# TECHNICKÁ UNIVERZITA V LIBERCI

**Fakulta mechatroniky a mezioborových inženýrských studií**

Katedra softwarového inženýrství Školní rok: 2000/2001

# ZADÁNÍ DIPLOMOVÉ PRÁCE

pro: Marka Pavlíka

studijní program: 2612T – Elektrotechnika a informatika

obor: Automatické řízení a inženýrská informatika

Vedoucí katedry Vám ve smyslu zákona o vysokých školách č.111/1998 Sb. určuje tuto diplomovou práci:

Název tématu:

Editor barevných vlastností materiálů pro grafickou knihovnu OpenGL

Zásady pro vypracování:

1. Seznámit se s tvorbou osvětlených prostorových scén pomocí grafické knihovny OpenGL.
2. Vytvořit aplikaci, která bude umožňovat editaci barevných vlastností materiálů pro grafickou knihovnu OpenGL.
3. Realizovat export výsledků editace ve vhodném tvaru použitelném při dalším programování pomocí knihovny OpenGL.
4. Umožnit opětovné načtení a následnou editaci souborů touto aplikací dříve vytvořených.

Rozsah grafických prací: dle potřeby dokumentace

Rozsah průvodní zprávy: cca 40 až 50 stran

Seznam odborné literatury:

[1] NEIDER, J. - DAVIS, T. - WOO, M.: OpenGL Programming Guide : The Official Guide to Learning OpenGL, Release 1. Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.

[2] ŽÁRA , J. - BENEŠ, B. - FELKEL, P.: Moderní počítačová grafika. Computer Press, Praha, 1998.

[3] CANTU, M.: Delphi 4: podrobný průvodce programátora. Grada, Praha, 1999.

[4] Elektronická dokumentace vývojového prostředí programovacího jazyka Delphi.

Vedoucí diplomové práce: Ing. Jiřina Královcová

Konzultant:

Zadání diplomové práce: 25. 10. 2000

Termín odevzdání diplomové práce: 25. 5. 2001

Technická univerzita v Liberci * Fakulta mechatroniky a mezioborových inženýrských studií

L.S.

Ing. Petr Tůma, CSc.

.....

Vedoucí katedry

Prof. Ing. Vojtěch Konopa, CSc.

.....

Děkan

V Liberci dne 25. 10. 2000

# TECHNICKÁ UNIVERZITA V LIBERCI

FAKULTA MECHATRONIKY A MEZIOBOROVÝCH INŽENÝRSKÝCH STUDIÍ


**Studijní program :** 2612T Elektrotechnika a informatika

**Studijní obor :** Automatické řízení a inženýrská informatika

## Editor barevných vlastností materiálů pro grafickou knihovnu OpenGL

(The Editor of coloured material properties for graphics library OpenGL)


**Autor:** Marek PAVLÍK

**Adresa:** Dobiášova 860/14

46006, Liberec 6

**Vedoucí práce:** Ing. Jiřina Královcová


**Počet**

| Stran | Obrázků | Tabulek |
|---|---|---|
| 49 | 17 | 3 |

V Liberci dne 24.5.2002

ANOTACE:

Cílem mé diplomové práce bylo vytvořit editor barevných materiálů pro grafickou knihovnu OpenGL, který bude sloužit jako doplněk při výuce a také jako pomocný nástroj programátora. Editor umožňuje editaci barev a dále pak export výsledků ve tvaru použitelném při dalším programování pomocí knihovny OpenGL a také umožňuje opětovné načtení a následnou editaci souborů touto aplikací dříve vytvořených.

SUMMARY:

The point of my thesis was to create the editor of coloured material properties for graphic library OpenGL, which will do as addition for teaching and as help tool for programmer. The editor allows editing colours and further to export events in a form applicable for next programming and it also allows reloading and editing of files created by this application before.

Prohlášení o původnosti práce:

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

V Liberci dne: 24.5.2002

Marek Pavlík

Marek Pavlík

Prohlášení k využívání výsledků DP:

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském zejména § 60 (školní dílo).

Beru na vědomí, že Technická univerzita v Liberci (TUL) má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **souhlasím** s případným užitím mé diplomové práce (prodej, zapůjčení, kopírování, apod.).

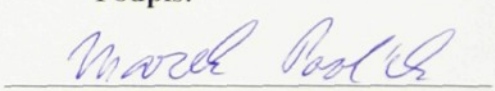
Jsem si vědom toho, že: užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše). Diplomová práce je majetkem školy, s diplomovou prací nelze bez svolení školy disponovat.

Beru na vědomí, že po pěti letech si mohu diplomovou práci vyžádat v Univerzitní knihovně Technické univerzity v Liberci, kde bude uložena.

**Autor:**
Marek PAVLÍK

**Adresa:**
Dobiášova 860/14
46006, Liberec 6

**Podpis:**



**Datum:**
24.5.2002

# OBSAH

Seznam příloh

# 1 Úvod

Grafická knihovna OpenGL (Open Graphics Library – dále jen OpenGL) je dnes standardem v oblasti grafických aplikací. Cílem mé práce bylo vytvořit Editor barevných materiálů pro grafickou knihovnu OpenGL, který bude sloužit jako doplněk při výuce a také jako pomocný nástroj programátora. Editor umožňuje editaci barev a dále pak export výsledků ve tvaru použitelném při dalším programování pomocí knihovny OpenGL a také umožňuje opětovné načtení a následnou editaci souborů touto aplikací dříve vytvořených.

Pro účely práce bylo potřeba se seznámit s knihovnou OpenGL a především s tvorbou osvětlených scén. Takto získané poznatky pak implementovat do vlastní aplikace editoru barevných vlastností.

# 2 OpenGL

## 2.1 Co je OpenGL

Grafická knihovna OpenGL je definována jako „a *software interface to graphics hardware*" (programové rozhraní pro grafický hardware). Představuje jednotné API (Application Programming Interface) mezi programem a grafickým hardware. Vlastní knihovna je soubor asi 150 funkcí, které umožňují specifikovat geometrické objekty ve dvou nebo třech dimenzích. Takto popsané objekty potom umožňuje dále zpracovávat.

Hlavním rysem OpenGL je nezávislost na cílové platformě a použitém programovacím jazyku, proto autor programu nemusí detailně znát konfiguraci počítače, na kterém jeho program poběží. Při značně zjednodušeném pohledu lze říci, že programátor pouze udává souřadnice vrcholů trojúhelníků společně s informací o mapování textury, vlastní vykreslování už za něj vykoná OpenGL. Má přitom možnost zapínat a vypínat různé její dílčí funkce, a tak přímo ovlivňovat výslednou kvalitu vytvářeného obrazu. Komplexnější vykreslení scény znamená často velmi malý nárůst kódu.

## 2.2 Historie

OpenGL je relativně nový průmyslový standard, který se jen během několika málo let stal široce uznávaným a podporovaným na poli grafických aplikací, CAD/CAM/CAE inženýrských systémů, virtuální reality a tvorby her.

Předchůdcem OpenGL byla GL od Silicon Graphics (SGI). Jejich „IRIS GL" byla hardwarově optimalizována pouze na grafické stanice IRIS, a proto při pokusu použít tuto knihovnu na jiné hardwarové konfiguraci se objevily problémy.

OpenGL je výsledek snahy SGI zlepšit přenositelnost IRIS GL na jiné hardwarové platformy a jiné operační systémy. První specifikace této knihovny (OpenGL 1.0) byla představena 1.července 1992.

Každá implementace, která usiluje o označení OpenGL, musí projít sérií testů (Conformance Tests), které musí zcela splnit. Na to, zda je implementace v souladu se

specifikacemi OpenGL dohlíží konsorcium ARB (Architecture Review Board), do kterého patří firmy Digital Equipment Corporation (DEC), Hewlett-Packard, IBM, Intel, Intergraph, Microsoft, Silicon Graphics a Sun Microsystems.

Protože je OpenGL používána již osm let, nevyhnula se za dobu své existence určitým inovacím. Současná verze knihovny je 1.2. Podobně jako předchozí verze, i verze 1.2 do sebe pojala nové rysy, které byly dříve dostupné pouze jako systémová rozšíření.

V současnosti je knihovna OpenGL implementována na téměř všech platformách, z nichž jmenujme ty nejvýznamnější – IRIX, Unixy, Windows 95 a novější, DOS, Linux, Solaris, MacOS, OS/2, NeXTSTeP, OPENStep a BeOS. Trochu nečekaně lze OpenGL nalézt i na některých palmtopech. OpenGL využívá značné množství komerčních programů. Z vizualizačních programů například Kinetix 3D Studio MAX, Alias, Wavefront Maya, CaligaritrueSpace nebo LightWave 3D. Z oblasti CAD potom MicroStation, CATIA nebo SolidWorks. Možností OpenGL knihovny využívají i některé hry jako například Quake 2, Unreal, Sin nebo Half Life.

## 2.3 Možnosti OpenGL

Rendrovací možnosti OpenGL jsou poměrně široké. Od zobrazování jednoduchých drátových modelů těles, přes ploché stínování (flat shading) až po komplexní zobrazení scény s definovanými světelnými zdroji, průhlednými či lesklými stěnami objektů, texturami, Goraudovým stínováním, mlhou atd.

Autoři OpenGL zvolili přístup, kde jsou objekty scény popsány ne svým objemem, ale plošnými útvary, které určují jejich povrch. Scéna se komponuje z elementárních objektů (primitiv), jejichž množina je značně redukovaná:

- bod
- úsečka
- mnohoúhelník
- bitmapa

Primitiva jsou dány skupinou jednoho nebo více vrcholů. Vrchol (vertex) přitom znamená bod v trojrozměrném prostoru. Význam vrcholů v definici elementárního objektu může být různý. Podle typu potom může znamenat počáteční a koncový bod úsečky, vrchol mnohoúhelníku, apod. Veškeré další informace o scéně, jako například barva, normálový vektor nebo souřadnice textury jsou spjaty právě s vrcholy primitiv. Jednotlivé vrcholy, z nichž se elementární objekt skládá se dají zpracovávat nezávisle v daném pořadí.

Právě na proudovém zpracování obrazových elementů staví OpenGL svůj grafický výkon. Koncept zobrazování OpenGL s sebou nese i jistá omezení. Například je obtížné základními prostředky pracovat s objektem, který je oblý. Parametricky popsané plochy (Bézierovy plochy, NURBS) se nejprve musí převést na síť mnohoúhelníků, a teprve potom se mohou zobrazovat. Jednotlivé komponenty obrazu se zpracovávají odděleně a to má vliv na výslednou kvalitu obrazu. Například výpočet osvětlení se vztahuje vždy lokálně k danému tělesu. Bez detailní simulace šíření světla ve scéně jsou výsledky ne zcela přesvědčivé. Kvalita obrazu výsledné scény se tak pouze blíží metodě sledování paprsku (raytraycing), tedy metodě, která nejlépe aproximuje fyzikální model. Tato omezení jsou vyvážena rychlostí a jednoduchostí výpočtu.

Grafická knihovna OpenGL nabízí mnoho věcí:

- řešení viditelnosti objektů ve 3D
- různé druhy transformace grafických objektů
- definice barev popř. stanovení barevných vlastností povrchů
- osvětlení scény (minimálně 8 světel) včetně nastavení vlastností jednotlivých světel
- výpočet barvy v interakci se světly
- vytvoření průhledných objektů
- vyhlazování (antialiasing)
- mapování textur
- možnost sdružení několika příkazů do jediného makropříkazu pro jeho pozdější vyvolání

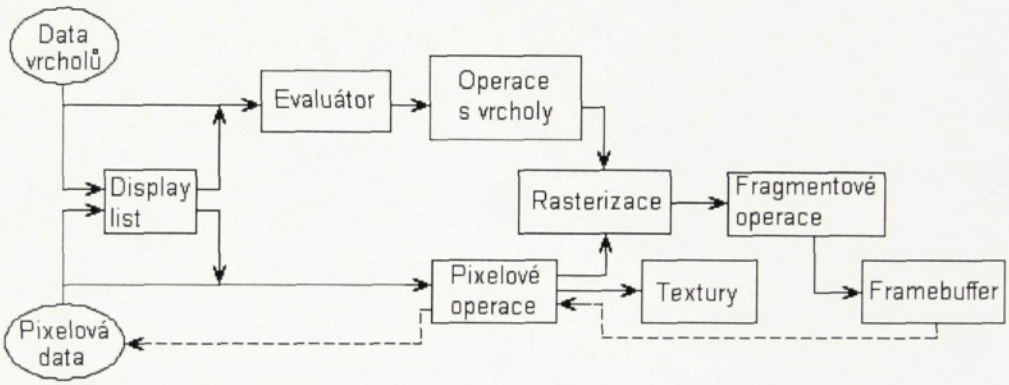
a mnoho dalšího. OpenGL sama o sobě nenabízí prostředky k objektovému popisu scény na hierarchickém principu. K vyřešení některých těchto nedostatků se používají pomocné knihovny:

- GLU (OpenGL Utility Library) – tato knihovna obsahuje funkce pro projekci textur, rozkladu na trojúhelníky (teselace), vykreslování B-spline křivek, některé útvary (válce, koule, disky)
- GLUT (OpenGL Utility Toolkit) – knihovna obsahuje další funkce například pro obsluhu vstupních zařízení jako myš a klávesnice

# 3 Implementace OpenGL

## 3.1 Rendrovací postup OpenGL

Většina implementací OpenGL má podobný sled operací, sérii procesních stavů, které se nazývají „OpenGL rendering pipeline". Tento sled operací znázorněn na obrázku 1 se může mírně lišit v různých implementacích, ale dává dobrou představu jak OpenGL pracuje.




*Obrázek 1 – schéma rendrovacího postupu OpenGL*

Tento diagram znázorňuje, jak OpenGL zpracovává data. Geometrická data (vrcholy, úsečky a mnohoúhelníky) zpočátku prochází jinou cestou než pixelová data. Oboje cesty se na konci spojí do jediné a oba typy dat prochází stejnými procesy (rasterizace, fragmentové operace) než je výsledek uložen do framebufferu. Framebuffer je část paměti ve které se uchovávají informace o zobrazované scéně (např. barva pixelů, vzdálenost pixelů od pozorovatele).

### 3.1.1 Display list

Zpracování elementárních objektů může probíhat buď v „immediate" (bezprostředním) módu nebo „display list" módu. Immediate mód spočívá v tom, že všechny akce s primitivy se provádějí přímo v ten okamžik, kdy jsou programem

zavolány. Tento přístup je vhodný víceméně akorát pro statické scény, kde se objekty neopakují. V případě, že chceme vytvořit v OpenGL animaci, může být tento přístup neefektivní. V každém snímku, třebaže se od předchozího liší jen minimálně, bychom museli znovu specifikovat všechny objekty na scéně. Z toho důvodu je do OpenGL zařazen i Display list mód, který využívá pomocnou datovou strukturu display list. V té se uchovávají předdefinované objekty nebo funkce OpenGL a na místě volání se potom pouze zavolá daný display list. Protože již není nutné např. zadávat znovu vrcholy apod. Dosáhne se tím značného urychlení vykreslování.

### 3.1.2 Evaluátor

Při průchodu touto fází, která je zajišťována podpůrnou knihovnou GLU se složitější grafické objekty jako křivky a polynomiální plochy převádí na jednodušší objekty, se kterými už umí OpenGL přímo zacházet.

### 3.1.3 Operace s vrcholy

Operace s vrcholy zahrnuje zpracovávání jednotlivých vrcholů. Jejich souřadnice jsou transformovány z 3D prostoru na 2D zobrazovací plochu. Pokud jsou použity textury, tak se v této fázi určují koordináty textur a jejich transformace. Při použití světla se nyní každému bodu přiřazuje konkrétní barva v závislosti na definované barvě objektů, světelných zdrojích, normálovému vektoru plochy a použitému světelném modelu. Nakonec se v této fázi provádí ořezávání na rozměry projekčního okna.

### 3.1.4 Rasterizace

Ve fázi rasterizace se spojují geometrická a pixel data do fragmentů. Z jednotlivých vrcholů se tvoří úsečky a vyplňují mnohoúhelníky a to vše s ohledem na nastavení OpenGL jako například tloušťka čar, velikost bodů, stínovací model atd.

### 3.1.5 Fragmentové operace

Fragmentu odpovídá jeden pixel výsledného obrázku, který nese navíc informace o barvě, hloubce, popřípadě obsahuje i souřadnice textury. Fragmenty lze dále zpracovávat a docílit tak různých efektů. Můžeme například chtít vykreslovat mnohoúhelníky s vyhlazenými hranami (antialiasing), ovlivňovat průhlednost nebo míchat barvy (blending). Patří sem celá množina efektů, které ovlivňují barvu bodu v závislosti na jeho vzdálenosti od pozorovatele. Tak lze docílit efektu mlhy a jiných atmosférických vlivů.

## 3.2 Syntaxe OpenGL funkcí

Funkce OpenGL používají prefix *gl* a počáteční velká písmena od každého slova tvořící název příkazu (např. *glClearColor*). Předpona funkce udává knihovnu, ze které pochází, tak jako „gl" označuje základní funkci OpenGL, „glu" potom funkci z OpenGL Utility Library (např. *gluLookAt*), apod. Podobně OpenGL definuje i konstanty, které začínají prefixem *GL*, všechna písmena velká a znak podtržítka („_") pro oddělení slov (např. *GL_COLOR_BUFFER_BIT*).

V OpenGL existují skupiny funkcí, které navenek dělají tutéž činnost a liší se pouze počtem a typem argumentů. Tyto funkce jsou od sebe odlišeny příponou, která může mít až čtyři znaky. Prvním znakem je číslice, která udává počet vstupních hodnot funkce. Tyto hodnoty mohou být předány buď přímo příslušným počtem argumentů ve volání funkce nebo je parametrem odkaz na vektor s daným počtem hodnot a v tom případě se na konci přípony objevuje znak „v". Druhý znak nebo dvojice znaků udává datový typ hodnot.

Jména funkcí OpenGL se tedy skládají z:

- prefix (označuje knihovnu ze které daná funkce pochází)
- vlastní jméno funkce (vyjadřuje činnost funkce)
- sufix (určuje počet a typ parametrů)

V tabulce 1 jsou uvedeny možné přípony funkcí, k nim odpovídající datový typ parametrů a jejich definice v knihovně OpenGL.

| Sufix | Datový typ | OpenGL definice |
|---|---|---|
| **b** | 8-bit integer | GLbyte |
| **s** | 16-bit integer | GLshort |
| **i** | 32-bit integer | GLint, GLsizei |
| **f** | 32-bit floating-point | GLfloat, GLclampf |
| **d** | 64-bit floating-point | GLdouble, GLclampd |
| **ub** | 8-bit unsigned integer | GLubyte, GLboolean |
| **us** | 16-bit unsigned integer | Glushort |
| **ud** | 32-bit unsigned integer | GLuint, GLenum, GLbitfield |

*Tabulka 1 - Datové typy OpenGL*

Jako příklad lze uvést funkci `glVertex*`, která definuje vrchol v trojrozměrném prostoru. Funkce existuje v mnoha modifikacích, mimo jiné:

`glVertex`***3f****(x, y, z)*

(v tomto případě jsou parametry předány přímo ve volání funkce a jsou jimi tři hodnoty typu GLfloat)

`glVertex`***2sv****(v[2])*

(dvě souřadnice x, y jsou zadány nepřímo pomocí odkazu na vektor se dvěmi hodnotami typu GLshort)

`glVertex`***4d****(x, y, z, w);*

(čtyři souřadnice ve formátu GLdouble)

# 4 Geometrie

## 4.1 Definice primitiv

Jak už bylo řečeno, základními stavebními prvky, s nimiž OpenGL pracuje a vytváří scénu jsou body, čáry a mnohoúhelníky. Všechny tyto útvary se definují pomocí vrcholů.

Homogenní souřadnice jsou výhodnou reprezentací, protože zjednodušují výpočty spojené s různými lineárními transformacemi scény (otočení, posunutí). Bod v ***n***-rozměrném prostoru je zde popsán ***n+1*** souřadnicemi. Souřadný systém se nazývá homogenní (stejnorodý), protože jakýkoliv násobek vektoru (***x***, ***y***, ***z***, ***w***) různý od nuly udává stále stejný bod v třírozměrném prostoru.

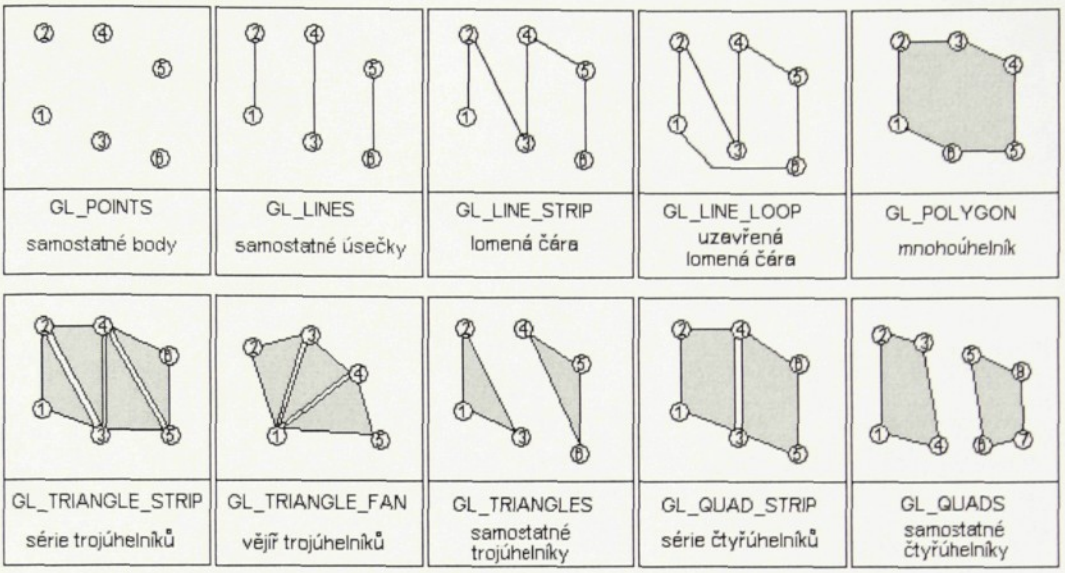
Mezi kartézskými a homogenními souřadnicemi platí jednoduchá převodní pravidla:

$$(x,y,z) \Rightarrow (x,y,z,1) \qquad (x,y,z,w) \Rightarrow \left(\frac{x}{w},\frac{y}{w},\frac{z}{w}\right)$$

Vrcholy mohou definovat bod v trojrozměrném prostoru udáním dvou, tří nebo čtyř souřadnic. V případech kdy určujeme jinak než homogenními souřadnicemi OpenGL si je převede právě do homogenních, které používá jako vnitřní reprezentaci.

Pokud definuji bod třemi souřadnicemi, čtvrtá souřadnice ***w*** se nastaví na implicitní hodnotu 1. V případě určení bodu pouze dvěmi souřadnicemi jsou implicitní hodnoty ***z*** rovno 0 a ***w*** 1.

Jednotlivé elementární objekty se v OpenGL definují v části kódu uvozené funkcemi *glBegin* a *glEnd*. Při definici elementárního objektu funkce *glBegin()* očekává jako argument typ nově vytvářeného objektu. Podstatné je, jak se mají chápat následující definice vrcholů. Na obrázku 2 je znázorněno všech deset možností, jakými se dají objekty zadávat.

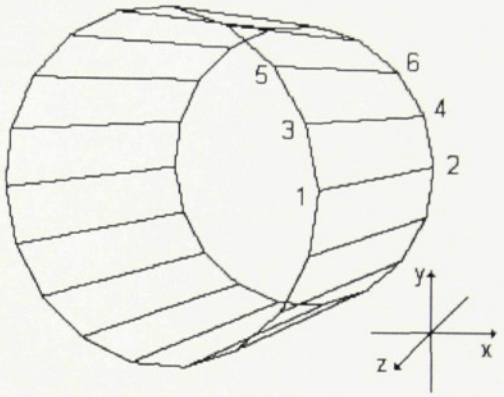



*Obrázek 2 – módy zadávání primitiv*

Příklad č. 1:

```
glBegin(GL_QUAD_STRIP);
     for cast_plaste:=0 to 18 do begin
          alfa:= cast_plaste*20*pi/180;
          x:=cos(alfa);
          y:=sin(alfa);
          glVertex3f(x,y,1);
          glVertex3f(x,y,-1);
     end;
glEnd;
```

Tento jednoduchý příklad vykreslí plášť válce, který se skládá z 18-ti čtyřúhelníků. Postup zadávání jednotlivých vrcholů je znázorněn na obrázku 3. Přední strany čtyřúhelníků při tomto postupu zadávání vrcholů budou ty na vnější straně pláště válce. Zadní strany pak uvnitř pláště.




*Obrázek 3 – postup zadávání vrcholů v příkladě 1*

Když definujeme vrcholy mnohoúhelníků nebo čtyřúhelníků (quads) je třeba zaručit, že nově definovaný objekt bude rovinný. Další problém nastává tehdy, když zadáváme mnohoúhelník, který není konvexní. Takový mnohoúhelník se musí rozdělit na menší konvexní trojúhelníky, protože OpenGL přímo pracuje pouze s konvexními.

U mnohoúhelníků OpenGL rozlišuje přední a zadní stěnu. Díky tomu je možné přední strany mnohoúhelníků vykreslovat odlišně od zadních stran a tak je odlišit. Rozlišení přední a zadní stěny závisí na tom jak jsou jednotlivé vrcholy zadávány. Implicitně je strana mnohoúhelníků, jejichž vrcholy byly zadány proti směru hodinových ručiček, přední a ta druhá strana pak zadní. Je ale také možnost nastavit pro rozlišení přední a zadní strany opačný postup.

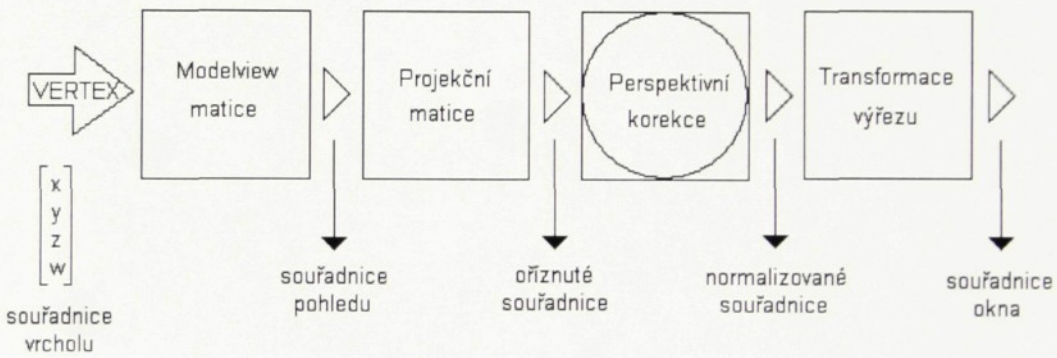
## 4.2 Transformace

Všechny transformace v OpenGL se dějí násobením dat vrcholů transformačními maticemi. Protože knihovna OpenGL používá pro výpočet homogenní čtyřrozměrné souřadnice, jsou pro všechny transformace používány matice 4x4.

V OpenGL je několik typů transformací:

- modelová transformace a transformace souřadného systému
- projekční transformace
- transformace výřezu

Místo změn souřadnic všech jednotlivých vrcholů se mění pouze potřebné matice a tak můžeme jednoduše měnit například polohu a natočení jednotlivých objektů vzhledem k souřadnému systému, místo ze kterého se budeme na scénu dívat, úhel pohledu a v neposlední řadě i způsob projekce 3D scény na obrazovku. Všechny tyto transformace se dějí pomocí násobení maticemi. OpenGL obsahuje funkce na některé základní transformace (posunutí, natočení, zkosení, změna měřítka) to znamená použití předdefinovaných matic, které tyto transformace provádějí. Je ale také možnost provést svou vlastní transformaci a to definováním hodnot transformační matice. Postup transformace počátečních zadaných vrcholů až k souřadnicím bodů na obrazovce je znázorněn na obrázku 4.




*Obrázek 4 – postup transformace vrcholů*

### 4.2.1 Modelová transformace a transformace souřadného systému

Tyto dva typy transformací spolu úzce souvisí. Liší se pouze tím, co si pod nimi představujeme. Modelovými transformacemi můžeme posouvat, natáčet a měnit měřítko jednotlivých modelů. Transformacemi souřadného systému naopak posouváme a natáčíme samotný souřadný systém. Je to, jako bychom posouvali a natáčeli kameru, kterou se díváme na scénu. Když posuneme model od místa pohledu, je výsledek stejný jako kdybychom posunuli kameru opačným směrem. Proto se také tyto dva typy transformací spojují v jednu, tzv. „Modelview matrix".

Protože není žádný principiální rozdíl mezi modelovou a zobrazovací transformací, podstatné je pořadí provádění geometrických transformací. Například posloupnost příkazů posunutí, rotace a vykreslení dává jiný výsledek, než rotace,

posunutí a vykreslení. Když se komponuje složitá dynamická scéna, je dobré postupovat od lokálních transformací ke globálním.

### 4.2.2 Projekční transformace

Projekční transformace se již týkají promítání scény do okna a převádí 3D scénu na 2D plochu. To jak se bude scéna mapovat do roviny určuje matice projekčního zobrazení. OpenGL nabízí dva vestavěné nejdůležitější druhy projekce – kolmou (ortogonální) a perspektivní projekci.

Při kolmé projekci nedochází ke zkreslení délek a rovnoběžky zůstávají i po aplikaci transformace rovnoběžné. Toto zobrazení však neodpovídá fyzikálnímu modelu. Jako transformační matice je použita jednotková matice. Kolmá (ortogonální) projekce nalézá uplatnění v různých technických aplikacích a CAD programech, kde je důležitější přesné zobrazení proporcí než realistický pohled. Oproti tomu perspektivní projekce jak napovídá název dodává zobrazení perspektivitu.

### 4.2.3 Transformace výřezu

Spolu s projekční transformací určují, jak bude scéna zobrazena na obrazovce. Projekční transformace specifikuje jak vypadá a transformace výřezu určuje, na jakém místě obrazovky se objeví.

## 4.3 Zásobník matic

Zásobník matic je užitečný při konstruování hierarchických modelů kde jsou složitější modely konstruovány z jednodušších. Když například vykreslujeme scénu a chceme aplikovat určitou transformaci pouze na jediný objekt a ne všechny co budou následovat, zásobník matic nám k tomu dává jednoduché řešení. Před prováděním transformací na daný objekt si stávající matici uložíme do zásobníku pomocí funkce *glPushMatrix()*. Teď můžeme provádět libovolné transformace a definovat objekty na

které se budou vztahovat. Poté si původní matici ze zásobníku vyzvedneme funkcí *glPopMatrix()* a všechny transformace provedené mezi těmito dvěmi voláními se „zapomenou" a nebudou se aplikovat na ostatní objekty.

V knihovně OpenGL je zásobník pro minimálně 32 „modelview" matic a alespoň pro 2 projekční matice.

# 5 Barvy a stínování

## 5.1 Vnímání barev

Z fyzikálního hlediska je světlo tvořeno fotony různých vlnových délek. Lidské oko vnímá barvu když určité útvary na sítnici, zvané tyčinky, reagují na dopadající fotony. Jsou tři druhy tyčinek a liší se tím, na které světlo reagují nejlépe. Jeden typ reaguje nejlépe na červené světlo, druhý na zelené a třetí na modré světlo. Další jiné barvy jsou pak jen tvořeny smícháním různých stupňů intenzity těchto tří složek. Na tomto principu funguje i barevná obrazovka. Každý pixel na ní emituje různé množství červeného, zeleného a modrého světla. Tyto složky se nazývají R, G a B hodnoty. Na tomto principu také pracuje barevná obrazovka. K zobrazení určité barvy monitor posílá každému pixelu na obrazovce červené, zelené a modré světlo různých intenzit.

## 5.2 Color-index mód

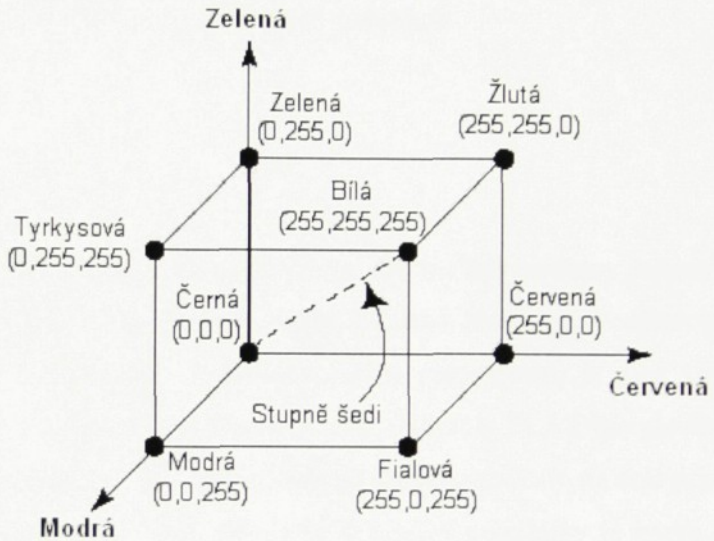
V color-index módu je barva pixelu udána indexem, který odkazuje na hodnoty RGB uložené v tabulce (paleta barev). Protože tyto tabulky má na starost operační systém, OpenGL neobsahuje funkce na práci s touto tabulkou. Měnit barvy v tabulce lze přes pomocnou knihovnu GLUT. V tomto módu počet současně zobrazitelných barev je limitován velikostí palety barev. Velikost palety je vždy mocnina 2 a typické velikosti jsou od 256 ($2^8$) do 4096 ($2^{12}$) různých barev. V RGBA módu je každý pixel nezávislý na ostatních, ale v color-index módu pixely se stejným indexem mají tutéž barvu a při změně barvy s tímto indexem se změní i barva všech těchto bodů. Tento mód se moc nevyužívá a proto se v dalším textu zaměřím pouze na RGBA mód.

## 5.3 RGBA mód

OpenGL používá pro výpočty reprezentaci barev v barevném modelu RGBA. RGB složka se skládá ze tří základních barev: červená – zelená – modrá. Písmeno A je

zkratka pro tzv. „alfa kanál“, který určuje průhlednost barvy. Nelze použít zároveň RGBA a color-index mód.

Protože se k zobrazení barev používají tři složky, můžeme si všechny dostupné barvy modelovat jako trojrozměrné těleso, které se nazývá RGB barevný prostor znázorněn na obrázku 5. RGB složky jsou specifikovány jako **x**, **y** a **z** souřadnice. Na počátku souřadnic (bod 0; 0; 0) kde je intenzita všech složek nulová, je výsledná barva černá.




*Obrázek 5 – RGB barevný prostor*

Na běžném PC je pro uložení RGB složek maximální počet 24 bitů, a tak na každou složku připadá maximálně 8 bitů. Každá jednotlivá složka tak může nabývat až 256-ti různých hodnot (0-255). Nula značí nenižší intenzitu a hodnota 255 pak intenzitu maximální. Tato „kostka barev“ obsahuje všechny možné barvy zobrazitelné na dnešních počítačích.

Pro definování aktuální barvy v RGBA módu je v OpenGL funkce *glColor*()*. Tato funkce existuje v mnoha variantách lišících se počtem a typem parametrů (viz. kapitola 2.2). Barvu určují 3 (RGB) nebo 4 parametry (RGBA). a mohou být použity všechny datové typy OpenGL nebo parametrem může být pointer na pole opět 3 nebo 4 parametrů všech typů. Při definování barvy pouze třemi parametry poslední složka má implicitní hodnotu 1,0.

Pro verzi příkazu *glColor*()*, která jako parametr akceptuje data reálná čísla, by hodnoty měly být v rozsahu mezi 0,0 a 1,0, což je minimální a maximální hodnota pro barevnou informaci která může být uložena do framebufferu. V případech kdy jsou parametry zadávány typem integer bez znaménka se celý číselný rozsah tohoto typu lineárně mapuje do intervalu <0,0; 1,0>. Při použití integer parametrů se znaménkem, se opět celý číselný rozsah lineárně mapuje na interval <-1,0; 1,0>. Ve všech případech se parametry před zápisem do framebufferu oříznou na rozsah <0;1> tzn. že vyšším hodnotám se přiřadí hodnota 1 a nižším pak hodnota 0.

## 5.4 Stínování

Úsečky nebo vyplněné mnohoúhelníky mohou být kresleny jednou barvou (flat shading) nebo plynulým přechodem barev (Gourad shading). Požadovaný stínovací model se nastavuje příkazem *glShadeModel()* a parametrem je buď GL_SMOOTH (implicitní nastavení) pro Gouraudovo stínování nebo GL_FLAT pro ploché stínování.

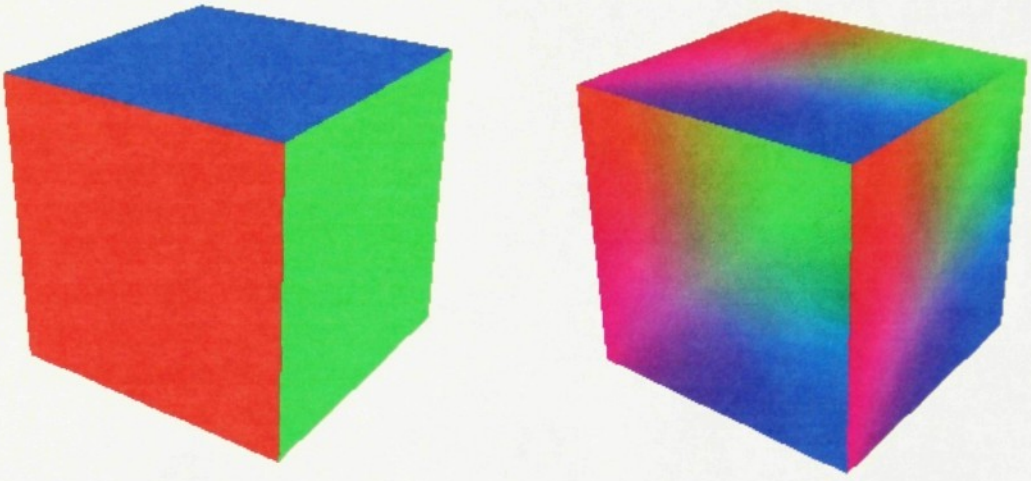
Při plochém stínování se barva jednoho vrcholu aplikuje na celé primitivum. Pro úsečky je barva dána koncovým vrcholem. Pro mnohoúhelníky je barva dána barvou která je nastavena v okamžiku zadávání určitého vrcholu. Který vrchol je ten rozhodující ukazuje tabulka 2. Vrcholy v tabulce jsou počítány od 1.

| **Typ mnohoúhelníku (viz. kapitola 3.1)** | **Vrchol použitý k určení barvy i-tého mnohoúhelníku** |
|---|---|
| samostatný mnohoúhelník | 1 |
| série trojúhelníků | i+2 |
| vějíř trojúhelníků | i+2 |
| samostatné trojúhelníky | 3i |
| série čtyřúhelníků | 2i+2 |
| samostatné čtyřúhelníky | 4i |

*Tabulka 2 – výběr barvy při plochém stínování*

U Gouraudova stínování se barva každého vrcholu bere nezávisle. Barva mezi vrcholy je u mnohoúhelníků bilineárně interpolována podle plošné polohy a u úseček je

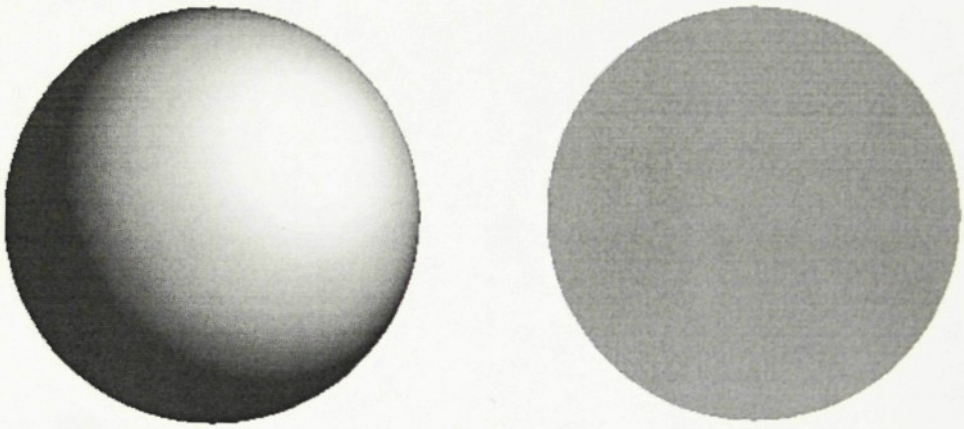
to lineární interpolace podle polohy mezi koncovými body. Vnitřním bodům mnohoúhelníku a úsečky se tak přiřadí barva smíchaná z barev vrcholů v poměru jejich vzdáleností od jednotlivých vrcholů. Výsledkem toho všeho jsou hladké barevné přechody viz. obrázek 6.



*Obrázek 6 - krychle vykreslená s plochým stínováním (vlevo) a Gouraudovým stínováním s interpolací barvy*

# 6 Světlo a materiály

Pokud chceme vykreslovat alespoň trochu realisticky vyhlížející scény, je třeba vytvořit osvětlenou scénu. K tomu je potřeba povolit výpočet osvětlení, nadefinovat světla, spustit je a pro každý povrch je nutno určit vlastnosti materiálu. Osvětlení scény také jednoduše pomáhá vnímat trojrozměrnost scény jak je vidět na obrázku 7 kde je zobrazena stejná scéna jen v jednom případě (levý objekt) se zapnutým a v druhém případě vypnutým světlem.



*Obrázek 7 – osvětlená (levá) a neosvětlená koule*

OpenGL aproximuje světlo tak, jako by mohlo být rozděleno na R, G a B složky. Proto je každý světelný zdroj charakterizován množstvím červeného, zeleného a modrého světla které vyzařuje. Povrchy materiálů jsou pak charakterizovány tím, kolik procent dopadajícího červeného, zeleného a modrého světla odrážejí do různých směrů. Zbytek světla je pak pohlcen povrchem. Rovnice šíření světla v OpenGL ne zcela odpovídají realitě, ale výpočty s nimi spjaté se dají realizovat velice rychle.

V OpenGL osvětlovacím modelu světlo ve scéně pochází z různých světelných zdrojů, které mohou být jednotlivě zapínány a vypínány. Některé světlo přichází z určité pozice a pod určitým směrem, jiné je naopak rovnoměrné po celé scéně a nedá se určit odkud přichází.

Světelné zdroje v OpenGL nemají žádný efekt dokud nedefinujeme ve scéně povrchy, které mohou absorbovat a odrážet světlo. Každý povrch může odrazit část dopadajícího světla do všech směrů a jinou část zas jen do určitého směru jako zrcadlo

nebo jiné lesklé povrchy. Může také vyzařovat své vlastní světlo, které se ale nepočítá jako světelný zdroj.

## 6.1 Složky světla a materiálů

OpenGL osvětlovací model dělí barevné vlastnosti světel a povrchů materiálů do čtyř nezávislých komponent. Všechny tyto komponenty jsou počítány nezávisle a potom složeny dohromady ve výslednou barvu. Tyto složky jsou:

- ambientní složka (ambient)
- rozptýlená složka (diffuse)
- směrová složka (specular)
- emisní složka (emission)

### 6.1.1 Ambientní složka (ambient)

Ambientní složka světla je „všudypřítomná složka". Pochází z určitého zdroje světla, ale je tak rozptýlená že nelze určit jeho směr. Je to jakoby přicházela ze všech směrů. Objekty jsou osvětlené tímto světlem stejně na všech místech a ve všech směrech.

### 6.1.2 Rozptýlená složka (diffuse)

Difusní složka světla přichází z konkrétního zdroje. Objekty blíž zdroji jsou osvětleny více a intenzita světla klesá se čtvercem vzdálenosti. Když se srazí se stěnou tělesa, rozptýlí se rovnoměrně do všech směrů.

### 6.1.3 Směrová složka (specular)

Přímé bodové osvětlení přichází z jednoho zdroje, podobně jako rozptýlené světlo. Na povrchu tělesa se odráží pod příslušným úhlem. Ideální zrcadlo odráží ze 100% právě směrové světlo díky dokonalé reflexi.

### 6.1.4 Emisní složka (emission)

Tato složka se definuje pouze u vlastností materiálů a ne u vlastností světel. Je to dáno podstatou této složky. Specifikováním této složky se povrch s touto vlastností jeví jakoby vyzařoval své vlastní světlo dané barvy. Tato složka vlastnosti materiálu není ovlivněna žádným zdrojem světla ani neovlivňuje své okolí. Je pouze připočtena k ostatním složkám.

Používá se chceme-li zobrazit zdroj světla (např. lampa apod.). Umístěním světla a objektu (např. malá koule jako žárovka), jehož emisní složka je totožná s barvou světla, do stejné pozice ve scéně docílíme „zviditelnění" světelného zdroje. To znamená že v místě odkud pochází určité světlo je objekt který vyzařuje barvu shodnou s barvou světla tohoto světelného zdroje.

## 6.2 Definice světel

Světelné zdroje mají mnoho vlastností jako například barevné vlastnosti složek, pozice nebo směr. Tyto vlastnosti se nastavují voláním funkce *glLight*()*, která má tři parametry.

*glLight*(light, pname, value)*

**light** - určuje světlo jehož vlastnosti chceme měnit (GL_LIGHT0, GL_LIGHT1 atd.)

**pname** - vlastnost, kterou chceme nastavit

- GL_AMBIENT (ambientní složka světla)
- GL_DIFFUSE (rozptýlená složka)

- GL_SPECULAR (směrová složka)
- GL_POSITION (typ světla a v závislosti na typu buď pozici nebo směr)
- GL_SPOT_DIRECTION (směr pozičního světla)
- GL_SPOT_EXPONENT (určuje změnu intenzity světla ve světelném kuželu)
- GL_SPOT_CUTTOFF (velikost světelného kužele, zadává se pouze poloviční úhel, úhel mezi osou kužele a hranou)
- GL_CONSTATNT_ATTENUATION (konstantní útlum)
- GL_LINEAR_ ATTENUATION (lineární útlum)
- GL_QUADRIC_ ATTENUATION (kvadratický útlum)

**value** – vektor hodnot přiřazovaných dané vlastnosti

Barva světelného zdroje se nastavuje pomocí ambientní, rozptýlené a směrové složky světla. Parametrem je RGBA hodnota odpovídající intenzitě složky světla daného světelného zdroje.

Pomocí vlastnosti GL_POSITION určuje typ světla ve scéně. Parametrem je vektor čtyř hodnot. Čtvrtá hodnota určuje typ světla. Je-li čtvrtá hodnota vektoru rovna nule, definujeme směrové světlo (např. slunce) a pak první tři hodnoty vektoru určují směr šíření tohoto světla. Paprsky světla jsou počítány jakoby dopadaly rovnoběžně na celou scénu. Ve všech ostatních případech kdy je čtvrtá hodnota vektoru nenulová je daný světelný zdroj zdrojem pozičního světla (např. lampa). V tomto případě první tři hodnoty definují pozici světla. Implicitně i toto světlo svítí do všech směrů, ale lze u něj definovat směr, velikost světelného kužele a jeho intenzitu.

V reálném světě intenzita světla klesá se vzrůstající vzdáleností od zdroje světla. Jelikož je směrové světlo bráno jako by bylo umístěno nekonečně daleko, není u tohoto typu útlumový faktor povolen. Tento útlumový faktor lze použít pouze u pozičního světla a je počítán následovně:

$$faktor = \frac{1}{k_c + k_l d + k_q d^2}$$

d ... vzdálenost počítaného vrcholu od pozice světla

$k_c$ ... GL_CONSTANT_ATTENUATION

$k_l$ ... GL_LINEAR_ ATTENUATION

$k_q$ ... GL_QUADRIC_ ATTENUATION

Implicitní nastavení $k_c$ je 1 a $k_l$, $k_q$ rovno 0, čili nedochází k útlumu světla se vzdáleností.

Aby bylo světlo do scény přidáno, musíme nejprve povolit výpočet světel funkcí *glEnable(GL_ LIGHTING)* a zapnout jednotlivá nadefinovaná světla pomocí funkce *glEnable(GL_LIGHTi)* kde **i** určuje číslo světelného zdroje. Každá implementace OpenGL umožňuje vytvořit minimálně 8 světelných zdrojů.

Příklad č. 2:

```
const LAmbient:array [0..3] of GLFloat=(1,1,1,1);
      LDiffuse:array [0..3] of GLFloat=(1,1,1,1);
      LSpecular:array [0..3] of GLFloat=(1,1,1,1);
      LPos:array [0..3] of GLFloat=(0,0,3,1);
      LDir:array [0..2] of GLFloat=(0,0,-1);
begin
  glLightfv(GL_LIGHT0, GL_AMBIENT, @LAmbient);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, @LDiffuse);
  glLightfv(GL_LIGHT0, GL_SPECULAR, @LSpecular);
  glLightfv(GL_LIGHT0, GL_POSITION, @LPos);
  glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, @LDir);
  glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 30);
end;
```

Tento příklad demonstruje definici světla ve scéně. V tomto případě se jedná o poziční bílé světlo s maximální intenzitou (hodnoty R, G a B u všech složek rovny 1) umístěném na pozici (0; 0; 3) a svítícího směrem záporné části osy **z**. Velikost světelného kužele je 60°.

## 6.3 Definice vlastností materiálu

Nastavení barevných vlastností materiálu je podobná jako nastavení barevných vlastností světla. V případě materiálů se vlastnosti definují pomocí příkazu *glMaterial*()* opět se třemi parametry.

*glMaterial*(face, pname, value)*

**face** – strana mnohoúhelníků pro kterou vlastnost definujeme

- GL_FRONT (přední strana mnohoúhelníků)
- GL_BACK (zadní strana mnohoúhelníků)
- GL_FRONT_AND_BACK (obě strany)

**pname** - udává vlastnost, kterou chceme měnit

- GL_AMBIENT (ambientní složka)
- GL_DIFFUSE (rozptýlená složka)
- GL_AMBIENT_AND_DIFFUSE (obě složky zároveň a budou obsahovat stejné hodnoty)
- GL_SPECULAR (směrová složka)
- GL_SHININESS (velikost a jas směrové složky)
- GL_EMISION (emisní složka)

**value** – ukazatel na vektor hodnot dané vlastnosti

V případě barevných vlastností materiálů hodnota RGBA značí procentuelní množství odrážené složky světla. To znamená, že pokud na povrch dopadá rozptýlená složka světla s RGBA hodnotou (1, 1, 1, 1) a povrch má stejnou složku nastavenou na hodnotu (1; 0,5; 0,2; 1) výsledná barva v rozptýlené složce bude (1*1; 1*0,5; 1*0,2; 1*1) = (1; 0,5; 0,2; 1) (pokud nebudeme brát v úvahu další světelné zdroje a faktory).

Pokud nijak nedefinujeme barevné vlastnosti materiálů tak jsou všechny povrchy vykresleny se základním nastavením viz. tabulka 3.

| Parametr | Implicitní hodnota |
|---|---|
| GL_AMBIENT | (0,2; 0,2; 0,2; 1,0) |
| GL_DIFFUSE | (0,8; 0,8; 0,8; 1,0) |
| GL_SPECULAR | (0,0; 0,0; 0,0; 1,0) |
| GL_SHININESS | 0,0 |
| GL_EMISSION | (0,0; 0,0; 0,0; 1,0) |

*Tabulka 3 – základní nastavení vlastností materiálu*

Příklad č. 3:

```
const
  Am_MATERIAL:array [0..3] of GLFloat=(0.333,0.333,0.169,1.000);
  Di_MATERIAL:array [0..3] of GLFloat=(0.608,0.608,0.000,1.000);
  Sp_MATERIAL:array [0..3] of GLFloat=(0.251,0.502,0.502,1.000);
  Em_MATERIAL:array [0..3] of GLFloat=(0.000,0.000,0.000,1.000);
begin
  glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, @Am_MATERIAL);
  glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, @Di_MATERIAL);
  glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, @Sp_MATERIAL);
  glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, @Em_MATERIAL);
  glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 13);
end;
```

V tomto příkladě nastavíme všechny barevné vlastnosti materiálu na hodnoty zadané v příslušných vektorech. Barevné vlastnosti materiálu lze zadávat jen pomocí odkazu na vektor hodnot. Jediný parametr, který může být zadán přímo nebo pomocí odkazu, je parametr GL_SHININESS. V tomto případě je zadáván přímou hodnotou.

Alternativním způsobem nastavování vlastností materiálů je možnost měnit jednotlivé složky materiálu pomocí změny barvy funkcí *glColor*()*. Tento způsob se ale nejdříve musíme povolit zavolaním funkce *glEnable(GL_COLOR_MATERIAL)*. A potom nastavit které vlastnosti materiálů a kterých stran mnohoúhelníků se změny prováděné příkazem *glColor*()* budou týkat. K tomu slouží funkce *glColorMaterial()* se dvěma parametry. První určuje strany mnohoúhelníků a druhý vlastnost materiálu. Potom každým nastavením barvy se příslušně změní definovaná složka vlastností

materiálu. Tento způsob je výhodný pokud měníme jen jednu vlastnost, jinak je lepší používat funkci *glMaterial*()*.

## 6.4 Definice normálových vektorů

Pro výpočet odrazu světla na objektu je důležité znát normálové vektory jeho vrcholů, které nám určují orientaci plochy vůči světlu. Informace o normálovém vektoru se sdružuje s jednotlivými vrcholy mnohoúhelníků.

Pokud ve svém programu použijeme knihovní funkce GLU, které vykreslují předdefinované objekty, jako kouli, válec nebo prstenec, jsou už většinou normálové vektory definovány. U nově vytvářených objektů je třeba tyto informace dodat. S OpenGL je možné nastavit normálový vektor ke každému vrcholu mnohoúhelníku. Nejjednodušší případ je, když vrcholy mnohoúhelníku mají stejný normálový vektor. Pokud ale určíme normálový vektor ke každému vrcholu zvlášť, můžeme dodat plošce optické vlastnosti zakřiveného povrchu. Vnitřním bodům mnohoúhelníků se při rasterizaci přiřadí příslušné mezihodnoty.

U normálového vektoru je podstatný jeho směr. Pro výpočet osvětlených scén je potřeba aby normálové vektory byly v tzv. normalizovaném tvaru, kdy je jejich délka rovna jedné. Normálové vektory zůstávají v normalizovaném tvaru i po aplikaci některých modelovacích transformací, třeba posunutí a otočení. Pokud se ale provede jiná transformace, jako změna měřítka nebo zkosení, je třeba normálové vektory znovu normalizovat. OpenGL může provádět automatickou normalizaci normálových vektorů po aplikaci modelovací transformace za cenu zvýšení výpočetních nároků. Automatická normalizace lze povolit funkcí *glEnable(GL_NORMALIZE)*.

# 7 Míchání barev (alpha blending)

Další faktor který má vliv na barvu povrchu plochy je „alfa kanál". Při technice míchání barev se právě využívá tato informace. Bez využití techniky míchání barev každý fragment, který je zapsán do framebufferu, přepíše jakoukoliv informaci o barvě tam již uloženou. Naopak s využitím míchání barev můžeme určit jak a jak moc má být

barva již uložená zkombinována s novou barvou. Tak můžeme vytvořit průhledné barvy a skrze objekty s těmito barvami jsou pak vidět i objekty ležící za nimi.

# 8 Texturování

Texturování nám dovoluje na geometrická primitiva nanášet textury (obrázky). Můžeme například na mnohoúhelník nanést obrázek cihlové zdi a vykreslit celou zeď jako jeden mnohoúhelník. Texturování zajistí že se textura na daném polygonu bude správně spolu s objektem transformovat a vykreslovat.

Textury lze nanášet na plochy tvořené několika mnohoúhelníky nebo na zakřivené plochy. Textura se může opakovat oběma směry tak, aby pokryla celou plochu, může být dokonce jednodimenzionální. Užitím texturování lze docílit zrcadlových efektů. Textury mohou být nanášeny různým způsobem. Nanášeny přímo, mohou měnit barvu kterou je objekt vykreslován nebo lze míchat barvu textury s barvou plochy.

# 9 Editor barevných vlastností

Barevné vlastnosti povrchu jsou závislé zejména na vlastnostech materiálu (viz. kapitola 6.3) a vlastnostech světla (viz. kapitola 6.2). Výsledkem práce je aplikace která byla vyvíjena jako demonstrační program pro výuku OpenGL na Technické Univerzitě v Liberci a jako pomůcka pro vývojáře osvětlených scén. Protože výuka probíhá ve vývojovém prostředí Borland Delphi byla aplikace v tomto prostředí také vyvíjena.

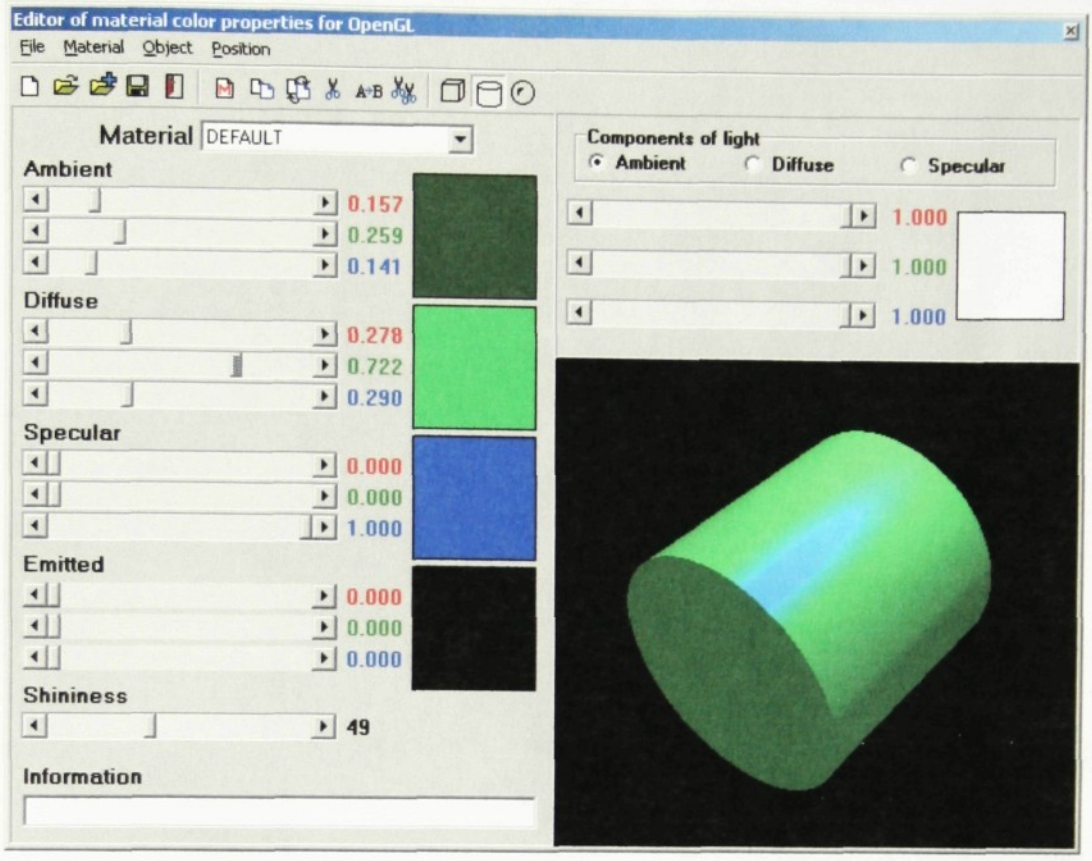
Program názorně demonstruje závislost barevných vlastností materiálů a světelného zdroje na výsledné barvě objektu. Umožňuje:

- zobrazení různých objektů
- specifikace vlastností materiálu
- specifikace barevných vlastností světla
- práce s více materiály
- uložení výsledku do souboru
- načtení definic materiálů ze souboru

Textové popisky, zprávy i menu jsou v anglickém jazyce. Tento jazyk jsem volil z důvodu, že aplikace byla pojata i jako volně šiřitelný software. Přestože prvotní záměr je pro výuku, není anglický jazyk překážkou také proto, že studenti při výuce požívají vývojová prostředí s anglickým prostředím.

## 9.1 Celkový náhled

Na obrázku 8 je zobrazeno hlavní okno aplikace. Je rozděleno na několik částí. V levé části se nachází panel pro editaci barevných vlastností materiálů. V pravé horní části je možno měnit barevné vlastnosti pro světelný zdroj, který osvětluje model umístěný v pravé dolní části. Okno aplikace samozřejmě obsahuje i menu a také nástrojovou lištu.

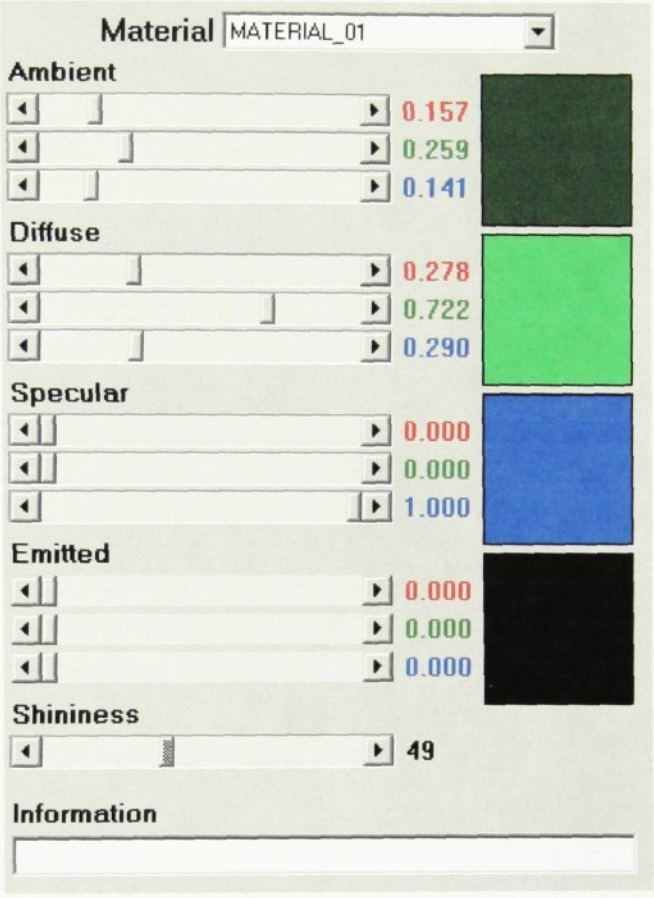



*Obrázek 8 – hlavní okno aplikace*

## 9.2 Panel pro editaci materiálu

V horní části panelu pro editaci materiálu je seznam všech dostupných materiálů. Pod seznamem pro každou ze čtyř složek barevných vlastností materiálu jsou k dispozici tři posuvníky pro příslušné R, G a B složky barvy. Vedle posuvníků se zobrazuje aktuální hodnota zobrazena na tři desetinná místa.

Ve spodní části je možno nastavit parametr GL_SHININESS pro směrovou složku. A nakonec je zde editační řádek pro případné informace o materiálu, které nemají s vlastností materiálu nic společného a slouží jen pro případné poznámky (např. si zde můžeme zaznamenat pro jaký objekt daný materiál použijeme).

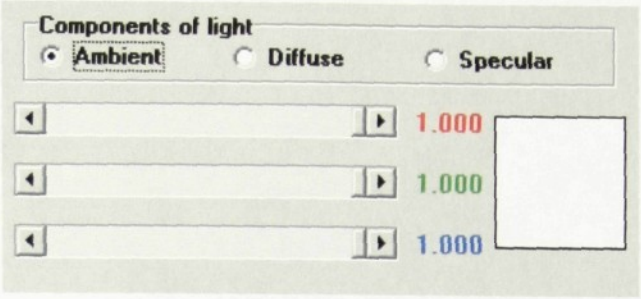



*Obrázek 9 – Panel editace materiálů*

Alternativním způsobem nastavení barvy jsou barevné čtverce vedle posuvníků. Slouží nejen k zobrazení ale i k výběru barvy z palety. Tuto nabídku vyvoláme klepnutím libovolného tlačítka myši nad barevným čtvercem. Objeví se standardní nástroj operačního systému Windows pro výběr a editaci barev. Zároveň barevný čtverec zobrazuje barvu příslušné složky.

## 9.3 Panel pro změnu vlastností světla

Měnit barevné vlastnosti světla je možné obdobným způsobem jako u materiálů. Jediný rozdíl je v tom, že zde nemáme k dispozici posuvníky pro všechny složky, proto je nutné požadovanou složku vybrat v nabídce nad posuvníky.

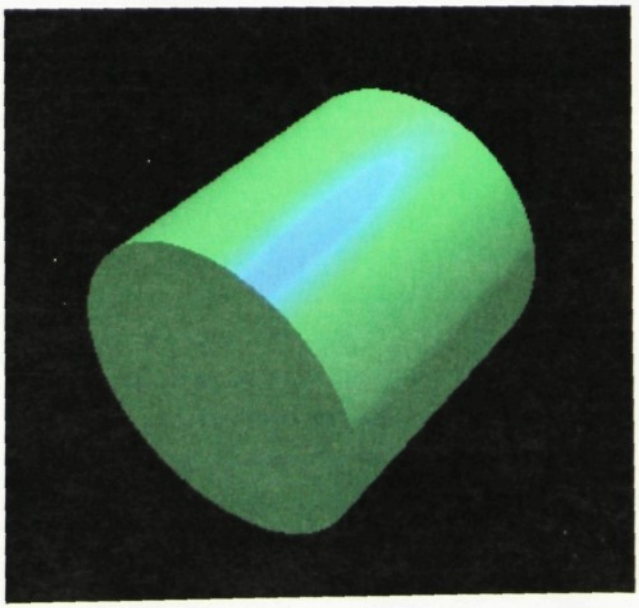



*Obrázek 10 – panel pro změnu vlastností světla*

Při základním nastavení světelného zdroje jsou všechny jeho složky nastaveny na bílou barvu s maximální intenzitou (všechny RGB hodnoty složek jsou rovny 1).

## 9.4 Náhledové pole

V náhledovém poli je zobrazen zvolený objekt jehož barevné vlastnosti povrchu korespondují s barevnými vlastnostmi právě vybraného materiálu. Objektem je možno natáčet pomocí myši. Při současně stisknutém levém tlačítku myši a jejím pohybu se objekt natáčí okolo svého středu. Pohybem do stran se natáčí podél svislé osy a pohybem nahoru a dolu se natáčí podél osy horizontální.



*Obrázek 11 – náhledové pole*

Lze také měnit umístění světelného zdroje. Jeho umístění se mění stejným principem jako u natáčení objektu, pouze je použito pravé tlačítko myši. Světelný zdroj se při změně polohy pohybuje po kulové ploše jejíž střed je totožný se středem otáčení modelu.

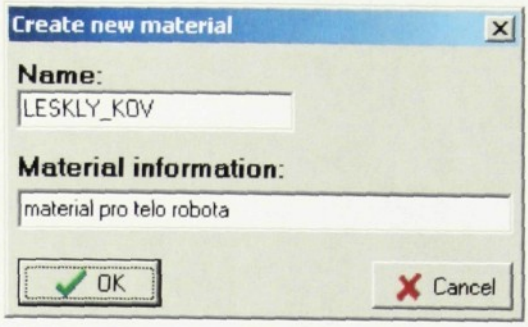
## 9.5 Operace s materiály

S materiály lze provádět řadu základních operací. V uvedeném seznamu jsou v závorkách uváděny možnosti k vyvolání dané funkce. Jednak umístění v menu, případná klávesová zkratka a odpovídající ikona na nástrojové liště.

- vytvoření nového materiálu ( Material / New, Ctrl+N, )
- kopírování materiálu (Material / Copy, Ctrl+C, )
- přiřazení materiálu (Material / Assign, Ctrl+A, )
- přejmenování materiálu (Material / Rename, Ctrl+R, )
- smazání materiálu (Material / Delete, Ctrl+D, )
- hromadné mazání materiálů (Material / Multiple delete, Ctrl+M, )

### 9.5.1 Vytvoření nového materiálu

Při vytváření nového materiálu se nám objeví dialog viz. obrázek 12. Jediné co je třeba pro vytvoření udělat, je zadat jméno materiálu. Jméno může být tvořeno pouze písmeny, číslicemi a podtržítkem. Jiné znaky ani nelze zadat. Toto omezení je dáno tím, že jméno materiálu tvoří část jména procedury v jazyce Object Pascal. Při zadávání jsou malá písmena automaticky převáděna na písmena velká.

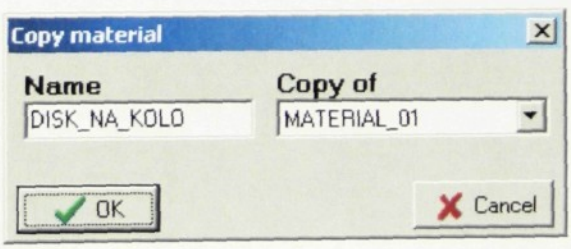



*Obrázek 12 – dialog pro vytvoření nového materiálu*

Pokud již materiál se zadaným jménem existuje, je na to uživatel upozorněn a nový materiál se nevytvoří. Pokud se nám podaří materiál úspěšně vytvořit, jeho barevné složky jsou naplněny implicitními hodnotami pro materiál v OpenGL. Pole s informací o materiálu je nepovinné a lze jej později měnit.

### 9.5.2 Kopírování materiálu

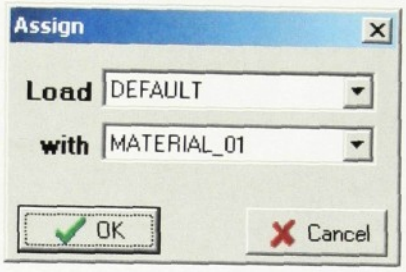
Na obrázku 13 je znázorněn dialog pro kopírování materiálu. Zadání jména materiálu se řídí stejnými pravidly jako u vytváření nového. V seznamu se vybere materiál jehož kopii uživatel vytváří. Vytvoří se nový materiál se zadaným jménem a všechny jeho vlastnosti (barvy, informace) se zkopírují z vybraného materiálu.




*Obrázek 13 – dialog pro kopírování materiálu*

### 9.5.3 Přiřazení materiálu

Při přiřazení jednoho materiálu druhému dojde k naplnění vlastností jednoho materiálu hodnotami jiného materiálu. Tím nám vzniknou dva materiály se stejně definovanými vlastnosti jako při kopírování, ale nevytváří se žádná nová položka.

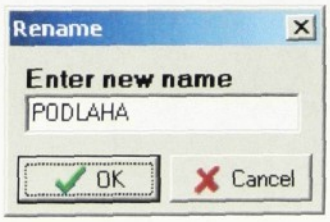



*Obrázek 14 – dialog pro přiřazení materiálů*

V prvním seznamu si vybereme materiál do kterého se zkopírují vlastnosti materiálu vybraného v druhém seznamu.

### 9.5.4 Přejmenování materiálu

Přejmenování materiálu slouží ke změně právě vybraného materiálu. Pravidla pro název materiálu jsou opět totožná jako v případě vytváření a kopírování.




*Obrázek 15 – dialog pro přejmenování materiálu*

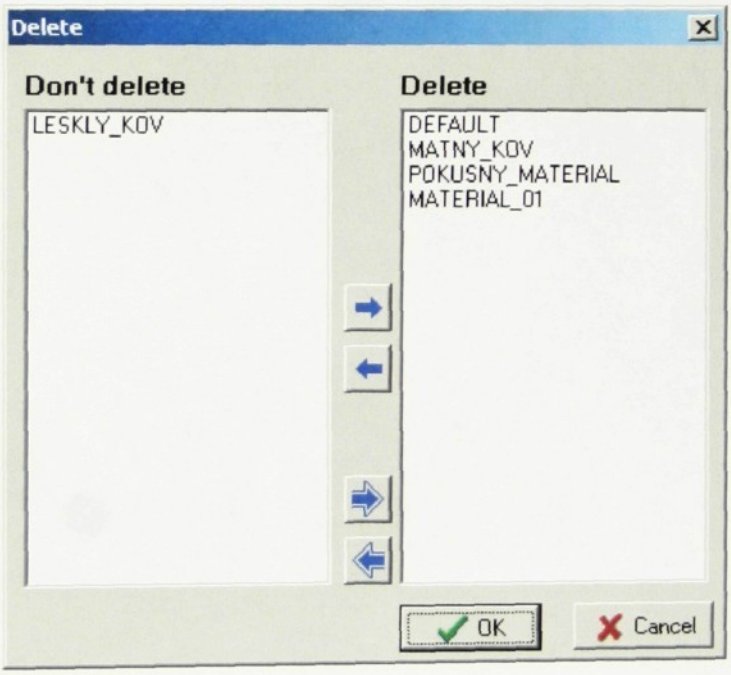
### 9.5.5 Smazání materiálu

Smazání materiálu nám umožňuje odstranit materiál. Vyžaduje potvrzení ke kterému jsme vyzváni dialogovým oknem. Pokud aktivujeme tuto volbu, je odstraněna

definice materiálu, který je pravě vybrán v seznamu na panelu editace vlastností materiálu.

### 9.5.6 Hromadné mazání

Hromadné mazání vyvolá dialog zobrazen na obrázku 18 a umožní nám smazat najednou více definic materiálů. V pravém okně jsou materiály určené ke smazání, v levém okně pak materiály které se mazat nebudou. Zpočátku jsou všechny dostupné materiály v pravém okně určeném ke smazání. Pomocí tlačítek  a  lze vybrané materiály přesouvat z jednoho okna do druhého ve směru šipky. Tlačítka  a  přesouvají všechny materiály na příslušnou stranu ve směru šipek.




*Obrázek 16 – dialog pro hromadné mazání materiálů*

Pokud smažeme všechny materiály, výsledek je stejný jako při použití položky menu File / New ( ) kdy po potvrzení máme opět k dispozici jen materiál DEFAULT.

## 9.6 Objekty a pozice

Barevné vlastnosti materiálu si můžeme prohlédnout na třech různých jednoduchých objektech:

- krychle (Object / Cube, Alt+1, )
- válec (Object / Cylinder, Alt+2, )
- koule (Object / Sphere, Alt+3, )

Tyto objekty jsou voleny z důvodů možnosti prohlédnutí si vlastností materiálu na rovinném, oblém a kulově zakřiveném povrchu.

Protože při otáčení objektem nebo změně pozice světelného zdroje se občas dostaneme do situace kdy při snaze otáčení objektem na jednu stranu se objekt otáčí přesně opačně, jsou zde volby pro nastavení výchozí pozice objektu (Position / Reset object position, Ctrl+O), světelného zdroje (Position / Reset light position, Ctrl+L) nebo obojího současně (Position / Reset both, Ctrl+B).

## 9.7 Ukládaní a načítání ze souboru

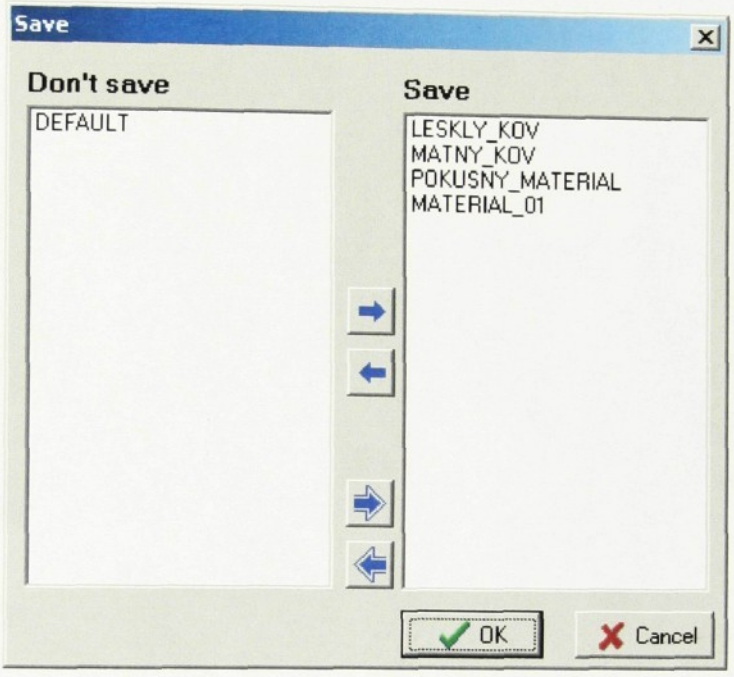
Protože program byl vyvíjen také jako pomůcka při vytváření osvětlených scén, má uživatel možnost si barevné vlastnosti materiálů vytvořené v tomto programu uložit do souboru. Takto uložené definice materiálů umí program zpětně načíst a dál s nimi pracovat. Jelikož byla aplikace psána v prostředí Delphi je výstupním souborem modulová jednotka tohoto programovacího jazyka.

Při ukládání do souboru z menu File / Save, zkratkou Ctrl+S nebo tlačítkem se nám objeví okno téměř totožné s oknem při hromadném mazání. Rozdíl je jen v tom, že zde určujeme které materiály chceme uložit do souboru a které ne.

Při načítání ze souboru přes volbu File / Open, klávesu F3 nebo tlačítko ztrácíme všechny neuložené materiály protože se před načtením nových materiálů ze souboru všechny ostatní mažou.

Další možností jak načíst data ze souboru je volba File / Append (Shift+F3, ), která ponechá všechny materiály a přidá k nim definice materiálů ze souboru. Pokud se

vyskytne materiál se stejným názvem, přidá se na konec jeho jména číslice 2 (např. pokud se v načítaném souboru vyskytuje materiál se jménem KOV a materiál se stejným jménem je již v programu vytvořen pak se načtený materiál bude jmenovat KOV2).




*Obrázek 17 – dialog pro uložení do souboru*

Načítaný soubor musí mít stejnou strukturu jako soubor výstupní. Ze souboru s pozměněnou strukturou (např. po nevhodné editaci souboru uživatelem) se definice materiálu nenačtou.

## 9.8 Výstupní soubor

Výstupní soubor má strukturu modulové jednotky jazyka Borland Pascal (Delphi) a tudíž se dá jednoduše připojit k programu a využívat tak nadefinované materiály. Název procedury Delphi která nastavuje daný materiál je tvořen prefixem *SetMaterial_* a zbytek tvoří název materiálu tak jak byl vytvořen pomocí editoru. Parametry jednotlivých vlastností jsou definovány jako konstanty uvnitř jednotlivých příkazů. Pouze parametr GL_SHININESS je definován přímou hodnotou ve volání

funkce *glMaterialf()*. Vlastnosti materiálů jsou definovány pro přední i zadní strany mnohoúhelníků. Informace k materiálu je uložena jako poznámka.

Příklad č. 4: (formát výstupního souboru)

```
// This file contains procedures to set material color properties
unit pokus;

interface

  procedure SetMaterial_MATERIAL_01;

implementation

uses OpenGL;

procedure SetMaterial_MATERIAL_01;
// pokusny material
const
  Am_MATERIAL_01:array [0..3] of GLFloat=(0.333,0.333,0.169,1.000);
  Di_MATERIAL_01:array [0..3] of GLFloat=(0.608,0.608,0.000,1.000);
  Sp_MATERIAL_01:array [0..3] of GLFloat=(0.251,0.502,0.502,1.000);
  Em_MATERIAL_01:array [0..3] of GLFloat=(0.000,0.000,0.000,1.000);
begin
  glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, @Am_MATERIAL_01);
  glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, @Di_MATERIAL_01);
  glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, @Sp_MATERIAL_01);
  glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, @Em_MATERIAL_01);
  glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 46);
end;

end.
```

# 10 Závěr

Mým úkolem bylo vytvořit aplikaci umožňující editaci barevných vlastnost materiálů pro grafickou knihovnu OpenGL, uložení těchto vlastností ve tvaru vhodnén k dalšímu použití a také umožnit opětovné načtení a editaci souborů touto aplikac vytvořených. K tomu jsem se musel seznámit s knihovnou OpenGL a především s tvorbou osvětlených prostorových scén pomocí této knihovny. Při tvorbě takovýchto scén je pro barvy důležité nastavení barevných vlastností povrchů objektů, které jsou závislé zejména na vlastnostech materiálu a vlastnostech světla. Na základě získaných znalostí byla vytvořena aplikace (Editor), který tuto závislost názorně demonstruje a uživateli ulehčuje práci spojenou s vytvářením vhodných materiálových vlastností. Vytvořené materiály lze uložit do souboru. Výsledné výstupní soubory editoru mají formát modulové knihovny (unit) jazyka BP, Delphi a tudíž je lze snadno začlenit do programu při vývoji aplikace v těchto programovacích jazycích. Pro programátory, kteří používají jiná vývojová prostředí mají ilustrativní význam. I tak jim moje aplikace usnadní hledání vhodných nastavení materiálů.

# 11 Seznam literatury

1. ŽÁRA, J., BENEŠ, B., FELKEL, P.: Moderní počítačová grafika. Computer Pr
Praha, 1998.

2. CANTÚ, M.: Delphi 4: podrobný průvodce programátora. Grada, Praha, 1999.

3. Waite Group Press: OpenGL Super Bible. Macmillan Computer Publishing, 1996.

4. Neider, J., Davis, T., Woo, M., Shreiner, D.: OpenGL Programming Guide. Addi
Wesley Publishing Company, 1999.

5. Elektronická dokumentace vývojového prostředí programovacího jazyka Delphi.

# Seznam příloh

1. Disketa (3.5“) obsahující aplikaci a zdrojový kód aplikace.

Příloha č. 1

Disketa (3.5“) obsahující aplikaci a zdrojový kód aplikace.