VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

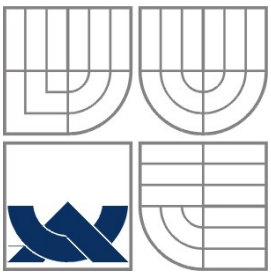
3D PROJEKCE FOTEK V OPENGL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

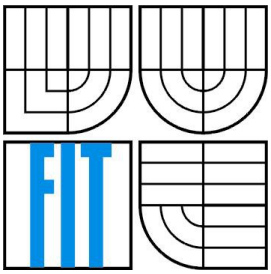
AUTOR PRÁCE
AUTHOR

MARIÁN ONDREJÍČEK

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D PROJEKCE FOTEK V OPENGL

3D PHOTO SLIDESHOW USING OPENGL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARIÁN ONDREJÍČEK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. MICHAL SEEMAN

BRNO 2010

Abstrakt

Účelem této práce bylo navržení spůsobu projekce fotek v 3D prostoru za pomocí OpenGL a implementování těchto technik ve formě fungující aplikace. Opisovány jsou také principy Qt toolkitu, ve kterém je naprogramováno uživatelské rozhraní programu. Dále jsou vysvětleny základy interpolace kvaterniónů, které jsou využity při animaci fotek a tvoří jádro kamerového systému. Druhá část je věnována praktické realizaci a popisu jednotlivých částí projektu. Aplikaci se podařilo úspěšně implementovat. V závěru jsou zhodnoceny dosažené výsledky a nastíneny možnosti budoucího vývoje.

Abstract

The aim of this thesis was to design a photo slideshow in 3D space using OpenGL and to implement functional application. Also there is description of Qt toolkit, which was used to design graphical user interface of the program. Furthermore there are explained basics of quaternion interpolation, which is the core of animation and camera system. The second part describes practical realization and each part of the project in details. Application was successfully implemented. In the end, the results are discussed, including possibilities of future development.

Klíčová slova

OpenGL, Qt, CML, 3D projekce, prezentace, čočky, rotace, zoom, kvaterniony, interpolace

Keywords

OpenGL, Qt, CML, 3D projection, slideshow, lenses, rotation, zoom, quaternions, interpolation

Citace

Marián Ondrejíček: 3D projekce fotek v OpenGL, bakalářská práce, Brno, FIT VUT v Brně, 2010

3D projekce fotek v OpenGL

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Michala Seemana. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Marián Ondrejíček
17.05.2010

Poděkování

Chcel by som sa poďakovať svojmu vedúcemu Ing. Michalovi Seemanovi za odbornú pomoc a konzultácie, ktoré mi pomohli pri tvorbe tejto práce.

Obsah

1 Úvod

V dnešnej dobe sú digitálne fotoaparáty cenovo dostupné pre bežných užívateľov a tak možnosť tvorby kvalitných fotografií už nie je záležitosťou len profesionálnych fotografov. Proces fotografovania však nekončí vytvorením snímky a prípadnými úpravami. Vytvorené fotky je treba skôr či neskôr zobraziť, či už v papierovej forme, na displeji alebo pomocou projektoru. Zobrazovanie na počítači nám dáva veľa možností, ako tento proces urobiť zaujímavejším. Účelom tejto práce je práve tento posledný krok, teda efektné prezentovanie fotiek pomocou počítačovej aplikácie.

Výbavou každého desktopového počítača či notebooku býva v dnešnej dobe grafická karta, ktorá dokáže akcelerovať grafické operácie a vykonávať ich efektívnejšie ako procesor. Mojim cieľom je navrhnúť a implementovať nástroj na tvorbu a prehrávanie sekvencií z fotiek, ktorý umožňuje základné funkcie ako zoom a rotácia pozorovateľa. Toto všetko za použitia 3D počítačovej grafiky. Preto je vhodné využiť možnosti, ktoré grafické karty ponúkajú a vykonávať čo najviac výpočtov pomocou nich. Vo svete existuje niekoľko podobných nástrojov. Hlavnou odlišnosťou tejto aplikácie je, že v procese zobrazovania využíva optické vlastnosti šošoviek. Toto umožňuje vytvárať panoramatické sekvencie, ktoré nadobúdajú väčší dojem reálnosti.

Tento dokument popisuje proces tvorby spomínanej aplikácie od počiatočnej teoretickej prípravy, cez implementáciu, až po finálny produkt a testovanie. Po úvode, v druhej kapitole, zhodnocujem súčasný stav programov, ktoré disponujú podobnou funkčnosťou. Tretia kapitola je zameraná na viac-menej teoretický popis knižníc, ktoré aplikácia používa. Prvou z nich OpenGL, čo je multiplatformové API na prácu s 2D a 3D grafikou, ktoré umožňuje výpočty akcelerovať cez grafickú kartu. Keďže cieľom je vyvinúť aplikáciu s príjemným užívateľským rozhraním, musel som zvoliť vhodný nástroj na jeho tvorbu. Naša aplikácia k tomuto účelu používa Qt toolkit. Základné princípy tohto toolkitu sú vysvetlené v rovnakej kapitole. Stručne sa zmienim aj o knižnici CML, ktorá umožňuje výpočty s kvaterniónmi. Tie používam pri rotáciách pozorovateľa a v animačnom systéme, pomocou ktorého je určovaná poloha fotiek v priebehu prehrávania. Štvrtá kapitola vysvetľuje základnú teóriu, ktorú bolo potrebné preskúmať pred samotnou implementáciou. Toto zahŕňa už spomenuté kvaternióny, ďalej interpoláciu, princíp zoomovania a základy projekcie v 3D priestore. Piata kapitola je venovaná samotnej implementácii. Popisujem jednotlivé triedy a dátové štruktúry našej aplikácie, pričom začínam tými najdôležitejšími a postupne prechádzam k detailom. V šiestej kapitole je stručne popísané užívateľské rozhranie a jeho ovládanie. V predposlednej kapitole testujem aplikáciu z hľadiska použiteľnosti v reálnom prostredí. Zameriavam sa predovšetkým na pamäťovú náročnosť. V záverečnej kapitole hodnotím prácu a aplikáciu ako celok, pričom naznačujem možnosti budúceho vývoja.

2 Súčasný stav

Vo svete existuje pomerne mnoho nástrojov na tvorbu 3D prezentácií z fotiek, či už komerčných alebo voľne dostupných. Mal som možnosť vyskúšať niektoré z nich. V tejto kapitole stručne opisujem vybrané programy, poukazujem na odlišnosti voči našej aplikácii a hľadám spoločné znaky.

Aleo 3D Flash Slideshow Gallery Maker

Web: http://www.aleosoft.com/flashslideshowmaker/

Tento nástroj je jedným z množiny grafických nástrojov od firmy Aleo Software. Slúži na tvorbu galérií vo formáte Adobe Flash a je vhodný najmä pre webové prezentácie. Galérie dokáže vytvárať nielen z fotiek, ale aj z videosúborov. Implementuje nástroje pan a zoom ako aj naša aplikácia. Možnosť exportu vo forme videa je taktiež vítaná vlastnosť. Nevýhodou je, že sa jedná o komerčný produkt. Jednotlivé nástroje je však možné podľa potreby kupovať aj oddelene.

Karsten SlideShow

Web: http://karsten.sourceforge.net/

Jedná sa o open-source projekt, ktorý je napísaný v Delphi. Program dokáže tvoriť prezentácie z obrázkov alebo z videosúborov. Jeho prednosťou je, že sekvencie umožňuje zobrazovať aj na pracovnej ploche alebo formou šetriča obrazovky. S takto hlbokou integráciou do systému však súvisí aj obmedzenie. Program požaduje k svojmu behu platformu MS Windows.

Imagination

Web: http://imagination.sourceforge.net/

Táto aplikácia predstavuje komplexný nástroj na tvorbu prezentácií z fotiek so zameraním pre nosiče DVD. Disponuje veľkou databázou prevažne 2D efektov. Príjemným spestrením je možnosť obohatiť prezentáciu o hudbu na pozadí. Nevýhodou môže byť, že program beží len na operačnom systéme Linux.

SMILE

Web: http://smile.tuxfamily.org/

SMILE alebo Slideshow Maker In Linux Environment, ako už názov napovedá, je nástroj na tvorbu videosekvencií pre prostredie Linux. Z hľadiska porovnania s našou aplikáciou, sa jedná o najviac podobný program, aký som mal možnosť vyskúšať. Fotky je možné ľubovoľne umiestňovať do 3D

priestoru. Podporuje niekoľko efektov a prechodov medzi slajdami. Výslednú prezentáciu je možné vyrenderovať a uložiť vo zvolenom video formáte.

Po vyskúšaní týchto aplikácií mi bolo jasné, ktoré vlastnosti by mal mať kvalitný nástroj na tvorbu sekvencií z fotiek a taktiež som narazil na ich obmedzenia a nedostatky. Pozitívne hodnotím užívateľské rozhranie, ktoré je intuitívne a často podporuje rôzne vylepšenia ako napríklad drag & drop. Za najväčší nedostatok považujem nízku podporu rôznych operačných systémov. Napriek tomu, že existujú aplikácie pre MS Windows aj Linux, väčšina z nich je stavaná iba pre jeden zo spomínaných systémov. Preto som si okrem požadovaných bodov zadania za cieľ stanovil aj prenositeľnosť medzi rôznymi platformami. Aplikácia je vyvíjaná pre prostredie Linux aj MS Windows. S tým súvisí aj výber použitých knižníc. Tie opisujem v následujúcej kapitole.

3 Použité knižnice

Pri tvorbe každého rozsiahlejšieho softvérového projektu sa nezaobídeme bez použitia externých knižníc. V našej aplikácii som ich využil hneď niekoľko. Prvou z nich je knižnica OpenGL, ktorá slúži na prácu s grafiku a bola požadovaná samotným zadaním. Ďalšie knižnice som si mohol zvoliť. Na matematické výpočty som použil knižnicu CML. Na prácu so šošovkami mi vedúci práce Ing. Michal Seeman poskytol sadu tried a metód. Grafické užívateľské rozhranie bolo vytvorené pomocou rozsiahleho toolkitu Qt, ktorý som použil aj na načítavanie fotiek, ukladanie súborov a mnohé ďalšie činnosti. Prečo som sa rozhodol pre tieto knižnice a aké základné princípy sú v nich uplatňované, to všetko je vysvetlené v tejto kapitole. Vychádzal som pritom zo zdrojov [1], [5], [6] a [9].

3.1 OpenGL

OpenGL (Open Graphics Library) je multiplatformové API na prácu s 2D a 3D grafikou s podporou hardvérovej akcelerácie pomocou grafickej karty. Bolo vyvinuté firmou Silicon Graphics v roku 1992. Aktuálna špecifikácia vo verzii 4.0 bola publikovaná v marci 2010. Knižnica OpenGL bola pôvodne napísaná v jazyku C, ale v súčasnosti je portovaná do väčšiny populárnych programovacích jazykov ako napríklad C++, Java, Python, C# a mnohé ďalšie.

3.1.1 Scéna v OpenGL

Vytvorenie a vykreslenie scény je neoddeliteľnou súčasťou našej aplikácie. Následujúce kroky stručne popisujú proces, akým to je realizované v OpenGL. Podľa [1], strana 21:

1. *„Skonštruovanie tvarov zo základných geometrických primitív a vytvorenie matematického popisu objektov v scéne.*
2. *Rozmiestnenie objektov do scény a nastavenie pozície kamery, teda pozorovateľa. Určí sa tzv. viewpoint – bod, z ktorého bude scéna pozorovaná.*
3. *Spočítanie farby objektov. Farba je určená priamo, osvetlením, alebo je daná nanesenou textúrou.*
4. *Prevod matematického popisu objektov a ich farieb na pixely. Tento proces sa v počítačovej grafike nazýva rasterizácia.*
5. *Spočítané pixely sú prenesené do video bufferu a vykreslené na obrazovku.“*

3.1.2 Stavový automat

OpenGL pracuje ako stavový automat, ktorý nastavuje a mení stavové premenné. Pomocou volania príslušných funkcií prechádza do rôznych stavov, ktoré ostávajú platné až do ďalšej zmeny. Najjednoduchším príkladom tejto myšlienky je zmena farby. Farba je stavová premenná. Pri zmene farby sa nová hodnota uloží a všetky objekty od tohto okamžiku sú vykresľované novou farbou až do ďalšej zmeny. Na zmenu stavových premenných poskytuje OpenGL sadu funkcií. Logické premenné možno meniť pomocou `glEnable()` a `glDisable()`. Aktuálne hodnoty premenných je možné získať, až na niekoľko výnimiek, použitím funkcií tvaru `glGet*()` v závislosti na type, napríklad `glGetBooleanv()`, `glGetDoublev()`, `glGetFloatv()`.

3.1.3 Transformačné matice

Transformácie v OpenGL sú reprezentované násobením matíc. Toto zahŕňa operácie ako rotácia, zmena mierky, posunutie, ale aj výpočet perspektívnej projekcie. OpenGL disponuje sadou funkcií, ktoré vykonávajú prácu s maticami za nás. Nastali však aj prípady (trieda `Camera`), kedy som sa nevyhol vytváraniu vlastných matíc. Preto bolo dôležité pochopiť, akým spôsobom sa s maticami pracuje.

Transformačné matice majú rozmer 4 x 4, skladajú sa teda zo 16 čísel, ktoré sú typu `float`. V prípade že deklarujeme maticu ako dvojrozmerné pole, musíme mať na pamäti, že OpenGL indexuje jednotlivé prvky inak ako je tomu v C/C++. Všetky OpenGL funkcie používajú tzv. column-major poradie, zatial čo jazyk C++ uvažuje tzv. row-major poradie. Odlišnosť ilustruje následujúci príklad:

$$M_C = \begin{bmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{bmatrix} \qquad M_{OpenGL} = \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix}$$

Problém nenastane, ak maticu deklarujeme ako jednorozmerné pole s veľkosťou 16 prvkov.

Medzi najzákladnejšie funkcie patrí `glLoadIdentity()`, ktorá nastaví počiatočný stav - nahradí aktuálnu maticu jednotkovou maticou. Potom môžeme vykonávať požadované operácie, napríklad rotácie alebo posuny pomocou príslušných OpenGL funkcií, ktoré vytvárajú matice a násobia ich s aktuálnou za nás. Ak vytvárame vlastnú maticu, tak na vynásobenie použijeme funkciu `glMultMatrix()`. Túto možnosť som využil pri implementácii kamery.

Pri tvorbe zložitejších scén potrebujeme často pracovať s viacerými maticami. OpenGL ich umožňuje ukladať do zásobníka. Zásobníky sú štyri - modelview, projection, texture a color. Funkcie

na prácu s nimi sú rovnaké, preto je treba najprv určiť, ktorý zásobník sa bude používať. K tomu výberu slúži funkcia `glMatrixMode()`. Najčastejšie vykonávame operácie nad modelview zásobníkom, ktorý umožňuje manipulovať so scénou a objektami v nej. Zmestí sa do neho až 32 matíc, prípadne viac, v závislosti na implementácii. Na uloženie aktuálnej matice sa používa `glPushMatrix()`, ktorá ju skopíruje na vrchol zásobníka. K odstráneniu slúži funkcia `glPopMatrix()`. Zásobníky bývajú implementované hardvérovo, preto sú operácie s nimi efektívne.

3.1.4 OpenGL vs. DirectX

Za hlavného konkurenta je považované API DirectX od firmy Microsoft. Toto API dominuje v prostredí MS Windows, kde sa používa hlavne v oblasti počítačových hier. Na rozdiel od OpenGL, DirectX neslúži len na prácu s grafikou, ale zahŕňa aj podporu pre vstupné zariadenia, spracovanie zvuku a komunikáciu po sieti. Poskytuje tak kompletný balík pre vývoj multimediálnych aplikácií. Ak sa rozhodneme použiť OpenGL, tak sa nezaobídeme bez použitia externých knižníc. Na druhej strane, programy napísané v OpenGL sú bez problémov prenositeľné na rôzne platformy, čo je rozhodujúce pre aplikácie ako je tá naša.

3.2 Qt

Keďže mojou úlohou bolo naprogramovať desktopovú aplikáciu, musel som si zvoliť knižnicu, ktorá umožňuje tvorbu užívateľských rozhraní. V súčasnej dobe medzi najpoužívanejšie multiplatformové toolkity patria Qt, GTK+ a wxWidgets. Rozhodol som sa použiť knižnicu Qt, pretože je v aktívnom vývoji, obsahuje kvalitnú online dokumentáciu, existuje k nej príjemné vývojové prostredie Qt Creator a je v nej zahrnutá priama podpora pre OpenGL. V tejto časti popíšem základné princípy, ktoré sú uplatňované v každej Qt aplikácii.

3.2.1 Meta-Object systém

Tento systém umožňuje použiť princípy a konštrukcie, ktoré nie sú priamo podporované v programovacom jazyku C++. Toto zahŕňa signály a sloty na komunikáciu medzi objektami, dynamické vlastnosti objektov, ktoré je možné pridávať a odoberať za behu aplikácie. Systém sa skladá z niekoľkých základných častí. Trieda `QObject` tvorí bázovú triedu pre všetky objekty, ktoré využívajú meta-object systém. Makro `Q_OBJECT` sa používa v deklaráciách tried, ktoré používajú dynamické vlastnosti alebo signály a sloty. Qt obsahuje vlastný kompilátor, tzv. moc (Meta-Object Compiler). Počas kompilácie prechádza zdrojový kód a hľadá v ňom triedy, ktoré používajú makro

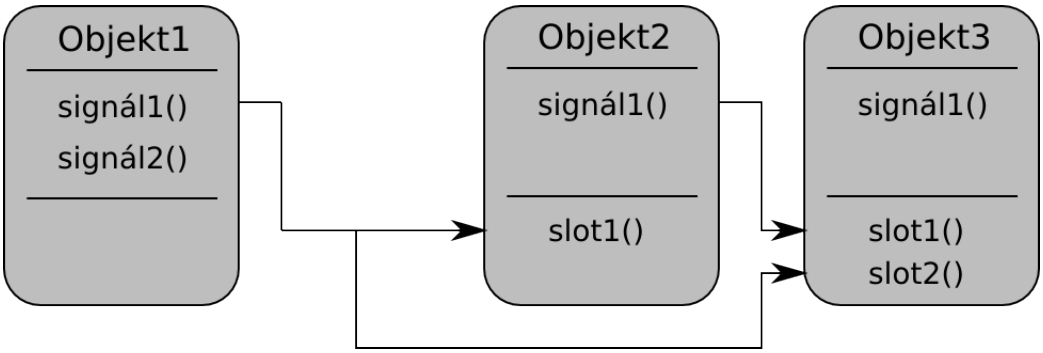
Q_OBJECT. Všetky tieto triedy spracuje tak, že kód špecifický pre Qt nahradí čistým C++ kódom. Vygeneruje nové súbory, ktoré je už možné skompilovať štandardným C++ prekladačom.

3.2.2 Signály a sloty

Signály a sloty slúžia na komunikáciu medzi rôznymi objektmi alebo aj v rámci jedného objektu. Všetky objekty, ktoré ich používajú, musia byť potomkami základného objektu **QObject**. Syntakticky sa deklarácia signálu a slotu podobá klasickej metóde.

```
signals:
      void mySignal(int value);
public slots:
      void mySlot(int value);
```

Sloty musia obsahovať definíciu svojho chovania, podobne ako ostatné metódy. Signály neobsahujú definíciu, sú iba deklarované. Pomocou signálu je možné prenášať rôzne hodnoty, formou parametrov. Na spojenie signálu so slotom slúži funkcia **connect()**.



Obrázok 1: Príklad možného prepojenia pomocou signálov a slotov

Signál je vyslaný pomocou kľúčového slova **emit**, napríklad **emit mySignal(10)**. Toto spôsobí zavolanie príslušného slotu a vykonanie jeho kódu. Signál je možné napojiť na viac slotov. Umožňuje to vytvárať čistejšie komunikačné rozhranie medzi objektami. Koncept signálov a slotov je používaný takmer v každej aplikácii postavenej na knižnici Qt.

3.2.3 Viacjazyčnosť

Qt má pokročilú podporu pre programovanie viacjazyčných aplikácií. Na uchovávanie reťazcov sa používa trieda **QString**. Reťazec sa skladá zo znakov, ktoré sú typu **QChar**. Tieto sú 16-bitové a

odpovedajú znakom Unicode 4.0. Umožňujú tak používať rôzne národné abecedy. Užívateľské rozhranie našej aplikácie je primárne napísané v angličtine, s možnosťou zmeny jazyka na slovenský alebo český.

Jazyková podpora je zabudovaná už aj v samotnej triede **`QObject`** a teda aj vo všetkých triedach, ktoré od nej dedia. Obsahuje metódu **`tr()`**, ktorá ako parameter požaduje reťazec **`QString`** a vracia jeho lokalizovanú verziu podľa nastaveného jazyka. Ak neexistuje preklad, tak vracia nezmenený reťazec. Je vhodné používať túto metódu pri všetkých reťazcoch, ak plánujeme aplikáciu lokalizovať. Vytvorenie samotného prekladu prebieha v troch krokoch:

1. Pridanie ciest k jazykom do projektového súboru.
2. Spustenie programu lupdate, ktorý prečíta zdrojové kódy a vyhľadá v nich všetky preložiteľné reťazce. Výstupom sú súbory s príponou „ts“, ktorých cesty boli špecifikované v predchádzajúcom kroku. Jedná sa o XML súbory. Tie je možné otvoriť v editore Qt Linguist, ktorý umožňuje preložiť reťazce do požadovaných jazykov.
3. Po preložení textu je treba spustiť program lrelease, ktorý vytvorí binárne súbory s príponou „qm“. Tieto sa už dajú použiť v aplikácii.

Na prepínanie medzi jazykmi slúži trieda **`QTranslator`**. Aplikácii je treba predať ukazovateľ, aby vedela odkiaľ sa budú brať textové reťazce. K tomu sa používa metóda **`installTranslator()`** triedy **`QApplication`**. Toto sa typicky robí v konštruktore hlavného okna, pretože potrebujeme mať inštanciu triedy **`QTranslator`** dostupnú počas celého behu programu. Ak chceme zmeniť jazyk, použijeme jej metódu **`load()`**, ktorá ako parameter berie cestu k súboru s prekladom (qm súbor). Voľbu jazyka je vhodné uložiť a načítať pri spustení aplikácie, napríklad pomocou triedy **`QSettings`**. Dôležitou funkciou je **`retranslateUi()`**, ktorá okamžite aktualizuje reťazce v užívateľskom rozhraní. Ak by sme ju po výbere jazyka nezavolali, tak by sa zmena prejavila až po reštarte aplikácie.

3.3 CML

CML (Configurable Math Library) je open-source matematická knižnica pre C++. Je založená na šablónach. CML je primárne určená pre aplikácie pracujúce s grafikou a pre hry. Poskytuje API pre výpočty s vektormi, maticami a kvaterniónmi. Naša aplikácia ju využíva k interpolácii medzi dvoma kvaterniónmi počas animácie.

4 Teória

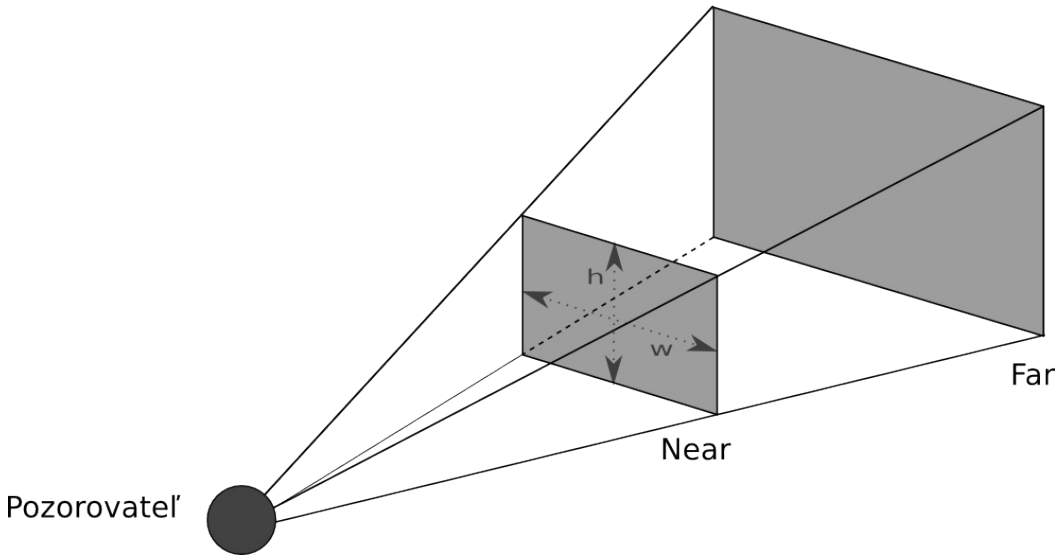
Vývoj tejto aplikácie by nebol možný bez teoretickej prípravy. Počas návrhu som si musel ujasniť, aké princípy a algoritmy použijem. Táto kapitola popisuje základnú teóriu a dôležité postupy, ktoré sa uplatňujú v našej aplikácii. Pri písaní som použil zdroje [2], [3] a [4].

4.1 Projekcia v 3D priestore

Pri práci s 2D grafikou stačí previesť objekty na formát, ktorému rozumie zobrazovacie zariadenie a daný výsledok vykresliť. Situácia s 3D grafikou je zložitejšia. Súčasné monitory pracujú v 2D režime, preto je potrebné previesť všetky trojrozmerné objekty do 2D priestoru. K tomu prevodu slúži tzv. projekcia. Ako sa píše v [2], strana 66: *„Projekcia je transformácia, ktorá realizuje redukciu dimenzie 3D priestoru na 2D priestor pri zachovaní parametrov použitého zobrazenia.“* Týmto prevodom sa síce stráca určitá matematická informácia, ale z hľadiska zobrazenia na monitore sa výsledok javí ako keby sa nachádzal v 3D priestore. Medzi základné druhy projekcie patrí paralelná a perspektívna projekcia. My sa budeme zaoberať druhou spomenutou.

Definícia podľa [2], strana 69: „*Perspektívna projekcia je nelineárna projekcia, ktorá zobrazuje vrcholy premietaných objektov prostredníctvom lúčov pretínajúcich sa v jednom bode, v strede projekcie. Objekty, ktoré sa nachádzajú bližšie k priemetne, sa javia ako väčšie a opačne. Táto projekcia nezachováva rovnobežnosť hrán.“* Perspektívna projekcia sa používa všade tam, kde je cieľom zobrazovať 3D objekty spôsobom, akým ich vníma ľudské oko v reálnom svete. Toto zahŕňa virtuálnu realitu, počítačové hry a aj našu aplikáciu.

S projekciou úzko súvisí pohľadový objem (viewing volume). Je to časť priestoru, ktorá obsahuje viditeľné objekty. Je zbytočné vykresľovať tie, ktoré nebudú zobrazené. Preto sú všetky objekty mimo pohľadový objem vynechané a objekty na jeho hrane sú orezané. Tvar pohľadového objemu závisí na zvolenej projekcii. V prípade perspektívnej projekcie sa jedná o zrezaný ihlan. Šírka záberu kamery závisí na uhle pri vrchole ihlanu. Štandardne sa veľkosť uhla pohybuje v rozsahu 40 až 60 stupňov. Pohľadový objem je ohraničený prednou (near) a zadnou (far) obmedzujúcou rovinou. Ich úloha je vysvetlená v [3], strana 316: „*Obmedzujúce roviny pri orezaní zaisťujú odstránenie príliš blízkych objektov, ktoré bránia vo výhľade a príliš ďalekých objektov, ktoré sú z hľadiska pozorovateľa nezaujímavé a ktorých spracovanie spomaľuje proces zobrazovania. Niekedy býva priestor zobrazovaných objektov transformovaný tak, že sa zmení jeho mierka, aby sa celý pohľadový objem vošiel do jednotkovej kocky. Tým sa zjednodušia výpočty v orezávacích algoritmoch.“*



Obrázok 2: Pohľadový objem při použití perspektívnej projekcie

4.2 Rotácie

Horizontálny pohyb alebo rotácia kamery okolo osy Y býva niekedy označovaný ako panning. Pohyb vo vertikálnom smere, okolo osy X, sa nazýva tilting. V našej aplikácii je možné vykonať horizontálny pohyb, vertikálny pohyb alebo ich kombináciu. Vo výsledku tieto nástroje simulujú posunutie snímky na obrazovke, resp. zmenu pohľadu pozorovateľa, podľa toho, čo považujeme za pevný bod. Posledným typom rotácie je otáčanie okolo vlastnej osy. Všetky tri spomenuté možnosti sa dajú v našej aplikácii kombinovať, čo dovoľuje umiestniť fotku do polohy, ktorá vyhovuje užívateľovi.

Rotácie v trojrozmernom priestore sú v OpenGL reprezentované maticami o veľkosti 4x4. Matematický zápis je následovný:

$$R_X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_Y = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_Z = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ľubovolná rotácia je definovaná osami X, Y, Z a uhľom. V OpenGL sú rotácie reprezentované transformačnými maticami. Takéto vyjadrenie je vo väčšine prípadov postačujúce, avšak pri zložitejších transformáciách (rotácia okolo viacerých osí súčasne, napríklad v animácii) nie je ich použitie intuitívne. Navyše obsahujú nadbytočné údaje a sú ťažko interpolovateľné. Preto som sa rozhodol použiť kvaternióny. Tie sú síce zložitejšie na implementáciu, ale ich použitie je intuitívnejšie, najmä pri zložitejších rotáciách. V našej aplikácii sú operácie panning, tilting aj otáčanie okolo vlastnej osi vykonávané jednotným spôsobom – ako rotácia pomocou kvaterniónov.

4.3 Kvaternióny

Kvaternióny sú rozšírením komplexných čísel. Boli definované írskym matematikom W. R. Hamiltonom v roku 1843. Zaujímavosťou je, že keď ich Hamilton vynašiel, tak práve odpočíval pod mostom Brougham Bridge v rodnom Dubline. Aby spečatil tento významný objav, tak pomocou vreckového nožíka vytesal do skaly ich základnú rovnicu [4]:

$$i^2 = j^2 = k^2 = ijk = -1$$

V praxi nachádzajú uplatnenie v počítačovej grafike, kde sa používajú pri animačných úlohách. Umožňujú efektívne realizovať rotácie v 3D priestore. Pre naše potreby si vystačíme s jednotkovými kvaterniónmi. Kvaternión obsahuje jednu reálnu a tri imaginárne zložky. Je ho možné zapísať ako lineárnu kombináciu prvkov 1, i, j, k:

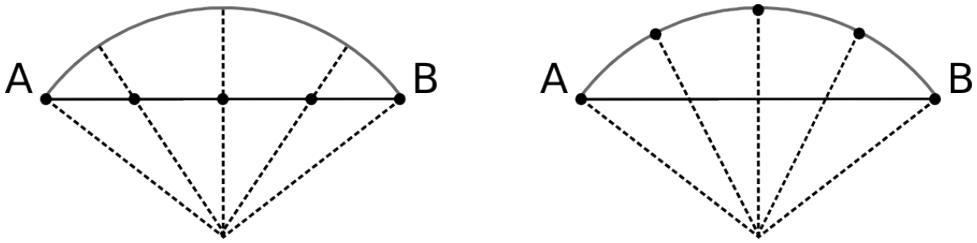
$$q = w + ix + jy + kz$$

Pre reálne použitie je potrebné nadefinovať sadu operácií ako sčítanie, násobenie a normalizáciu. Tieto všetky operácie sú však už implementované v matematickej knižnici CML, ktorú naša aplikácia využíva. Súčasné grafické karty ale nedokážu priamo pracovať s kvaterniónmi. Preto výsledný rotačný kvaternión musíme pred použitím previesť na transformačnú maticu pre OpenGL:

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

4.4 Interpolácia

Kvaternióny bývajú najčastejšie v praxi využívané práve k interpolácii. Interpoláciou je myslený prechod z polohy A do polohy B za určitý čas. Požadovanou vlastnosťou je plynulosť tohto prechodu. Najjednoduchší spôsob interpolácie je posun s konštantným krokom po úsečke. Je jednoduchý na implementáciu a výpočtovo nenáročný. V našom prípade ho však nie je možné použiť, pretože rotácia pozorovateľa opisuje kružnicu. Pohyb by nebol vo všetkých úsekoch rovnako plynulý. Musíme sa posúvať s určitým krokom po kružnici, teda rotovať s konštantným uhľom. Rozdiel v interpolácii po úsečke a po kružnici je znázornený na obrázku. Situácia je prezentovaná v dvojrozmernom priestore, avšak problém je analogický aj pri viacerých dimenziách.

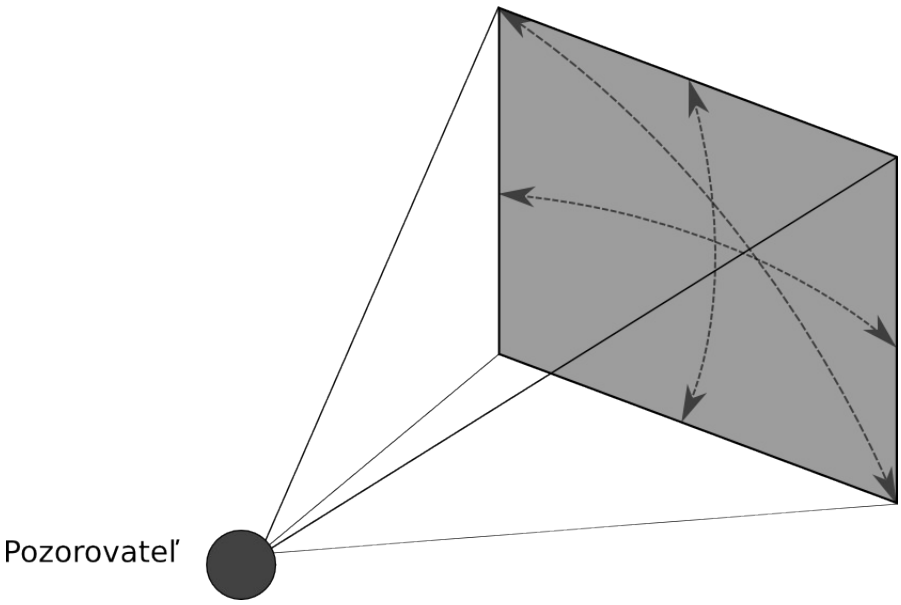


Obrázok 3: Interpolácia po úsečke a po kružnici

Existujú viaceré metódy na plynulú interpoláciu medzi dvomi kvaterniónmi. Najznámejšou z nich je metóda sférickej lineárnej interpolácii SLERP (Spherical Linear intERPolation), ktorú predstavil Ken Shoemake v osemdesiatych rokoch. Táto je považovaná za základnú metódu, ostatné z nej vychádzajú. SLERP je pomerne výpočtovo náročná a preto sa v praxi nahradzuje inými metódami. Ja som zvolil techniku normalizovanej lineárnej interpolácie NLERP (Nomalized Linear intERPolation), ktorá síce nevykonáva interpoláciu po guľovej ploche, ale vďaka normalizácii je rozdiel oproti SLERP voľným okom nepostrehnuteľný. Nami použitá matematická knižnica CML obsahuje funkcie na interpoláciu kvaterniónov, vrátane metód SLERP a NLERP.

4.5 Zoom

Technika približovania a oddaľovania objektov v 3D scéne je založená na zmene zorného poľa, resp. zmene jeho zorného uhľa. Zorný uhoľ je možné určiť vzhľadom na rôzne roviny:



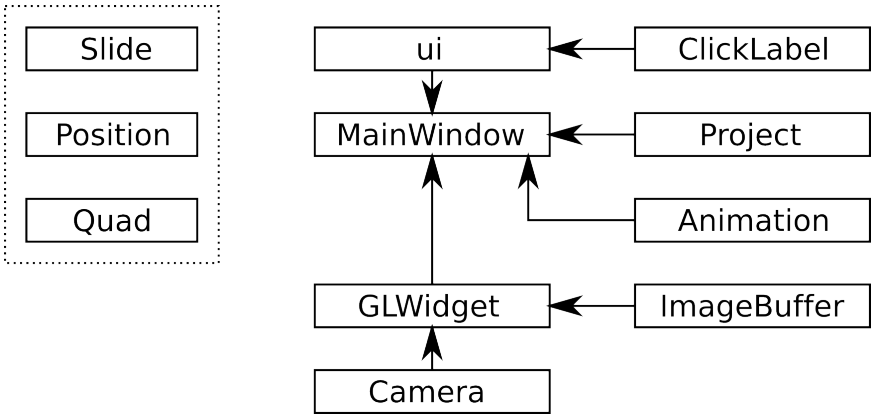
Obrázok 4: Zorný uhoľ vo vertikálnom, horizontálnom a diagonálnom smere

Pre naše použitie je významný uhoľ v rovine YZ, ktorý je v OpenGL označovaný ako fovy. Hodnota musí byť v intervale <0, 180> stupňov. Pre lepší výsledok nie je vhodné používať ani krajné hodnoty. Čím väčší je uhol, tým väčšia časť scény sa premietne na obrazovku a tým pádom sa objekty javia menšie a opačne. Zvyšovaním hodnoty fovy simulujeme oddaľovanie, znižovaním zase približovanie. Pri zmene aplikácia zavolá funkciu OpenGL **`gluPerspective()`**, ktorá aktuálny zoom reflektuje do scény.

5 Implementácia

Aplikácia je naprogramovaná v jazyku C++, pomocou vývojového prostredia Qt Creator. Užívateľské rozhranie bolo navrhnuté v GUI dizajnérovi, ktorý je do uvedeného prostredia integrovaný. Projekt je písaný objektovo, skladá sa z niekoľkých tried a iných dátových štruktúr. Najvýznamnejšie triedy a ich dôležité metódy sú popísané v tejto kapitole.

Následujúci diagram znázorňuje objekty, ktoré som navrhol a vzťahy medzi nimi. Z ilustrácie je vidieť, že všetky z nich sú priamo či nepriamo podradené triede **`MainWindow`**. Toto je typické pre aplikácie napísané v Qt toolkite [6]. V rámčeku sú zobrazené dátové štruktúry, ktoré slúžia na uchovávanie údajov. Disponujú minimálnym počtom metód, pretože nevykonávajú samostatnú činnosť. Ostatné triedy ich používajú ako úložisko dát. Niektoré ich využívajú aj pri komunikácii pomocou signálov a slotov.



Obrázok 5: Dátove štruktúry, triedy a vzťahy medzi nimi

5.1 Hlavné okno

Významnú časť našej aplikácie tvorí trieda **`MainWindow`**, ktorá reprezentuje hlavné okno. Jednou z jej úloh je manažment grafických prvkov (okná, menu, tlačidlá a pod.), ktoré boli vygenerované pomocou dizajnérskeho nástroja. Druhou úlohou je prepojenie týchto prvkov pomocou signálov a slotov a implementácia udalostí, ktoré sa vykonajú pri ich aktivácii.

Hlavné okno sa skladá z niekoľkých celkov – menu, hlavná lišta, lišta s prehrávačom, dokovací panel s oknami, stavový riadok a widget s OpenGL scénou. Menu je objekt triedy **`QMenuBar`** a umožňuje ovládať základné operácie ako načítavanie alebo ukladanie vytvorenej prezentácie, zmenu jazyka alebo nastavenie pomeru strán scény. V hlavnej lište, ktorá je objektom **`QToolBar`**, sú niektoré z týchto činností reprezentované tlačidlami. Lišta s prehrávačom slúži na spustenie

prezentácie a navigáciu v nej. Stavový riadok **QStatusBar** zobrazuje popis ostatných prvkov rozhrania alebo iné dôležité správy, napríklad chybové hlášky. Ak máme otvorený slajd, tak sa v pravej časti stavového riadku zobrazuje aktuálna poloha kamery. Najzložitejšou časťou užívateľského rozhrania je dokovací panel, ktorý sa skladá z troch okien. Sú to objekty triedy **QDockWidget**, ktoré sa dajú premiestňovať a združovať pomocou myši, tak aby ich rozloženie vyhovovalo užívateľovi.

Jedným z týchto okien je zoznam fotiek, ktoré sa budú zobrazovať. Je to komponenta užívateľského rozhrania, ktorá sa počas vytvárania prezentácie často používa. Preto je dôležité, aby bola práca so zoznamom jednoduchá a intuitívna. K implementácii som použil **QListWidget**. Je to štandardný Qt widget, ktorý umožňuje zobrazovať prvky formou zoznamu. Nad ním je umiestnená štvorica tlačidiel **QToolButton**, ktorá slúži na pridávanie a odoberanie fotiek a presúvanie vybranej fotky v zozname smerom hore alebo dole.

Dôležité je, že tento widget slúži len na zobrazovanie už spracovaných dát. Jediný údaj, ktorý sa z neho spätne získava, je index vybraného slajdu. Všetky potrebné informácie o aktuálnej prezentácii a všetkých fotkách sú uložené v ďalšom zozname. Ten je typu **QList**, čo je Qt varianta triedy **list** zo štandardnej knižnice C++. Prvky tohto zoznamu sú objektami triedy **Slide**, ktorá je popísaná v časti 5.5. Zoznamy sú teda dva – jeden slúži ako zdroj informácií a druhý ako prostriedok na ich zobrazovanie. Operácie nad prvkami (pridanie, odstránenie, posun) sa vykonávajú nad zoznamom s dátami a súčasne v užívateľskom rozhraní, aby zmeny boli viditeľné navonok.

Pridanie fotky do prezentácie je vyvolané stlačením príslušného tlačidla a následne pomocou dialógového okna alebo použitím drag & drop. Po získaní cesty súboru je fotka spracovaná funkciou **addSlideFile()**, ktorá sa ju pokúsi načítať pomocou triedy **QImage**. Po úspešnom načítaní sa zavolá **addSlideToWidget()**, ktorá vytvorí náhľad zmenšením fotky a pridá ju do zoznamu v hlavnom okne. Ihneď potom sa vytvorí objekt **Slide** s východzími parametrami a je pridaný do druhého, vnútorného zoznamu.

Druhé okno v paneli umožňuje meniť vlastnosti vybraného slajdu. Typ šošovky je možné zmeniť pomocou widgetu **QComboBox**. Ohnisková vzdialenosť sa nastavuje posuvníkom **QSlider** alebo použitím komponenty **QSpinBox**, ktorá umožňuje zapísať hodnotu aj pomocou klávesnice. Tento widget dovoľuje definovať rozsah povolených hodnôt pomocou vlastností **minimum** a **maximum**, takže máme zaručené, že vstupná hodnota bude číselná a korektná. Východzia farba pozadia je čierna. Užívateľ ju môže nastaviť pri každom slajde zvlášť. K tomu som použil tlačidlo **QToolButton**, ktorého farba sa nastavuje pomocou **setPalette()**. Jeho stlačením sa vyvolá dialógové okno **QColorDialog**, ktoré slúži na výber farby. Pomocou jeho statickej metódy **getColor()** získame farbu, nastavíme ju v aktuálnom slajde a aj v OpenGL scéne. Hoci sa dajú

farby ľubovolne striedať medzi slajdami, je odporúčané dodržať rovnakú farbu počas celej prezentácie, aby nebol výsledný efekt rušivý.

Posledné z trojice okien nastavuje parametre animácie. Najdôležitejšie položky sú začiatočný a koncový pohľad. Tie určujú natočenie kamery a zoom. Zobrazená hodnota je vo formáte [pitch,yaw,roll/zoom], kde prvé tri položky predstavujú rotácie okolo jednotlivých osí. Pozícia je zobrazená pomocou widgetu **`ClickLabel`**. Je to potomok triedy **`QLabel`**, ktorý som rozšíril, aby reagoval na kliknutie myšou. Dosiahol som to reimplementovaním metódy **`mousePressEvent()`**, ktorá emituje signál **`clicked()`**. Po kliknutí sa kamera nastaví na danú pozíciu. Práca s kamerou prebieha tak, že užívateľ pomocou myši alebo tlačidiel v hlavnej lište zvolí počiatočný pohľad a uloží ho pomocou príslušného tlačidla. To isté urobí aj pre koncový pohľad. Pri prehrávaní potom kamera plynule interpoluje medzi počiatočnou a koncovou polohou. Dĺžku trvania tohto prechodu je možné nastaviť v tomto okne. Tiež sa odporúča nastaviť pauzu na začiatku a na konci. To je doba, počas ktorej bude fotka v nehybnej polohe. Náhly prechod medzi dvoma slajdami by bez nej pôsobil rušivo. Východzie hodnoty sú preto nastavené na jednu sekundu. Poslednou vlastnosťou, ktorá sa v tomto okne nastavuje, je nasledovanie predchádzajúceho slajdu. Ak je povolené, tak sú súradnice kamery počiatočného pohľadu rovnaké ako súradnice koncového v predchádzajúcom slajde. Toto značne rozširuje možnosti pohybu. Pridaním niekoľkých slajdov s rovnakou fotkou je možné skombinovať prechody medzi nimi a vytvoriť tak komplexný pohyb po scéne. Prípadná zmena súradníc sa automaticky prejaví v následujúcom slajde.

Pred zobrazením hlavného okna sa aplikácia pokúsi načítať nastavenia uložené na disku. K tomu používa objekt typu **`QSettings`**. Táto trieda umožňuje uchovávať ľubovolné hodnoty, ktoré sa dajú konvertovať na reťazec znakov. Jednotlivé položky sú ukladané ako dvojica kľúč-hodnota. V závislosti na operačnom systéme sa ukladajú napríklad do domovského adresára (Linux) alebo do registrov (MS Windows). Naša trieda definuje metódy **`saveSettings()`** a **`loadSettings()`**, ktoré pracujú s triedou **`QSettings`**. Medzi ukladané hodnoty patria poloha, rozmery okna a tiež poloha ostatných grafických prvkov (toolbar, zoznam slajdov a pod.). Užívatelia si tak môžu prispôsobiť grafické rozhranie podľa rozmerov obrazovky a osobných preferencií.

Ďalšou vlastnosťou, ktorá zjednoduší prácu užívateľom, je drag & drop. Umožňuje pridávať fotky do našej aplikácie pretiahnutím súboru do okna pomocou myši. Qt má pre túto techniku priamu podporu. Stačí implementovať dve metódy. Prvou z nich je **`dragEnterEvent()`**. Zisťuje sa v nej, či sú vkladané dáta vo formáte text/uri-list. Cesty k súborom majú práve tento formát. Druhou metódou je **`dropEvent()`**, kde aplikácia iteruje zoznamom vložených súborov a pokúša sa ich načítať a pridať do zoznamu slajdov. Naraz je teda možné pridať viac súborov.

5.2 Okno s 3D scénou

Ako bolo úž spomenuté, Qt má priamu podporu pre OpenGL. Základná komponenta, ktorá nám umožňuje prácu s touto grafickou knižnicou, sa nazýva **QGLWidget**. Túto triedu je potrebné zdediť a reimplementovať niektoré jej metódy. Novú triedu som nazval **GLWidget** a okrem implementácie potrebných metód obsahuje aj ďalšiu funkcionalitu, napríklad spracovanie myši.

Metóda **initializeGL()** vykonáva inicializáciu, povolí textúry, nastaví počiatočnú farbu pozadia a zavolá funkciu **setProjection()**, ktorá nastaví perspektívnu projekciu. Pre korektné chovanie pri zmene veľkosti okna je potrebné reimplementovať metódu **resizeGL()** a upraviť pohľad na základe nových rozmerov.

Najdôležitejšou metódou je **paintGL()**, v ktorej prebieha samotné vykresľovanie. Na začiatku je nutné zistiť, v akom móde sa momentálne aplikácia nachádza. Podľa toho sa nastaví kamera. V animačnom móde (počas prehrávania) je potrebné vypočítať aktuálnu polohu kamery, pretože sa neustále mení. K tomu slúži trieda **Animation**. **GLWidget** iba zavolá metódu **interpolateTo()** triedy **Camera** a predá jej tieto spočítané hodnoty. Situácia je jednoduchšia v normálnom, editačnom móde. Vtedy stačí nastaviť polohu kamery na základe posledných uložených hodnôt. Po tejto operácii vykreslíme pozadie scény farbou aktuálneho slajdu. Nakoniec vyrenderujeme textúru, teda fotku, postupne po jednotlivých obdĺžnikoch. Metóda na prekresľovanie scény býva volaná mnohokrát v priebehu sekundy, aby bol pohyb v 3D priestore plynulý. K tomu používam funkciu **updateGL()**, čo je vlastne slot, napojený na časovač **QTimer**.

Načítavanie fotiek prebieha v metóde **loadSlide()**. Algoritmus je podrobne popísaný v časti 5.3. Na vytvorenie textúry používam funkciu **bindTexture()**, čo je Qt alternatíva ku klasickej OpenGL funkcii **glBindTexture()**. Jej výhodou je, že dokáže vytvoriť textúru na základe cesty k súboru, alebo z objektu **QImage**. Netreba riešiť rôzne formáty fotiek. Dokáže pracovať s mnohými populárnymi formátmi ako napríklad JPEG, PNG, BMP, TIFF, GIF a ďalšie. Funkcia **bindTexture()** umožňuje generovať aj mipmapové textúry pre OpenGL. Mipmapa predstavuje textúru (sadu textúr), ktorá je uložená v rôznych veľkostiach. Prvá z nich je v pôvodnej veľkosti, každá ďalšia má polovičnú šírku a polovičnú výšku tej predchádzajúcej. Veľkosť poslednej je 1 x 1 pixel. Počas vykresľovania sa zvolí vhodná textúra podľa vzdialenosti. Za cenu mierne vyššej spotreby pamäte získame obraz, ktorý je jemnejší a vypadá krajšie. Samotné generovanie však zaberá niekoľkonásobne viac času ako pri použití štandardných textúr. Pri vytváraní prezentácie sú mipmapy z tohto dôvodu vypnuté, keďže ich generovanie by spôsobovalo nižšiu odozvu užívateľského rozhrania. Mipmapové textúry sú však zapnuté počas celoobrazovkového (fullscreen) režimu, pretože počas prehrávania je kvalita dôležitým faktorom.

Trieda `GLWidget` sa stará aj o spracovanie myši. K tomu používa niekoľko metód. Najvýznamnejšia z nich je `mouseMoveEvent()`. Knižnica Qt túto funkciu zavolá vždy, ak nastane pohyb myši nad daným widgetom. Pomocou parametra, ktorý je objektom triedy `QMouseEvent`, je možné zistiť aktuálne súradnice kurzora. Na základe rozdielu nových a predchádzajúcich hodnôt otočíme kameru v horizontálnom alebo vertikálnom smere, alebo vykonáme rotáciu okolo osi Z. Počas fullscreen režimu sa `GLWidget` stará aj o spracovanie klávesnice. Pri stlačení klávesy je zavolaná metóda `keyPressEvent()`, ktorá svojim parametrom typu `QKeyEvent` umožňuje identifikovať klávesu a vykonať požadovanú akciu.

5.3 Umiestnenie fotky do scény

Každá fotka zobrazovaná našou aplikáciu, je pred samotným vykreslením spracovaná v niekoľkých krokoch:

1. **Načítanie do vyrovnávacej pamäte**

   Súbor s obrazovými dátami je načítaný do vyrovnávacej pamäte, ak sa tam už nenachádza. Viac detailov v kapitole 5.6.

2. **Vytvorenie siete bodov**

   Aplikácia prechádza fotku postupne po dĺžke a po šírke s určitým konštantným krokom. V každej iterácii zavolá príslušnú metódu triedy na prácu so šošovkami, predá jej aktuálne súradnice a získa súradnice v 3D priestore pre danú šošovku.

3. **Vytvorenie obdĺžnikov z bodov**

   Zo získaných bodov vytvoríme objekty typu `Quad`, ktoré reprezentujú obdĺžniky. Každý z nich obsahuje štvoricu bodov, ktoré predstavujú polohu daného obdĺžnika na fotke a tiež štvoricu, ktorá uchováva koordináty pre textúru.

4. **Vytvorenie textúry pre OpenGL**

   O vytvorenie textúry, ktorú je možné vykresliť pomocou OpenGL, sa postará knižnica Qt počas volania funkcie `bindTexture()`.

5. **Vykreslenie textúry**

   Vytvorenú textúru je potrebné pred vykreslením namapovať na súradnice v priestore. Tie už máme vypočítané a uložené v zozname obdĺžnikov. Postupne vykreslujeme jednotlivé

obdĺžniky, až kým nezobrazíme celú fotku. Tento krok, na rozdiel od predchádzajúcich, prebieha opakovane a vykonáva sa pri každom prekreslení scény.

5.4 Kamera

Kamera určuje, ktorá časť scény bude zobrazená na obrazovke. Zmenou jej polohy môžeme simulovať pohyb alebo rotáciu, čo presne vyhovuje našim potrebám. Ako bolo už spomenuté v predchádzajúcich kapitolách, naša aplikácia k tomu účelu používa kvaternióny. Triedu `Camera` som založil na kóde, ktorý zverejnil Vic Hollis na stránkach NeHe tutoriálov [7]. Kameru som rozšíril o niekoľko ďalších funkcií a pridal som možnosť interpolácie. Celý kód som prepísal tak, aby všetky výpočty s kvaterniónmi prebiehali pomocou knižnice CML.

Pri otvorení hotovej prezentácie potrebujeme nastaviť kameru na polohu aktuálneho slajdu. K tomu slúžia metódy `setPitch()`, `setYaw()`, `setRoll()`, alebo môžeme nastaviť súradnice naraz pomocou `set()`. Ak ukladáme počiatočný alebo koncový pohľad (natočenie kamery a zoom), tak potrebujeme prečítať aktuálne hodnoty premenných. Tie je možné získať pomocou metód `getPitch()`, `getYaw()`, `getRoll()`.

Kameru je možné ovládať pomocou troch základných funkcií – `changePitch()`, `changeYaw()`, `changeRoll()`, ktoré umožňujú rotácie okolo jednotlivých osí. Tieto berú ako parameter uhol v stupňoch, o ktorý sa má kamera (pozorovateľ) otočiť. Pri zmene sa nová hodnota pripočíta k jednej z vnútorných premenných `m_pitchDegrees`, `m_yawDegrees`, alebo `m_rollDegrees`. Táto trojica jednoznačne určuje polohu kamery. Všetky potrebné výpočty súvisiace s týmito súradnicami sú uskutočňované v rámci triedy `Camera`. Kvaternión reprezentujúci aktuálnu rotáciu kamery sa vypočíta práve z týchto troch premenných. K výpočtu sa použije funkcia `axisAngleToQ()`. Keď už máme spočítaný aktuálny kvaternión, je potrebné vytvoriť transformačnú maticu pre OpenGL. K tomuto prevodu slúži funkcia `createMatrix()`, ktorá funguje na základe vzorca uvedeného v kapitole 4.3. Vypočítanú maticu vynásobíme s aktuálnou maticou OpenGL a vykonaná rotácia sa prejaví pri najbližšom prekreslení scény.

5.5 Trieda Slide

`Slide` je jednoduchá trieda, ktorá slúži na uchovávanie informácií o konkrétnej fotke. Pamätá si názov súboru, typ šošovky, ohnisková vzdialenosť a mnohé ďalšie hodnoty. Všetky členské premenné som sa rozhodol deklarovať ako verejné, pretože sa k nim často pristupuje a ľahšie sa tak s nimi manipuluje. Aby bola zaistená validita dát, obsahuje táto trieda metódu `isValid()`, ktorá skontroluje premenné a rozsahy hodnôt. Metóda `reset()` nastaví členské premenné na ich

počiatočné hodnoty. V následujúcej tabuľke je popísaný význam jednotlivých premenných. Všetky tieto položky je potrebné uchovať pri ukladaní prezentácie. Okrem toho trieda `Slide` obsahuje aj dočasné premenné, ktoré majú význam len počas behu aplikácie.

Dátový typ	Názov	Význam
QString	file	Absolútna alebo relatívna cesta k súboru s fotkou.
QString	name	Názov zobrazený v zozname slajdov.
QColor	bgColor	Farba pozadia.
int	lensType	Typ šošovky. Hodnota je v rozsahu enumerácie LensType.
int	flength	Ohnisková vzdialenosť šošovky.
float	duration	Dĺžka trvania prehrávania (sek.).
float	startDelay	Pauza pred prehrávaním (sek.).
float	endDelay	Pauza po prehrávaní (sek.).
Position	startPos	Súradnice kamery pre počiatočný pohľad.
Position	endPos	Súradnice kamery pre koncový pohľad.
bool	follow	Náväznosť na predchádzajúci slajd. Ak je true, tak počiatočný pohľad aktuálneho slajdu bude zhodný s koncovým predchádzajúceho.

Tabuľka 1: Členské premenné triedy Slide

Pričom `Position` je jednoduchá štruktúra obsahujúca súradnice kamery.

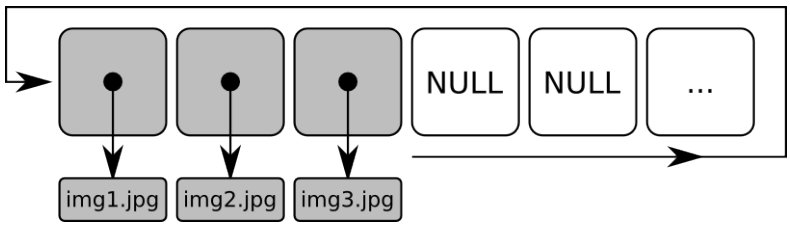
Dátový typ	Názov	Význam
float	pitch	Rotácia okolo osi X. Hodnota je v stupňoch (0-360).
float	yaw	Rotácia okolo osi Y. Hodnota je v stupňoch (0-360).
float	roll	Rotácia okolo osi Z. Hodnota je v stupňoch (0-360).
float	zoom	Hodnota zoomu. Čím väčšie číslo, tým väčšie oddialenie a opačne. Hodnota je v rozsahu 1.0 až 179.0.

Tabuľka 2: Štruktúra Position uchovávajúca súradnice kamery

5.6 Trieda ImageBuffer

Grafické aplikácie často pracujú s veľkým objemom dát v pamäti. Naša aplikácia nie je výnimkou. Preto je dôležité zabezpečiť efektívny manažment pamäťových zdrojov. Na tento účel som navrhol triedu `ImageBuffer`. Aplikácia požiada túto triedu o načítanie určitého súboru, teda fotky. Tá sa najprv pozrie do svojho bufferu a zistí, či sa v ňom nachádza požadovaný súbor. Ak bol súbor predtým načítaný, tak vráti ukazateľ na obrazové dáta. V prípade neúspešného vyhľadania sa súbor

načíta z disku a je uložený do bufferu. Buffer je implementovaný ako cyklická fronta. Nové súbory sa ukladajú na prvé voľné miesto. V prípade plnej fronty sa uvoľňujú najstaršie fotky a sú nahradené novými.



Obrázok 6: Obsah buffera po načítaní troch fotiek

5.7 Trieda Animation

Táto trieda slúži na posun v animačnom čase. Pri spustení prezentácie (animácie) sa čas vynuluje a spustí sa časovač **`QTimer`**. Tento časovač v pravidelných intervaloch emituje signál, ktorý je napojený na slot **`timeOutSlot()`**. V jeho tele je implementované jadro animačného mechanizmu. Najprv sa posunie animačný čas. V prípade, že aktuálny čas spadá do intervalu v ktorom prebieha interpolácia, tak sa vypočíta interpolačný parameter a hodnota zoom. Tieto hodnoty sú následne emitované formou signálu a prijímané triedou **`GLWidget`**, ktorá ich pri najbližšom prekreslení aplikuje na scénu. Ak je animácia aktuálneho slajdu ukončená a v prezentácii sa nachádzajú ďalšie fotky, tak trieda **`Animation`** zabezpečí prechod na ďalší slajd. O tejto akcii informuje triedou **`MainWindow`**, takisto pomocou signálov.

5.8 Ukladanie

Na ukladanie vytvorených prezentácií som zvolil otvorený formát XML, pretože je bez problémov prenositeľný medzi rôznymi platformami a je priamo podporovaný v Qt. Prezentácie je možno ukladať dvomi spôsobmi. Prvým z nich je uloženie iba samotného projektu, do súboru s príponou „3do“. Je to súbor obsahujúci informácie o slajdoch a absolútne cesty k fotkám. Tento formát je vhodné použiť počas vytvárania prezentácie alebo ak ju plánujeme spúšťať iba na počítači, kde bola vytvorená. Ak chceme aby bola prenositeľná medzi rôznymi počítačmi, je treba zvoliť uloženie vo formáte „3dp“. Tento súbor je tiež XML súborom, ktorý ale obsahuje relatívne cesty k fotkám. Okrem projektového súboru sa vytvorí adresár rovnakého názvu so sufixom „_data“, do ktorého sa počas ukladania nakopírujú všetky fotky. Aby nenastala kolízia v prípade rovnakých názvov, sú názvy týchto súborov generované pomocou **`qChecksum()`**. Táto funkcia počíta kontrolný súčet

CRC-16. Vypočíta ho na základe cesty súboru a času poslednej modifikácie. Názov nového súboru je teda číslo s príponou (jpg, png a pod.). Prezentácie je možné obojsmerne konvertovať medzi týmito dvoma formátmi podľa potreby. Aktuálne otvorenú prezentáciu spravuje trieda **`Project`**. Metóda **`load()`** slúži na načítanie projektu, metóda **`save()`** na ukladanie a konvertovanie medzi formátmi 3do a 3dp. Cestu k aktuálnemu projektu je možné získať pomocou **`getPath()`**. V prípade relatívnych ciest sa používa metóda **`getDir()`**, ktorá vracia názov adresára s fotkami. Následujúci výpis je príkladom súboru s prezentáciou, ktorá obsahuje jeden slajd:

```
<!DOCTYPE slideshowXML>
<slideshow dir="Test_data" complete="1">
 <slide>
  <file>7564.jpg</file>
  <name>Example.jpg</name>
  <bgcolor>#55ffff</bgcolor>
  <lenstype>2</lenstype>
  <flength>30</flength>
  <duration>4.5</duration>
  <startdelay>1</startdelay>
  <enddelay>1</enddelay>
  <follow>0</follow>
  <startpos zoom="60" yaw="22" pitch="15.8" roll="0"/>
  <endpos zoom="16" yaw="-3.4" pitch="-6.25" roll="0"/>
 </slide>
</slideshow>
```

6 Ovládanie

V tejto časti opíšem ovládanie programu a tvorbu jednoduchej prezentácie. Keďže sa jedná o grafickú aplikáciu, nezaobídeme sa bez použitia počítačovej myši. Niektoré úkony je možné vykonávať pomocou klávesnice a klávesovými skratkami, ktoré sú zobrazené v menu.

Po spustení binárneho súboru sa otvorí hlavné okno s prázdnou scénou bez slajdov. Začneme výberom jazyka. V menu **Settings->Language** máme na výber angličtinu, slovenčinu a češtinu. Ďalej budem uvádzať anglické názvy. Ak potrebujeme zistiť, k čomu slúži určitý prvok rozhrania, tak stačí nad neho umiestniť myš. Jeho význam sa zobrazí v stavovom riadku, ktorý sa nachádza v spodnej časti okna. Pred vytváraním prezentácie je vhodné aspoň približne nastaviť pomer strán monitora alebo projektora, na ktorom bude výsledok zobrazovaný. Prednastavená hodnota je 16:10. Zmenu vykonáme v menu **View->Aspect ratio**. Všetky tieto nastavenia, vrátane rozloženia panelov a veľkosti okien, sa uchovajú po vypnutí aplikácie a netreba ich teda vykonávať opakovane.

Môžeme pokračovať pridaním fotiek. V zozname slajdov klikneme na tlačidlo  . Otvorí sa dialógové okno, pomocou ktorého vyberieme súbor s fotkou. Môžeme pridať aj viac súborov súčasne. Náhľad fotky sa zobrazí v zozname slajdov **Slide list** a po kliknutí sa načíta do 3D scény. Nastavíme niektoré vlastnosti slajdu v okne **Slide properties**. Vyberieme šošovku, nastavíme požadovanú ohniskovú vzdialenosť, prípadne zmeníme aj farbu pozadia. Ďalej potrebujeme definovať počiatočný a koncový pohľad kamery. Ľavým tlačítkom myši klikneme na scénu a pohybom otáčame kameru v požadovanom smere. Rotáciu fotky môžeme urobiť tak, že vykonávame horizontálny pohyb myšou, pričom mám stlačené pravé tlačítko. Alternatívne je možné použiť tlačidlá  a  z lišty. Približovanie a oddaľovanie fotky je najintuitívnejšie pomocou kolieska na myši. V prípade, že nie je dostupné, jeho funkcionalitu vykonávajú aj tlačidlá  a  . Po nasmerovaní kamery a dosiahnutí požadovaného pohľadu na scénu, klikneme na prvé tlačidlo  v okne **Slide animation** označené ako **Start view**. Toto spôsobí uloženie počiatočného pohľadu. Zmeníme polohu a rovnakým spôsobom uložíme koncový pohľad, pomocou druhého tlačidla  , ktoré je označené ako **End view**. Pridávame ďalšie fotky a postup opakujeme pre všetky slajdy.

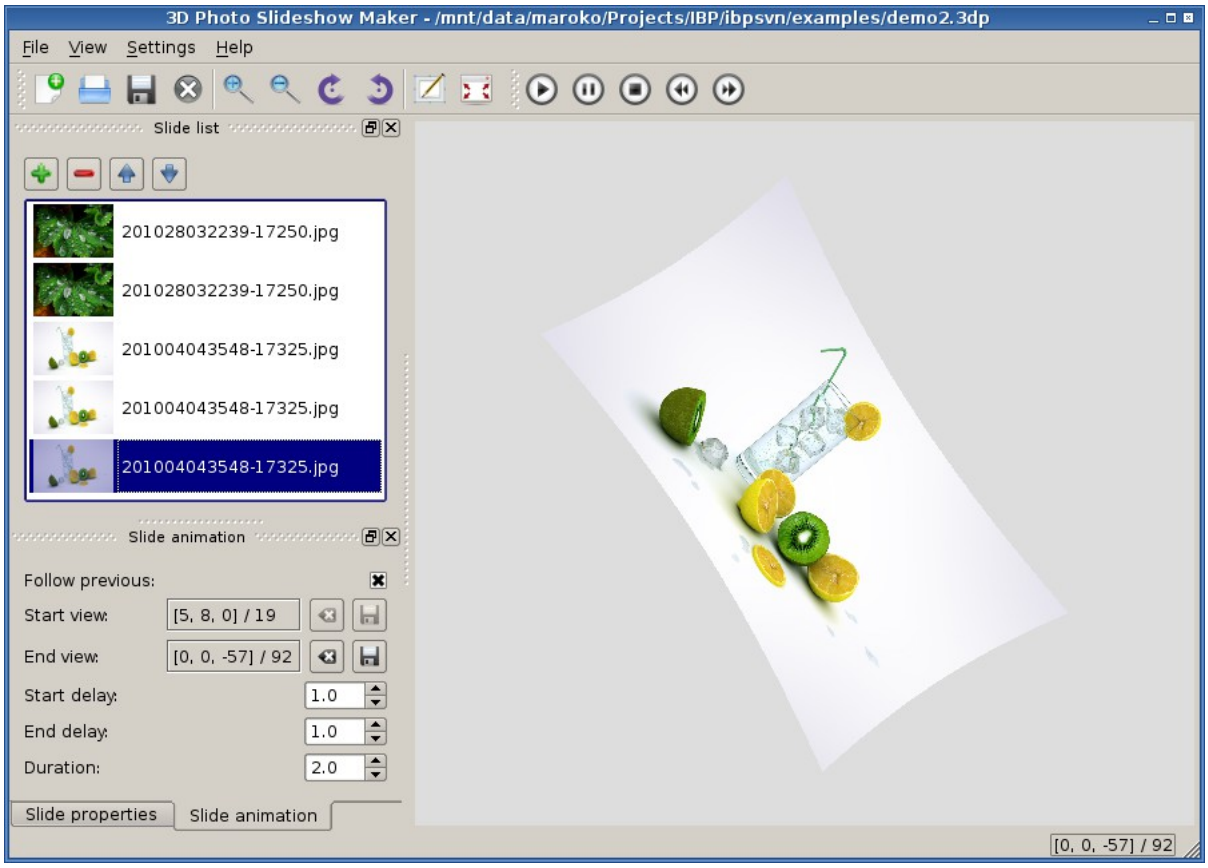
Prezentáciu máme vytvorenú, môžeme ju spustiť. Klikneme na tlačidlo  v lište s prehrávačom. Pre kvalitnejší zážitok je odporúčané prepnúť aplikáciu do celoobrazovkového (fullscreen) režimu, kliknutím na  . V tomto režime môžeme ovplyvniť priebeh prehrávania pomocou klávesnice:

Medzerník – Pozastavenie prehrávania

Šípky – Prechod na predchádzajúci alebo nasledujúci slajd

Esc/F11/f – Ukončenie fullscreen režimu

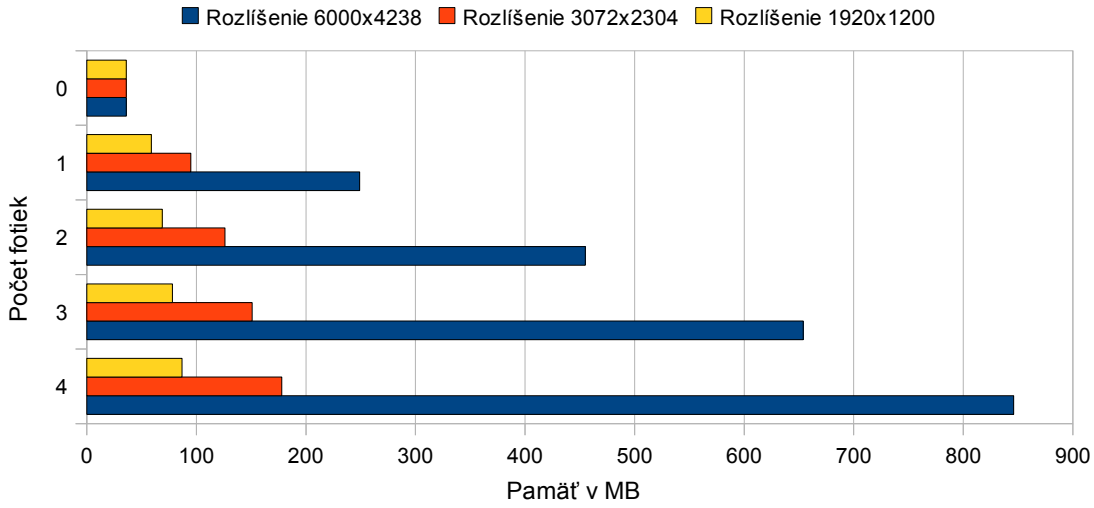
Vytvorenú prezentáciu uložíme pomocou menu **File->Save** pre budúce použitie. Môžeme uložiť buď len samotný súbor prezentácie, alebo všetko, vrátane fotiek. Druhá možnosť je vhodná ak prenášame prezentáciu medzi dvoma počítačmi.



Obrázok 7: Grafické užívateľské rozhranie

7 Testovanie

Aby bola aplikácia reálne použiteľná, musí byť dôkladne otestovaná a to nielen voči chybám, ale aj na nároky na procesor, grafickú kartu a operačnú pamäť. Keďže používam iba základné OpenGL príkazy a vykresľujem len jednu textúru (fotku) v danom momente, tak nároky na CPU a GPU nie sú veľké. Žiadny z testovaných počítačov nezaznamenal problémy v tomto smere. Na druhej strane požiadavky na pamäť zohrávajú dôležitú úlohu, pretože fotky vo vyššom rozlíšení zaberajú nezanedbateľné množstvo pamäte. Táto kapitola obsahuje porovnanie pamäťových nárokov našej aplikácie pri použití fotiek v rôznych rozlíšeniach. Výsledky som vyhodnotil pre Linux (32 bit, kernel 2.6.33) a MS Windows 7 (64 bit), pričom testovacia zostava bola: Core 2 Duo 2.2 GHz, NVIDIA GeForce 8600M GS (256 MB), 2 GB RAM.



Obrázok 8: Pamätové nároky aplikácie v prostredí Linux

Graf pre MS Windows neuvádzam, pretože rozdiely oproti Linuxu boli zo štatistického hľadiska nezaujímavé. Z výsledkov vyplýva, že fotky s nižším a stredne vysokým rozlíšením nerobia našej aplikácii žiadny problém, ani pri väčšom počte. Fotky vo vysokom rozlíšení ale zaberajú značné množstvo operačnej pamäte. Zaujímavým zistením je rozdielny spôsob riešenia nedostatku pamäte na oboch platformách. V prípade Linuxu je aplikácia ukončená systémom, zatiaľ čo v prostredí MS Windows beží ďalej, avšak prestane vykreslovať textúru.

Možným riešením nedostatku pamäte je zmenšiť veľkosť bufferu, ktorý fotky ukladá (detaily v kapitole 5.6), za cenu zníženej rýchlosti počas prepínania slajdov (fotky by sa museli častejšie načítavať z disku). Toto by však obmedzovalo užívateľov, ktorí majú pamäte dostatok. Optimálne

riešenie by bolo meniť veľkosť bufferu dynamicky, v závislosti na rozlíšení fotiek a na množstve dostupnej pamäte v systéme. Túto variantu ponechávam ako možné vylepšenie do budúcnosti. V aktuálnej verzii je veľkosť bufferu konštantná, pričom dokáže uchovať šesť fotiek. Táto hodnota je pri veľkosti dnešných pamätí rozumným kompromisom .

8 Záver

Účelom tejto práce bolo vytvoriť nástroj na tvorbu prezentácií z fotiek v 3D priestore, ktorý bude fungujúci a dá sa prakticky používať. Program sa mi podarilo úspešne implementovať, pričom spĺňa všetky požadované vlastnosti zo zadania.

Aplikácia bola vytváraná pomocou moderných knižníc s dôrazom na prenositeľnosť. Napriek tomu sa vyskytli mierne rozdiely medzi platformami, ktoré sa však podarilo úspešne odstrániť. Program bol úspešne otestovaný pod operačným systémom MS Windows a Linux.

K vytváraniu priestorového dojmu používam knižnicu na prácu so šošovkami a tak demonštrujem ich optické vlastnosti. Pomocou šošoviek je možné zobrazovať panoramatické sekvencie takým spôsobom, že rotácie pozorovateľa vytvárajú dojem reálnosti. Toto je podľa mňa najväčší prínos tejto aplikácie z pohľadu užívateľa, pretože existuje veľa podobných aplikácií, no žiadna z nich neimplementuje túto možnosť. Veľkým prínosom pre mňa bolo, že som sa zoznámil s teóriou kvaterniónov, ktorá zohráva v oblasti počítačovej grafiky a animácie významnú úlohu, a aplikoval som získané poznatky vo forme funkčného programu.

V budúcnosti by bolo vhodné projekt rozšíriť o ďalšie vlastnosti. Jednou z nich je možnosť definovať vlastné typy šošoviek, ktorých špecifikácie by mohli byť uložené v súboroch a boli by dynamicky načítavané. Príjemným vylepšením by bolo aj pridávanie textu do slajdov alebo možnosť exportovať výsledok ako video.

Literatúra

[1] Kršek, P.: Základy počítačové grafiky: grafická API a OpenGL, Brno, 2009.

[2] Kršek, P.: Základy počítačové grafiky: studijní opora, Brno, 2006.

[3] Žára, J., Beneš, B., Sochor, J., Felkel, P.: Moderní počítačová grafika. Brno, Computer Press, 2004, ISBN 80-251-0454-0.

[4] Schwerd, D.: Quaternions [online]. 2006 [cit. 2010-04-22]. Dostupné z WWW: <http://www.hamilton2005.ie/quaternions.html>.

[5] Shreiner, D.: OpenGL programming guide: the official guide to learning OpenGL, version 2.0. Upper Saddle River, Addison-Wesley, 2006, ISBN 0-321-33573-2.

[6] Molkentin, D.: The book of Qt 4: the art of building Qt applications. San Francisco, No Starch Press, 2007, ISBN 978-1-59327-147-3.

[7] Hollis, V.: Quaternion Camera Class [online]. 2003 [cit. 2010-04-11]. Dostupné z WWW: <http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=Quaternion_Camera_Class>.

[8] Molofee, J., Turek, M.: NeHe OpenGL tutoriály v čestine [online]. 2004 [cit. 2010-04-12]. Dostupné z WWW: <http://nehe.ceske-hry.cz/>.

[9] Qt Reference Documentation [online]. 2010 [cit. 2010-04-16]. Dostupné z WWW: <http://doc.qt.nokia.com/4.6/>.

Zoznam príloh

Príloha 1. CD – Zložka bin – spustitelná aplikácia pre MS Windows a Linux (32/64 bit)
– Zložka doc – technická správa vo formáte PDF a jej zdrojové súbory
– Zložka src – zdrojové súbory aplikácie