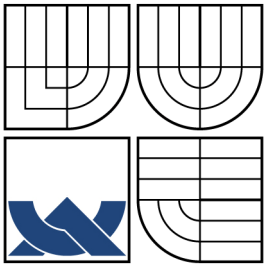
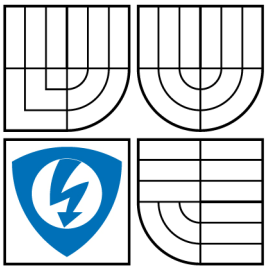


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV VÝKONOVÉ ELEKTROTECHNIKY A ELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF POWER ELECTRICAL AND ELECTRONIC ENGINEERING

NÁVRH ZAPOJOVACÍHO ROZHRANÍ VIRTUÁLNÍCH LABORATOŘÍ

Connection interface design for virtual laboratory

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Petr Kučera
AUTHOR
VEDOUCÍ PRÁCE — doc. Dr. Ing. HANA KUCHYŇKOVÁ
SUPERVISOR

BRNO 2007

Abstrakt:

Tato bakalářská práce je zaměřena na popis programování 3D aplikací v jazyce C++ pomocí OpenGL. Zaměřuje se jednak na obecný popis vlastností OpenGL s přihlédnutím na jeho možnosti při využití tvorby virtuálních laboratoří. Dále je práce zaměřena na popis možností načítání externích souborů pro použití v OpenGL. Cílem projektu je praktická realizace virtuální 3D laboratoře a podrobný popis její tvorby.

Klíčová slova: Virtuální laboratoř; OpenGL; C++

Abstract:

This bachelor project deals with a description of 3D application programming in C++ language using the OpenGL. It is focused on a general description of features of OpenGL taking account of its capabilities for programming the virtual laboratory. Further, the work aims to describe loading external files for use in OpenGL. The goal of the project is a practical realization of the virtual 3D laboratory and a detailed description of its creation.

Keywords: Virtual laboratory; OpenGL; C++

Bibliografická citace mé práce:

KUČERA, P. *Návrh zapojovacího rozhraní virtuálních laboratoří.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 34 s. Vedoucí bakalářské práce doc. Dr. Ing. Hana Kuchyňková.

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma Návrh zapojovacího rozhraní virtuálních laboratoří jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ……. Podpis autora …….

Poděkování:

Děkují své vedoucí diplomové práce paní Doc. Dr. Ing. Haně Kuchyňkové za poskytnutí cenných rad, připomínek a za její trpělivost.

Seznam obrázků

Obsah

1 OpenGL

1.1 Co je OpenGL

OpenGL je průmyslový standard specifikující multiplatformní rozhraní pro tvorbu aplikací využívajících 3D grafiku. Mezi tyto aplikace patří různé CAD programy, vizualizační vědeckotechnické programy, programy pro simulaci virtuální reality a v neposlední řadě také počítačové hry. Protože se jedná o standard, musí být tento standard někde přesně specifikován. O tuto specifikaci a o údržbu se stará konsorcium (tedy sdružení organizací nebo firem) dříve označované jako ARB (Architecture Review Board) a nyní OpenGL Working Group. Členy tohoto sdružení jsou firmy jako například SGI (Silicon Graphics Inc.), Microsoft, nVidia, ATI, a další.

1.2 Historie

Knihovna OpenGL (Open Graphics Library) byla navržena firmou SGI jako aplikační programové rozhraní (Application Programming Interface - API) k akcelerovaným grafickým kartám resp. celým grafickým subsystémům. Předchůdcem této knihovny byla programová knihovna IRIS GL (Silicon Graphics IRIS Graphics Library). OpenGL byla navržena s důrazem na to, aby byla použitelná na různých typech grafických akcelerátorů a aby ji bylo možno použít i v případě, že na určité platformě žádný grafický akcelerátor není nainstalován - v tom případě se použije softwarová simulace. V současné době lze knihovnu OpenGL použít na různých verzích unixových systémů (včetně Linuxu a samozřejmě IRIXu), OS/2 a na platformách Microsoft Windows. V roce 1992 se zformovalo nezávislé konsorcium, které rozhodovalo o budoucnosti OpenGL. Toto konsorcium bylo označené jako ARB (Architecture Review Board). V září roku 2006 ARB přešlo pod konsorcium KHRONOS a je nadále označováno jako OpenGL Working Group. Seskupení KHRONOS bylo založeno v roce 2000 vedoucími firmami v oblastech výpočetní techniky. Z těch nejznámějších jsou to například ATI, Discreet, Intel, nVidia, SGI, Sun Microsystems a další. Seskupení KHRONOS se zabývá otevřenými standardy aplikačních rozhraní (API).

1.3 Implementace

Implementace je praktická realizace určitých algoritmů, metod a postupů k dosažení daného cíle. Implementace OpenGL existují v dnešní době téměř pro všechny počítačové platformy, které jsou schopny používat grafiku. Tato implementace může být (a většinou také je) přímo v hardware grafické karty, ale je možná i softwarová implementace pro použití OpenGL v případech, kdy hardwarová není k dispozici. Většinou je softwarová implementace jen jako doplňkové řešení, protože je podstatně pomalejší než implementace hardwarová. Její hlavní výhodou je možnost doplnění některých funkcí, které nejsou v hardware, a umožní tak dosáhnout požadovaného cíle i když něco hardware nepodporuje.

1.4 Struktura

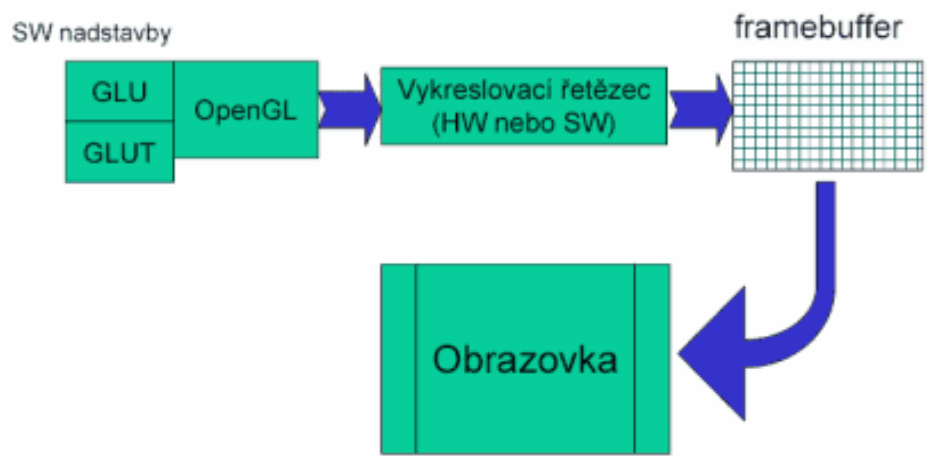
Základní funkcí OpenGL je vykreslování do takzvaného obrazového rámce – framebufferu. Vykreslovány jsou základní grafické objekty jako jsou body, úsečky, trojúhelníky a mnoho-úhelníky. Všechny tyto objekty jsou však vykreslovány podle aktuálních nastavení různými způsoby. Veškerá činnost se v OpenGL řídí vydáváním příkazů pomocí volání jednotlivých funkcí a procedur, kterých je zhruba okolo 250. Každý elementární grafický prvek je definován

především svými okrajovými body. Každý bod má přiřazena určitá data podle toho, v jakém režimu je grafický prvek vykreslován. Rozhraní OpenGL je založeno na architektuře typu Klient-Server.

Je tím myšleno to, že program jakožto klient vydává jednotlivé příkazy, které grafický adaptér jakožto server vykonává. Tato koncepce umožňuje jistou nezávislost, takže program, který vydává příkazy, může fyzicky běžet na jiném počítači, než na jakém jsou příkazy zpracovávány a případně vykreslovány. Ne všechny implementace OpenGL jsou však takto koncipovány.

OpenGL se chová jako stavový automat. Jednotlivé elementární grafické prvky (označované též jako grafická primitiva) jsou vykreslovány podle toho, v jakém stavu se právě OpenGL nachází. Stav se mění pomocí příkazů a zůstává nezměněn, dokud nepřijde další příkaz, který tento stav změnil. Tato koncepce má výhodu v tom, že jeden jediný příkaz může ovlivnit celou vykreslovanou scénu a dále v tom, že jednotlivé příkazy nemají zbytečně mnoho parametrů.

Vykreslování scény probíhá pomyslnou cestou od aplikace až k zobrazovacímu adaptéru (obrazovka, display, případně tiskárna). Začátek cesty tvoří vlastní aplikace, která používá příkazy buď přímo aplikačního rozhraní OpenGL nebo některé příkazy z nadstaveb tohoto API. Z nejznámějších nadstaveb je to například GLU (OpenGL Utilities) a GLUT (The OpenGL Utility Toolkit). Jedná se o rozšiřující knihovny, které dále zvyšují funkcionalitu základního OpenGL. Dále v cestě následuje vlastní implementace OpenGL, která může být jak hardwarová tak softwarová nebo smíšená. Funkce a metody této implementace zapisují data přímo do paměťového bloku takzvaného framebufferu. Odtud už data putují k zobrazovacímu adaptéru. Aby byl celý proces plynulý a uživatel nepostřehnul, jak jsou data postupně vykreslována do framebufferu, jsou tyto bufery nejčastěji dva. Ale může jich být i více, aby se ještě zvýšila plynulost celého procesu. Například termín „triplebuffering“ říká, že jsou použity při vykreslování tři bufery. Výše popsaný proces je dobře patrný z Obrázku 1-1.



Obrázek 1-1 Pomyslná vykreslovací cesta OpenGL

Na začátku cesty vstupují data reprezentující celou scénu v podobě čísel majících různý význam. Jsou to například vrcholy jednotlivých grafických primitiv, barvy vrcholů, texturové souřadnice, normály ploch a další. Všechno jsou to data, která (až na výjimku v podobě textur a různých výškových a deformačních map) netvoří obrazová data. Tedy data, která sama o sobě nereprezentují obraz. Z těchto dat je teprve prostřednictvím implementace OpenGL výsledný obraz vypočítán a vystupuje na konci naší pomyslné cesty v podobě barvy jednotlivých pixelů (obrazových bodů) uložených ve *framebufferu*.

1.5 Proč právě OpenGL

Rozhraní OpenGL není samozřejmě jediné rozhraní umožňující zobrazování 3D scén. Dalším a velice často používaným je DirectX. DirectX je programátorská knihovna obsahující nástroje pro tvorbu multimediálních aplikací a her. DirectX je produktem firmy Microsoft. Na rozdíl od OpenGL, které se zabývá výhradně grafikou, obsahuje DirectX podporu pro více účelů. Obsahuje *DirectImput* (pro podporu vstupních zařízení typu myš, joystick, gamepad a dalších), *DirectPlay* (pro podporu hraní po síti), *DirectSound* a *DirectMusic* (pro podporu zvukových efektů a hudby) a v neposlední řadě *DirectDraw* a *Direct3D* pro podporu grafiky. Právě mezi Direct3D a OpenGL je jistá analogie, protože obě dvě mají za úkol vypočítat obrazovou reprezentaci ze vstupujících dat. Proč tedy používat OpenGL, když podporuje „jenom“ 3D grafiku a nezvolit DirectX který obsahuje podporu spousty dalších potřebných funkcí? Jak bylo výše uvedeno, DirectX je produktem firmy Microsoft a jakožto takový běží pouze na operačních systémech stejné firmy – tedy na operačních systémech Windows. OpenGL je otevřený standard, jenž má implementace téměř pro všechny existující platformy počítačů jako jsou například LINUX, UNIX, MacOS a další. Mezi další výhody patří licenční politika OpenGL. Na rozdíl od DirectX je použití OpenGL bezplatné. Licence OpenGL je definována na stránkách http://www.sgi.com/products/ /software/opengl/license.html

Jiným hlediskem ve výběru grafického rozhraní je také složitost základního programového kódu, který je nutný napsat, aby se dalo rozhraní použít a zobrazil se požadovaný objekt. OpenGL je pro začátečníka mnohem jednodušší na pochopení základního konceptu a pokud se programují pouze jednoduché scény je programový kód OpenGL kratší a přehlednější. Při programování složitějších aplikací se tento rozdíl vytratí a alespoň podle mého názoru jsou OpenGL i DirectX stejně složité a je už jen na zvyku programátora, v čem se mu pracuje lépe.

2 Datové formáty pro uložení reprezentace 3D tělesa

2.1 Co se rozumí pod pojmem datový formát

Pod pojmem datový formát se rozumí jistá určitá organizace dat uložených v souboru. Data jsou v podstatě informace nějakého druhu. Může to být prostý text, obrázek, zvuk, video nebo cokoliv jiného. Ale aby měla data nějaký smysl, musí být definováno, co ta data vlastně znamenají a jak jsou organizována. Každá data je vhodné (a většinou i nutné) organizovat nějakým způsobem. Pokud jsou data organizována přesně daným způsobem, má to obrovskou výhodu v tom, že každý, kdo s nimi pracuje, ví, jak jsou organizována a může s nimi efektivně pracovat a většinou je znalost jejich organizace nezbytně nutnou podmínkou, toho aby s daty mohl vůbec nějak nakládat a aby jim rozuměl. Na první pohled by bylo rozumné stanovit, že data budou organizována přesně jedním jediným způsobem. Bylo by tím zajištěno, že jim bude každý rozumět a mohl by s nimi nakládat jak potřebuje. Ale věci většinou bývají složitější, než se zdá na první pohled. Je potřeba zajistit, aby byla data uložena efektivně a nezabírala více místa než je skutečně potřeba a aby se s nimi dalo efektivně manipulovat. Každá data však musí být uložena jinak, aby byla uložena efektivně. Například takový obrázek je vhodné uložit jiným způsobem než třeba prostý text a jiným způsobem než zvuk. Ale naopak dva obrázky je vhodné uložit stejným způsobem, aby se zjednodušila manipulace s nimi. Pak totiž se znalostí toho, jak jsou data uložena v prvním obrázku, lze zacházet i s druhým obrázkem, aniž by se muselo zjišťovat, jak jsou v něm data uložena. A tato myšlenka je analogicky přenesena i na další datové typy (text, zvuk, video, cokoliv).

Datový formát tedy jednoznačně definuje, jakým způsobem jsou data uložena nebo jak je ukládat, aby je bylo možno efektivně použít. Operační systémy firmy Microsoft používají pro určení toho, o jaký datový formát se jedná, příponu souboru, ve kterém jsou data uložena. Například přípony „jpg“, „png“, „bmp“ znamenají, že data uložená v souboru reprezentují obrázek. Jiné operační systémy (například Linux) mohou používat přímo data uložená v souboru k identifikaci toho, o jaký datový formát se jedná.

2.2 Formáty používané pro 3D tělesa

Formáty používané pro uložení reprezentace 3D tělesa obsahují především různá data potřebná pro zobrazení tělesa. Některé složitější formáty obsahují informace nejen o tělese jako takovém, ale mohou obsahovat data reprezentující celou scénu. Pod pojmem scéna se v tomto případě rozumí rozmístění grafických objektů v 3D prostoru, jejich osvětlení, jejich vlastnosti, pozice kamery a úhel jejího záběru a další. Obsah formátu se také liší podle specifického zaměření, pro který je formát určen. Jiné požadavky jsou kladeny na čistě vizualizační programy a jiné zase na konstruktérské. Obě oblasti se ovšem prolínají.

Z nejznámějších vizualizačních grafických programů lze uvést alespoň některé. Každý z nich má svůj vlastní datový formát. Jsou to „3D Studio MAX“, „Maya“, „Cinema 4D“, „Rhino“

Další programy jsou z oblasti konstruktérské. Také tyto programy mají každý svůj formát uložení dat. Jsou to „Inventor“, „SolidEdge“, „SolidWorks“, „Unigraphics“

2.3 Základní data pro reprezentaci 3D tělesa

3D těleso je v oblasti výpočetní techniky definováno nikoliv svým objemem, jak je tomu v reálném prostředí ve kterém žijeme, ale svým povrchem. Přestože jsou v programech objekty rozděleny na plošné a na objemové, jsou i ty objemové objekty tvořeny plochou. Jediný rozdíl mezi nimi je ten, že objemové objekty mají plochu „uzavřenou“. To znamená, že z vnitřku objemového tělesa se nedostaneme ven, aniž bychom prošli plochou, která jej obklopuje. Pro vizualizační účely je tedy možné zobrazením určité plochy zobrazit objemové těleso. V reálném světě také vidíme pouze povrch tělesa a nikoli jeho obsah (když neberu v úvahu průhledné objekty).

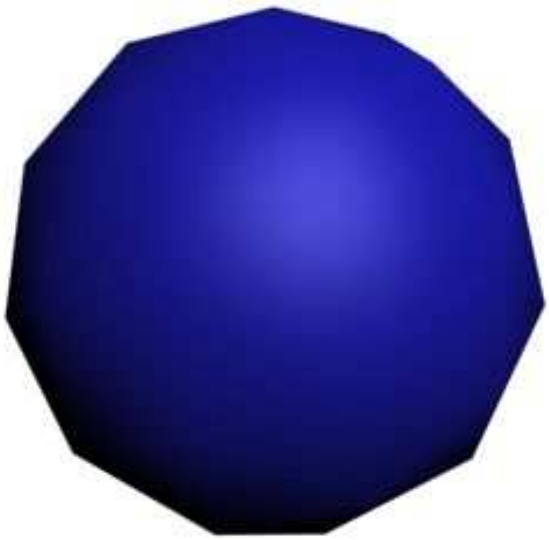
Dostali jsme se tedy k ploše. Přesněji řečeno jsme se dostali k všeobecné ploše, která se dá s určitou ztrátou přesnosti převést na určité množství rovinných ploch (viz třeba následující obrázek). Rovinnou plochou se obvykle myslí ohraničená plocha ležící v určité rovině. Rovina je jednoznačně definována třemi body. Tyto tři body také mohou sloužit k ohraničení plochy ležící na rovině, kterou body definují. Tím vznikne jakýsi trojúhelník ve 3D prostoru. A jsme tam, kde jsme chtěli být, protože právě trojúhelník je nejzákladnější grafický objekt, který je zpracováván grafickými rozhraními.

Základními daty reprezentujícími 3D těleso jsou tedy body, které tvoří vrcholy trojúhelníku. Z trojúhelníků jsou pak vytvořeny plochy reprezentující 3D těleso.

Dalšími daty, která blíže specifikují 3D objekt, může být barva povrchu objektu, nebo dokonce barevná textura. Zaznamenávají se také vektory (normály) kolmé k ploše objektu v jednotlivých bodech tvořících vrcholy trojúhelníku. Normálu k ploše každého trojúhelníku tvořícího povrch tělesa sice můžeme lehce spočítat, ale většinou se tyto normály liší od normál v jednotlivých bodech. Jako ukázka jsou uvedeny dva obrázky. Na obrázku 2-1 jsou normály v jednotlivých bodech tvořících trojúhelník shodné s normálou trojúhelníku, který tvoří.

Obrázek 2-1 Normály shodné s normálami trojúhelníků

Na obrázku 2-2 jsou normály bodů, tvořících jeden trojúhelník, různé od normály trojúhelníku, který tvoří. Rozdíl je patrný na první pohled.



Obrázek 2-2 Normály různé od normál trojúhelníků

Normály každého bodu na Obrázku 2-2 jsou kolmé k povrchu koule, kterou se pomocí trojúhelníků snažíme vytvořit a tím, že se zpřesnily alespoň tyto normály, jsme se po vizualizační stránce více přiblížili tomu, co se má vizualizovat. V tomto případě to byla koule. Za použití stejného počtu trojúhelníků byla s různým stupněm úspěšnosti zobrazena koule.

Pro shrnutí - obrazová data Obrázku 2-1 byla tvořena pouze body, které tvořily vrcholy jednotlivých trojúhelníku, z nichž se těleso skládá. Ostatní informace (včetně normál) byly dopočítány pro zobrazení v 3D rozhraní. Na Obrázku 2-2 přibyla data normál, která napomohla tomu, že se výsledný obrázek více přibližuje požadovanému vzhledu.

Přidáváním dalších dat jakou jsou textury (obrázky mapované na povrch tělesa) a texturové koordináty by se výsledný dojem opět zlepšil.

Cílem této kapitoly však nebyl vyčerpávající popis všech dat, která se mohou použít pro popis 3D objektu, ale pouze ukázat ty nejzákladnější.

3 Datový formát 3DS

3.1 Stručný úvod

Datový formát „3DS“ byl navržen firmou AutoDesk pro její produkt 3D Studio Max. Byl navržen ještě v dobách DOSu a v dnešní době se stále používá, protože jej může načíst velká část grafických programů. Přestože přesný popis tohoto formátu jako standardu nebyl firmou Autodesk veřejně uvolněn, popis jeho základních části je možné na internetu nalézt v různých dokumentech. Přestože dnešní verze 3D Studia Max datový formát 3DS nativně nepoužívá, je možný export do tohoto formátu. Soubor obsahuje informace sloužící k sestavení celé 3D scény. Těmito informacemi jsou například pozice jednotlivých kamer, klíčové snímky animací, osvětlení a další. Pro projekt virtuálních laboratoří jsou však nejdůležitější data týkající se jednotlivých 3D objektů umístěných ve scéně. Datový formát 3DS jsem zvolil pro použití v aplikaci virtuálních laboratoří pro jeho značnou rozšířenost. Dále proto, že obsahuje i informace o materiálu povrchu 3D objektu. Z následujícího popisu bude zřejmé, že materiály jsou poměrně komplexní, což je nutné pro věrnou podobu vizualizace 3D objektů, na které jsme zvyklí ze svého okolí. Použití správného materiálu v kombinaci se světelným modelem má značný vliv na reálnost vizualizace.

3.2 Vnitřní struktura souboru

Data jsou v souboru uložena binárně, a jakožto taková jsou pro člověka nečitelná. Jednotlivé informace jsou v souboru uloženy do takzvaných CHUNKů. CHUNK je blok dat, která mají nějakou souvislost. Podstatné je že tyto bloky mohou a také většinou jsou vnořeny. Znamená to tedy, že jeden blok může obsahovat jeden nebo více dalších bloků. Tyto mohou zase obsahovat další. Tato informace je podstatná pro způsob čtení tohoto formátu, který je rekurzivní. Každý CHUNK, tedy blok může obsahovat (v závislosti na jeho typu) nějaká data, nebo slouží pouze jako jednotící prvek pro další bloky v něm obsažené. Každý blok musí obsahovat *identifikátor* o velikosti dvou bytů. Dále musí obsahovat svou *délku* o velikosti 4 byty. Tato délka udává velikost vlastních dat (včetně dvou bytů pro identifikátor a čtyř pro délku) a velikost dalších bloků, které obsahuje. Lepší představu získáme z následující Tabulky 3-1.

Tabulka 3-1: Uspořádání dat v bloku

Offset	Délka	Položka
0	2	Identifikátor
2	4	Délka (6 + n + m)
6	n	Vlastní data
6+n	m	Vnořené bloky

Identifikátor bloku určuje jaká data a jaké vnořené bloky můžeme v daném bloku nalézt. Podle identifikátoru se rozhoduje jakým způsobem dále postupovat ve čtení bloku, jaká vlastní data (pokud nějaká jsou) blok obsahuje a jaké další vnořené bloky můžeme očekávat. Je to dáno hierarchií jednotlivých bloků. Tím, že čtení souboru probíhá rekurzívně, prochází se jednotlivé bloky a jejich vnořené bloky a hierarchii není téměř potřeba znát. Rozhodující je pouze znalost vlastních dat, která blok obsahuje na základě typu bloku. Vlastní data bloku se načtou podle jejich typu a velikosti a následuje načítání případných dalších vnořených bloků, které se načítají rekurzivně.

Následující seznam ukazuje jména identifikátorů bloků a jejich dvoubytový kód. Uvedený seznam neobsahuje všechny identifikátory, ale pouze určitou skupinu, která je užitečná pro použití v projektu. Každá položka seznamu obsahuje anglický název bloku a jeho dvoubytový kód. Začátek kódu „0x“ značí v jazyce C šestnáctkovou číselnou soustavu.

- **MAIN CHUNK 0x4D4D**
  - **VERSION 0x0002**
  - **3D EDITOR CHUNK 0x3D3D**
    - **OBJECT BLOCK 0x4000**
      - **TRIANGULAR MESH 0x4100**
        - **VERTICES LIST 0x4110**
        - **FACES DESCRIPTION 0x4120**
          - **FACES MATERIAL 0x4130**
        - **MAPPING COORDINATE LIST 0x4140**
    - **MATERIAL BLOCK 0xAFFF**
      - **MATERIAL NAME 0xA000**
      - **AMBIENT COLOR 0xA010**
      - **DIFUSE COLOR 0xA020**
      - **SPECULAR COLOR 0xA030**
      - **SHININESS 0xA040**
      - **TRANSPARENCY 0xA050**
      - **TEXTURE MAP 0xA200**
        - **MAP FILE 0xA300**
        - **MAP U SCALE 0xA354**
        - **MAP V SCALE 0xA356**
        - **MAP U OFFSET 0xA358**
        - **MAP V OFFSET 0xA35A**
        - **MAP ANGLE 0xA35C**

Většina z výše uvedených bloků obsahuje vlastní data, která jsou svým typem specifická pro daný blok. Následující popis jednotlivých bloků upřesňuje, jaký typ dat obsahují a jejich velikost.

MAIN CHUNK – Hlavní blok dat. Obsahuje všechny ostatní bloky. Tento blok v podstatě tvoří obsah celého souboru.

VERSION – Blok dat obsahující pouze verzi souboru. Data jsou formátu **dword** – tedy velikosti 4 byty. Program virtuálních laboratoří kontroluje tuto hodnotu, aby byla menší než 3, aby měl jistotu, že je soubor kompatibilní a bude jej schopen správně zpracovat.

3D EDITOR CHUNK – Obsahuje informace týkající se 3D objektů obsažených v souboru. Nemá žádná vlastní data a slouží pouze jako nositel vnořených bloků dat.

OBJECT BLOCK – Slouží jako blok dat sdružující informace jednoho 3D objektu. Těchto bloků může být v nadřazeném bloku několik. Jeho vlastní data tvoří řetězec znaků sloužící jako název 3D objektu. Dále obsahuje bloky tvořící vlastní 3D objekt.

TRIANGULAR MESH – Je blok dat který sdružuje informace o „drátovém“ modelu. Sám o sobě neobsahuje žádná data, ale obsahuje další vnořené bloky.

VERTICES LIST – Obsahuje informace o vrcholových souřadnicích 3D objektu. Jeho vlastními daty jsou počet vrcholů, které obsahuje (velikost **word** = 2 byty) a následují jednotlivé vrcholy o velikosti 3 * **float** (tedy 3 * 4 byty).

FACES DESCRIPTION – Obsahuje indexy do pole vrcholů. Odkazuje se tedy na data obsažená v bloku *VERTICES LIST*. Vždy tři indexy tvoří jednu plošku. Tyto plošky pak tvoří povrch 3D tělesa. Jeho vlastní data začínají dvěma byty (tedy **word**) nesoucí informaci o počtu plošek v bloku. Následuje 4 * **word** (tedy 8 bytů) pro každou plošku. První tři obsahuji indexy tří vrcholů. Čtvrtý **word** obsahuje příznaky viditelnosti jednotlivých hran plošky. Tato informace však není v programu virtuálních laboratoří využita.

FACES MATERIAL – Je vnořený ve *FACES DESCRIPTION* a jeho vlastní data obsahují seznam ploch majících stejný materiál. Odkazuje se tedy na data z *FACES DESCRIPTION*. Jeho vlastní data začínají nulou zakončeným řetězcem (jak je v jazyce C běžné) obsahujícím název materiálu pro tuto řekněme „materiálovou skupinu“ plošek. Následuje údaj o počtu plošek (velikost **word**) a pak příslušný počet indexů (opět velikosti **word**) odkazujících se na jednotlivé plošky.

MAPPING COORDINATE LIST – Obsahuje informace o texturových koordinátách. Tyto koordináty náleží každému vrcholu objektu a nesou informace nutné pro správné namapování textury (tedy obrazového povrchu) na 3D objekt. Vlastními daty jsou počet těchto koordinátů (velikosti **word**), který se musí shodovat s počtem vrcholových souřadnic. Následují souřadnice. Každá souřadnice má dvě položky U a V velikosti **word**. Tyto souřadnice přiřazují každému vrcholu 3D objektu souřadnici v obrázku textury.

MATERIAL BLOCK – Sdružuje informace o materiálu použitém pro část 3D objekt (případně pro celý objekt). Každému jednotlivému materiálu náleží jeden tento blok. Neobsahuje žádná vlastní data, pouze další vnořené bloky.

MATERIAL NAME – Obsahuje ve vlastních datech jméno materiálu. Jde o nulou ukončený řetězec. Podle tohoto jména jsou potom materiály používány pro jednotlivé „materiálové skupiny“ jak je popsáno v *FACES MATERIAL*.

AMBIENT COLOR – Neobsahuje žádná vlastní data ale pouze vnořené bloky, z nichž jedním je *RGB CHUNK* nesoucí informaci o třech barevných složkách. Stejné uspořádání mají i **DIFUSE COLOR** a **SPECULAR COLOR**.

RGB CHUNK – Obsahuje ve vlastních datech tři položky (R, G a B), každá velikosti jednoho bytu.

SHININES – Obsahuje pouze vnořený blok obsahující procentuální velikost odleskové složky materiálu. Jedná se o číslo od nuly do sta. Stejné složení má i blok **TRANSPARENCY** určující stupeň průhlednosti materiálu. Na základě průhlednosti materiálu je volen i způsob vykreslování pomocí OpenGL.

TEXTURE MAP – Sdružuje informace o primární textuře použité pro materiál. Texturových bloků (s různými identifikátory) je více pro složitější materiály. V tomto programu je však použita jediná primární textura pro materiál. Tento blok neobsahuje žádná vlastní data, pouze vnořené bloky.

MAP FILE – Obsahuje ve vlastních datech nulou zakončený řetězec se jménem souboru, ze kterého se má načíst obrázek textury.

MAP U SCALE – Je blok obsahující ve vlastních datech jedinou položku typu **float** určující měřítko použité textury. Stejného typu je blok **MAP V SCALE**.

MAP U OFFSET – Obsahuje podobně jako předcházející blok položku typu **float**, která určuje posunutí textury. Obdobně je tomu u bloku **MAP V OFFSET**.

MAP ANGLE – Má ve vlastních datech také jedinou položku **float**, která určuje úhel natočení textury kolem osy kolmé na osy U a V.

Přestože soubor formátu 3DS obsahuje ještě další data, není jejich znalost nezbytně nutná ke správnému přečtení souboru. Je to dáno právě jeho blokovou strukturou, která umožňuje neznámé bloky, na základě znalosti jejich velikosti, jednoduše přeskočit a zpracovat pouze bloky známého typu. Konkrétní implementaci čtení souboru si zájemce může prohlédnout v příloze ve zdrojových souborech. Pro načítání souborů je vytvořena třída (viz objektově orientované programování jazyka C++), jejíž metody soubor zpracují a uloží do podoby vhodné pro použití v prostředí OpenGL. Podrobnější popis této třídy a jejich metod je uveden v následující kapitole, která se zabývá konkrétní realizací.

4 Praktická realizace virtuální laboratoře

4.1 Použití OpenGL v jazyce C++

Aby bylo možné použít implementaci OpenGL rozhraní, je zapotřebí mít způsob, jak s ní komunikovat. Tímto způsobem jsou hlavičkové soubory a příslušné knihovny obsažené ve vývojovém prostředí, které pro vývoj aplikace používáme. V rámci této práce je použito vývojové prostředí firmy Microsoft. Je jím Visual C++ verze 6.0. Důvodem výběru právě tohoto prostředí je jednak to, že jej škola vlastní (toto vývojové prostředí je placený produkt), a druhým aspektem je to, že toto vývojové prostředí obsahuje všechny potřebné hlavičkové soubory a knihovny pro práci s OpenGL. Toto prostředí také obsahuje rozsáhlou databázi nápovědných textů, ukázek a doporučení, jak funkce OpenGL efektivně používat.

OpenGL je psáno v holém jazyce C bez použití novějších možností jazyka C++, jimiž jsou třeba podpora objektově orientovaného programování. Je to především kvůli efektivnosti a přenositelnosti zdrojového kódu. Také se tím OpenGL stává snáze naučitelné pro začátečníky, kteří se teprve seznamují s jazykem C jako takovým. Není však problém si funkce OpenGL zapouzdřit do objektů v případě, že chce někdo využít výhod objektově orientovaného programování. Tohle je zvoleno i v této práci, protože v tomto projektu nejde o maximální možnou efektivitu výsledného kódu, ale spíše o přehlednost a snadnou rozšiřitelnost o další funkce. Dále se předpokládá spolupráce více autorů při projektu virtuálních laboratoří, kde srozumitelnost a přehlednost zdrojového kódu bude hrát velkou roli.

4.2 Všeobecný úvod týkající se realizace programu

Objektově orientované programování jazyka C++ umožňuje logicky a funkčně oddělit jednotlivé části programu. Prostřednictvím metod jednotlivých tříd lze definovat komunikační rozhraní, pomocí něhož jsou jednotlivé třídy propojeny a vzájemně spolupracují. Zapouzdřením funkčnosti do jednotlivých tříd lze docílit jejich použitelnosti pouze na základě znalosti jejich komunikačních metod bez potřeby znalosti přesné a konkrétní implementace té které třídy. Toto je výhodné zejména při spolupráci více lidí na jednom projektu. Dále se tímto přístupem s pomocí virtuálních metod a dědičnosti tříd dosáhne relativně jednoduché rozšiřitelnosti funkčnosti jednotlivých tříd a to opět bez nutné znalosti její konkrétní vnitřní implementace.

V následujícím textu budou postupně popsány jednotlivé třídy z hlediska jejich funkčnosti a také jejich vzájemné propojení v jeden celek. Jednotlivé třídy jsou uloženy v samostatných definičních souborech a pro jejich spojení jsou použity jejich takzvané hlavičkové soubory obsahující jejich deklarace.

Pro programování aplikace pro operační systém Windows je využito MFC (Microsoft Foundation Class) což je ve své podstatě objektové zapouzdření Windows API (Application Programming Interface). MFC je obsaženo přímo ve vývojovém prostředí Visual C++ 6.0, které proto nabízí rozsáhlou nápovědu a s použitím MFC tedy není žádný problém.

Celý projekt je založen na dialogovém okně, v jehož pracovním prostoru je vyčleněna určitá plocha pro vykreslování pomocí OpenGL. Dialogové okno bylo vybráno pro možnost jeho snadného začlenění do jiného programu. Veškerá funkčnost programu je směřována do této plochy a minimálně využívá samotné grafické rozhraní systému Windows. Je to hlavně z důvodu případné přenositelnosti do jiného operačního systému.

4.3 Popis jednotlivých tříd a jejich začlenění v programu

4.3.1 Třída COGL_Base

Deklarace této třídy je umístěna v hlavičkovém souboru „OGL_Base.h“. Tato třída je jedna ze základních tříd, od nichž jsou odvozovány třídy další. Tato třída je čistě virtuální, což znamená, že není možné vytvořit její instanci, ale je určena čistě k odvození dceřiné třídy. Tedy třídy, která je od této odvozena a má ji jako rodičovskou. Tyto vztahy hrají v objektově orientovaném programování důležitou roli. Tato třída zprostředkovává styk s operačním systémem a je proto odvozena od třídy CWnd, která je základní třídou pro „okno“ v rámci MFC. Úlohou této třídy je především získání takzvaného „rendrovacího kontextu“ pro jeho použití v OpenGL. Zjednodušeně řečeno je to získání určité plochy v rámci rodičovského dialogového okna, ve kterém celá aplikace běží. Vykreslování do tohoto okna není v režii operačního systému Windows, ale stará se o něj právě OpenGL. Tato třída je pojata koncepčně jako knihovní prvek, tedy s cílem jejího co nejsnazšího rozšíření a použití. Tato třída tedy implementuje pouze základní metody, jejichž konkrétní funkčnost se většinou pro různé aplikace příliš neliší. Naopak metody, jejichž funkčnost závisí na konkrétní aplikaci, jsou deklarovány jako čistě virtuální a dceřiná třída je musí implementovat.

Další funkcí této třídy je zprostředkování interakce s uživatelem zachycením určitých událostí v systému Windows. Jedná se především o zachycení událostí způsobených pohybem myši a stisknutí tlačítek myši. Tyto události jsou touto třídou zachyceny, ale o konkrétní zpracování těchto zpráv se musí postarat třída od této třídy odvozená, tedy dceřiná. Předání těchto událostí zděděné třídě umožňuje takzvaný polymorfismus objektově orientovaného programování.

Třída sama o sobě nevykresluje do plochy, kterou pro OpenGL získala, ale zajišťuje mechanismus vykreslování. Jednak reaguje na požadavek na překreslení od systému Windows a také sama spustí časovač, který zhruba každých 30 milisekund požádá o překreslení plochy. Při každém požadavku o překreslení je volána virtuální metoda dceřiné třídy ***OGL_Paint***.

Tento mechanismus je na první pohled docela krkolomný, ale jedná se o jakési logické oddělení a rozhraní mezi operačním systémem a vlastním programem. Upravení této třídy umožňuje bez dalších zásahů a bez detailní znalosti ostatních tříd použití pod jiným operačním systémem, který má implementaci OpenGL.

Třída ***COGL_Base*** má také implementován mechanismus výběru, takzvaný ***picking***. Jedná se o mechanismy založené na možnostech OpenGL. ***Picking*** – tedy výběr umožňuje uživateli interakci s 3D objekty vykreslovanými ve 3D scéně. Mechanismus výběru je kvůli rychlostní optimalizaci trochu složitější a bude mu věnována samostatná kapitola.

V hlavičkovém souboru je deklarována ještě jedna čistě virtuální třída se jménem ***CBaseObject***. Tato třída slouží jako společný předek všem dalším objektům určeným k vykreslování prostřednictvím mechanismu vykreslování tak jak je implementován v ***COGL_Base***. Tato třída obsahuje virtuální metody pro vykreslení 3D objektu. Jsou to metody ***Draw*** a ***DrawObject***. Dále obsahuje virtuální metody pro mechanismus výběru ***DrawPrePicking*** a ***DrawPicking*** a pro interakci metodu ***Action***. Konkrétní funkce těchto metod záleží na jejich implementaci v dceřiné třídě. Tento mechanismus ale umožňuje volat tyto metody, aniž by bylo v době sestavování a překladu programu známo o jaké konkrétní metody se jedná. Teprve až za běhu programu je na tyto metody odkazováno skrze *tabulku virtuálních metod*.

4.3.2 Třída COGL_Wnd

Tato třída je odvozena od třídy ***COGL_Base***. Její metody jsou psány konkrétně pro aplikaci virtuální laboratoře, a to měření akumulátoru automobilu. Tato třída tvoří jakýsi hlavní prvek aplikace. Stará se o vykreslování všech objektů, které prostřednictvím třídy ***CLoad3DS*** načetla. Samotná třída ***CLoad3DS*** bude popsána dále. Zajišťuje interakci s uživatelem tím, že vhodně reaguje na veškeré události, jako je pohyb myši, stlačení tlačítka a další. Tato třída používá ke své funkčnosti další třídy a jejich schopnosti. Je tedy jakýmsi centrem veškerého dění a pojí vše v jeden celek.

Metoda ***OGL_Init*** se stará o veškeré nastavení vykreslovacího režimu OpenGL. Nastavuje osvětlení, načítá 3D objekty ze souborů a přiřazuje počáteční hodnoty vnitřním proměnným, (proměnnou je v tomto kontextu myšlena určitá paměťová oblast určená pro uchovávání různých hodnot, ať už číselných, znakových nebo jiných).

Metoda ***OGL_Paint*** pravidelně (zhruba každých 30 milisekund) vykresluje celou scénu a zjišťuje prostřednictvím mechanismu výběru interakci s kurzorem myši.

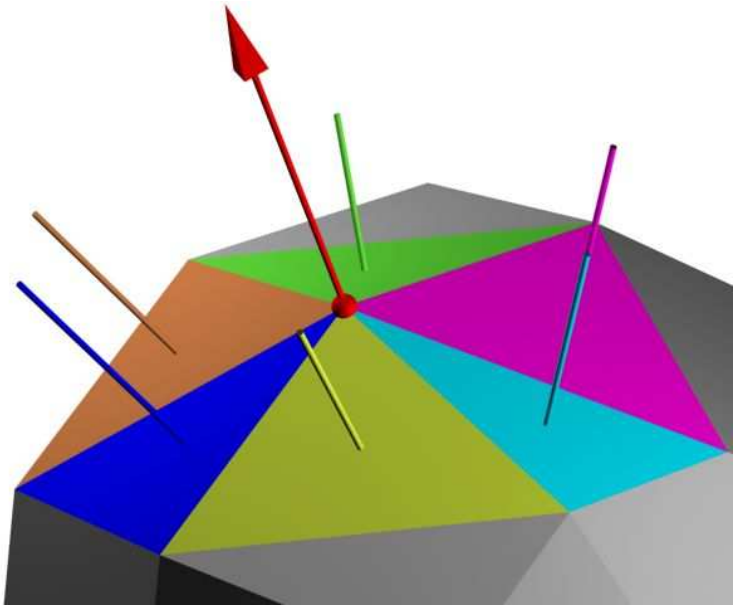
Třída ***COLG_Wnd*** se chová jako stavový automat a na základě vnitřních proměnných je určeno jakým způsobem se chová. Jedním takovým stavem je například tvorba drátového propojení v rámci zařízení. Dalším stavem je například práce v menu nějakého přístroje a podobně.

4.3.3 Třída CLoad3DS

Tato třída slouží k načítání dat ze souborů ve formátu 3DS do vnitřních proměnných třídy ***C3DModel*** (bude popsána dále). O vlastní načtení se stará metoda ***Import3DS*** jejímiž vstupními parametry jsou ukazatel na třídu ***C3DModel*** a jméno souboru který chceme načíst. Jméno souboru je udáváno bez plné přístupové cesty. Jako výchozí přístupový bod je brán adresář, ze kterého je spouštěn program. Přístup do podadresářů tohoto výchozího je možný zadáním jména tohoto podadresáře do jména souboru (například „Objekty\Objekt1.3DS“).

Jak bylo napsáno výše při popisu vnitřní struktury datového formátu 3DS, probíhá načítání souboru rekurzivně. Metoda ***Import3DS*** na začátku složí celou přístupovou cestu i se jménem souboru a otevře soubor v režimu binárního čtení. Od této cesty odstraní jméno souboru a uloží si cestu pro pozdější načítání případných textur. Načte se hlavička svazku (*CHUNKu*) a určí se, o jaký se jedná typ. Pokud to není primární svazek, tedy s hodnotou identifikátoru 0x4D4D, tak se nejedná o platný obsah 3DS souboru a v jeho čtení se nepokračuje. Pokud je vše v pořádku, tak se zavolá vnitřní metoda ***ProcessNextChunk***. Tato metoda přečte vlastní data a z rozdílu velikosti vlastních dat, který je dán typem svazku, a velikosti udané v hlavičce svazku určí, zdali svazek neobsahuje další vnořený svazek, případně více svazků. Pokud ano, tak metoda zavolá sebe sama, zde je ona rekurzivita zpracování souboru, a pokračuje ve zpracování vnořeného svazku. Ten může a nemusí obsahovat další vnořené svazky. U svazků, u kterých není znám jejich typ, se použije pouze jejich velikost a svazek se přeskočí. Tímto způsobem se přečte celý soubor a příslušná data, o která máme zájem, se uloží do vnitřní struktury třídy C3DModel. Jedná se o vrcholové souřadnice 3D objektu, popis jednotlivých povrchových plošek a informací o materiálu použitém na jednotlivé plošky. Jsou také načteny texturové koordináty. O načtení textur pro jednotlivé materiály se musíme postarat samostatně. Je to z toho důvodu, aby chybějící soubor jediné textury neměl za následek nenačtení celého 3D modelu. Model s chybějícím souborem textury se tedy načte v pořádku, ale jeho povrch, nebo část jeho povrchu nebude vypadat tak jak by měla.

Pro správné osvětlení modelu je potřeba, aby model obsahoval také normálové vektory pro každý vrcholový bod. V kapitole „základní data pro reprezentaci 3D tělesa“ byl vysvětlen rozdíl mezi použitím normál pro jednotlivé plošky a normál pro jednotlivé vrcholy. Pro hladší povrch tělesa je potřeba vypočítat normálové vektory pro každý vrchol. Jedna ploška je definována třemi vrcholovými souřadnicemi a přísluší jí tedy i tři normálové vektory. Soubor formátu 3DS informace o normálových vektorech neobsahuje a je tedy nutné si je dopočítat až po načtení souboru. O tento výpočet se stará vnitřní metoda ***ComputeNormals***. Princip výpočtu je následující: Metoda prochází postupně všechny vrcholové souřadnice a k nim dopočítává normálové vektory. Ke každé vrcholové souřadnici si projde všechny plochy a uloží si ty plochy, ve kterých se nachází vrcholová souřadnice, ke které se počítá normála. Pomocí vektorového součinu dvou vektorů tvořících hrany plošky se spočítá normála každé plošky, která k danému vrcholu náleží. Veškeré normály jsou normalizovány (jsou tedy jednotkové). Poté se všechny normály sečtou a výsledek se opět normalizuje (na jednotkový vektor). Tato poslední operace se dá chápat jako jakési zprůměrování všech vektorů. Na výpočet jediného normálového vektoru se prochází celá datová struktura ploch a poté se počítá minimálně se třemi plochami jejich normály a jejich průměrem. Toto se opakuje pro každý vrchol objektu. Výpočet je proto poměrně časově náročný a při spouštění programu chvilku trvá, než se všechny objekty načtou a dopočítají se normálové vektory. Metoda ***ComputeNormals*** je volána automaticky při načítání každého souboru a není proto nutné (na rozdíl od načítání textur) ji volat zvlášť.



Obrázek 4-1 Výpočet normály vrcholového bodu

Na obrázku je vidět názorně, jakým způsobem je získána normála vrcholového bodu objektu. Z každé plochy vychází její normála. Z bodu, který je společný všem těmto plochám, vychází normála. Tato normála by byla shodná se všemi normálami pouze tehdy, pokud by ležely všechny plochy v jedné rovině a byly by shodné tedy i jejich normálové vektory.

4.3.4 Třída C3DModel

Tato třída slouží pro uložení všech informací načtených ze souboru formátu 3DS. Dále se stará o jejich vykreslování v několika režimech. Třída je odvozena od ***CBaseObject*** a implementuje všechny její virtuální metody, kromě metody ***Action***, jejíž funkce se liší podle toho, co daný model představuje. Nejdůležitější je metoda ***Draw***, jejímiž parametry jsou počátek jména objektu k vykreslení a typ vykreslování. Typů vykreslování je definováno pět.

Jedná se o tyto typy:

1. **TP_ALL** – vykreslí všechny objekty v modelu
2. **TP_OPACQUE** – vykreslí pouze neprůhledné objekty
3. **TP_TRANSP** – vykreslí pouze průhledné objekty
4. **TP_PICK** – vykreslí objekty zjednodušeným způsobem
5. **TP_EXT_MAT** – nepoužívá vlastní materiály pro kreslení

Rozdělení na průhledné a neprůhledné objekty je důležité, protože průhledné objekty se musí kreslit až po neprůhledných. Typ **TP_PICK** je důležitý pro rychlostní optimalizaci vykreslování pro mechanismus výběru. Při tomto vykreslování se nepoužívá osvětlení, s tím souvisí to, že není potřeba vykreslovat normálové vektory a nenastavuje se různý materiál. Typ **TP_EXT_MAT** slouží k tomu, aby bylo možné použít k vykreslení určitého objektu nebo celého modelu jiný materiál než se kterým je model vytvořen. Jednotlivé objekty modelu, vybrané pomocí počátku jména (je vysvětleno dále), jsou vykreslovány standardním způsobem pouze s vynecháním nastavování materiálových vlastností.

Důležitým parametrem při vykreslování je také počátek jména objektu k vykreslení. Pomocí tohoto mechanismu si mohu vybrat, které objekty v rámci modelu chci vykreslit na základě jejich jména, případně řetězce kterým jejich jméno začíná. To je důležité při mechanismu výběru a také k určení interakce a typu interakce objektu. Dále v této práci bude popsán způsob, jakým musí být jednotlivé objekty v rámci modelu popsány a jak musí být uspořádány, aby byl objekt v rámci scény interaktivní. Důležitost tohoto parametru je dobře vidět v kapitole 4.5 Pojmenování objektů v rámci modelu.

Dalšími metodami jsou ***SetRot*** a ***SetPos***. Těmito metodami je nastavováno relativní umístění objektu ve scéně a také jejich relativní natočení. Na umístění a natočení modelu je třeba dávat pozor už při jeho vytváření, protože vrcholové souřadnice všech objektů v sobě nesou i informaci o jejich absolutní pozici a natočení. Proto je důležité už při vytváření modelu mít určené, kde je počátek souřadné soustavy modelu. V číselném vyjádření kde je nula v jednotlivých osách.

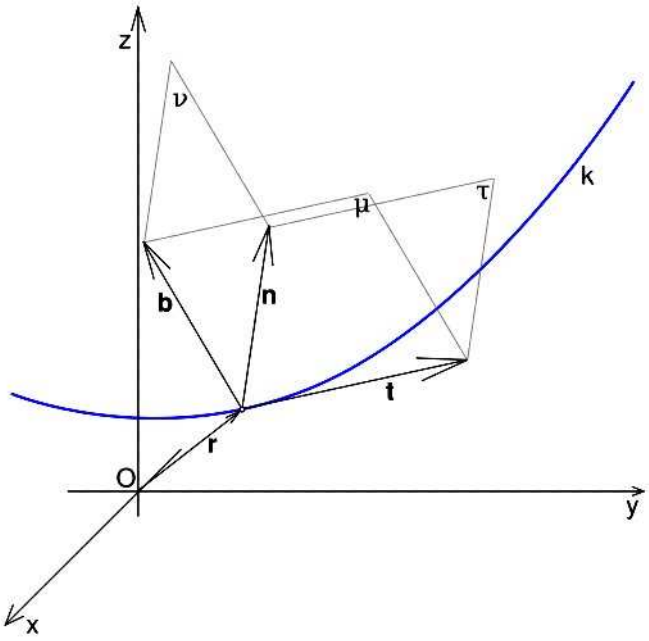
4.3.5 Třída CBezierCurve

Tato třída, stejně jako všechny třídy určené ke kreslení a výběrový mechanismus, je odvozena od ***CBaseObject***. Slouží k vytváření „drátů“ k propojování přístrojů a dalších objektů v rámci virtuální laboratoře. Jak už název napovídá, dráty jsou v prostoru vedeny po Beziérových křivkách. Beziérovy křivky jsou k této úloze velice vhodné svým zadáním. Tedy počátečním a koncovým vektorem. Počáteční vektor udává směr, ze kterého drát vychází a koncový vektor směr, do kterého drát na konci směřuje. Tyto vektory jsou získány z objektů, které jsou určeny pouze pro to, aby nesly tuto informaci. Tyto objekty jsou pojmenovány tak, aby nebyly při zpracování scény vykreslovány – začínají znakem podtržítka a písmenem B tedy „_B“. Písmeno B značí, že jde o objekt určený pro tvorbu Beziérových křivek. Pravidla pro pojmenovávání objektů v rámci modelu jsou popsány dále.

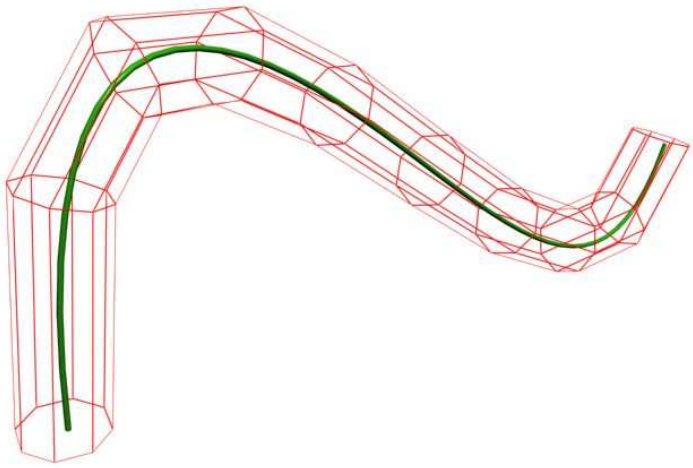
Matematický popis Beziérovy křivky je poměrně jednoduchý a výsledný dojem se blíží tomu, jak by dráty v reálném světě mohly za určitých podmínek vést. Pro jednoduchost není kontrolován objem jednotlivých drátů, zdali nekolidují s jiným objektem, ať už dalším drátem nebo třeba přístrojem. Protože jsou pro zadání křivky použity dva vektory tedy čtyři body, použijí se Beziérovy kubické křivky, které jsou složeny z lineární kombinace Bernsteinových polynomů třetího stupně.

Beziérova křivka jako taková nemá žádný objem. Je proto nutné ji nějakým způsobem „obalit“, aby z ní vznikl objekt podobající se drátu. To je řešeno tak, že je kolem Beziérovy křivky sestrojen pomyslný válec. Pro výpočet vrcholových souřadnic tohoto válce je nutné nejdříve sestrojit kružnici, která leží na ploše kolmé k tečně křivky v bodě, kterým daná plocha prochází. K tomu je nutné spočítat kolmici k Beziérově křivce. K tomu je použit takzvaný **Frenetův trojhran** (Obrázek 4-2 Frenetův trojhran), pomocí kterého je spočítána binormála a normála křivky. Tyto dvě normály tvoří bázové vektory plochy, na níž jsou pomocí funkcí sinus a cosinus spočítány vrcholové souřadnice ležící na kružnici. Stejným způsobem jsou spočítány i normálové vektory, které jsou z hlediska potřeb osvětlovacího mechanizmu OpenGL normalizovány. Vrcholové body jednotlivých sousedních kružnic jsou spolu náležitým způsobem pospojovány tak, aby tvořily výslednou válcovitou plochu kolem Beziérovy křivky.

Třída ***CBezierCurve*** má stejně jako ***C3DModel*** metody pro vykreslení zjednodušeného modelu pro mechanismus výběru a metody pro posun a rotaci.



Obrázek 4-2 Frenetův trojhran



Obrázek 4-3 Tvorba objemu Beziérovy křivky

4.3.6 Třída C3DWires

Výsledků třídy ***CBezierCurve*** je využito v této třídě, která se stará o všechny dráty umístěné ve scéně, ať už jde o jejich tvorbu, nebo jejich rušení. K modelu drátu, který je vypočítán na základě počátečního a koncového vektoru je možné načíst ještě model svorky, aby drát vypadal realisticky. Model svorky je umístěn v externím souboru formátu 3DS a je použit při tvorbě každého nového drátu. Při tvorbě drátu je zadáno kromě počátečního a koncového vektoru drátu také materiál, ze kterého budou drát i svorka vytvořeny a také jméno souboru ve kterém je umístěn model svorky. Pokud je model svorky už jednou načtený, třeba při tvorbě minulého drátu, tak je použit již načtený model a nenačítá se znovu. Stejně tak je model svorky umístěn v paměti pouze jednou a při tvorbě drátu je k danému drátu pouze umístěn odkaz na tento model. Podle počátečního a koncového vektoru pro tvorbu drátu jsou vypočítány polohy pro svorky a model drátu je posunut o délku svorky.

Jednotlivé dráty jsou umístěny v dynamickém poli, které je aktualizováno při každém přidání nebo odebrání drátu. Pro přidání drátu slouží metody ***AddWire*** a k odebrání ***DeleteWire***. Jedním z parametrů ***AddWire*** je i délka přidávané svorky. Pokud je tato délka nulová (nebo není zadána), tak je tato délka dopočítána z délky modelu svorky a mírně zkrácena, aby drát v každém případě pěkně navazoval. Pro to, aby bylo možné svorku použít, je nutné její přesné umístění už při jejím návrhu. Svorka musí být umístěna tak, aby její spodní okraj byl v absolutním počátku souřadného systému, a musí směřovat vzhůru, tedy její nejdelší rozměr musí být v ose Y.

To, že jsou všechny dráty umístěny na jediném místě, tedy v této třídě, je také výhodné z hlediska kontroly zapojení.

4.3.7 Třída CMyArcBall

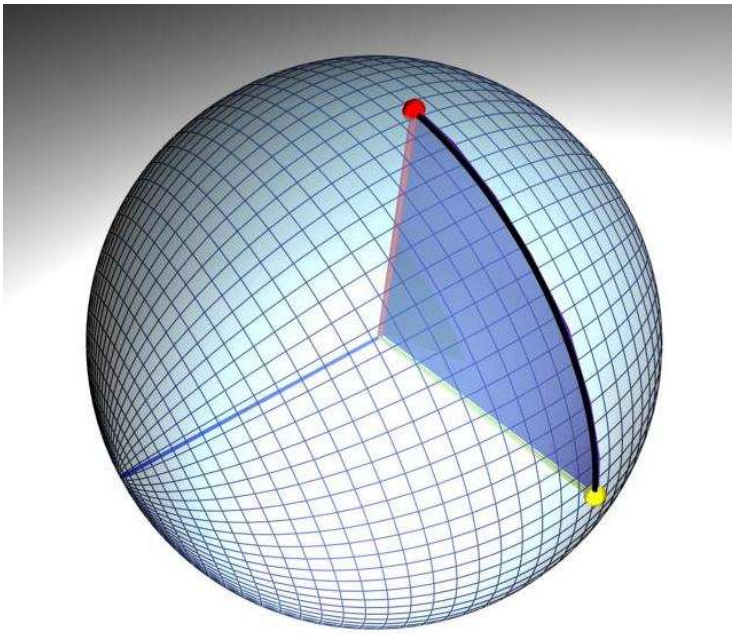
Pro správnou orientaci a práci ve 3D scéně je nutné mít možnost se na ni podívat z různých úhlů a správně si ji natočit. Pro natáčení ve 3D prostředí, tak aby to bylo intuitivní, by bylo nejlepší mít nějaké zařízení, jehož natáčením v požadovaném směru by se natáčela i celá scéna. Tato zařízení však nejsou v dnešní době standardním vybavením počítače a je proto potřeba si vystačit s myší. Problém převodu pohybu myši, která se pohybuje pouze ve dvou osách (tedy 2D), na otáčení ve třech osách (tedy 3D) je v tom, aby byl intuitivní a tím pádem pro člověka použitelný. Tento problém je řešen v 3D aplikacích různě, ať už více či méně zdařile. Existuje jeden velice dobrý algoritmus založený na principu natáčení kulové plochy – pro představu třeba balónu. Princip algoritmu spočívá v tom, že na obrazovce je možné zobrazit přivrácenou polovinu této kulové plochy a na jakýkoliv bod této plochy ukázat myší. Tento bod lze chytnout myší a přetáhnout do jiného místa ležícího na této kulové ploše. Přesto, že tato kulová plocha nemusí být ani na obrazovce vykreslena, tak je natáčení prostřednictvím tohoto algoritmu velice intuitivní.

Nejlepší představu o právě popsaném principu si lze udělat z obrázku (Obrázek 4-4 Otáčení pomocí ArcBallu) na následující stránce.

Z bodu A (nahoře) se do bodu B (dole) lze dostat pomocí rotace kolem osy, která je kolmá k vektorům ze středu kulové plochy do jednotlivých bodů. Jednotlivé body na kulové ploše se jednoduše spočítají ze souřadnic myši (tedy X a Y) a Z se dopočítá podle známého vzorce

$$X^2 + Y^2 + Z^2 = r$$

Mohou nastat dva případy. Buďto se kurzor myši nachází uvnitř kružnice o poloměru shodném s poloměrem kulové plochy a Z lze dopočítat, nebo se nachází mimo tuto kružnici a potom Z je nulové a hodnoty X a Y se dopočítají tak, aby se nacházely přesně na kružnici – tedy na kulové ploše. Z počátečního, koncového a středového bodu je lehce spočítán úhel i osa rotace. Při tažení myší je počáteční bod bodem kliknutí a koncový bod je průběžně počítán při každém překreslení plochy podle současné pozice kurzoru myši.



Obrázek 4-4 Otáčení pomocí ArcBallu

4.4 Mechanismus výběru – PICKING

Pro interakci uživatele a objektů ve scéně je nutné nějakým způsobem zjistit, na který objekt uživatel právě ukazuje kurzorem myši, případně který objekt vybral kliknutím myši. Způsobů jak toho dosáhnout je více.

Jedním z nich je přepočet souřadnic myši do 3D. Přesněji řečeno z kurzoru myši je veden paprsek směřující skrze celou scénu tak, aby byl v ose kamery. Paprsek pak protíná jednotlivé objekty a na základě toho poté vybere nejbližší. Pro výpočet toho, který objekt byl protnut, se však musí vypočítat protnutí paprsku se všemi trojúhelníky, které tvoří objekt. To může být časově velice náročné a navíc to vše musí počítat procesor počítače (CPU) a nikoliv procesor grafické karty (GPU).

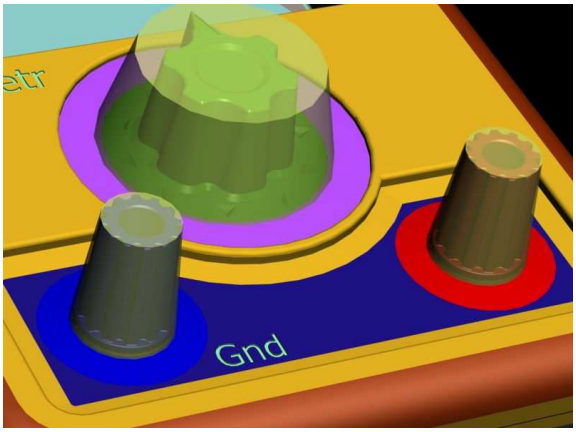
Jiným řešením je nastavit velice malou vykreslovací oblast pod kurzorem, vykreslit všechny objekty, u kterých nás zajímá, jestli na ně „ukazuje“ kurzor myši a poté vyhodnotit, které objekty byly kresleny do této oblasti a které ne. Tím se přesunou veškeré výpočty na stranu grafické karty, která je na to optimalizovaná.

OpenGL má mechanismus založený právě na tomto principu, kdy se vybere oblast několika pixelů (obrazových bodů), nastaví se speciální vykreslovací režim a postupně se nastavují jména objektů (v rámci kontextu OpenGL se hovoří o jménech, ale ve skutečnosti se jedná o číselné hodnoty) a tyto objekty se vykreslují. Pokud se objekt nachází ve vybrané oblasti, tak se vytvoří záznam, který obsahuje jméno objektu a maximální a minimální hloubku, kterou objekt při kreslení zasáhnul. Hloubka je určena z takzvaného *Z bufferu* (někdy označovaného jako Depth Buffer), jehož primární funkcí je řešení viditelnosti jednotlivých pixelů při normálním vykreslovacím režimu, kdy dojde k vykreslení pouze při splnění určité podmínky (většinou při hodnotě vyšší nebo rovné hodnotě aktuální). Ve výběrovém režimu je však použita jeho hodnota pro určení hloubky, ať už je pixel viditelný z hlediska pozorovatele či nikoliv.

4.4.1 Implementace mechanismu výběru v aplikaci virtuálních laboratoří

Mechanismus výběru je začleněn do třídy ***COGL_Base***. Používá metody ***InitPicking***, ***ProcesPicking*** a ***FinishPicking***.

Pro rychlostní optimalizaci je výběr prováděn ve více etapách. V první etapě jsou kresleny speciální zjednodušené objekty (Obrázek 4-5 Zjednodušené objekty pro optimalizaci výběru), které se svým tvarem podobají objektu určenému pro testování na „zásah“ kurzorem. Tyto objekty jsou kresleny bez jakéhokoliv materiálu a osvětlení tak aby jejich vykreslování zabíralo co nejméně času. Pro každý objekt, u kterého dojde k „zásahu“, se vytvoří záznam, který obsahuje ukazatel na kreslený objekt a jméno počátečního řetězce s jakým byl kreslen. Důvodem této etapy je to, aby se kreslilo co nejméně složitých objektů. Ve druhé etapě je podle seznamu objektů zasažených z minula provedeno kreslení přesného objektu. Kreslení se však provádí bez nastavování materiálu a bez výpočtu osvětlení aby bylo co nejrychlejší. Opět se dělá seznam objektů, u kterých dojde k „zásahu“. V poslední etapě se tento seznam prochází a vybere se ten objekt, který je nejblíže.



Obrázek 4-5 Zjednodušené objekty pro optimalizaci výběru

InitPicking vynuluje vnitřní proměnné, ve kterých je uložen záznam o vybraných objektech z minula. Dále nastaví projekční matici tak, aby obsahovala oblast o velikosti 4 x 4 pixely (obrazové body) pod kurzorem myši. Jako poslední akci nastaví OpenGL Buffer pro ukládání záznamů o „zásahu“ objektu kurzorem.

ProcesPicking provádí kreslení velmi zjednodušených 3D objektů a vytváří seznam objektů, u kterých došlo k „zásahu“ kurzorem myši. Jde o jakýsi před-výběr – *PrePicking*.

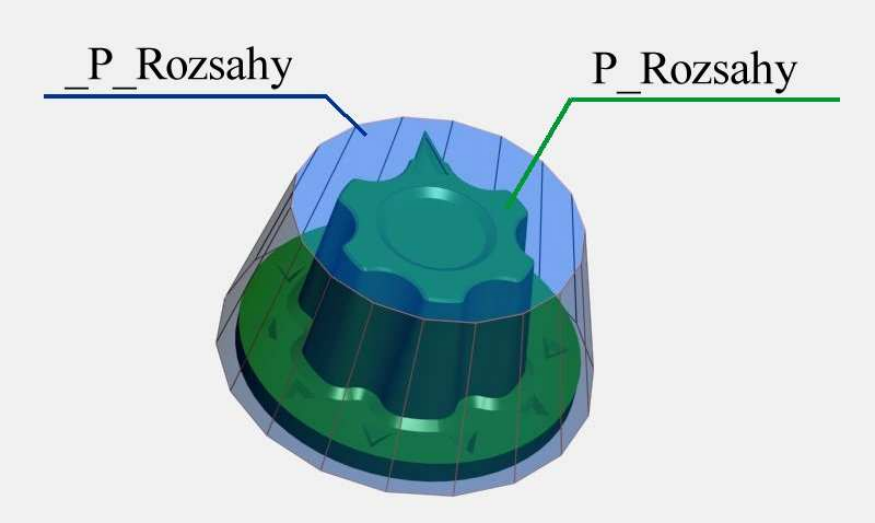
FinishPicking vykresluje podle seznamu z předešlého kreslení detailní objekty a opět vytváří záznam „zasažených“ objektů. Po vykreslení všech objektů se seznam prochází a vybírá se ten, který byl nejblíže.

Metody ***ProcesPicking*** a ***FinishPicking*** pracují se třídou ***CBaseObject,*** protože nemohou vědět, jestli je zpracovávaným objektem třída ***C3DModel***, ***C3DWires*** nebo jiná. Proto je nutné, aby byly všechny tyto třídy odvozeny od třídy ***CBaseObject*** a mechanismus polymorfizmu se postará o to, aby byly volány při vykreslování ty správné metody.

4.5 Pojmenování objektů v rámci modelu

Proto, aby bylo možné objekty v modelu správně identifikovat a použít příslušným způsobem, je nutné zavést určitý systém jejich pojmenovávání. Jména objektů jsou jediným možným způsobem identifikace jednotlivých objektů v rámci modelu. Modelem je v tomto kontextu brán soubor všech 3D objektů nacházejících se v souboru formátu 3DS. Nezáleží při tom, zda se jedná o objekt plochy, čáry nebo objemový.

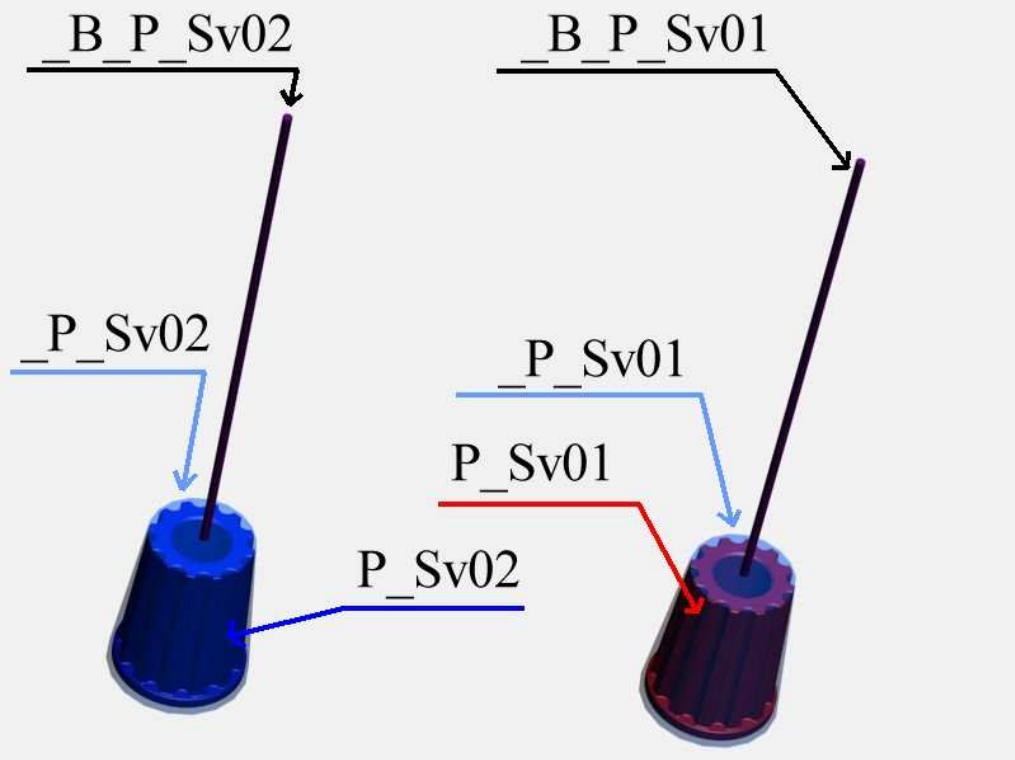
V modelu jsou primárně data určená k zobrazení. Je ale možné použít určitá data i k jiným účelům. V rámci tohoto projektu je zavedena konvence pojmenování, které určuje, zdali je objekt vykreslován nebo používán jiným způsobem. Všechny objekty, které nejsou určeny k zobrazení, jsou pojmenovány tak, aby začínaly znakem podtržítka „_“. Při vykreslování jsou pak takové objekty vyloučeny. Dále jsou určeny všechny objekty, které mají určitým způsobem umožňovat interakci s uživatelem označeny písmenem velké „**P**“ (odvozeno od *Picking*). Kombinací podtržítka a písmene „**P**“ vzniká další důležitý typ objektu určený pro takzvaný *PrePicking* – tedy před-výběr (viz předešlá kapitola). Podtržítko musí být jako první výsledný tvar, tedy je „**_P**“. Takovýto objekt tedy není ve scéně vidět (vliv podtržítka), ale při první fázi výběru je vykreslován místo detailního objektu (funkce optimalizace). Mechanismus výběru nebude fungovat s objektem, který nemá tento zjednodušený model, přestože jeho název bude začínat písmenem „**P**“. Je to proto, že takový objekt neprojde procesem před-výběru. Stejně tak je důležité, aby se jméno zjednodušeného objektu lišilo pouze oním podtržítkem, protože jméno k vykreslení detailního objektu je získáno právě odstraněním podtržítka. Situaci znázorňuje Obrázek 4-6 Pojmenování interaktivních objektů.



Obrázek 4-6 Pojmenování interaktivních objektů

Dalším písmenem, které má specifický význam, je velké „**B**“. Jedná se o objekt určený k tvorbě „drátu“ – tedy svorkovnice přístrojů a elektrických zařízení. Protože jde o předmět, který nemá být vidět, vystupuje jméno vždy s podtržítkem, tedy „**_B**“. Aby bylo možné pomocí tohoto objektu vytvořit „drát“, musí být s tímto objektem svázán nějaký interaktivní objekt. Tedy objekt, který začíná písmenem „**P**“ (a k němuž musí náležet i zjednodušený objekt „**_P**“). Navíc musí být tento objekt nakreslen v 3D Studiu jako čára začínající u svorky a vedoucí ve směru, ve kterém bude pokračovat Beziérova křivka. Je to proto, že se z tohoto objektu berou první dvě vrcholové souřadnice, z nichž je tvořen vektor pro tvorbu křivky (ať už počáteční nebo koncový). Důležitý je zde jak směr vektoru tak jeho umístění a velikost. Příslušnost jednotlivých objektů je dána jejich

stejným jménem, lišícím se pouze začátkem. Celou situaci znázorňuje Obrázek 4-7 Značení objektů pro tvorbu Beziérových křivek.



Obrázek 4-7 Značení objektů pro tvorbu Beziérových křivek

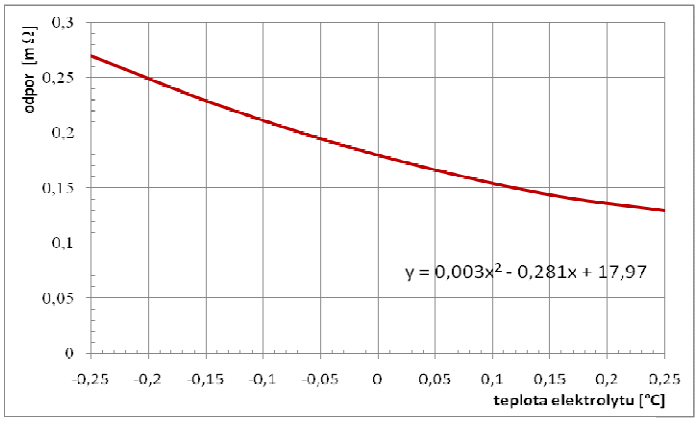
5 Měřená úloha

Podstata virtuální laboratoře spočívá v simulaci měření nějaké úlohy. Bylo vybráno měření na automobilovém akumulátoru. Důvodem je relativně jednoduché zapojení této úlohy a jednoduchý matematický popis celého měření. Jednoduchost matematického popisu vyplývá samozřejmě z řady zjednodušujících předpokladů majících na svědomí jistou nepřesnost. Cílem tohoto projektu však není přesné matematické modelování měřených jevů, ale vytvoření programového základu, který umožní simulaci měření. Tvorba přesnějších matematických modelů a jednotlivých úloh pro tyto laboratoře je předpokládána jako pokračování tohoto projektu.

V automobilovém akumulátoru probíhají komplexní chemické reakce, na které má vliv spousta vnějších faktorů. Těmi jsou například velikost vybíjecího/nabíjecího proudu, teplota elektrolytu, vnitřní mechanické uspořádání akumulátoru, chemické složení elektrolytu a další. Matematický model v tomto projektu zohlední pouze vnitřní odpor akumulátoru v závislosti na teplotě (se zanedbáním přechodných jevů vlivem chemického ustálení a konstantní teplotě) a dále na kapacitu baterie v závislosti na odebíraném proudu a teplotě. Teplota je brána jako konstantní během měření (zanedbají se jak změna teploty okolí tak teplo vznikající vlivem elektrochemických reakcí uvnitř článků).

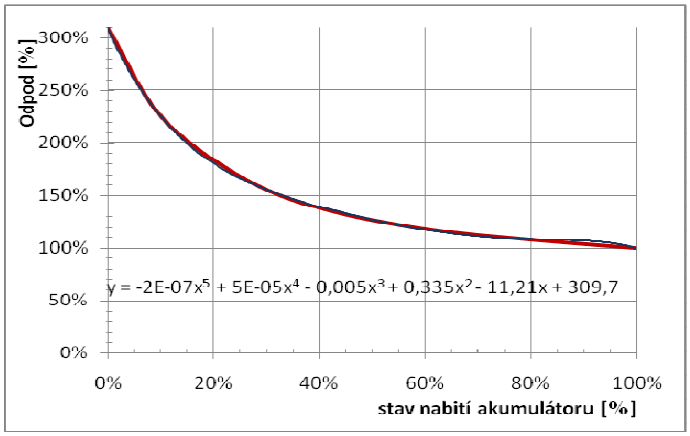
Závislosti na teplotě a proudu jsou vzaty ze známých průběhů v grafickém vyjádření. Pro potřebu jejich matematického vyjádření je použita interpolace polynomem vyššího stupně. Vnitřní odpor akumulátoru je součtem odporů jednotlivých článků a jejich spojek. Na vnitřní odpor akumulátoru má odebíraný proud minimální vliv, takže je zanedbán. Podstatně větší vliv má

teplota akumulátoru. Odpor při teplotě -18°C bývá až dvojnásobný oproti teplotě při +25°C. Na vnitřní odpor má také vliv stav nabití akumulátoru. Při téměř vybitém akumulátoru je vnitřní odpor více než trojnásobný. Grafickou závislost odporu elektrolytu na teplotě ukazuje Obrázek 5-1 Závislost odporu na teplotě.



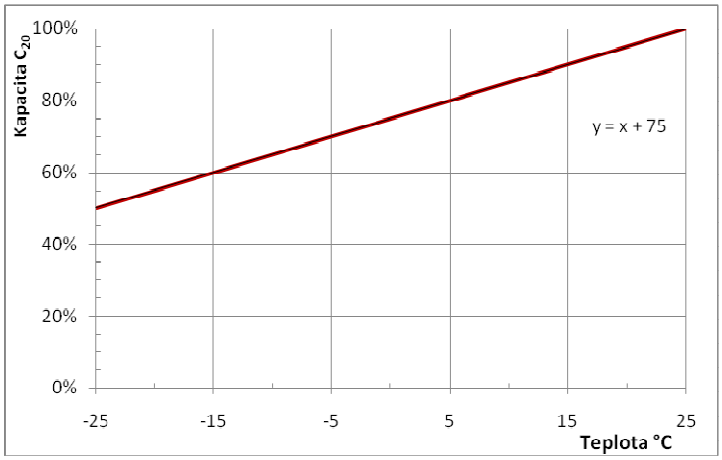
Obrázek 5-1 Závislost odporu na teplotě

Vnitřní odpor akumulátoru také značně závisí na stavu nabití akumulátoru. Pro dostatečně přesnou interpolaci je potřeba polynom pátého stupně, který není ideální v oblasti 80 až 100%. Řešením by bylo použít jinou metodu interpolace, například spline křivku. Pro tuto aplikaci je však použit polynom pátého stupně, protože přesnost simulace není v tomto projektu prioritou.



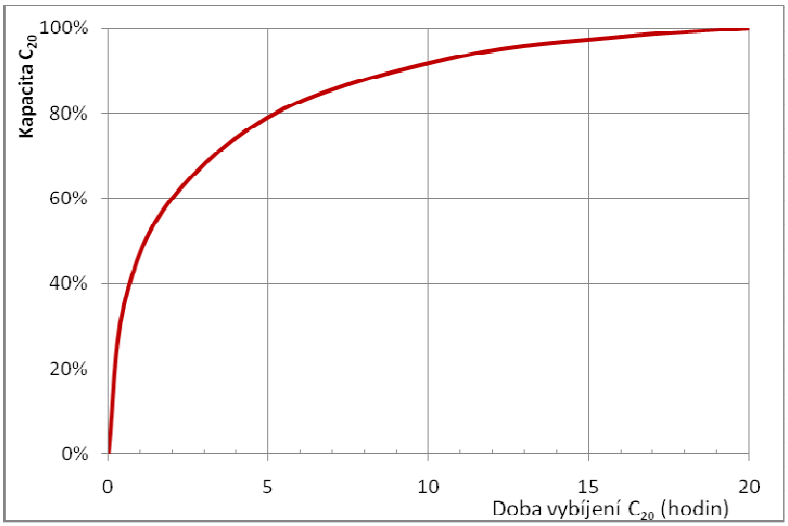
Obrázek 5-2 Závislost odporu na stavu nabití akumulátoru

Závislost jmenovité kapacity na teplotě elektrolytu ukazuje Obrázek 5-3 Závislost kapacity na teplotě



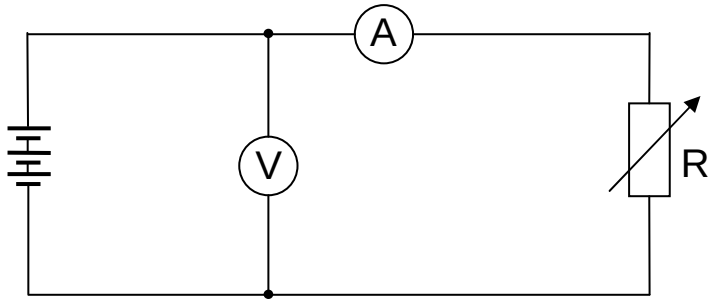
Obrázek 5-3 Závislost kapacity na teplotě

Poslední simulovanou závislostí v tomto projektu je závislost kapacity na odebíraném proudu.



Obrázek 5-4 Závislost kapacity na odebíraném proudu

Při měření úlohy se předpokládá zapojení, které ukazuje Obrázek 5-5 Zapojení měření.



Obrázek 5-5 Zapojení měření

Podle tohoto schématu se bude provádět kontrola správnosti zapojení před spuštěním vlastní simulace měření. V průběhu simulace je kontrolováno, zda měřené veličiny nepřesahují nastavené rozsahy. Pokud měřená veličina přesáhne nastavený rozsah o více než dvacet procent, tak je měření ukončeno a vypsána chybová hláška. Stejně tak je kontrolováno maximální výkonové zatížení rezistoru, které se nesmí v průběhu měření přesáhnout.

6 Postup měření v rámci virtuální laboratoře

Po spuštění programu virtuálních laboratoří se objeví pracovní stůl, na kterém jsou umístěny všechny objekty, které jsou pro danou úlohu zapotřebí. V úloze měření automobilového akumulátoru se jedná o akumulátor, dva multimetry a zatěžovací rezistor s možností nastavení jeho rezistivity a maximálního výkonového zatížení.

Úkolem měření bude provést správné zapojení všech objektů a nastavení měřicích rozsahů multimetrů a nastavení měřené veličiny (tedy volty nebo ampéry). Dále je potřeba před spuštěním měření nastavit stav nabití akumulátoru a teplotu, při níž se měření provádí. Po zapojení je spuštěna kontrola správnosti zapojení. Pokud je zapojení správné, tak lze odečíst měřené veličiny z příslušných přístrojů. Měřené veličiny se na přístroji zobrazují analogově pomocí ručičky přístroje. To však slouží spíše k orientačnímu účelu. Pro přesné odečtení hodnoty se klikne na display daného přístroje a v dialogovém okně se objeví přesná naměřená hodnota. V průběhu měření je možné nastavovat hodnotu zatěžovacího rezistoru, aby bylo možno sestrojit zatěžovací charakteristiku a určit vnitřní odpor akumulátoru. Pro měření vybíjecích charakteristik je možné určit koeficient plynutí času a měření (které by jinak trvalo poměrně dlouho) tak zrychlit.

7 Závěr

Cílem této práce bylo vytvoření jednoduché virtuální laboratoře z oblasti silnoproudé elektrotechniky. Důraz byl kladen na maximální možnou rozšiřitelnost a znovu použitelnost již napsaného zdrojového kódu bez potřeby hlubších znalostí vnitřní implementace. Pro demonstraci funkčnosti byla naprogramována úloha z oblasti automobilové elektrotechniky – měření akumulátoru. Po přečtení této práce by neměl být pro člověka znalého C++ a OpenGL problém použít zdrojové soubory a dále rozšířit funkčnost této laboratoře o další úlohy k měření, což je hlavní záměr této práce.

V první kapitole je popsáno rozhraní OpenGL, které zpřístupňuje možnosti současných grafických akcelerátorů a umožňuje tak simulaci laboratoře v trojrozměrném prostředí v reálném čase.

Druhá kapitola všeobecně popisuje data, která slouží pro reprezentaci 3D objektů v oblasti počítačového zpracování, ať už jde o vizualizaci či nikoliv. Dále uvádí některé často používané datové formáty pro ukládání těchto dat.

Třetí kapitola detailně popisuje datový formát souborů s příponou „.3DS“, který je vhodný pro reprezentaci trojrozměrných objektů. Vhodnost formátu spočívá v jeho relativní rozšířenosti, dostupnosti jeho vnitřní struktury a také proto, že obsahuje veškerá data, která jsou potřeba pro věrnou vizualizaci objektů v něm uložených.

Čtvrtá kapitola podrobněji popisuje základní stavební prvky, na nichž je aplikace virtuální laboratoře vystavěna. Jedná se o vlastní zpřístupnění funkcí OpengGL v rámci operačního systému Microsoft Windows a v neposlední řadě o způsob, jakým jsou tyto funkce využity pro potřeby aplikace virtuální laboratoře. Jsou popsány jednotlivé třídy, které jsou naprogramovány vhodně k jejich opětovnému použití a snadné modifikaci jejich funkčnosti.

Pátá kapitola stručně popisuje vybranou měřenou úlohu a uvádí prvky, které jsou v rámci této implementace simulovány.

Šestá kapitola uvádí postup, jakým probíhá vlastní virtuální měření.

Podstatnou částí této práce jsou bohatě komentované zdrojové soubory, které jsou umístěny na přiloženém CD.

Použitá literatura

[1] http://www.root.cz/slovnicek/opengl/

[2] http://www.root.cz/serialy/graficka-knihovna-opengl/

[3] http://opengl.org/about/

[4] http://nehe.ceske-hry.cz/tut_obsah.php

[5] Dave Shreiner, Mason Woo, Jackie Neider, Tom Dawis. ***OpenGL Průvodce programátora.*** Computer Press, 2006, 680 stran. ISBN 80-251-1275-6

[6] http://hippo.nipax.cz/docs/3ds_format.rtf
http://mediasrv.ns.ac.yu/extra/fileformat/3d/3ds/3ds.rtf

Přílohy

CD obsahující:

1. Zdrojové soubory
2. Spustitelný soubor virtuální laboratoře
3. Modely použitých 3D objektů