Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Demonstrace OpenGL

Program pro demonstraci možností počítačové grafiky

2015

Lukáš Vrajík

Vedoucí práce: Michal Krupka, doc. RNDr. Ph.D.

Studijní obor: Informatika, prezenční forma

Bibliografické údaje

Autor:	Lukáš Vrajík
Název práce:	Demonstrace OpenGL (Program pro demonstraci možností počítačové grafiky)
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2015
Studijní obor:	Informatika, prezenční forma
Vedoucí práce:	Michal Krupka, doc. RNDr. Ph.D.
Počet stran:	37
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Lukáš Vrajík
Title:	Demonstration of OpenGL (A program for demonstration of computer graphics)
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2015
Study field:	Computer Science, full-time form
Supervisor:	Michal Krupka, doc. RNDr. Ph.D.
Page count:	37
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Práce pojednává o počítačové grafice s využitím existujícího API OpenGL, popisuje vytvořený program Demonstrace OpenGL, který slouží jako doplňující prezentační nástroj pro popularizační přednášky, během kterých jsou žákům středních škol představeny základy počítačové grafiky. Program obsahuje vybrané prvky OpenGL, které doplňují teoretickou část přednášek.

Synopsis

This thesis deals with computer graphics using existing OpenGL API. It describes the application OpenGL Demonstration , which is an additional presentation tool for popularizing lectures, during which the basics of computer graphics are introduced to the secondary school students. The program contains selected elements of OpenGL that complement the theoretical part of the course.

Klíčová slova: OpenGL; Common Lisp; CAPI; FLI; grafická primitiva; transformace

Keywords: OpenGL; Common Lisp; CAPI; FLI; primitive; transformations

Rád bych poděkoval svému vedoucímu doc. RNDr. Michalu Krupkovi, PhD. za cenné rady, odborné připomínky a vstřícnost při konzultacích v průběhu zpracování mé bakalářské práce.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

1 Úvod **8**

2 LispWorks **8**
2.1 CAPI . . . 9
2.2 FLI . . . 9
2.3 WIN32 . . . 9
2.4 OpenGL v prostředí LispWorks . . . 9

3 OpenGL **11**
3.1 Historie a vznik OpenGL . . . 11
3.2 Co je tedy OpenGL? . . . 11
3.3 Vývoj OpenGL . . . 12
3.4 Popis OpenGL . . . 13
3.4.1 Vykreslovací řetezec . . . 13
3.4.1.1 Vertex data . . . 13
3.4.1.2 Pixel data . . . 14
3.4.1.3 Per-vertex operace . . . 14
3.4.1.4 Per-pixel operace . . . 14
3.4.1.5 Rasterizace . . . 15
3.4.1.6 Per-fragment operace . . . 15
3.4.1.7 Frame-buffer . . . 15
3.4.2 Grafická primitiva . . . 16
3.4.3 Transformace . . . 20
3.4.4 Barva a světlo . . . 21

4 Aplikace Demonstrace OpenGL **23**
4.1 Uživatelská příručka . . . 23
4.1.1 Spuštění programu . . . 23
4.1.2 Okno editoru . . . 24
4.1.2.1 Prostředí . . . 25
4.1.2.2 Prezentace . . . 25
4.1.2.3 Slajdy . . . 25
4.1.2.4 Plátno . . . 26
4.1.2.5 Vrchol . . . 26
4.1.2.6 Normála . . . 26
4.1.2.7 Změna barvy . . . 26
4.1.2.8 Změna pozice . . . 26
4.1.2.9 Rotace . . . 27
4.1.2.10 Změna velikosti . . . 27
4.1.2.11 Změna vykreslení primitiva . . . 27
4.1.2.12 Práce s modelview maticí . . . 27
4.1.2.13 Změna typu stínování . . . 27
4.1.2.14 Řídící slajd . . . 28

4.1.3 Okno Projektoru 29
4.2 Programátorská příručka 30
4.2.1 CAPI 30
4.2.2 Třída workplace 30
4.2.3 Třída variable-environment 30
4.2.4 Třída slideshow 30
4.2.5 Třída primitive-slideshow 31
4.2.6 Vykreslení prezentace 31
4.2.7 Třída slide 31
4.2.8 Rozšíření o nové prvky 32
4.2.9 Přehled vytvořených tříd 32

Závěr **34**

Conclusions **35**

5 Obsah přiloženého CD/DVD **36**

Literatura **37**

Seznam obrázků

1 vykreslovací řetězec OpenGL 1.0, zdroj obrazku: [1] 13
2 rozdíl pořadí zadávání vrcholů 16
3 jednotlivé body 16
4 jednotlivé úsečky 17
5 řetěz z úseček 17
6 smyčka z úseček 17
7 jednotlivé trojúhelníky 18
8 trojúhelníky tvořící vějíř 18
9 pás trojúhelníků 18
10 jednotlivé čtyřúhelníky 19
11 pás čtyřúhelníků 19
12 konvexní polygon 19
13 Phongův model osvětlení, zdroj [2] 22
14 Rozdíl typů stínování 22
15 Vliv typu stínování na rasterizaci 22
16 Editor OpenGL 24
17 Plátno OpenGL 29

Seznam tabulek

List of Theorems

Seznam zdrojových kódů

1 Příklad prostředí 32

1 Úvod

Zadáním bakalářské práce bylo vytvořit aplikaci, kterou bude využívat doc. RNDr. Michal Krupka, PhD. pro názorné ukázky během svých popularizačních přednášek na středních školách. Důraz byl kladen na rychlost a jednoduchost předvedení jednotlivých prvků knihovny. Dále bylo cílem, aby bylo možné se zobrazovanou scénou pohybovat a aby bylo možno krok za krokem předvést, jak probíhá její vykreslení. Téma demonstrace počítačové grafiky s využitím prvků grafické knihovny OpenGL jsem si vybral kvůli počítačové grafice, která mě zajímá, a chtěl jsem vytvořit nástroj, který by pomohl zájemcům o tento obor v pochopení základních principů. Počítačová grafika je velmi populární a vyspělá, avšak stále potřebuje ohromné množství výkonu počítače. Myslím si, že výzkum v této oblasti je velmi aktuální a žádaný. V textu budou představeny použité technologie, následovány popisem samotného OpenGL a pak popisem aplikace, která je rozdělena na uživatelskou a programátorskou příručku.

2 LispWorks

LispWorks je vývojové prostředí implementující jazyk ANSI Common Lisp, které běží pod operačními systémy Windows, Linux, OS X. Vývoj LispWorks se datuje od roku 1989 a v současné době vyšla poslední, zatím pouze placená, verze 7.0. LispWorks se pro osobní potřeby a pro potřeby vzdělávání distribuují zdarma pod označením LispWorks Personal Edition. Tato verze má však svá omezení, například velikost alokované paměti v systému, nemožnost vytvořit spouštěcí soubor či pěti hodinový limit, po kterém se LispWorks vypne a je nutné jej spustit znovu. Tato omezení nejsou samoúčelná, jsou zde proto, že i tato bezplatná verze je plnohodnotným vývojovým prostředím. Vývojové prostředí samotné je napsáno v jazyce Common Lisp a lze snadno rozšiřovat o chybějící funkce. Avšak samotné nabízí spoustu nástrojů, které jsou pro vývoj software nepostradatelnou součástí. Jednou z nich je například Inspektor, který dovoluje nahlédnout i do samotného okna LispWorks a doplnit tak chybějící funkcionalitu.

Svou práci jsem začal na systému OS X, avšak okolnosti mě donutily nainstalovat prostředí na systém Linux, kde jsem se ale potýkal s úskalími spojenými s OpenGL a CAPI. I přes vyřešení problémů s ohledem na to, že aplikace by měla být k dispozici žákům středních škol a také že stále nejrozšířenějším systémem je Windows, jsem přešel právě na tento systém, ve kterém jsem práci i dokončil a pro který je aplikace optimalizována. Kromě různých licencí, kdy jsem z Personal Edition kvůli výše zmíněným omezením přešel na Professional Edition, jsem měl možnost vyzkoušet i různé verze samotných LispWorks, konkrétně verzi 5.1 Professional s updatem na 5.1.2 a verzi 6.1.1 Personal. Rozdíly jsem zaznamenal hlavně v balíku CAPI, kde mi u verze 5.1 chyběly některé sloty z verze 6.1.1.

2.1 CAPI

Common Application Programming Interface, zkráceně CAPI, je balík z vývojového prostředí LispWorks určený pro vytváření grafického uživatelského rozhraní. Obsahuje základní množinu prvků jako jsou tlačítka, menu, textová pole, stromy, oddělovače a mnohé jiné. Odstiňuje programátora od samotné práce s okny, které je u každého systému jiné, a proto je výsledný kód snadno přenositelný mezi jednotlivými platformami. Výsledek potom vypadá až na drobné detaily stejně.

Díky zmíněnému odstínění programátora od práce s okny jsem narazil na rozdíly mezi verzemi 5.1.2 a 6.1.1. Aby se okno po startu aplikace zobrazovalo pokaždé na stejném místě je nutné při vytváření definovat počáteční x-ovou a y-novou souřadnici levého horního rohu. Tady je to ještě stejné u obou verzí. V čem se ale verze liší jsou právě sloty pro tyto souřadnice ve vnitřní reprezentaci daného okna. Zatímco verze 6.1.1 má tyto sloty po zobrazení okna vyplněné, ve verzi 5.1.2 tyto sloty úplně chybí. Pro zjištění daných aktuálních souřadnic okna jsem tedy musel použít externí funkci C++, která dané souřadnice na základě popisovače okna zjistí.

2.2 FLI

Foreign Language Interface neboli FLI je rozšíření prostředí LispWorks, skrze které lze volat funkce napsané v jiném jazyce než je Common Lisp a naopak volat funkce napsané v lispu funkcí z jiného jazyka. Protože připojované jazyky mohou mít jiný typový systém než Common Lisp, je pro potřeby výměny dat mezi funkcemi nutné definovat vzájemnou korespondenci typů. FLI pak definujeme jakou reprezentaci mají datové typy či struktury z daného jazyka vůči lispu. Dále definujeme FLI funkci, která vezme argumenty, konvertuje je, zavolá funkci a vrátí opět konvertovanou hodnotu.

Ve své práci jsem toto rozhraní použil pro zjištění souřadnic a rozměrů aktuálního okna. Tyto údaje spolu s jinými jsou ukládány do regitrů, ze kterých se opět při spuštění aplikace načítají.

2.3 WIN32

WIN32 je dalším balíkem v prostředí LispWorks, který poskytuje funkce pro různé aplikační rozhraní Microsoft Windows. Tento balík není podporovanou implementací WinAPI, proto lze bezpečně využívat pouze dokumentované funkce. Mě zajímalo rozhraní pro přístup k registrům. Díky tomuto rozhraní lze jednoduše vytvářet, mazat, a ověřovat klíče, podklíče a přiřazovat jim hodnoty.

2.4 OpenGL v prostředí LispWorks

OpenGL je do prostředí LispWorks implementováno jako samostatná knihovna v rámci implementace LispWorks a nachází se v adresáři examples. V této složce

se nachází například i ukázky zdrojových kódů pro práci s CAPI. Tento adresář OpenGL obsahuje soubory, které definují jednotlivé datové typy OpenGL, OpenGL kontext - což je aktuální stav proměnných, bufferů a okna OpenGL, dále obsahuje soubory pro svázání OpenGL s CAPI a také FLI funkce, které volají funkce z knihoven závislých na daném operačním systému. Tedy v případě Windows volají funkce z knihoven opengl32.dll a glu32.dll. Takto implementovanou knihovnu OpenGL nepovažuji za vyhovující řešení, protože standard OpenGL se neustále rozšiřuje o takzvané Extensions, což jsou nové funkce OpenGL, které se v knihovnách operačních systémů nacházejí, ale s jejich aktualizací se bohužel nepromítnou do implementace v LispWorks.

3 OpenGL

3.1 Historie a vznik OpenGL

V 80. letech se nevyskytoval žádný univerzální grafický software, který by umožňoval pracovat s více typy hardware. Vývojáři museli pro každý kus hardware napsat odpovídající software, což bylo velmi nákladné a zdlouhavé. V 90. letech byla jedničkou v 3D grafice firma SGI. Jejich IRIS GL API byla považována za nejmodernější a stala se průmyslovým standardem. IRIS vynikala svou jednoduchostí v použití a její tehdejší konkurent PHIGHS byl naopak velmi kostrbatý a obtížný na vývoj aplikací. SGI takřka dominovala trhu. To se nelíbilo jiným společnostem jako SUN Microsystems, HP a IBM. Tyto firmy se proto rozhodly, ve snaze o ovlivnění trhu v jejich prospěch, dodat vlastní 3D hardware s využitím právě konkurenčního PHIGSu, kterému zajistili masivní podporu.

Tento krok spolu s narůstajícím počtem výrobců hardware vedl ke snížení podílu firmy SGI a jejich IRIS na trhu a ta se ve snaze o opětovné získání suverenity rozhodla změnit uzavřenou IRIS GL na volně dostupný, otevřený standard OpenGL. Toto ale nebylo úplně možné, protože IRIS byla zatížena patenty a funkcemi, které byly vázány na konkrétní hardware. Proto existovaly IRIS GL a OpenGL společne. Tuto vlastnost elegantně obešel OpenGL tím, že funkce, které hardware nepodporoval, nahradil ekvivalentními softwarovými. Tímto také OpenGL tlačil odpovědnost za vývoj přímo na výrobce hardware a přenesení funkcí z hardware na software na výrobce operačních systémů. Díky této abstrakční bariéře mohli vývojáři pracující s 3D vyvíjet software na vyšších úrovních.

Pro správu standardu OpenGL bylo vytvořeno průmyslové konsorcium ARB, které je v současnosti součástí Khronos Group a které dohlíží na údržbu a rozšíření specifikace OpenGL. Mezi firmami v konsorciu byl i Microsoft, který rok po vstoupení do konsorcia zveřejnil svoje vlastní API Direct3D jako konkurenci k OpenGL. Později se vyskytl společný projekt SGI a Microsoftu mající za úkol spojit to nejlepší z Direct3D a OpenGL. To se však kvůli nedostatku financí u SGI a strategickým důvodům Microsoft nestalo a projekt byl zrušen. Od té doby si jdou každý vlastní cestou.

V současnosti jsou k dispozici kromě OpenGL i jiná API pro práci s grafikou. Direct3D, který je pouze pro Microsoft Windows, Mantle od AMD pouze pro karty AMD, Apple API Metal zaměřeno na OS X a iOS a nástupce OpenGL Vulkan, od Khronos Group.

3.2 Co je tedy OpenGL?

OpenGL je zkratka pro Open Graphic Library, označující knihovnu fukcí pro práci s počítačovou grafikou, která splňuje určitý standard a přesná implementace závisí přímo na dané platformě, která tuto knihovnu využívá. Tyto implementace jsou nejčastěji přímo na grafické kartě a programátoři využívají funkce komunikující s hardware. Existují i implementace softwarové, které dovolují použít OpenGL. Tyto implementace jsou však velmi pomalé a většinou standard

nesplňují.

OpenGL se používá zejmnéna pro tvorbu počítačových her, virtuální realitu, pro tvorbu inženýrských programů jako jsou různé CAD systémy a jiné účely. Například nástroj založený na OpenGL používá Škoda Auto pro vizualizaci budoucích modelů automobilů. OpenGL také využívá systém společnost Valve ve svém operačním systému SteamOS, který je založený na linuxové distribuci Debian a je zaměřený na hráče počítačových her.

3.3 Vývoj OpenGL

První verze OpenGL vznikla v roce 1992 a od té doby prošla celou řadou změn a rozšíření. Mezi nejvýznamnější změny patří změna způsobu vykreslování. V ranných verzích se používaly příkazy glBegin a glEnd. Tyto příkazy způsobovaly volání knihovny OpenGL a s rostoucím počtem vykreslovaných objektů rostl i počet volání těchto funkcí, což vedlo ke značnému zpomalení. Proto vznikla pole vrcholů, ve kterých se definovala celá scéna, a vykreslení proběhlo jedním nebo více voláními daných funkcí. Dalším vylepšením byla možnost jedním voláním vykreslit stejné objekty, ale s různými vlastnostmi jako je poloha, barva či velikost. V současnosti je maximální snaha omezit přenos dat mezi procesorem a grafickou kartou.

Další důležitou změnou a vylepšením je změna fixní pipeline na programovatelnou. Fixní pipeline se rozumí pevně dané pořadí vykonávání funkcí nad daty. Toto pořadí bylo ovlivněno aktuálním stavem OpenGL. Programovatelnou pipeline přineslo OpenGL s verzí 2.0. Princip je založený na použití tzv. shaderů, což jsou programy ovlivňující konkrétní část vykreslovacího řetězce. Shadery byly nejprve psány v nízkoúrovňovém jazyce grafické karty podobnému assembleru a vyšší jazyky přišly až s novějšími grafickými kartami. Mezi takové jazyky patří například GLSL nebo Cg od firmy NVIDIA. Grafická karta má k dispozici omezený počet shaderovacích jednotek (např. GeForce 970GTX jich má 1664), které můžeme rozdělit na unifikované a neunifikované. Unifikované jsou specializované a mohou provádět pouze operace pro které jsou určeny, oproti tomu neunifikované jsou univerzální a lze je použít dle potřeby. Nevýhodou je potom menší výkon.

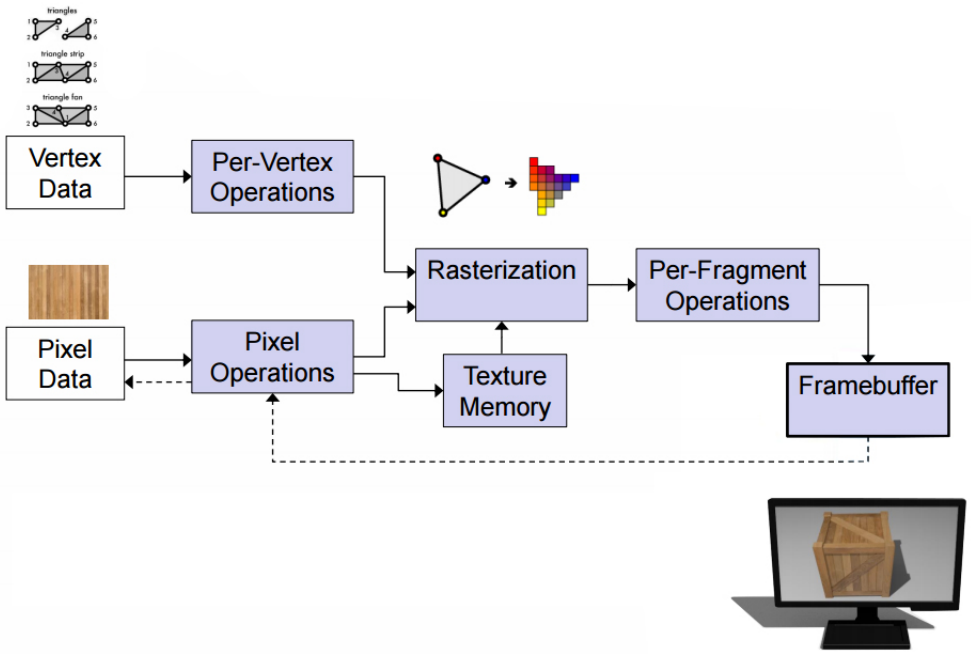
U vrcholu hovoříme o vertex shaderu, u fragmentů o fragment shaderu. Vertex shadery se používají pro určení aktuálních souřadnic jednotlivých vrcholů, fragment shadery potom pro určení barvy nebo načtení textury u jednotlivých pixelů (fragmentů). Mezi další shadery patří geometry shader, který oproti vertex shaderu dovoluje měnit počet vrcholů. Používá se například při modelování trávy nebo k úpravě existujících objektů a to v reálném čase. Od verze 3.2 přibyly také tesselation shadery. Tyto shadery také dovolují přidat počet vrcholů, a tím i detailů, v návaznosti na vzdálenosti objektu od pozorovatele, tedy objekty vzdálené jsou složeny z menšího počtu polygonů než objekty, které jsou blízko. Teselaci provádí přímo hardware, tedy grafická karta.

3.4 Popis OpenGL

Ve své práci používám prvky OpenGL verze 1.0 spolu s příkazy glBegin a glEnd, dále se budu zabývat popisem právě této verze. OpenGL se chová jako stavový automat, který je definován mnoha stavovými proměnnými. Na ty se můžeme dotazovat pomocí funkce glGet a nebo je nastavovat pomocí glSet. Stavy se mohou také povolit a zakázat příkazy glEnable a glDisable. Vykreslování objektu začíná funkcí glBegin, jejímž parametrem je typ grafického primitiva, a končí glEnd. Mezi tyto příkazy zadáváme vrcholy, normály nebo barvu. Dalšími funkcemi, které se používají mimo blok glBegin a glEnd, jsou hlavně funkce pro transformace (otočení, posun, škálování), pro práci se zásobníkem matic, stavové funkce na vymazání bufferů, nastavení kamery a jiné.

3.4.1 Vykreslovací řetezec

Průchod dat systémem OpenGL je reprezentován pomocí vykreslovacího řetězce. V případě verze 1.0 jde o fixed-function pipeline. Vstupní data rozdělujeme na vertex data a pixel data.



Obrázek 1: vykreslovací řetězec OpenGL 1.0, zdroj obrazku: [1]

3.4.1.1 Vertex data

Vertex data nesou informace o jednotlivých vrcholech. Mezi ně patří mimo jiné souřadnice, barva a normála. Pro zadávání souřadnic vrcholů je k dispozici několik funkcí, v práci používám funkci glVertex4D a její lispovou alternativu gl-

vertex-4d. Tato funkce bere 4 argumenty, jimiž jsou hodnoty typu double, tvořící homogenní souřadnice [x, y, z, w]. První tři jsou souřadnice vrcholu x, y, z a čtvrtá hodnota w udává, zda se jedná o bod vlastní či nevlastní. Kartézské souřadnice jsou pak vypočítány jako podíl [x/w, y/w, z/w].

3.4.1.2 Pixel data

Pixel data jsou rastrová data. Použití naleznou při zobrazení bitmap, což jsou obrázky složené pouze z jedné barvy. Jelikož OpenGL samotné nepodporuje práci s textovými řetězci, používá se právě bitmapa se všemi znaky daného fontu a jen se vybírají souřadnice potřebného znaku.

Dalším použitím rastrových dat jsou takzvané pixmapy. To jsou rastrové obrázky, kde barva jednoho pixelu je reprezentována více než jedním bitem.

Posledním typem jsou textury, které bývají uloženy v paměti přímo na grafické kartě nebo v hlavní paměti, a odtud se pouze nanášejí na povrch tělesa během rastrování.

3.4.1.3 Per-vertex operace

Nad vertex daty jsou provedeny per-vertex operace. Tím se rozumí operace provádějící transformaci pomocí ModelView matice. Tato matice tvoří přechodovou matici od báze modelu k bázi světa. Báze je způsob, jakým jsou dané vrcholy reprezentovány v prostoru. Každý objekt je vytvářen ve své vlastní bázi a do celé scény se zobrazí právě vynásobením každého vrcholu zmíněnou přechodovou maticí. Pokud jsou zadány textury, jsou i souřadnice textury transformovány.

Dalším krokem je výpočet osvětlení. Osvětlení jednotlivých vrcholů se počítá s použitím normál a informací o materiálu daného vrcholu. OpenGL používá Phongův osvětlovací model, který je složen ze tří typů světla. Z okolního světla - rovnoměrně dopadající paprsky na těleso, difúzního světla - část světla rovnoměrně se odrážející od tělesa do všech směrů, a odlesků - část světla, která se od tělesa odráží jedním směrem.

Po výpočtu osvětlení přichází na řadu ořezání neboli clipping. Ořezání se provádí pro zrychlení a usnadnění vykreslování. Aby nedošlo k nevykreslení polygonů, u kterých byly odřezány pouze některé vrcholy, jsou přidány další vrcholy tak, že se zbytek polygonu vykreslí.

Nakonec následuje vynásobení vrcholů maticí Projection, která tvoří matici přechodu od báze světa k bázi pozorovatele, a opět se provede ořezání tak, že se vykreslí pouze ty vrcholy, které jsou v zorném poli.

3.4.1.4 Per-pixel operace

Během provádění operací nad jednotlivými pixely jsou dekódovány barvy do interní formy grafické karty. Následuje uložení pixelů do paměti pro textury na grafické kartě nebo jsou předány přímo rasterizační jednotce.

3.4.1.5 Rasterizace

Rasterizace je proces rozložení grafického primitiva na jednotlivé body. Během rasterizace jsou primitiva převedena na dvourozměrný obrázek. Každý bod tohoto obrázku nese informaci o svojí barvě a hloubce. Rasterizace se tedy skládá z určení, do kterého čtverce v mřížce okna primitiv zasahuje, a také jakou má daný čtverec barvu a hloubku. Během této operace se vytváří alias, což je objekt se zubatými hranami nebo interference u pravidelných tvarů vzdálených odpozorovatele.

3.4.1.6 Per-fragment operace

Data z rasterizace jsou dále zpracovávána pomocí per-fragment operací. Jedná se o sadu testů a funkcí, které mohou fragment upravit nebo vyřadit z vykreslování. Mezi takové operace patří test, zda OpenGL kontext vlastní pixel, tedy jestli není renderovací okno překryto jinou aplikací. Dále prochází testem zvaný Scissor, ten má za úkol zjistit, zda je pixel v uživatelem definovaném obdélníku. Používá se například ve hrách, kde část obrazovky je pokryta GUI a není potřeba pod ním vykreslovat. Mezi další patří Stencil test, který se používá pro vytváření stínů. Objekty se nechají vykreslit bez testování hloubky. Tím se všechny vykreslí na rovinu a vznikne vzor, který tvoří šablonu pro následné vykreslení stínů. Depth buffer test ověřuje pořadí fragmentů. Pokud máme vyplý tento test, objekty se vykreslují v pořadí, v jakém byly nadefinovány. Blending test kombinuje hodnotu Alpha s hodnotami RGB a vytváří tak průhledný fragment. Pro vytváření průhledných fragmentů je potřeba vykreslit nejprve objekty nacházející se za objektem průhledným. Následuje Dithering, což je tvorba více odstínů dané barvy pomocí různé vzdálenosti bodů od sebe. A jako poslední jsou logické operace, které se provádějí pro každý pixel zvlášť a kombinují vstupní hodnoty s hodnotami uloženými ve framebufferu.

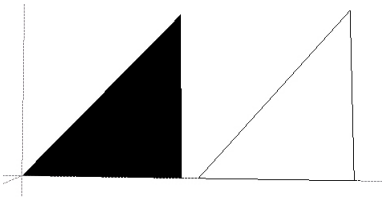
3.4.1.7 Frame-buffer

Posledním prvkem ve vykreslovacím řetežci je Frame-buffer. Je to množina pixelů uspořádaných do dvourozměrného pole. Každý pixel se sestává z většího počtu bitů. Tento počet je dán implementací OpenGL. Odpovídající bity tvoří logické celky, kterým říkáme color, depth nebo také někdy Z, stencil a accumulation buffry. Color buffer se skládá ze 4 buffrů front left, front right, a back left a back right buffr. Pro nás je důležité, že se na barevném monitoru zobrazuje front left a front right buffer.

3.4.2 Grafická primitiva

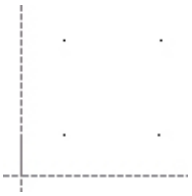
Grafická primitiva jsou základními stavebními kameny celé scény. Jsou to nejjednodušší elementy OpenGL, ze kterých se tvoří geometrické objekty, a z nich potom větší celky. Do bloku, který je tvořen příkazy glBegin a glEnd, se vkládají body a jsou to právě grafická primitiva, která určují jejich význam. Typ grafického primitiva se zadává jako povinný parametr příkazu glBegin.

Pro zadávání vrchlů existuje konvence a je velice důležité ji dodržovat. Jde o pořadí zadávání vrcholů. Vrcholy mohou být zadávány po směru hodinových ručiček nebo proti směru hodinových ručiček. Způsob zadávání vrcholů se na začátku vykreslování nastaví do jedné z mnoha stavových proměnných OpenGL. Pokud například nastavíme, že vrcholy budeme zadávat proti směru hodinových ručiček z pohledu kamery, a vrcholy tak zadáme, bude grafické primitivum čelem k nám. To je důležité proto, že u grafických primitiv lze zadat, jak jsou vykreslována. Existují tři druhy vykreslení, která je možno pro každou stranu nastavit zvlášť. Prvním typem je vykreslení pouze daných vrcholů jako body. Druhým je vykreslení pouze úseček mezi vrcholy a posledním způsobem je vykreslení vyplněného primitiva. Toto rozlišení si nesmíme plést se samotnými primitivy, mezi která také patří samostatné body, úsečky. Pro ilustraci jsou použity obrázky z aplikace Demonstrace OpenGL.



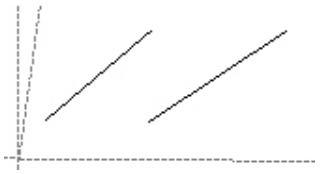
Obrázek 2: rozdíl pořadí zadávání vrcholů

Mezi grafická primitiva patří izolované body. Jedná se o nejjednodušší typ z primitiv, kde každý bod je reprezentován jedním vrcholem.



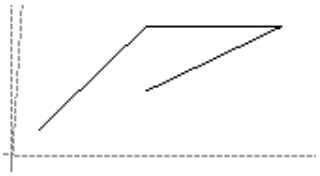
Obrázek 3: jednotlivé body

Dalším zástupcem primitiva jsou jednotlivé úsečky. Pro vykreslení jedné úsečky je potřeba zadat právě dva vrcholy. Z toho plyne, že úsečky jsou zadávány po párech vrcholů, a v případě, že by počet vrcholů, byl lichý, bude poslední bod ignorován. Způsobů, jak vykreslit čáry, je v OpenGL několik.



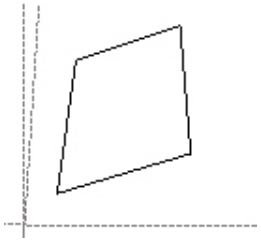
Obrázek 4: jednotlivé úsečky

Vrcholy lze také reprezentovat jako řetězec úseček. V tomto případě vždy dvě úsečky sdílí jeden bod. První úsečka je zadána pomocí dvou bodů a pro vykreslení následující úsečky stačí zadat jeden vrchol. Druhá úsečka bude definována druhým a třetím zadaným vrcholem. Třetí úsečka pak třetím a čtvrtým vrcholem a tak dále.



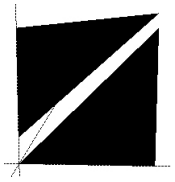
Obrázek 5: řetěz z úseček

Úsečky, které tvoří smyčku z úseček, jsou dalším primitivem. Postup vykreslení je stejný jako v případě řetězce z úseček s tím rozdílem, že první a poslední vrchol tvoří poslední úsečku.



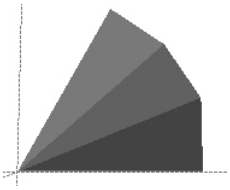
Obrázek 6: smyčka z úseček

Pokud jako grafické primitivum zvolíme trojúhelníky, tvoří trojice po sobě jdoucích vrcholů v bloku jeden trojúhelník. K vykreslení jednoho trojúhelníku jsou tedy nutné minimálně tři body. Pokud počet vrcholů není dělitelný třemi, jsou přebytečné vrcholy ignorovány. Lze vytušit, že tato metoda je velice paměťově náročná a pro vykreslení n trojúhelníků je zapotřebí $3n$ vrcholů.



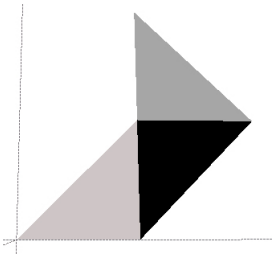
Obrázek 7: jednotlivé trojúhelníky

Z trojúhelníků je možné vytvořit vějíř, který je dalším grafickým primitivem. Po zadání prvního trojúhelníku, jenž je definován třemi vrcholy, je zde každý další trojúhelník reprezentován jedním bodem novým, bodem předchozím a prvním zadaným bodem, kolem kterého se vějíř konstruuje. Tento bod je společný pro všechny trojúhelníky.



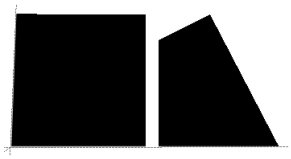
Obrázek 8: trojúhelníky tvořící vějíř

Dalším typem primitiva, který lze vytvořit z trojúhelníků, je trojúhelníkový pás. Opět jako u vějíře je další trojúhelník definován novým vrcholem, ale při tvorbě nového trojúhelníku jsou brány dva vrcholy z trojúhelníku předešlého. Spolu s předchozím poskytují efektivnější využití paměti než u jednotlivých trojúhelníku.



Obrázek 9: pás trojúhelníků

Stejně jako u trojúhelníků lze zadávat samostatné čtyřúhelníky. Tentokrát jsou vrcholy brány po čtyřech a vrcholy, které netvoří čtveřici jsou ignorovány. U tohoto primitiva je důležité zaručit, aby všechny čtyři vrcholy ležely v jedné rovině. Pokud by neležely, nemusí být primitivum vykresleno správně. To, jak bude vykreslení vypadat, záleží na daném hardwaru.

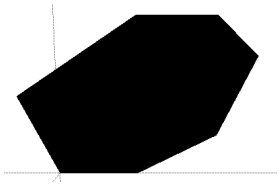


Obrázek 10: jednotlivé čtyřúhelníky

Z čtyřúhelníků lze také vytvořit pás. Každý další čtyřúhelník je definovaný pomocí nových dvou vrcholů. Pokud zadáváme vrcholy po směru hodinových ručiček, vkládáme vrcholy dle písmene N. Opět je důležité dodržet, aby v případě prvního čtyřúhelníku ležely všechny čtyři vrcholy v jedné rovině. V případě dalších čtyřúhelníků postačí, aby ležely v rovině pouze dva vrcholy. Tato rovina musí být otočená dle osy, která je tvořena použitými dvěma vrcholy z předchozího čtyřúhelníku.

Obrázek 11: pás čtyřúhelníků

Posledním typem primitiva je polygon. Toto primitivum pro vykreslení vyžaduje zadat minimálně tři vrcholy. Také platí podmínka o zaručení, aby všechny vrcholy ležely v jedné rovině a navíc body musí tvořit konvexní tvar. Konvexní tvar je takový obrazec, ve kterém pro každou úsečku mezi dvěma libovolnými body platí, že celá leží v daném tvaru. Pokud toto není splněno, provede se rozdělení polygonu na trojúhelníky, které jsou vždy konvexní. Tomuto jevu se říká teselace.



Obrázek 12: konvexní polygon

3.4.3 Transformace

Pro manipulaci s grafickými primitivy používáme v OpenGL transformace. Ty jsou reprezentovány pomocí matic o rozměru 4×4. Matice reprezentuje bázi, která je složena ze tří vektorů (první tři sloupce) a počátečního bodu (poslední sloupec). Transformace je provedena vynásobením v okamžiku vykreslování aktuální matice s daným vrcholem. Postupným násobením transformačních matic vznikají složitější transformace a vrchol je pak násoben výslednou maticí. Na počátku máme matici identity, která je reprezentována jednotkovou maticí. Matice jsou v OpenGL reprezentovány 16ti prvkovým vektorem. Pro získání dané transformace můžeme použít dva přístupy. První způsob je přímá manipulace matic a druhou možností je použít OpenGL poskytované funkce, které práci s maticemi udělají za nás. Protože práce s maticemi je základem rozmísťování objektů a primitiv na scéně, je matice velmi často měněna. Jelikož se OpenGL chová jako stavový stroj, jednou změněná matice tak zůstane do té doby, než se opět změní. V případě, že bychom se chtěli vrátit k předchozímu stavu matice, museli bychom použít opačnou transformaci a vrácení o více než jeden krok by bylo náročné. K tomuto účelu poskytuje OpenGL zásobník, na který se aktuální matice po zavolání funkce glPushMatrix uloží a po zavolání glPopMatrix je matice vyjmuta a aktuální matice se nastaví na matici, která je na vrcholu zásobníku.

Mezi základní typy transformací patří posunutí, rotace a změna velikosti. Pořadí těchto operací je důležité dodržovat, protože výsledek posunutí před rotací je jiný než rotace následována posunutím. Pravidlem je, že se nejdříve objekt posune, poté je provedena rotace a nakonec změna velikosti.

Posunutí je realizováno pomocí posunutí počátku.

$$posunutí_{X,Y,Z} = \begin{pmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Změna velikosti je provedena změnou velikosti vektorů definujících bázi.

$$velikost_{X,Y,Z} = \begin{pmatrix} X & 0 & 0 & 0 \\ 0 & Y & 0 & 0 \\ 0 & 0 & Z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

U rotací rozlišujeme podle které osy neboli podle kterého vektoru bázi otáčíme. Ten pak zůstává stejný a pomocí goniometrických funkcí vyjádříme rotaci zbylých vektorů. Rozlišujeme tedy tři transformační matice.

$$rotace\ x_{\varphi} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\ \varphi & -sin\ \varphi & 0 \\ 0 & sin\ \varphi & cos\ \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$rotace\ y_\varphi = \begin{pmatrix} cos\ \varphi & 0 & sin\ \varphi & 0 \\ 0 & 1 & 0 & 0 \\ -sin\ \varphi & 0 & cos\ \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

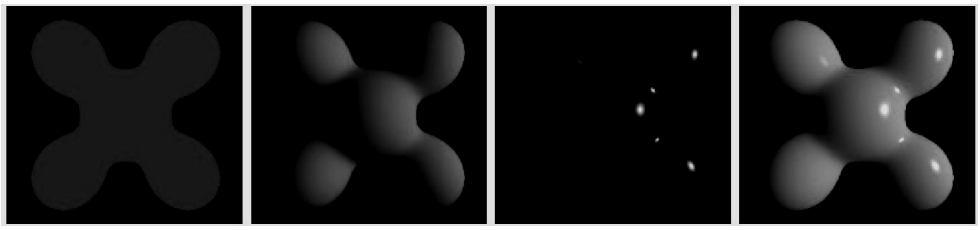
$$rotace\ z_\varphi = \begin{pmatrix} cos\ \varphi & -sin\ \varphi & 0 & 0 \\ sin\ \varphi & cos\ \varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3.4.4 Barva a světlo

V OpenGL se převážně využívá aditivního barevného modelu RGB. Tento model využívají monitory či televize. Jednotlivé složky barvy jsou vnitřně reprezentovány čísly v plovoucí řádové čárce v intervalu [0, 1]. Tato reprezentace je přesnější než interval [0, 255], ale je paměťově náročnější. Barva se zadává pro vrcholy a opět platí, že dokud se nezmění, bude se vše vykreslovat aktuálně nastavenou barvou. Při vytváření fragmentů z primitiv se barva pixelů mezi vrcholy definující primitivum vypočítává lineární interpolací. Konečnou barvu pixelu dále ovlivňuje osvětlení, pokud je zapnuto a materiál.

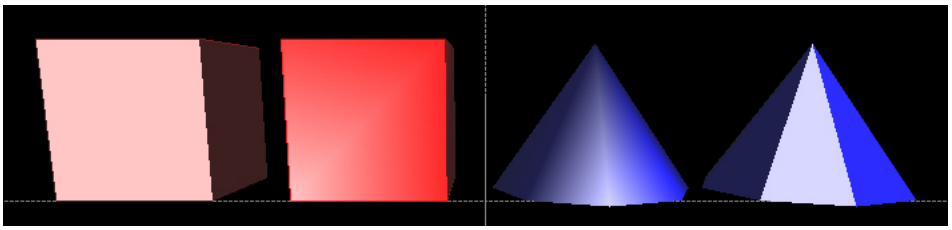
OpenGL používá Phongův model osvětlení. Výpočet osvětlení se snaží co nejvíce přiblížit vlastnostem reálného světla s ohledem na efektivitu a rychlost výpočtu. Implementuje pouze část malou část fyzikálních vlastností, proto nedovoluje vytvořit některé světelné efekty. Nelze například vykreslit efekty související s nepřímými odrazy paprsků. Základem je rozdělení světla do tří složek. Am-bient složka, reprezentující zbytkové okolní světlo, které dopadá na celý povrch objektu rovnoměrně a tvoří náhradu za odražené paprsky od okolních objektů. Všechny části objektu jsou osvětleny stejně, takže nevytváří trojrozměrný dojem. Druhou částí je difúzní složka. Tato část světla je tvořena rozptýleným světlem do všech směrů od povrchu objeku. Nejlépe pozorovatelné je na matném povrchu. Difúzní světlo je tím intezivnější, čím je úhel mezi paprskem a normálou menší. Normála je jednotkový vektor kolmý ke grafickému primitivu a je nezbytnou součástí pro správné vypočtení světla. Poslední složku tvoří specular světlo. Je to složka světla, která se od tělesa odráží přibližně stejným směrem k pozorovateli. Pro všechny složky Phongova modelu lze nastavit jinou barvu. Na následujícím obrázku je zleva ambient, difuse, specular složka, a výsledné sečtení všech tří složek tvořící Phongův model osvětlení.

Barvu lze ovlivnit zadáním vlastností materiálu. Jsou to další stavové proměnné, které se nevztahují k danému primitivu či objektu, kdy změněné hodnoty se použijí pro veškeré další vykreslování. Vlastnostmi materiálu ovlivňujeme jeho reakci na světlo. Tvoří ho stejné složky jako samotné světlo, tedy ambientní, difúzní a lesklá složka. Navíc lze zadat lesklost materiálu a míru distribuce světelných paprsků v primitivu. Lesklost materiálu ovlivňuje velikost plochy, na kterou je aplikována specular složka světla.



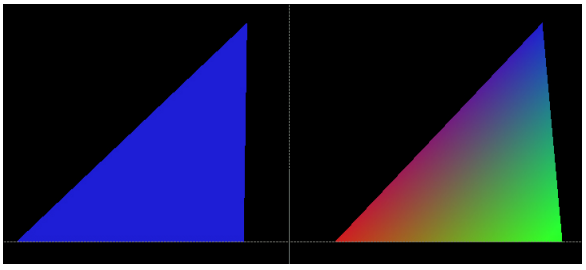
Obrázek 13: Phongův model osvětlení, zdroj [2]

Poslední možností, jak ovlivnit barvu primitiva, je typ stínování. V OpenGL rozlišujeme mezi dva typy stínování. Jedním je plochý typ stínování a druhým typem je Goraudovo stínování. V prvním případě je zvolena jedna ze zadaných normál a ta je použita pro všechny pixely daného primitiva. Ve druhém případě dochází k interpolaci mezi normálami zadaných u vrcholů. Tímto typem stínování lze jednoduše vykreslit zaoblené tvary. Na obrázku je u vnějších objektů použit typ plochého stínování a u vnitřních je nastaveno Goraudovo stínování.



Obrázek 14: Rozdíl typů stínování

Typ stínování neovlivňuje pouze výpočet světelných vlastností, ale také rasterizování primitiva. Následující obrázek zobrazuje dva stejné trojúhelníky, vlevo vykreslen při plochém typu stínování a vpravo při použití Goraudova stínování.



Obrázek 15: Vliv typu stínování na rasterizaci

4 Aplikace Demonstrace OpenGL

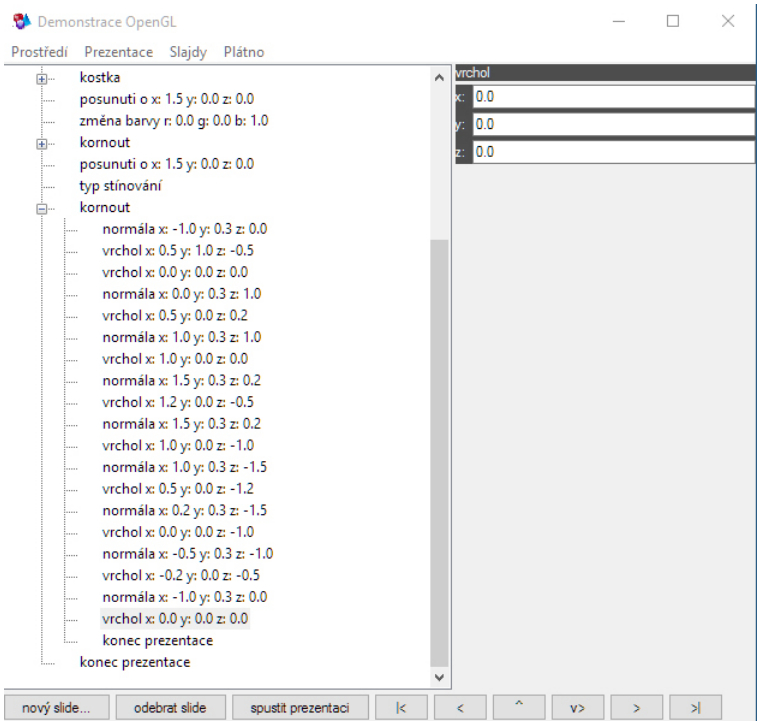
4.1 Uživatelská příručka

Demonstrace OpenGL je program určený pro doplnění přednášek zaměřených na představení základů počítačové grafiky žákům středních škol. Implementuje část prvků grafické knihovny OpenGL a klade důraz na názorné předvedení jednotlivých prvků. Samotné příkazy OpenGL jsou zadávány pomocí grafického rozhraní tak, aby ho mohli využívat i žáci, kteří se s programováním ještě nesetkali. Tímto postupným vkládáním příkazů je možné vytvořit jednoduché i složitější scény bez nutnosti psaní textových příkazů. Textové příkazy se dají vkládat také, a to pomocí speciálních prvků programu, viz dále řídící slajd. Aplikace je koncipována jako prezentační nástroj, ve kterém jsou jednotlivé scény reprezentovány prezentacemi. Danou scénu je tak možno krok po kroku vykreslit do úplných detailů. Každý krok je tvořen jedním slajdem, který představuje příkaz OpenGL. Na slajd lze vložit místo příkazu i celou prezentaci.

Program není třeba instalovat, pouze je ke spuštění nutné mít v počítači základní OpenGL knihovny opengl32.dll a glu32.dll. Nacházejí se ve složce /windows/system32, případně jsou k dispozici na přiloženém CD. Aplikace se skládá ze dvou oken, okna editoru a projektoru. V okně editoru se vytvářejí jednotlivé prezentace a v okně projektoru probíhá jejich vykreslení.

4.1.1 Spuštění programu

Po spuštění programu se zobrazí dvě okna. Jedno představuje okno editoru, ve kterém se na pravé straně ve stromě zobrazují prezentace a jejich prvky a na druhé straně se po zvolení prvku ze stromu zobrazí grafická reprezentace funkce, kterou zvolený prvek představuje. V druhém okně, projektoru, se pak zvolený prvek vykreslí. Po startu aplikace je okno editoru i okno projektoru prázdné. V tuto chvíli lze vytvořit nové prostředí nebo otevřít prostředí ze souboru. Obě volby se nacházejí pod položkou prostředí. Při volbě možnosti otevření, po otevření souboru jsou ve stromu zobrazeny jednotlivé prezentace. Kliknutím na jednu z nich dojde k jejímu načtení, v pravé části editoru se zobrazí její název a seznam proměnných, které se v dané prezentaci nacházejí. Stisknutím "+"vedle názvu prezentace dojde k rozbalení a zobrazení jejích slajdů. Šipkami se lze pohybovat ve stromu. Zároveň je v okně projektoru vykreslována prezentace. Po vytvoření nového prostředí levá část obsahující strom zbělá. Znamená to, že prostředí je prázdné. Při vytváření scén se postupuje od nejjednodušších prvků, které jsou spojovány do složitějších celků. Základním vykreslovaným prvkem jsou grafická primitiva, která vytvoříme prezentací pro kreslení primitiva. Spojení do větších celků zajišťuje prezentace pro vytváření objektů z primitiv. Do jednotlivých prezentací se vkládají slajdy, které představují jednotlivé příkazy jazyka OpenGL.



Obrázek 16: Editor OpenGL

4.1.2 Okno editoru

Okno editoru se skládá ze tří částí. Na levé straně se nachází strom prezetací, který slouží k orientaci a také k vybrání slajdu nebo celé prezentace k vykreslení. Prvky se vždy vykreslují zpětně, takže právě vybraný prvek není zobrazen. Lze je okamžitě vykreslit stisknutím klávesy ENTER, nebo dvojím poklepáním myši. V pravé části je zobrazen náhled prvku stromu. Pokud je vybrán slajd, obsahuje náhledy jednotlivých příkazů OpenGL. Pokud je zvolena prezentace, nalezneme zde informace o proměnných, které daná prezentace obsahuje, a v případě prezentace, která je určena k vykreslování grafického primitiva, je zde i výběr daného typu primitiva. Ve spodní části nalezneme tlačítko "nový slide", které zobrazí nabídku zda vložit slajd před aktuální prvek nebo na konec prezentace, tlačítko "odebrat slide" pro odebrání právě zvoleného slajdu, a tlačítka pro procházení prezentace. Pro procházení prezentace je zde 6 tlačítek.

|< skok na začátek prezentace

< posun o jeden slajd směrem na začátek prezentace

^ jestliže se nacházíme v prezentaci, která je součástí slajdu prezentace o úroveň výš, způsobí vystoupení z prezentace a návrat na místo, ze kterého jsme do prezentace vstoupili

∨> slouží k přesunu na další slajd nebo vstoupení do prezentace, která je obsahem slajdu

> posun o jeden slajd směrem ke konci prezentace

>| skok na konec prezentace

4.1.2.1 Prostředí

V horním menu nalezneme položky pro práci s prostředím, prezentacemi, slajdy a plátnem. Prostředím je myšleno pracovní prostoředí, ve kterém se nacházejí všechny prezentace. Toto prostředí se ukládá do souboru. Pod položkou **"Prostředí"** se nachází volba "Nové prostředí...", která po zadání jména do dialogového okna vytvoří nové prázdné prostředí. Dále je tu možnost "Otevřít...", která otevře již existující prostředí. Touto volbou se zobrazí dialogové okno, ve kterém je potřeba zvolit soubor, který dané prostředí reprezentuje. Poté se prezentace, které jsou obsahem prostředí zobrazí v navigačním stromu. Pro uložení práce v prostředí jsou zde dvě možnosti. První možnost pod položkou "Uložit"provede uložení prostředí do souboru, ze kterého bylo prostředí otevřeno. Druhou možnost představuje položka "Uložit jako...", která vyvolá dialogové okno s dotazem na jméno souboru, do kterého se má prostředí uložit.

4.1.2.2 Prezentace

Druhá položka **"Prezentace"** představuje možnosti operací s prezentacemi. Prezentace jsou rozděleny na dva typy. První typ je určený pro zadáví grafického primitiva. Druh typ prezentace je obecná prezentace, která umožňuje kombinovat ostatní prvky OpenGL a primitiva do složitějších scén. Zvolením první možnosti "Nová prezentace pro kreslení primitiva..." nebo druhé "Nová prezentace pro vytváření objektů z primitiv..." dojde k otevření okna pro zadání jména prezentace. Po zadání jména je vytvořena prázdná prezentace. Rozlišení mezi první volbou a druhou hraje roli při vkládání slajdů. Třetí volba "Odstranit prezentaci" provede vymazání prezentace ze všech míst, kde se prezentace nachází. Poslední volbou pro prezentace je možnost "Vložit proměnnou". Zvolením této možnosti dojde k vyvolání okna pro zadání názvu proměnné a poté k zadání číselné hodnoty proměnné. Proměnné lze ve slajdech libovolně použít jako číselnou hodnotu a také změnit aktuální hodnotu na jinou.

4.1.2.3 Slajdy

Třetí položka **"Slajdy"** obsahuje dvě volby pro vložení slajdu a jednu pro odstranění slajdu. Volbou "Vložit slajd před aktuální" dojde k vyvolání nabídky příkazu, který slajd reprezentuje. Tato nabídka se liší v závislosti na typu prezentace, do které chceme slajd vkládat. Pokud chceme vložit slajd do prezentace, která provádí vykreslení primitiva, jsou v nabídce možnosti pro příkaz vložení

vrcholu, změnu barvy a zadání normály. Jestliže chceme vložit slajd do prezentace druhého typu, nabídka obsahuje možnosti pro příkaz výběru typu vykreslení přední strany polygonu, změnu barvy, posunutí, změnu velikosti, rotaci, uložení matice, vyjmutí matice, změnu typu stínování a vložení řídícího slajdu. Dále jsou v nabídce zobrazeny prezentace z prostředí, které je možné vložit jako obsah slajdu. Druhou volbou pod položkou "Slajdy" je možnost vložit nový slajd na konec prezentace. Kromě pozice vložení je chování stejné jako v prvním případě. Volbou "Odstranit slajd" dojde ke smazání aktuálního slajdu. Pokud je obsahem slajdu prezentace, je smazána pouze v místě daného slajdu.

4.1.2.4 Plátno

Poslední položkou v menu je **"Plátno"** obsahující jedinou volbu a to "Zobrazit plátno OpenGL". Je zde pro případ zavření okna Projektoru. Tato volba zároveň nastaví scénu do výchozí polohy.

4.1.2.5 Vrchol

Pro vytvoření vrcholu je nutné nacházet se v prezentaci, která slouží pro kreslení grafického primitiva. Následně je potřeba zvolit z menu položku slajd, jednu z možností kam slajd vložit a z následující nabídky zvolit "vrchol". V náhledu slajdu se zobrazí tři textová pole, reprezentující jednotlivé souřadnice vrcholu. Do těchto polí lze vložit číselnou hodnotu nebo název proměnné, která je vytvořena v prezentaci.

4.1.2.6 Normála

Vytvoření normály probíhá stejným způsobem jako vložení vrcholu. Tentokrát textová pole reprezentují souřadnice vektoru kolmého k danému primitivu. Normála je vztažena k následujícímu vrcholu, proto by vytvoření normály mělo předcházet vytvoření vrcholu.

4.1.2.7 Změna barvy

Slajdem pro změnu barvy měníme barvu kreslení primitiva. Možnost vložení slajdu reprerezentující změnu barvy se nachází jak v primitivní slideshow tak i v obecné. Opět se v náhledu slajdu nachází tři textová pole. Do těchto polí, reprezentující jednotlivé složky barvy, vkládáme číselnou hodnotu. Vhodné hodnoty se nacházejí v intervalu [0, 1]. Jednička znamená 100% zastoupení dané složky, nula naopak.

4.1.2.8 Změna pozice

Náhled tohoto slajdu obsahuje tři textová pole, které reprezentují souřadnice, o které bude báze posunuta. Jednoduše si můžeme představit, že budou tyto hodnoty přičteny ke každému vrcholu.

4.1.2.9 Rotace

Náhled se skládá z rozbalovací nabídky, která obsahuje volby *x, y* a *z*, což jsou osy, okolo kterých bude rotace provedena. Dále je tu textové pole, sloužící k zadání velikost úhlu, do kterého lze vložit číselnou hodnotu nebo proměnnou.

4.1.2.10 Změna velikosti

Slajd realizující tuto transformaci obsahuje tři textová pole, kdy každé z nich představuje koeficient, kterým je vynásoben bázový vektor příslušný osám *x, y* nebo *z*. Základní hodnotou je 1. Pro zvětšení je třeba zadat hodnotu větší a pro zmenšení naopak. Velikost můžeme měnit pro každou osu nezávisle.

4.1.2.11 Změna vykreslení primitiva

Změnu vykreslení primitiva je možné provést vložením odpovídajícího slajdu. Tento slajd obsahuje rozbalovací nabídku, která sestává z možností "jednotlivé body", "linky", "vyplněný". Ve výchozím nastavení programu je primitivum vykreslováno vyplněné. Změnou hodnoty dojde ke změně vykreslení předních stran následujících primitiv. V prezentaci pro vytvoření primitiva se na prvním slajdu nachází volba typu primitiva. Potřebný typ primitiva je možno zvolit v rozbalovací nabídce, obsahující výčet možných typů. Je důležité nezaměňovat typ vykreslení primitiva a typ primitiva. Pokud například zvolíme typ primitiva jako čáry, nastavením vykreslení primitiva na "vyplněný"nedojde k vykreslení vyplněného primitiva.

4.1.2.12 Práce s modelview maticí

Pro vytváření složitějších scén se v OpenGL verze 1 pracuje se zásobníkem matic. Na tento zásobník ukládáme aktuální matici "světa". Tento postup lze názorně vysvětlit například na tvorbě ruky: vykreslíme dlaň, posuneme se o délku dlaně, uložíme matici, vykreslíme první článek prstu, posuneme se o délku článku, vykreslíme druhý článek, posuneme se o délku článku, vykreslíme třetí článek a posuneme se o délku článku. Nyní pro vykreslení druhého prstu bychom se museli posunutím dostat zpátky k dlani, místo toho provedeme vyjmutím matice ze zásobníku, které je jednodušší. Pro zadání těchto dvou příkladů slouží slajdy pro uložení matice a vyjmutí matice. V náhledu slajdu se zobrazí pouze text, značící o jakou operaci se zásobníkem se jedná.

4.1.2.13 Změna typu stínování

Pro nastavení plochého nebo Goraudova stínování slouží slajd "typ stínování". Tento slajd obsahuje v náhledu rozbalovací nabídku, ze které lze požadované stínování zvolit. Daná volba pak bude platit od momentu vložení slajdu do další změny, kterou je možno provézt vložením dalšího slajdu totoho typu na následující slajdy v prezentaci.

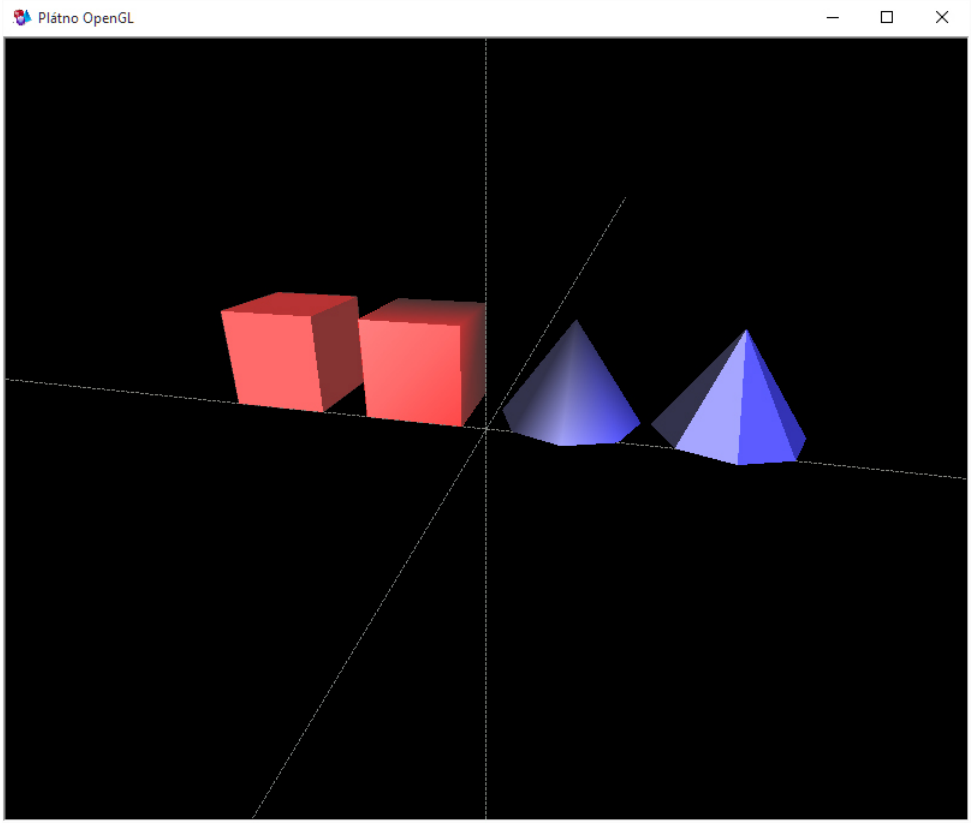
4.1.2.14 Řídící slajd

"Řídící slajd" obsahuje textové pole, do kterého lze vložit připravené příkazy pro nastavení proměnné, skok na začátek a příkaz pro podmínku. Příkaz pro nastavení proměnné se zadává ve tvaru (nastav *název_proměnné hodnota*). Hodnotou může být jiná proměnná, číslo nebo symbolický výraz v lispové syntaxi. Příkladem může být zvýšení hodnoty proměnné x, kterou máme nastavenou na 0, o jedničku. Celý příkaz by pak vypadal následovně: (nastav x (+ x 1)). Dalším příkazem je (skoc-na-zacatek), znamenající okamžité navrácení na začátek prezentace, ve které se řídící slajd s tímto příkazem nachází. Posledním příkazem, který doplňuje předchozí, je příkaz (kdyz *neco pak nebo*), který provede vyhodnocení podmínky, a pokud je pravdivá, provede se výraz za podmínkou, pokud není pravdivá, provede se poslední výraz, který je nepovinný. Následuje příklad řídícího slajdu, který pokud bude hodnota x < 3, provede skok na začátek, a v opačném případě nastaví x na 0. Obsah řídícího slajdu: (kdyz (< x 3) (skoc-na-zacatek) (nastav x 0)). Těmito příkazy lze provádět cyklení. V takovém případě by slajd vypadal následovně: (nastav x (+ x 1)) (kdyz (< x 10) (skoc-na-zacatek)). Tento slajd bude opakovat prezentaci dokud nebude x rovno 10. Poté bude pokračovat dál v prezentaci.

Řídící slajd zpracovává nejen předpřipravené příkazy, ale i všechny ostatní lispové příkazy. Tím se stává velmi mocným nástrojem, který by měl být použitý jen pokud uživatel ví, co dělá.

4.1.3 Okno Projektoru

Projektor obsahuje plátno OpenGL, do kterého se vykreslují prvky z OpenGL vytvořené v Editoru. Pro pohyb se scénou slouží myš a klávesnice. Skrolováním myši dojde k přiblížení nebo oddálení ve směru kam se dívá kamera. Pohyb myši spolu se stisknutým levým tlačítkem způsobí rotaci scény. Klávesami W, S, A, D je realizován horizontální posun scény po osách x a y. Klávesami Q a E je provedeno vertikální posunutí po ose y. Okno je možné zavřít a znovu otevřít z menu v okně Editoru. Tím je i zajištěno nastavení scény do původní polohy.



Obrázek 17: Plátno OpenGL

4.2 Programátorská příručka

Pro vytvoření aplikace bylo využito prostředí LispWorks, implementující jazyk Common Lisp. Uživatelské rozhraní je vytvořeno pomocí multiplatformního balíku CAPI, poskytující abstrakční bariéru nad práci s okny. Dále jsou využity některé funkce skrze rozhraní FLI. Pro využití funkcí OpenGL je použita implementace z LispWorks.

4.2.1 CAPI

Program se skládá ze dvou capi:interface: první, hlavní pro editor, a druhé pro plátno s kontextem OpenGL. Okno editoru je tvořeno horním menu složeného ze čtyř nabídek, pod menu pak v levé části je pro navigaci použitý capi:tree-view, který bere data z otevřeného pracovního prostředí workplace, které je uloženo do globální proměnné *workplace*. V pravé části je prostor pro náhled slajdu, ten se generuje dynamicky v závislosti na slajdu zvoleném ve stromu. Při zavření okna OpenGL je uložena jeho pozice a velikost do registrů. Pokud je zavřeno hlavní okno, je ukončena aplikace a je zavřeno i okno OpenGL. V tomto případě jsou uloženy do registrů pozice a rozměry obou oken.

4.2.2 Třída workplace

Hlavní třídou je třída pro pracovní prostředí. Obsahuje slot pro název, který se promítne do názvu souboru při uložení, slot pro aktuální prezentaci a slot content, který nese celý obsah prostředí, tedy prezentace v něm vytvořené. Prostředí tvoří hiearchii tříd prezentací a slajdů. Při vytváření stromu v okně editoru je prostředí zpracováno do seznamu, vhodného pro naplnění stromu.

4.2.3 Třída variable-environment

Třída pro prostředí proměnných je použita v jednotlivých prezentacích a poskytuje možnost vkládat jednotlivé prezentace víckrát za sebe nebo pro použití v cyklech. Zaručí, že každá prezentace bude mít vlastní proměnné a jejich aktuální hodnoty se nebudou míchat mezi jednotlivými prezentacemi. Obsahuje slot pro původní načtenou proměnnou, sloužící pro návrat k původní hodnotě proměnné a je použita při uložení prezentace. Další sloty jsou pro dočasnou hodnotu proměnné, kdy je hodnota v průběhu prezentace změněna. Posledním slotem je historie hodnot, která je použita pro cykly v prezentaci. Do historie se ukládá dočasná proměnná ve funkci (skoc-na-zacatek). Při zobrazování prezentace, která je v cyklu je prezentace zobrazena vždy pro všechny hodnoty v historii. Pro proměnné existují odděleně sloty pro název a hodnoty.

4.2.4 Třída slideshow

Prezentace jsou rozděleny do dvou dříd podle toho, zda je prezentace určena pro vykreslování grafického primitiva nebo se jedná o prezentaci, která obsahuje

prezentace pro grafická primitiva jako prvky slajdu, nebo ostatní prvky OpenGL, které se nevkládají do bloku gl-begin a gl-end. Obě třídy nesou slot pro číslo aktuálního slajdu, jméno prezentace, obsah prezentace, slot určující, zda se jedná o původní prezentaci nebo kopii, a slot pro prostředí, které obsahuje proměnné.

4.2.5 Třída primitive-slideshow

Třída zastupující prezentaci pro kreslení grafického primitiva má slot navíc, který určuje, o jaké grafické primitivum se jedná. Také v metodě opengl-element, obsahuje před samotným procházením obsahu příkaz gl-begin a na konci gl-end. Dále je odlišná metoda capi-element, která v náhledu prvního slajdu prezentace pro primitivum zobrazí rozbalovací nabídku, ve které je možno určit typ grafického primitiva.

4.2.6 Vykreslení prezentace

Zobrazení prezentace je vykonáno zavoláním metody opengl-element s prvním argumentem jako aktuální prezentací a cestou. V případě, že prezentace je kořenová, tedy je přímým potomkem pracovního prostředí, je cesta prázdná. Pokud je prezentace součástí jiné, je cesta vytvořena spojením jména prezentace a aktuálního slajdu z globální proměnné *parent-list*. Pro orientaci v hierarchii je zde proměnná way, která slouží k jednoznačné identifikaci aktuálního slajdu. Děje se tak přidáním názvu prezentace a čísla aktuálního slajdu k cestě, kterou metoda opengl-element dostane. Tuto novou cestu pak při průchodu obsahem prezentace předá jednotlivým prvkům ze svého obsahu. V průběhu procházení jednotlivých slajdů je cesta generována a porovnávána s cestou, která odkazuje na aktuální slajd. Pokud vygenerovaná cesta odkazuje na slajd, který je shodný s cestou aktuálního slajdu, je procházení prezentací zastaveno. Pokud prezentace obsahuje cyklus, je procházení zastaveno až při posledním průchodu cyklu.

4.2.7 Třída slide

Vkládání slajdu do aktuální prezentace probíhá vytvořením nového prázdného slajdu, kde jeho typ je určen prvkem v jeho obsahu. Ten je zvolen hned po vytvoření prázdného slajdu a nelze měnit. Teprve potom je nový slajd zařazen do prezentace. V nabídce jsou možnosti pro vložení před aktuální slajd a na konec prezentace. Vložením slajdu do prezentace dojde k přepočítání slajdů a změně názvů všech slajdů. Název slajdu je vždy jeho pořadové číslo v dané prezentaci. Pokud je jako typ slajdu zvolena jiná prezentace, dojde k vytvoření její kopie a nastavení příznaku, že se jedná o kopii. V prezentaci je vytvořeno nové prostředí, kde jsou zkopírovány původní hodnoty proměnných.

Mazání prvků probíhá odstraněním aktuálního prvku, tedy toho, který je zvolen ve stromu v okně editoru. Pokud je označena prezentace, která je součástí jiné, je odstraněna kopie. Pokud je označena prezentace, která není součástí jiné, dojde k rekurzivnímu odstranění ze všech prezentací, které se nacházejí

v prostředí. Jestliže je označen prvek, který není prezentací, jedná se o slajd a ten je pak odstraněn z aktuální prezentace.

4.2.8 Rozšíření o nové prvky

Jednotlivé prvky OpenGL tvoří samostatné třídy a jejich sloty se liší v závislosti na tom, o jaký příkaz OpenGL se jedná. Pro vytvoření nových prvků OpenGL je potřeba definovat metodu opengl-element, která se stará o příkazy OpenGL, capi-element, která má na starosti zobrazení v náhledu slajdu. Dále metodu update-data, která je volána po změně hodnot v capi a zajistí nastavení hodnot v instanci třídy. Poté metodu to-tree, která je volána při vytváření stromu, metodu print-object, sloužící k zobrazení třídy v čitelné podobě Printerem a také k uložení do souboru. Dále je potřeba upravit funkce read-data-from-slide-preview, kde je třeba přidat podmínku do condu, aby došlo ke správnému přečtení dat z capi a rozšířit funkci left-brack-reader, která se stará o přečtení ze souboru a vytvoření instance dané třídy prvku OpenGL. Pro tyto potřeby je přeprogramován Reader tak, aby četl [ ] syntax, která reprezentuje dané třídy. Tímto způsobem je možné napsat prezentaci v textovém editoru a v programu Demonstrace OpenGL pouze spustit.

```
1  [workplace "nove"
2  ([primitive-slideshow "trojuhelnik" *GL-TRIANGLES*
3    ([slide 0 ([normal "trojuhelnik" 0.0 0.0 1.0])]
4     [slide 1 ([vertex "trojuhelnik" 0.0 0.0 0.0])]
5     [slide 2 ([normal "trojuhelnik" 1.0 0.0 1.0])]
6     [slide 3 ([vertex "trojuhelnik" 1.0 0.0 0.0])]
7     [slide 4 ([normal "trojuhelnik" 1.0 1.0 1.0])]
8     [slide 5 ([vertex "trojuhelnik" 1.0 1.0 0.0])])])]
```

Zdrojový kód 1: Příklad prostředí

4.2.9 Přehled vytvořených tříd

Vertex - zastupuje příkaz gl-vertex4-d, který požaduje čtyři argumenty typu double-float. První tři hodnoty reprezentují souřadnice na osách *x, y* a *z*. Tyto hodnoty jsou zadávány pomocí tří textových polí, která jsou zobrazena v náhledu slajdu v okně editoru. Do textového pole je možno zadat číselnou hodnotu nebo použít proměnnou, která je zadána v rámci prezentace, do které vrchol patří. Čtvrtá hodnota je zadána automaticky. Jedná se o hodnotu 1.0D0, která značí, že daný bod je vlastní a nejedná se o vektor.

Normal - zastupuje příkaz gl-normal3-d, požadující tři argumenty typu double-float. Normálu zadáváme před vrcholem a vždy se vztahuje k následujícímu vrcholu. Tento slajd je nutné vložit pro správné vypočtení osvětlení primitiva. Opět

je reprezentován třemi textovými poli, do kterých je možné zadadat číselnou hodnotu nebo proměnnou.

Color - zastupuje příkaz gl-color4-d se čtyřmi double-float argumenty. Reprezentováno textovými poli s možností vložit číselnou hodnotu nebo proměnnou. První tři hodnoty označují jednotlivé složky barevného spektra tedy R - červenou, G - zelenou a B - modrou. Poslední hodnotou je hodnota alpha, která určuje stupeň průhlednosti. Ve slajdu pro vložení barvy je nastavena na 1.0D0, čili 100% krytí.

Translate - zastupuje příkaz gl-translated, který vyžaduje tři argumenty double-float. Jedná se o hodnoty vyjadřující posunutí po jednotlivých osách *x, y* a *z*. V náhledu slajdu jsou reprezentovány textovými poli, do kterých lze vkládat číselné hodnoty nebo proměnnou.

Scale - zastupuje příkaz gl-scaled. Tento příkaz vyžaduje tři argumenty typu double-float, které jsou zadány jako číselná hodnota nebo název proměnné do textových polí v náhledu slajdu. První hodnota určuje změnu velikosti v ose *x*, druhá v ose *y* a třetí v ose *z*. Hodnoty nazýváme koeficienty a základní hodnotou je 1.0D0. Větší hodnota znamená zvětšení a menší naopak zmenšení objektu. Změnu velikosti můžeme provádět pro každou osu nezávisle.

Rotate - zastupuje příkaz gl-rotated, kde prvním argumentem je velikost úhlu rotace a dalšími argumenty jsou 3 hodnoty, které určují, dle které osy se má báze rotovat. Pro označení osy, která má být použita k rotaci, se použije hodnota 1.0D0 jinak 0.0D0. V náhledu slajdu v okně editoru je úhel zadán do textového pole jako číselná hodnota nebo proměnná. Osa, podle které bude probíhat rotace je vybrána z rozbalovací nabídky, která obsahuje možnosti *x, y* a *z*.

Polygon-mode - doplňuje parametr funkce gl-polygon-mode. Tato funkce má dva argumenty. První argument určuje, zda bude hodnota nastavena na přední, zadní nebo na obě strany polygonu. Druhý argument je pak hodnota, která určuje, jakým způsobem bude polygon zobrazen. V náhledu slajdu je na výběr pouze typ zobrazení, tedy jednotlivé body, linky nebo vyplněný polygon. Slajdem je ovlivňována pouze přední stranu polygonu. Zadní strana je implicitně nastavena na linky.

Shade-model - nastavuje parametr funkce gl-shade-model. V náhledu slajdu je hodnota vybírána z rozbalovací nabídky. Na výběr je volba plochého nebo Goraudova stínování.

Control-slide - je zastoupen textovým polem, do kterého se vkládají připravé funkce "skoc-na-zacatek", "kdyz" a "nastav". Chování těchto funkcí je bezpečné. Do textového pole je však možno zadat libovolné příkazy v syntaxi lispu. Zadáním jiných než připravených funkcí uživatel přebírá zodpovědnost za stav aplikace.

Závěr

Cílem této bakalářské práce bylo vytvoření programu pro doplnění popularizačních přednášek doc. RNDr. Michala Krupky, PhD., které vede na středních školách. Prezentační program měl být jednoduchý na použití, protože je poskytován samotným žákům, a zároveň měl do detailu vysvětlit, jak je vykreslení scény provedeno. Přesný vzhled ani funkce nebyly na začátku vypracování známé, a proto program prošel mnoha změnami, než získal svou finální podobu. Střídaly se fáze, kdy byl program rozdělen na dvě samostatná okna a nebo vše v jednom okně. Experimentovali jsme se samotným umístěním ovládacích prvků. Knihovna OpenGL obsahuje spoustu nastavení a ve snaze o předvedení co největšího počtu prvků se aplikace stávala nepřehlednou. Poslední verze je složena ze dvou oken. První zobrazuje pouze vykreslovanou scénu a druhé okno je složeno ze dvou hlavních částí. První je strom, který reprezentuje jednotlivé prvky scény, a ve druhé části se nachází grafický náhled funkce OpenGL. Každá scéna je koncipována jako prezentace, kde jednotlivé příkazy OpenGL jsou zastoupeny slajdy. Celou scénu je pak možno postupně příkaz po příkazu vykreslovat, čímž byl splněn požadavek na jednoduchost a názornost. Bohužel kvůli tomu, že dlouho nebyla výsledná podoba programu známá, nejsou ve výsledné verzi některé prvky, které byly funkční v průběžných verzích. Mezi tyto prvky patří manipulace se světly či materiály a jejich nastavením. V programu je pouze jedno přednastavené světlo. Dále byla plánována možnost okamžitého vykreslení prezentace stiskem klávesy. Také chybí prostředky pro práci s texturami. Tyto prvky jsou však snadno doplnitelné, protože aplikace byla pro rozšíření navržena. V konečné verzi programu se nacházejí prvky, které v zadání nebyly, ale jejich použití významně rozšiřuje jeho schopnosti. Jedná se o práci s proměnnými a jejich využití pro vytvoření cyklů.

Během zpracování práce jsem se seznámil s pokročilejšími možnostmi jazyka Common Lisp a jeho vývojovým prostředím LispWorks, jako je programování Readeru, rozhraní pro volání funkcí v jiném jazyce, vytvářením a ukládáním hodnot do registrů Windows a v neposlední řadě se samotnou knihovnou OpenGL. Knihovna OpenGL je nesmírně složitá a rozsáhlá. Aplikace Demonstrace OpenGL pokrývá pouze zlomek funkcí v knihovně a využívá pro své potřeby vyhovující avšak zastaralou verzi OpenGL. Během studia OpenGL jsem zjistil, že implementace do prostředí LispWorks je velmi neflexibilní vůči vývoji samotné knihovny OpenGL, a domnívám se, že významným krokem v rámci použití a rozšíření LispWorks pro vývoj 3D aplikací by bylo její reimplementování.

Conclusions

The aim of this thesis was to create a program to supplement the popularization lectures of doc. RNDr. Michael Krupka, PhD., being lectured to secondary schools. On the one hand, the presentation program should be simple to use because it is provided to learners, on the other hand, it had to explain in detail how the scenes rendering is performed. The exact appearance and functions were not known at the beginning of the devising and therefore, the program had undergone many changes before it reached its final form. We were experimenting with the actual location of the controls. OpenGL library contains a lot of settings and in an effort to demonstrate a large number of elements, the application became confusing. At the end the latest version consists of two windows. The first one shows only rendering scenes and the second one is composed of two main parts. The first part is a tree that represents the individual elements of the scene, and the second part is a graphical preview of OpenGL functions. Each scene is designed as a presentation where each OpenGL command is represented by slide. The whole scene can be then gradually drawn step by step, which meets the requirement for simplicity and clarity. Unfortunately, due to the fact that the final form of the program was not known for a long time, they are not some elements in the final version, which had been working in earlier versions. These elements include handling lights or materials and their settings. The program has only a single preset light. Furthermore, the planned option to immediately render presentation by one click is also working just partially. There are also missing the resources to work with textures. These elements, however, are easy to add because the application has been designed for expansion. In the final version of the program there are elements that were not in the assignment, but their use significantly expands ability of this program. It is the work with variables and their use to create cycles.

During processing I became acquainted with more advanced capabilities of Common Lisp and its development environment LispWorks like Reader programming, interface for function calls in another language, creating and storing values in the Windows registry and finally the actual library OpenGL. OpenGL library is extremely complex and extensive. Demonstration of OpenGL application covers only a fraction of the functions in the library and uses for its needs a satisfying but outdated version of OpenGL. During my OpenGL studies I found out that the implementation into LispWorks is very inflexible towards development OpenGL library itself and I believe that an important step in the use and expansion of LispWorks for development of 3D applications would be its re-implementing.

5 Obsah přiloženého CD/DVD

Na samotném konci textu práce je uveden stručný popis obsahu přiloženého CD/DVD, tj. jeho závazné adresářové struktury, důležitých souborů apod.

bin/
: Program Demonstrace OpenGL, spustitelný přímo z CD. Adresář obsahuje i všechny runtime knihovny a další soubory potřebné pro bezproblémový běh programu z CD.

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

src/
: Kompletní zdrojové texty programu Demonstrace OpenGL se všemi potřebnými (příp. převzatými) zdrojovými texty, knihovnami a dalšími soubory potřebnými pro bezproblémové vytvoření spustitelných verzí programu.

Navíc CD/DVD obsahuje:

data/
: Ukázková a testovací data použitá v práci a pro potřeby testování práce při tvorbě posudků a obhajoby práce.

install/
: Instalátory aplikací, runtime knihoven a jiných souborů potřebných pro vývoj programu Demonstrace OpenGL, které nejsou standardní součástí operačního systému určeného pro vývoj programu.

U veškerých cizích převzatých materiálů obsažených na CD jejich zahrnutí dovolují podmínky pro jejich šíření. Pro všechny použité (a citované) materiály, u kterých toto není splněno a nejsou tak obsaženy na CD/DVD, je uveden jejich zdroj v bibliografii nebo textu práce.

Literatura

[1] NĚMEC, Martin (sest.). *Popis grafické knihovny Blender: metodický materiál pro vyuku modelovacího programu Blender* [online]. Ostrava: Němec, Martin, 2014 [cit. 2014-12-12]. Dostupný z: ⟨http://www.blender.vsb.cz/zpg/prednaska2.pdf⟩.

[2] *Phong reflection model. : Computer graphic algorhitms* [online]. 2015 [cit. 2015-07-25]. Dostupný z: ⟨http://en.wikipedia.org/wiki/Phong_reflection_model⟩.

[3] SEIBEL, Peter. *Practical Common LISP*. First. 2005. D, 500 s. ISBN 1590592395.

[4] SELLERS, Graham. *OpenGL SuperBible*. Sixth. 2013. DCCCXLVIII, 848 s. ISBN 0321902947.

[5] *LispWorks User Guide and Reference Manual*. [online]. 2015 [cit. 2015-07-23]. Dostupný z: ⟨http://www.lispworks.com/documentation/lw61/LW/html/lw.htm⟩.

[6] *OpenGL Programming. : Computer programming libraries* [online]. [cit. 2015-03-01]. Dostupný z: ⟨http://en.wikibooks.org/wiki/OpenGL_Programming⟩.

[7] *OpenGL Software Development Kit*. [online]. 2015 [cit. 2015-02-15]. Dostupný z: ⟨http://www.opengl.org/sdk/docs/⟩.