1 + 7 \cdot 1 \\ 3 \cdot 0 + 1 \cdot 1 & 3 \cdot 2 + 1 \cdot 3 & 3 \cdot 1 + 1 \cdot 1 \end{pmatrix} = \begin{pmatrix} 6 & 22 & 8 \\ 7 & 29 & 11 \\ 1 & 9 & 4 \end{pmatrix} \tag{4.22}$$

Násobenie matice s vektorom dosiahneme tak, že každý riadok matice vynásobíme vektorom tak, že násobíme medzi sebou navzájom zodpovedajúce prvky a ich súčiny sčítame.

$$A \times v = \begin{pmatrix} -1 & -2 & 3 \\ 4 & 5 & 6 \\ 7 & -8 & 9 \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix} = \begin{pmatrix} (-1) \cdot 1 + 2 \cdot 0 + 3 \cdot 2 \\ 4 \cdot 1 + 5 \cdot 0 + 6 \cdot 2 \\ 7 \cdot 1 + (-8) \cdot 0 + 9 \cdot 2 \end{pmatrix} = \begin{pmatrix} 5 \\ 16 \\ 25 \end{pmatrix} \tag{4.23}$$

$$A \times v = \begin{pmatrix} [00] & [01] & [02] \\ [10] & [11] & [12] \\ [20] & [21] & [22] \end{pmatrix} \times \begin{pmatrix} [0] \\ [1] \\ [2] \end{pmatrix} = \begin{pmatrix} [00] \cdot [0] + [01] \cdot [1] + [02] \cdot [2] \\ [10] \cdot [0] + [11] \cdot [1] + [12] \cdot [2] \\ [20] \cdot [0] + [21] \cdot [1] + [22] \cdot [2] \end{pmatrix} = \begin{pmatrix} [0] \\ [1] \\ [2] \end{pmatrix} \tag{4.24}$$

4.4 Geometrické transformácie

4.4.1 Posunutie

Pusunutie je geometrická transformácia, ktorá poskytuje premiestnenie objektu z jednej pozície na druhú. Transformácia posunutia je určená vektorom posunutia

$$v = (Tx, Ty, Tz) = (X' - X,\ Y' - Y, Z' - Z),$$

kde (X, Y, Z) sú súradnice objektu v pôvodnej pozícii a (X', Y', Z') v cieľovej pozícii.

Transformačná matica pre posunutie v 3D je nasledovná:

$$A = \begin{pmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.25}$$

4.4.2 Rotácia

Transformácia objektu okolo pevného bodu po kruhovej dráhe sa nazýva rotácia. Je určená uhlom a stredom rotácie. Otočením bodu $A[x, y, z]$ okolo stredu súradnicovej sústavy $O[0,0,0]$ o uhol α dostávame nový bod A' so súradnicami $[X', Y', Z']$:

a) Rotácia okolo osi x o uhol α

$$X' = X \tag{4.26}$$

$$Y' = Y \times cos(\alpha) - \mathrm{Z} \times \sin(\alpha) \tag{4.27}$$

$$Z' = Y \times sin(\alpha) + \mathrm{Z} \times \cos(\alpha) \tag{4.28}$$

b) Rotácia okolo osi y o uhol α

$$X' = X \times cos(\alpha) + \mathrm{Z} \times \sin(\alpha) \tag{4.29}$$

$$Y' = Y \tag{4.30}$$

$$Z' = Z \times cos(\alpha) - \mathrm{X} \times \sin(\alpha) \tag{4.31}$$

c) Rotácia okolo osi z o uhol α

$$X' = X \times cos(\alpha) - \mathrm{Y} \times \sin(\alpha) \tag{4.32}$$

$$Y' = Y \times cos(\alpha) - \mathrm{X} \times \sin(\alpha) \tag{4.33}$$

$$Z' = Z \tag{4.34}$$

Transformačné matice pre jednotlivé prípady potom vyzerajú nasledovne:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.35}$$

a) b) c)

4.4.3 Skosenie

V trojrozmernom priestore máme skosenie rozdelené podľa smeru jednotlivých rovín xy, xz a yz. Hodnoty Sx, Sy a Sz udávajú mieru skosenia. Transformačné matice pre skosenia podľa jednotlivých rovín sú nasledovné:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ Sx & Sy & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ Sx & 1 & \mathrm{Sz} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A = \begin{pmatrix} 1 & Sy & Sz & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.36}$$

a) rovina xy b) rovina xz c) rovina yz

4.4.4 Zmena mierky

Pri tejto geometrickej transformácii dochádza k zmene veľkosti objektu v udaných smerov súradnicových osí. Ak sú koeficienty väčšie ako 1, objekt sa zväčšuje, ak sú naopak menšie ako 1 objekt sa zmenšuje. Transformačná matica pre 3D je nasledovná:

$$A = \begin{pmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.37}$$

5 Implementácia

Naledujúce kapitoly popisujú použité použité technológie, nástroje a knihovne, ktoré som pri implementácii aplikácie využil. Záverečná časť tejto kapitoly je venovaná samotnej implementácii, využitiu teórie v programovaní aplikácie.

5.1 Použité rozhrania a nástroje

Naskytuje sa nám mnoho spôsobov akými docieliť požadované ciele. Môžme si vybrať zo skupín či už programovacích jazykov, implementačných prostredí, rôznych vývojových prostredi ako aj rozličných knihovieň a nástrojov.

5.1.1 C++

Programovacie jazyky sa delia z mnohých hľadísk. Môžu byť napr. vyššie a nižšie, kompilované a interpretované, imperatívne a deklaratívne, štruktúrované a objektovo orientované, atď. Z najznámejších spomeniem azda programovacie jazyky ako C++, C#, Java, Pascal, Python, Perl, VHDL, shell, ai.

Vzhľadom k doterajším skúsenostiam a po dohode s vedúcim tejto bakalárskej práce som si pre programovanie aplikácie zvolil jazyk C++. Je to objektovo orientovaný programovací jazyk, vyvinutý Bjarnom Stroustrupom a ďalšími v Bellových laboratóriách AT&T rozšírením jazyka C. C++ podporuje niekoľko programovacích štýlov (paradigmat) ako je procedurálne programovanie, objektove orientované programovanie a generické programovanie, nie je teda jazykom čisto objektovým. V súčasnej dobe patrí jazyk C++ k najrozšírenejším programovacím jazykom.

5.1.2 Visual Studio 2005

Vývojové prostredie, IDE - Integrated development environment, je software uľahčujúci prácu programátorov, zameraný väčsinou na nejaký programátorský jazyk. Zvyčajne obsahuje editor zdrojového kódu, compiler a väčsinou i debugger. Niektoré obsahujú i rozhranie pre rýchly návrh grafického užívateľského rozhrania, GUI - Graphical User Interface. K vytvoreniu aplikácie som použil vývojové prostredie od spoločnosti Microsoft – Visual Studio 2005. Oproti predošlým verziám prešlo viacerými zmenami v mnohých ohľadoch,či už v bezpečnosti alebo komfortnosti práce. Každopádne sa však jedná o značne zdokonalenú verziu, ktorá okrem iného obsahuje aj predošlé spomínané súčasti pre zjednodušenie práce vyvója software.

5.1.3 WinAPI

Grafické užívateľské rozhranie (Graphical User Interface – GUI), je úživateľské rozhranie, ktoré zabezpečuje interakciu užívateľa s počítačom. Poskytuje nám interaktívne ovládacie prvky ako rôzne tlačítka, ikony, textové vstupy atď. Rozhranie je ovládané užívateľom prostredníctvom klávesových vstupov, myšou, atď. Existuje viacero prostredí pre tvorbu užívateľského rozhrania, najznámejšie z nich, ktoré pripadali do úvahy a sú plne kompatibilné s jazykom C++ sú wxWidgets a WinAPI.

wxWidgets, pri vzniku wxWindows, je multiplatformný, opensource widget toolkit. Je to vlastne knihovňa napísaná v C++ pre tvorbu grafických užívateľských rozhraní GUI. wxWidgets nám

umožňuje zkompilovať a spustiť program na hneď niekoľkých počítačových platformách za minimálnych, niekedy dokonca žiadnych zmien v zdrojovom kóde. Ako som už spomínal, je implementovaná v jazyku C++, avšak jej použitie je samozrejmé aj v ostatných bežných programovacích jazykoch ako napr. C#, Java, JavaScript, Ruby, Perl, Python atď.

Windows API, skratkou WinAPI je sada aplikačných rozhraní, dostupných v operačních systémoch MS Windows. Pravidelne aktualizovaná dokumentácia tohoto rozhrania je voľne dostupná v balíkoch Windows SDK. Pre WinAPI bolo navrhnuté veľké množstvo knihovien, ktoré sa dajú použiť pre uľahčenie vývoja aplikácií. Medzi ne patrí napríklad MFC (Microsoft Foundation Class), ktorá ponúka sadu C++ tried, obalujúcich kľúčové funkcie WinAPI. Ďalej je to ATL (Active Template Library), knihovňa šablón pre COM (Component Object Model) a jej rozšírenie WTL (Windows Template Library), vyvinutá ako odľahčená varianta už spomínanej MFC. Microsoft kladie veľký dôraz na spätnú kompatibilitu so staršími verziami WinAPI. To nie je najlahšie keď vezmeme v úvahu že prvá verzia mala okolo 450 funkcií, zatiaľ čo dnešné implementácie obsahujú tisíce.

Jednou z najväčších zmien v histórii WinAPI bol prechod zo 16-bitového rozhrania na 32-bitové. Win32 bolo pôvodne súčasťou už systému NT 3.1, ale aplikácie ho začali naplno využívať až od vypustenia Windows 95. Aby dizajnéri uľahčili vývojárom prechod medzi jednotlivými verziami, API obsahovalo rutiny, ktoré umožňovali volanie 16-bitového kódu z 32-bit aplikácie a naopak. Táto schéma bola použitá v celých Windows 95, aby sa ušetrila práca. WinAPI sa delí na niekoľko základných súčastí. Rovnako sú tu základné služby, ktoré zprostredkuvajú prístup k súborovému systému, zariadeniam, procesom, vláknam ako aj ošetreniam chýb. Pokročilé služby potom umožňujú prístup k registrom, plánovaniam procesov, užívateľským účtom a správe energie. Grafická časť umožňuje prístup k grafickým rozhraniam, takisto ako aj monitorom, tlačiarňam a vytváraniu okien so základnými ovládacími prvkami ako sú tlačítka, scrollbary, slidery. Umožňujú nám aj interakciu pomocou klávesnice alebo myši. Od Windows XP je toto spojené s predtým oddelenými časťami, Common Controls Library (obecné ovládacie prvky), poskytujúce ďalšie ovládacie prvky ako ukazovatele priebehu, stavové lišty, panely nástrojov a podobne. WinAPI taktiež poskytuje obecné dialogy, kam napríklad patrí aj známy dialog pre otvorenie súboru, výber fontov, farby a podobne. Máme umožnený aj prístup k windows shell, príkazovému riadku operačného systému MS Windows. Akousi poslednou vrstvou sú sieťové služby, kam patrí NetBIOS, Winsock, NetDDE, RPC.

Keďže neplánujem multiplatformné implementovanie a vystačím si so základnými ovládacími prvkami, úplne si vystačím s natívnym windows rozhraním WinAPI, bez akýchkoľvek dodatočných inštalácií.

5.1.4 GDI

Pre zobrazenie jednoduchej grafiky na oknách aplikácie, ktoré priamo nezobrazujú grafický výstup jednotlivých diem, je použitá knihovňa Graphics Device Interface (GDI). Tá je súčasťou WinAPI, v systéme je zodpovedná za úlohy ako kreslenie jednoduchých geometrických útvarov (čiary, obdĺžniky, kružnice, atď.), vykreslovanie fontov a manažment farebných paliet, keď je to potrebné.

Dôležitou vlastnosťou GDI je vysoká vrstva abstrakcie výstupného zariadenia, čo nám povoluje použiť jeden kód pre kreslenie na monitora aj na tlačiareň s rovnakými výsledkami.

GDI dokáže zobrazovať jednoduchú 2D grafiku, ale moc sa nám nebude hodiť na animácie, kedže neobsahuje žiadny spôsob synchronizácie s framebufferom a spôsobuje tak nepríjemné preblikávanie.

Rozšírením v novších verziách operačných systémov (XP) je knihovňa GDI+. Je založená na programovacom jazyku C++, pridáva antialiassing, desatinný súradnicový systém (súradnice v GDI pre vykreslovanie sú celé čísla), prechody farieb a vstavaná podpora grafických formátov ako JPEG

alebo PNG (GDI pracovalo len s formátom BMP). Taktiež vie interpretovať alfa kanál obrázkov a vykreslovať tak efekty s priehľadnosťou. Avšak, GDI+ má taktiež aj nejaké nevýhody. Napríklad v implementácii knihovni JPEG bola chyba, spôsobujúca spúšťanie kódu, obsiahnutého v JPEG obrázku, ktorá bola následne odstránená aktualizáciou zo septembra 2004. Ďalšou nevýhodou je rýchlosť, napríklad vykreslovanie textu je v GDI+ zhruba desaťkrát pomalšie.

Pre naše účely si bohate vystačíme s knihovňou GDI, ktorej použitie môžme pozorovať hneď v hlavnom okne.

5.1.5 OpenGL

Na Windows je dostupné obľúbené rozhranie OpenGL, vyvíjané spoločnosťou SGI. Na rozdiel od DirectX, ktorá odkrýva funkcie grafického hardware, OpenGL je špecifikácia grafických funkcií a ich chovanie. OpenGL doslova skrýva samotné rozhranie a možnosti grafického hardware-u, to znamená že keď náš hardware nepodporuje nejakú funkciu, OpenGL použije softwarovú emuláciu. Vlastná implementácia je pritom vytváraná vyrobcami grafických kariet. Existujú implementácie OpenGL pre Mac OS, Windows, Linux a mnohé ďalšie unixové operačné systémy. Existuje taktiež niekoľko implementácií, poskytujúcich funkcie OpenGL na platformách, kde hardwarová akcelerácia nie je dostupná, najznámejšia z nich je Mesa. OpenGL je neustále vyvíjané, ajkeď už nie firmou SGI. Od roku 2006 sa o vývoj stará Khronos Group. Je ustanovená rada pre údržbu architektúry OpenGL, ktorá pozostáva z výrobcov grafických kariet a ďalších firiem ktoré sa podieľajú na vývoji. Microsoft pôvodne podal návrh na jej založenie, ale opustil ju v roku 1993.

OpenGL nemá toľko verzií ako DirectX (posledná verzia OpenGL je 3.0, verzií DirectX je už 11). Nové funkcie sú prístupné ako rozšírenia, linkované dynamicky a pôvodné API zostáva nezmenené. Čo sa týka programoateľného tieňovania, OpenGL ponúka hneď tri jazyky, prvý a najstarší je veľmi podobný assembleru. Bežne sa nazýva aj nízkoúrovňovým. Druhý, vysokoúrovňový je veľmi podobný jazyku C. Nakoniec je tu tretia alternatíva firmy NVidia, programovací jazyk CG. CG má vynikajúci kompilátor, ktorý dokáže preskupiť operácie podľa obsadenej grafickej karty tak, aby sa maximálne využili jej vlastnosti. Na grafických kartách firmy NVidia, je tento kompilátor použitý i pre preklad vysokoúrovňového tieňovacieho jazyka OpenGL. Vzhľadom k vysokej rýchlosti rozvoja v oblasti grafických akcelerátorov dáva väčší zmysel použitie vyššieho jazyka, kedže v nových generáciách grafických kariet pravdepodobne pobežia optimálnejšie než assembler, ladený na kartách, dostupných v dobe vývoja aplikácie.

5.1.6 OpenGL v demonštráciách

Pre vykreslovanie demonštrácií je použité práve rozhranie OpenGL, a to hneď z niekoľkých jednoduchých dôvodov. Jednak je to moderné rozhranie poskytujúce pokročilé grafické funkcie ,a jednak je platformovo nezávislé a nakoniec väčšina jednoduchých funkcií je implenetovaná v rýchlom referenčnom rasterizéri, takže aplikácia pobeží i na staršej výpočetnej technike, s ktorou sa často v školstve stretávame.

Knižnica OpenGL obsahuje približne 250 funkcií pre definovanie a zobrazovanie objektov. Objekty v OpenGL pozostávajú zo základných geometrických primitív ako bod, priamka, trojuholník, polygón, úsečka, atď. Tieto primitíva sú znázornené na Obr. 4.4. Každej entite je možné nadefinovať rôzne atribúty, takže vieme presne naprogramovať, ako sa má daný element zobrazovať. Tieto primitíva / elementy môžeme hierarchicky spájať do štruktúr nazývaných *D-list*. Vytvorené objekty sú nadefinované v 3-dimenzionálnom priestore. Jedná sa o modelový priestor, v ktorom máme umožnené operácie nad tymito objektmi ako otáčanie, posúvanie, zmena mierky.

OpenGL pracuje ako stavový automat, väčsina funkcií je použitá práve k nastaveniu určitého stavu, ktorý napr. určuje akým spôsobom sa grafické primitíva budú kresliť (od typu projekcie až po farby, textúry, osvetlovanie). Druhá skupina funkcií slúži k samotnému vykreslovaniu grafických primitív. Nasledujúca časť obsahuje prehľad niektorých základných funkcií využitých v programe.

Pri programovaní v OpenGL sa ale nezaobídeme bez znalosti matematiky, hlavne čo sa týka analytickej geometrie. V kapitole 4 som opísal základné matematické entity a vzťahy, s ktorými sa pri programovaní v OpenGL každý stretne.

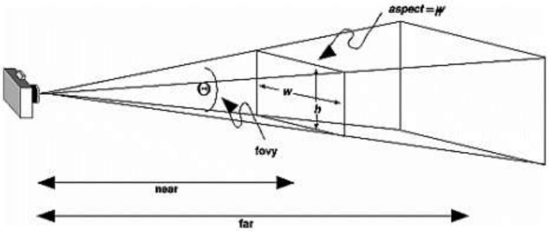
`glViewport(x, y, width, height)`

Príkazom definujeme oblasť v okne operačného systému, do ktorej sa bude naša OpenGL scéna vykreslovať. Jedná sa o transformáciu normalizovaných súradníc x_{nd}, y_{nd} scény na súradnice klasického Windows okna x_w, y_w. Tieto súradnice sú prepočítavané vzorcami (5.1) a (5.2). Parametre funkcie x, y špecifikujú ľavý dolný roh oblasti v pixloch. Width, height určujú rozmery oblasti. Po pripojení GL kontextu k oknu, tieto parameter sú nastavené vzhľadom k rozmerom okna.

$$x_w = (x_{nd} + 1)\left(\frac{width}{2}\right) + x \tag{5.1}$$

$$y_w = (y_{nd} + 1)\left(\frac{height}{2}\right) + y \tag{5.2}$$

Trojrozmerný priestor, do ktorého potom umiestňujeme vytvorené objekty môžeme definovať viacerými spôsobmi. Najznámejším príkazom pre definíciu pohľadového priestoru, v aplikácii použitým, je `gluPerspective()`.



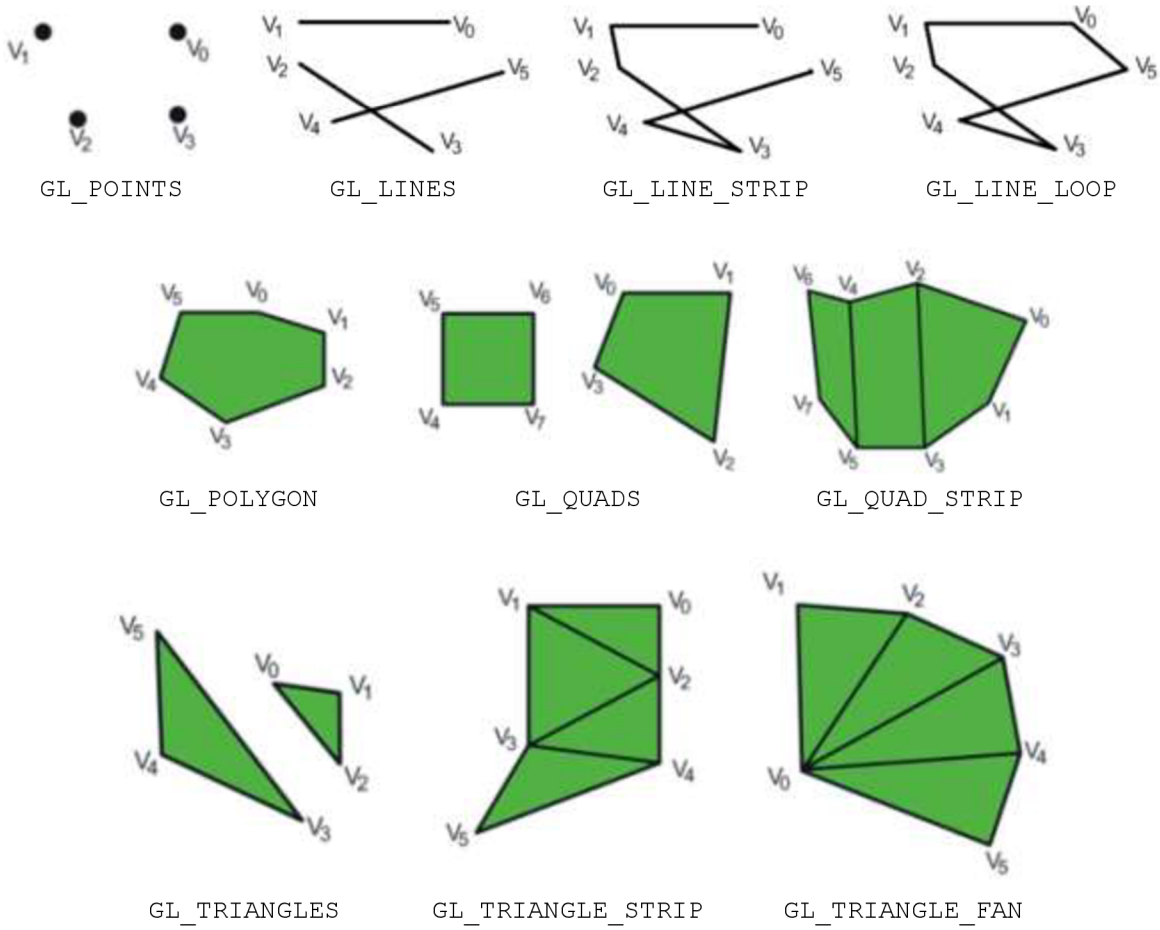
Obr. 5.1: Znázornenie definovania pohľadového priestoru

`glDepthMask()`

Táto funkcia s parametrami GL_TRUE / GL_FALSE slúži na povolenie / zákaz zápisu do hĺbkového Z buffera. Defaultne je tento zápis povolený.

`glBegin(GLenum mode)`

Tento príkaz slúži na zahájenie bloku pre vykreslovanie jednotlivých primitív v OpenGL. Za ním nasledujú príkazy s informáciami o geometrii, ktorú chceme vykreslovať. Parameter tejto funkcie môže nadobúdať hodnoty znázornené na Obr. 5.2.



Obr. 5.2 : Geometrické primitíva v OpenGL

`glEnd()`

Je protipárom k príkazu `glBegin()`, ukončuje blok vykreslovania geometrie.

`glVertex()`

Každý objekt v OpenGL dokážeme opísať pomocou sústavy navzájom pospájaných vrcholov. Každý vrchol nesie informácie o jeho položení v priestore. Na vytváranie vrcholov v OpenGL slúžia práve funkcie `glVertex()`.

`glColor3f( GLfloat r, GLfloat g, GLfloat b)`

Týmto príkazom sa v OpenGL nastavujú jednotlivé farebné zložky výslednej farby, ktorá je následne aplikovaná. Parametre r, g, b predstavujú intenzity červenej, zelenej a modrej zložky v rozmedzí 0.0f – 1.0f. Keďže sa v počítačovej grafike pracuje s aditívnym skladaním farieb (RGB), takže pri zavolaní funkcie s hodnotami 0.0f, 0.0f, 0.0f dostaneme výslednú čiernu a naopak pri 1.0f, 1.0f, 1.0f výslednú bielu farbu. V OpenGL predstavuje príkaz glColor3f() akýsi prepínač, po ktorého spustení sa daná farba aplikuje na všetky, po ňom aplikované vrcholy.

`glClearColor()`

Pred vymazaním zásobníku farieb je potrebné určiť hodnotu, akú nadobudnú všetky jeho vymazané hodnoty. Toto sa vykonáva pomocou vyššie uvedeného príkaza. Samotné vymazanie obsahu zásobníka potom vykonáva príkaz `glClear()`.

`glEnable()`

Tento príkaz spolu s jeho protipárom `glDisable()` slúžia na zapínanie / vypínanie jednotlivých stavov v OpenGL. Na prípadné zisťovanie aktuálneho stavu sú tu funkcie `glIsEnabled()` a `glGet()`.

`glMatrixMode(GLenum mode)`
Slúži na zmenu transformačnej matice. Ako parameter udávame typ transformačnej matice, ktorú budeme ďalšími príkazmi meniť:

- `GL_MODELVIEW` – buSSde sa meniť *MODELVIEW MATRIX*, to znamená, že budeme manipulovať s maticou, v ktorej sú uložené modelové a pohľadové tranformácie
- `GL_PROJECTION` – bude sa meniť *PROJECTION MATRIX*, čiže matica obsahujúca nastavenia perspektívnej alebo ortogonálnej projekcie kamery
- `GL_TEXTURE` – bude sa meniť *TEXTURE MATRIX*, takže matica, ktorá sa používa pri mapovaní textúr na objekty

`glLoadIdentity()`
Táto funkcia nahraje do aktuálne zvolenej transformačnej matice koeficienty zodpovedajúce jednotkovej matici. V praxi to znamená že vynulujeme všetky prvy, okrem prvkov hlavnej diagonály, ktoré budú nastavené na hodnotu 1.

Pri nasledovných funkciách sa nemanipuluje priamo s prvkami matíc, ale zadávajú sa základné lineárne transformácie – posun, rotácia, zmena mierky. Pre zadané transformácie sa vytvorí dočasná matica a aktuálna matica je touto maticou vynásobená.
`glTranslatef()`
Špecifikuje posun o vektor $[x, y, z]$. Ako som už vyššie písal, aktuálna matica sa vynásobí transformačnou maticou.
`glRotatef()`
Tento príkaz, alebo funkcia, špecifikuje transformáciu rotácia. Objekt je otočený o daný uhol okolo osy prechádzajúcej počiatkom súradnicovej sústavy a bodom so súradnicami x, y, z. Čiže sa jedná o rotáciu vektorom $[x, y, z]$. Uhol je zadávaný klasicky, v stupňoch.

5.1.7 TinyXML

TinyXML je jednoduchá knihovňa, napísaná v C++, implementujúca jednoduchý XML parser. Samotné data sú reprezentované ako strom elementov, v ktorom sa dá vyhľadávať, upravovať ho a ukladať späť do XML. TinyXML neposkytuje žiadnu konverziu znakových sád a interpretuje čítané data ako kódovanie UTF-8.

V aplikácii se pomocou XML ukládajú jednotlivé moduly. Samotný zdrojový kód neobsahuje žiadne špeciálne znaky a absencia konverzie kódovania v tom prípade nie je prekážkou. Súčasťou každého souboru ale má byť i obrázok vo formáte PNG, ktorý však bude obsahovať všetky binárne hodnoty a pri ich interpretácii ako UTF-8 by zrejme došlo k strate informácie. Preto je pre ukladanie obrázkov do XML využité kódovanie Base-64, ktoré prevádza binárne data na anglický text.

5.1.8 LibPNG

Knihovňa LibPNG je použitá k jednoduchému načítániu ikoniek diem vo formáte PNG (a možno bude použitá ku kompresii ďalších obrázkov, ktoré aplikácia môže obsahovať – ďalšie grafické ovládacie prvky, priloženie obrázkov do definícií a podobne).

5.2 Realizácia hlavného okna

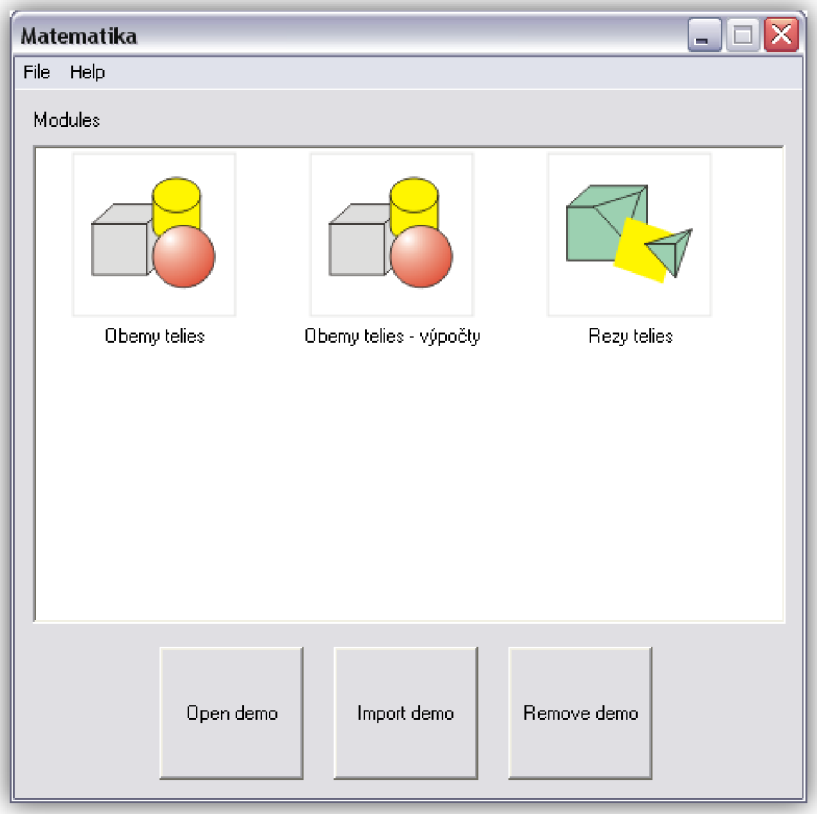
Návrh okna prostredníctvom vývojového prostredia Visual Studio sa stáva veľmi jednoduchým a intuitívnym, keďže obsahuje vizuálny editor zdrojov (z angl. resource editor). Tieto editory zdrojov

nám poskytujú rôzne techniky a nástroje pre vytváranie a upravovanie zdrojov rýchlo a efektne. V tomto procese sa definuje napr. rozmiestnenie prvkov v dialogu, popr. nastavenie ich východiskových atributov (napr. nastavenie identifikátora pre button, atď.) Výstupom takéhoto procesu je *.rc súbor. Samotné volania funkcií a jednotlivé akcie vykonávané užívateľom, ako napr. aj vytvorenie základného okna pre dialógy a rôzne ovládacie prvky však už "naklikať" nemôžme, tento účel nám spĺňa WinAPI.

Systém Windows (program vo WinAPI) je založený na princípe zasielaní správ. Tieto správy majú za úlohu poskytnúť informáciu o stlačení určitej klávesy, posune myšou, zmene veľkosti okna, atď. Každá správa sa skladá z identifikátora a dvoch parametrov.

Kľúčovou časťou takéhoto okna vo WinAPI je hlavná smyčka, ktorá zabezpečuje príjem a zpracovanie jednotlivých správ zaslaných aplikáciou. Táto smyčka je umiestnená vo funkcii WinMain, ktorá sa volá pri spustení okna aplikácie. Tu sa takisto prevádza inicializácia a tvorba hlavného okna.

Taktiež tam registrujeme a vytvárame triedu nášho okna. Parameterom `lpfnWndProc` definujeme názov funkcie, ktorá bude obsluhovať práve udalosti týkajúce sa okna. Je to v podstate funkcia, ktorá nám bude obsluhovať naše okno a zachytávať aj spomínané správy, reagovať na ne.



Obr. 5.3: Screenshot hlavného okna

5.3 Realizácia modulov

Každý modul je reprezentovaný dynamicky linkovanou knižnicou, ktorá obsahuje kód demonštračného programu. K nej patrí xml súbor, obsahujúci popis pluginu, odkaz na obrázok s náhľadom (thumbnail) a ďalej obsahuje všetky stringy, ktoré plugin zobrazuje. Stringy sú v xml uložené v troch jazykoch, jednak slovensky a potom ešte česky a anglicky. Jazyk sa vyberá

v hlavnom programe a plugin teda následne dostáva stringy v danom jazyku transparentne, bez potreby písania nejakého kódu. Ukážkový xml súbor vypadá nasledovne:

```
<?xml version="1.0" encoding="utf-8" ?>
<demo version="1.0">
	<title>
		<sk>Rezy telies</sk>
		<cs>Řezy těles</cs>
		<en>Slices demo</en>
	</title>
	<description>
		<sk>Demonštrácia rezov telies</sk>
		<cs>Demonstrace řezů tělesy</cs>
		<en>Volume slices demonstration</en>
	</description>
	<thumbnail>slices.png</thumbnail>
	<library>SlicesDemo.dll</library>
	<string-table>
		<sk>
			<string key="slider_phi">uhol x</string>
			<string key="slider_ro">uhol y</string>
			<string key="slider_dist">posuv</string>
			<string key="slider_sep">rez</string>
		</sk>
		<cs>
			<string key="slider_phi">úhel x</string>
			<string key="slider_ro">úhel y</string>
			<string key="slider_dist">posun</string>
			<string key="slider_sep">řez</string>
		</cs>
		<en>
			<string key="slider_phi">angle x</string>
			<string key="slider_ro">angle y</string>
			<string key="slider_dist">offset</string>
			<string key="slider_sep">cut</string>
		</en>
	</string-table>
</demo>
```

Toto je xml pre plugin rezy telies. Každý dokument obsahuje jeden element demo. Atribut version udáva verziu plug-inu, súčasné verzie sú “1.0”. Element title obsahuje názov plug-inu tak, ako sa zobrazuje v menu a v zozname s náhľadmi. Description potom obsahuje jeho popis (zobrazuje sa ako tool-tip-text v zozname s náhľadmi). Element thumbnail obsahuje url k súboru s náhľadom, format musí byť png. Element library obsahuje súbor s dynamicky linkovanou knihovňou, obsahujúcou vlastný kód demonštračného plug-inu. Nakoniec tabuľka string-table obsahuje reťazce, ktoré sú zobrazované v deme. Plugin má tieto reťazce k dispozícii ako asociatívne pole.

V programe je toto xml načítané pomocou knihovne tinyxml a je potrebné, aby jeho kódovanie bolo zhodné s kódovaním, ktoré je v operačnom systéme nastavené ako východiskové pre programy, ktoré nepoužívajú kódovanie unicode (xml stačí správne uložiť a správne prepísať hodnotu atributu encoding).

Väčšina kódu hlavnej aplikácie a pluginov sú napísané v jazyku C++. Tento jazyk nemá vo svojej špecifikácii udané, ako majú vypadať mená exportov funkcií v knihovniach dll. Preto býva

problém vytvoriť knihovňu dll v jednom kompilátore a potom ju použiť v inom. Preto sa väčšinou používa jednoduché rozhranie, napísané v jazyku "C" , pretože exporty v "C" majú štandardné mená a fungujú všade. Každý plugin verzie "1.0" demonštračnej aplikácie musí exportovať jedinú funkciu:

```
TPluginHandle *PluginInitFunc();
```

Táto funkcia vracia štruktúru, jednu inštanciu pluginu – podobne ako trieda v jazyku C++. Štruktúra `TpluginHandle` má nasledujúce pole:

```
typedef int OpenGUIFunc(TPluginHandle *plugin);
// returns 1 on ok, 0 on failure
typedef void CloseGUIFunc(TPluginHandle *plugin);
typedef int GetPositionStringFunc(TPluginHandle *plugin, char
*p_s_dest, int n_max_length);
// returns 0 = ok, -1 = error, n = number of characters needed
typedef int GetStateStringFunc(TPluginHandle *plugin, char
*p_s_dest, int n_max_length);
// returns 0 = ok, -1 = error, n = number of characters needed
typedef void SetPositionFunc(TPluginHandle *plugin, const char
*p_s_position);
// generated by GetPositionString()
typedef void SetStateFunc(TPluginHandle *plugin, const char
*p_s_state);
// generated by GetStateString()
typedef void ResetFunc(TPluginHandle *plugin);
// go to default state.
typedef void DestroyFunc(TPluginHandle *plugin);

struct TPluginHandle {
      char p_s_plugin_name[256];
      int n_version;

      OpenGUIFunc *OpenGUI;
      CloseGUIFunc *CloseGUI;
      GetPositionStringFunc *GetPositionString;
      GetStateStringFunc *GetStateString;
      SetPositionFunc *SetPosition;
      SetStateFunc *SetState;
      ResetFunc *Reset;
      DestroyFunc *Destroy;
};
```

Pole `p_s_plugin_name` obsahuje názov pluginu (malo by súhlasiť s názvom v xml, môže obsahovať copyright apod.). Verzia `n_version` je stokrát hlavný reťazec plus desaťkrát vedľajšia revízia, takže napríklad verzia 1.5 je 150. Nasledujú ukazatele na jednotlivé funkcie.

`OpenGUI` slúži k otvoreniu grafického rozhrania demonštrácie, naopak `CloseGUI` slúži k jeho násilnému zatvoreniu.

Pre účely uloženia stavu dema je potreba definovať nejaké rozhranie, ktoré by pomocou univerzálneho a prenositelného formátu dokázalo reprezentovať jak pozíciu, tak stav ovládacích prvkov a prípadne vnútorných premenných dema. Medzi najjednoduchšie reprezentácie patrí obecná binárna reprezentácia dátových štruktúr programu. Tá má však nevýhodu zložitej kontroly správnosti a nemusí byť zcela prenositeľná (little versus big endian kódovanie apod.). Preto bolo rozhodnuté, že

tieto hodnoty budú reprezentované pomocou obecného anglického reťazca v podobnom formáte, v akom sú parametry na príkazovom riadku.

K vlastnému prenosu parametrov z pluginu do hlavného okna slúži dvojica funkcií `GetPositionString` a `GetStateString`, ktoré majú za parametry výstupný string a jeho dĺžku. Funkcia nakopíruje svoj výstup do stringu a vráti nulu. Keď je výstupný string príliš krátky, vráti počet potrebných znakov (kladné číslo). Pokiaľ sa pri spracovaní vyskytne nejaká chyba, vráti `-1` (alebo iné záporné číslo). Týmito funkciami sa teda prenáša stav aplikácie do hlavného okna.

Prenos opačným smerom je realizovaný funkciami `SetPosition` a `SetState`, ktoré ako parameter dostanú string vygenerovaný ich párovými funkciami. Pokiaľ sa pri preklade textu vyskytne chyba, funkcie môžu vrátiť chybový kód, ale je nutné aby sa plugin nachádzal v použiteľnom stave. Tzn. Pokiaľ je napríklad predaná neplatná velikosť okna (nulová alebo záporná), je treba aby plugin otvoril okno s nejakou pôvodnou velikosťou. Toto sa potom vzťahuje aj na všetky ostatné načítané parametre pluginu. Je síce pravda, že plugin dostane vždy string, ktorý sám vygeneruje, ale obecne je potrebné aby očakával že bude obsahovať chyby (môžu byť spôsobené chybou v súbore, neodborným zásahom užívateľa, pri prechode na novšiu verziu pluginu).

Predposledná funkcia `Reset` slúži k uvedeniu pluginu do východiskového stavu. To sa týka skôr parametrov, nastavovaných užívateľom, ako veľkosti okna. Nepredpokláda sa, že by užívateľ dokázal zmeniť veľkosť okna tak, aby ju sám nemohol upraviť. Nakoniec je nutná funkcia pre uvoľnenie inštancie štruktúry `TPluginHandle` a ďalších dynamicky alokovaných štruktúr, ktoré obsahuje. V prípade pluginov sa nemôžme spoliehať na operačný systém, že prevedie uvoľnenie pamäte a ďalších prostriedkov, pretože knihovňa pluginu je načítaná po celú dobu behu programu a plugin tak môže byť opakovane i niekoľkokrát spustený a ukončený. Neuvoľňovanie pamäte by potom mohlo spôsobiť neblahé následky.

5.4 Implementované moduly

Spolu s kostrou aplikácie som implementoval dvojicu modulov, ktoré názorne ukazujú možnosti využitia počítačovej grafiky v danej problematike.

5.4.1 Povrch a objem telies

Demo objemy a povrchy telies zobrazuje rôzne trojrozmerné telesá (kocka, kváder, kužeľ, valce a ihlan) a demonštruje animáciu rozvinutia ich plášťa do roviny a následné zoskupenie súmerných častí a pomenovanie ich obsahov. K tomuto účelu tu existuje jednotné rozhranie `CunwrapDemoIface`:

```
class CUnwrapDemoIface {
public:
      virtual void Draw(float f_unroll, float f_explode) const = 0;
};
```

Jediná funkcia slúži k vykresleniu animácie, kde parameter f_unroll je miera rozvinutia do plochy v intervale $< 0, 1 >$ a f_explode je miera rozloženia jednotlivých súčastí plášťa v rovnakom intervale. Niektoré demá majú možnosť nastavovať f_unroll a f_explode nezávisle na sebe a iné demá požadujú aby f_explode bolo 0, pokiaľ f_unroll nebude 1 (teda najprv rozbalenie až potom rozloženie, krok za krokom). To však nie je omedzenie, pretože aplikácia zobrazuje animáciu tomto poradí. Parametre sú oddelené len, aby sa zachovali zásady objektového návrhu.

Každé demo má svoj vlastný konštruktor, ktorý prijíma rozmery telesa a prípadne ďalšie parametre. Napríklad konštruktor pre rozbalenie kvádra obsahuje nasledujúci konštruktor:

```
CBoxUnwrapDemo::CBoxUnwrapDemo(float a, float b, float c, float
f_padd = .1f),
```

kde parametre `a, b a c` sú samozrejme dĺžky strán kvádra, parameter `f_padd` je vzdialenosť jednotlivých strán pri rozloženom pohľade (z angl. padding). Konštruktor vygeneruje vrcholy objektu a následne generuje animáciu. Objekt je zložený z určitého počtu podobjektov, kde každému pripadá určitý počet grafických primitív (Obr. 5.2) a obsahuje nejaké animačné dáta.

Animácia rozbalenia plášťa je reprezentovaná ako rotácia okolo osy (teda osa rotácie ako trojrozmerný vektor a rozsah uhlov k tomu) a posunutie. Animácia následného rozloženia pohľadu je riešená už len transformáciou posunutie. Jednotlivé objekty sú usporiadané do stromovej štruktúry, kde koreň je v animácii väčsinou nehybný a jednotlivé podvetvy sa pohybujú. Platí však, že keď sa pohybuje rodič, hýbu sa aj deti. Tým sa dosiahne prirodzeného vzhľadu animácie bez väčších ťažkostí definície animácie. Tento systém spolieha na OpenGL zásobník transformačných matíc, ktorého výška je väčšinou omedzená na 32 matíc, čo znamená, že najzložitejší animačný strom môže mať výšku až 33. V programe nie sú takto zložité stromy vôbec implementované, bežná výška stromu je 3. Výnimkou sú rotačné telesá, kde je celý plášť v podstate zreťazeným zoznamom vetví s rovnakými rotáciami okolo zvislej osy. Ale aj tak je výška stromu rovná 16.

Objekty sú zobrazené v minimalistickom čiernobielom štýle, ktorý pri výuke nijak nerozptyluje a zároveň je dostatočne názorný. Plochy jednotlivých telies sú biele a hrany sú vytiahnuté čiernou farbou. U jednoduchých telies (kocka, kváder, apod.) je to jednoduché, stačí vytiahnuť hrany jednotlivých polygónov. Problém nastáva u rotačných telies, kde je jedna plocha reprezentovaná pomocou mnohých facetov a ich hrany by nemali byť zobrazené naraz. Normálne by zrejme bolo pre detekciu okrajových hrán využitie vertex shaderu, ale s ohľadom na minimálne nároky programu bol využitý jednoduchý trik. Celé teleso je najprv zobrazené v čiernej farbe o jeden pixel väčšie na všetkých stranách. Následne je prekreslené bielym v správnej veľkosti, čím dosiahneme ilúziu čierneho obrysu. Je síce potrebné ešte dokresliť niekoľko liniek, pre tie však už nie je potrebné rozhodovať či ich je alebo nie je potrebné vidieť. Tento trik však funguje len s jednoduchými telesami a len pokiaľ sú zložené z jedného kusu. Inak vznikne problém s prekrývajúcimi sa hranami.

Pre ovládanie dema bolo vytvorené jednoduché užívateľské rozhranie, kde užívateľ môže nastavovať ľubovoľný počet parametrov pomocou posuvníkov. Prvým posuvníkom sa prepína typ telesa a druhým sa ovláda stav animácie. Ďalšie posuvníky závisia od typu telesa a ich počet sa pohybuje od jedného (kocka-dĺžka strany) do troch (kváder-dĺžka troch strán). Zobrazenie dĺžok je pritom relatívne, najvyššia dĺžka je normalizovaná na dĺžku 1 a ostatné rozmery sú priamo úmerné. Tým je znížená šanca, že si užívateľ nešikovnou manipuláciou zväčší teleso tak, že prestane byť vidieť, o aké teleso sa jedná, alebo naopak zmenší tak, že ho nebude vidieť vôbec. Zmenou mierky je udržovaná približne rovnaká veľkosť telesa a v podstate sa mení len pomer strán. Screenshoty z tohto dema sú na obrázkoch 5.4 a 5.5.

5.4.2 Rezy telies

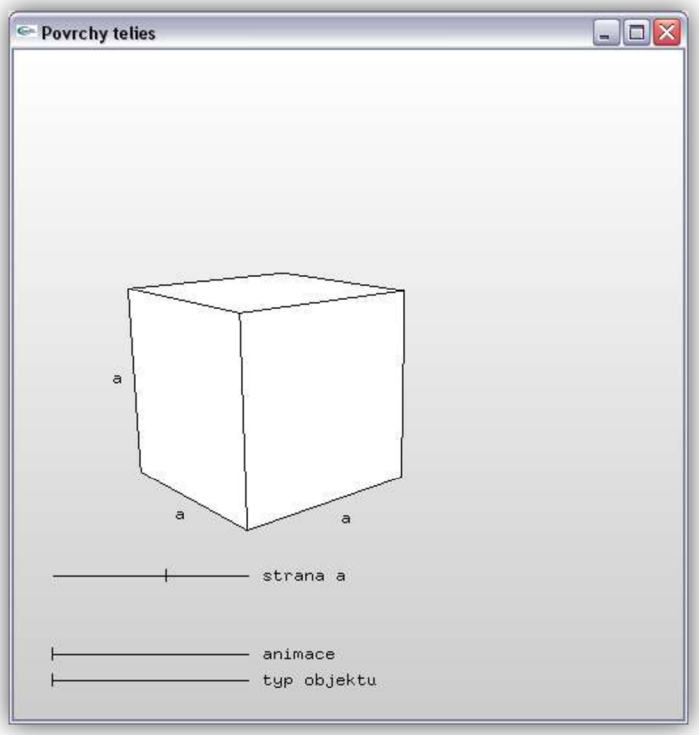
Demo rezy telies opäť zobrazuje rôzne primitíva a deliacu rovinu. Užívateľ si nastaví pozíciu a natočenie deliacej roviny a následne je možné demonštrovať rozdelenie telesa rovinou. S jednotlivými časťami telesa sa dá potom podľa potreby pohybovať myšou. S deliacou rovinou sa dá pritom manipulovať aj keď je teleso už rozdelené, pričom sa plynule mení.

Tu je použitá len jediná trieda dema, ktorá obsahuje polygonálnu reprezentáciu telesa, ktorého typ sa zadáva v konštruktore a ponúka funkcie k vykresleniu telesa a k zachytávaniu správ od užívateľa (posun myšou, kliknutie, apod.). Pre potreby výberu jednotlivých častí telesa pre ich posun

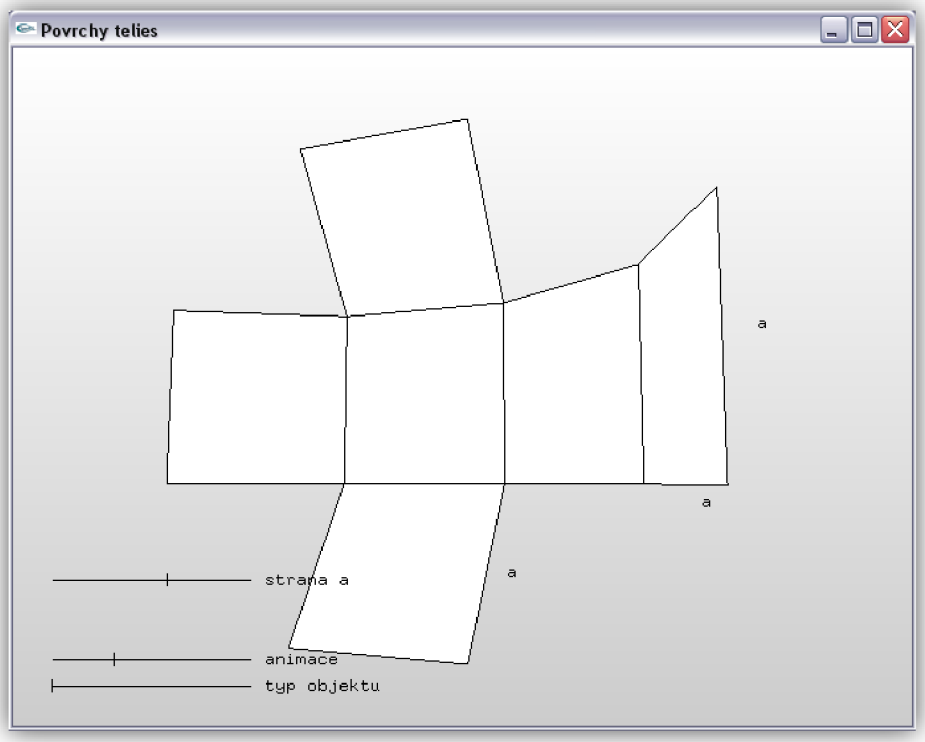
myšou je využité kreslenie telies pevnou farbou. Pri kliknutí sa jedna polovica telesa nakreslí červeno a druhá zeleno. Potom sa prečíta farba pixelu, ležiaceho pod kurzorom myši a určí sa, ktoré teleso bolo označené (alebo či bolo označené pozadie). Tento prístup sa dá ľahko rozšíriť na ľubovoľný počet telies (omedzený zhora farebným rozlíšením framebufferu). Pre následný posun telesa myšou je treba dopočítať vektor posunu. K tomu sa využíva spätná transformácia súradníc z priestora viewportu do priestora objektov. Načíta sa aktuálny modelview a projection matica, spočíta sa ich súčin, inverzia a transpozícia. Potom sa zoberie normalizovaná súradnica myši (x a y sú v intervale $<-1, 1>$) a transformuje sa do priestoru objektov so súradnicou s rovnou -1 a 1 (blízka a vzdialená rovina). Tým sa získa jednak pozícia oka a jednak vektor priamky, prechádzajúcej z oka pixelom pod kurzorom myši. Potom sa dopočíta priesečník s rovinou, v ktorej teleso leží a z dvoch priesečníkov (predchádzajúca a súčasná pozícia myši pri posune) vieme spočítať smer posunu telesa. Výsledkom je posun, kedy teleso pevne drží pod kursorom myši. Tým sa pohyb s objektom stáva intuitívnejší a jednoduchší, ako posun po jednotlivých osách, využívaný v bežných 3D aplikáciách.

Vizuálny štýl je opäť veľmi jednoduchý. Strohé čiernobiele objekty s farebne vyznačenou deliacou rovinou. Tentokrát ale nejde jednoducho vyfarbovať hrany, pretože pri každom reze sa potenciálne mení topológia objektov a už nemôžme využit trik, použiý pre kreslenie hrán v demu objemy a povrchy telies. Problém by sa však dal riešiť vertex shaderem, ktorý by mal pre každú hranu k dispozícii roviny povrchov, ktoré danú hranu zdieľajú.Pokiaľ sú súčasne obe roviny otočené ku kamere alebo od kamery, hrana nemá byť zvýraznená. Pokiaľ je však jedna rovina otočená ku kamere a druhá odvrátená od kamery, hrana je súčasťou siluety objektu. Hrany, ktoré vznikli rezom objektu sú viditeľné. Podobný princíp je využitý k extrakcii siluety pre algoritmus shadow volumes. Z dôvodu možnosti rozšírenia na kreslenie korektných siluet bol použiy explicitný kód delenia polygónov rovinou namiesto užívateľských klipovacích rovín OpenGL.

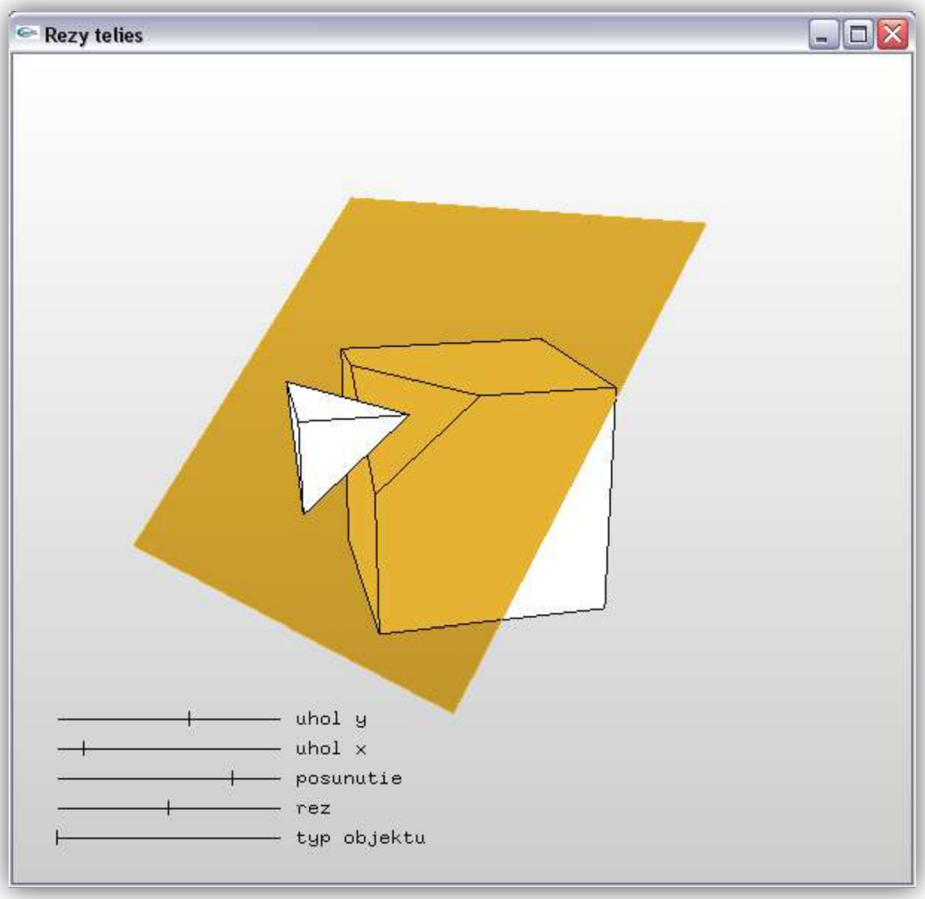
Ovládanie dema je náramne podobné predošlému demu. Pozostáva z posuvníkov, z ktorých 2 plnia rovnakú funkciu ako u dema objemy a povrchy telies, tj. voľba typu objektu a priebeh rezu. Ďalšie posuvníky nám poskytujú možnosť nastavovania uhlov a polohy roviny vzhľadom k jednotlivým osiam. Screenshot je možné vidieť na obrázku 5.6.



Obr. 5.4: Screenshot dema povrchy a objemy telies



Obr. 5.5: Screenshot dema povrchy a objemy telies



Obr. 5.6: Screenshot dema rezy telies

6 Záver

Bola vytvorená kostra aplikácie pre demonštráciu stredoškolského učiva matematiky prostredníctvom zásuvných modulov. Bolo definované ľahko rozšíriteľné rozhranie pre zásuvné moduly, spolu s jednoduchou možnosťou vytvárania ich prekladov, prípadne upravovania ich popisov.

Taktiež boli vytvorené tri ukážkové zásuvné moduly, demonštrujúce odvodenie vzorcov a názorné dopočítanie hodnôt pre objemy a povrchy telies a modul pre prezentáciu rezov telesami.

Aplikácia je navrhnutá tak, aby bola ľahko rozšíriteľná, ako aj šíriteľná. Na úkor tohto sa však nepodarilo tematicky pokryť celú stredoškolskú matematiku. V rámci ďalšieho pokračovania rozvoja tohto výukového programu, či už prostredníctvom diplomovej práce alebo svojvoľne by sa mohli implementovať nasledovné rozšírenia:

- voliteľné vizualizácie štýlov (farba, textúry objektov)
- anaglyph mód zobrazovania
- jednoduchý editor xml tagov
- použitie pokročilých grafických funkcií (shadery), ktoré z dôvodu jednoduchosti a kompatibility neboli použité
- prehrávanie obrazového a zvukového doprovodu k jednotlivým demonštráciám
- v neposlednej rade by bolo účelné vytvoriť, už spomínané , väčšie množstvo zásuvných modulov pre pokrytie širšieho pásma látky.

Literatúra

[1] Kršek, P.: Základy počítačové grafiky, Brno, 2006, Vysoké učení technické v Brně, Fakulta informačních technologií, [online], [cit. 02.05.2009]

[2] Peringer, P.: Modelování a simulace, Brno, 2008, Vysoké učení technické v Brně, Fakulta informačních technologií, [online], [cit. 02.05.2009]

[3] Ivan, J.: Matematika 1, Alfa, Bratislava, 1986

[4] Gatial, J. – Hejný, M.: Od pravouhlých súradníc k vektorom, 3. Doplnené vydanie, SPN, Bratislava, 1980

[5] Skriptá matematiky, STU, Bratislava, 2000

[6] Barstch, H.J.: Matematické vzorce, Academia, Praha, 2006

[7] Segal, M. – Akeley, K.: The OpenGL Graphics System: A Specification [online], version 2.1, 1.12.2006, [cit. 20.04.2009],
URL: http://www.opengl.org/registry/doc/glspec21.20061201.pdf

[8] Nikl, J.: Didaktické aspekty technických výukových prostředků, TU v Liberci, 2002

[9] Rambousek, V. a kol.: Technické výukové prostředky. Praha, SPN, 1989

[10] Dostál, J.: Výukový software a didaktické hry - nástroje moderního vzdělávání [online], URL: http://www.jtie.upol.cz/clanky_1_2009/dostal.pdf, [cit. 10.5.2009]

[11] Windows API [online], URL: http://en.wikipedia.org/wiki/Winapi, [cit. 5.5.2009]

[12] OpenGL [online], URL: http://en.wikipedia.org/wiki/Opengl, [cit. 5.5.2009]

[13] C++ [online], URL: http://en.wikipedia.org/wiki/C++, [cit.5.5.2009]

[14] Polygon Plane Partition [online],
URL: http://www.cgafaq.info/wiki/Polygon_plane_partition, [cit. 12.5.2009]

Zoznam príloh

Príloha 1. Manuál, priložený na CD
Príloha 2. CD, obsahujúce zdrojové kódy a binárky aplikácie