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

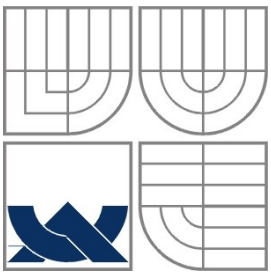
OPENGL APLIKACE NAD QT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

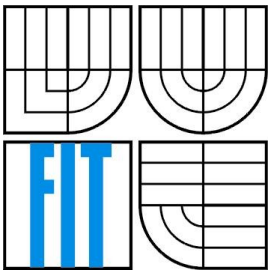
AUTOR PRÁCE
AUTHOR

DÁVID SMEJKAL

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

OPENGL APLIKACE NAD QT

OPENGL APPLICATION IN QT FRAMEWORK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — DÁVID SMEJKAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN NAVRÁTIL
SUPERVISOR

BRNO 2009

Zadání bakalářské práce/9240/2008/xsmejk13

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2008/2009

Zadání bakalářské práce

Řešitel: **Smejkal Dávid**
Obor: Informační technologie
Téma: **OpenGL aplikace nad Qt**
Kategorie: Počítačová grafika

Pokyny:

1. Seznamte se knihovnou OpenGL.
2. Prostudujte OpenGL nadstavbu nad Qt.
3. Navrhněte strukturu vhodné aplikace pracující s OpenGL a Qt.
4. Implementujte navrženou aplikaci v prostředí Qt.
5. Porovnejte výhody a nevýhody Qt proti ostatním C++ knihovnám.
6. Diskutujte dosažené výsledky a možnosti dalšího vývoje. Práci prezentujte plakátkem.

Literatura:

- Po domluvě s vedoucím.

Při obhajobě semestrální části projektu je požadováno:

- Bez požadavků

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

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

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

V první části práce jsou popsané Qt, OpenGL a jejich vzájemné propojení. Blíže jsou specifikované právě OpenGL techniky implementované v programu. Qt toolkit je porovnaný s třemi jinými multiplatformovo orientovanými toolkity (AWT/Swing, GTK+ a wxWidgets). Druhá kapitola má spíše praktické zaměření a jsou v ní popsané podrobnosti o programu. Také jsou zde specifikované požadavky na aplikaci, návrh a implementaci. Nakonec jsou zhodnoceny výsledky a možnosti dalšího vývoje aplikace.

Abstract

There are described Qt, OpenGL and their mutual interconnection in the first chapter of the thesis. Especially, OpenGL techniques implemented in the program are closely specified. Qt toolkit is compared to three other multiplatform toolkits (AWT/Swing, QTK+ and wxWidgets). The second, more practical chapter of the thesis, includes program specifications, program design and program implementation. There are presented results and possible future development at the end of the thesis.

Klíčová slova

OpenGL, Qt, hra, porovnání toolkitů

Keywords

OpenGL, Qt, game, comparison of toolkits

Citace

Dávid Smejkal: OpenGL aplikace nad Qt, bakalářská práce, Brno, FIT VUT v Brně, 2009

OpenGL aplikace nad Qt

Prohlášení

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

……
Dávid Smejkal
15. května 2009

Poděkování

Chtěl bych poděkovat vedoucímu své bakalářské práce, panu Ing. Janovi Navrátilovi, za odbornou pomoc, užitečné rady a připomínky při tvorbě této práce.

Obsah

1 Úvod

V tejto práci je spracovaná problematika výberu vhodného toolkitu pre implementáciu aplikácie s grafickým užívateľským rozhraním. Čitateľ sa dozvie ako je možné implementovať OpenGL aplikáciu v Qt toolkite. Takáto aplikácia bude nakoniec implementovaná a bude slúžiť ako ukážka skĺbenia OpenGL s Qt.

Prvá kapitola je zameraná skôr na teóriu. Oboznámime sa v nej s Qt, OpenGL a ich vzájomným prepojením. Ďalej porovnáme Qt s tromi inými multiplatformovo orientovanými toolkitmi (*AWT/Swing*, *GTK+* a *wxWidgets*).

Druhá kapitola má skôr praktické zameranie. Dozvieme sa v nej podrobnosti o programe. Dočítame sa aké sú požiadavky a priblížime si jeho návrh. Ukážeme si implementácie zaujímavých častí programu.

V poslednej kapitole sa dozvieme výsledky a predvedieme si ukážky z aplikácie. Následne v závere rozoberieme prínos a možnosti budúceho vývoja.

2 Teoretický základ

V tejto kapitole si priblížime softvérové rozhranie OpenGL, jeho charakteristické vlastnosti a techniky využívané v neskoršej implementácii aplikácie. Ďalej sa oboznámime s Qt a vysvetlíme si vzájomné prepojenie OpenGL s Qt. Dozvieme sa výhody a nevýhody Qt oproti iným multiplatformovo orientovaným toolkitom (z angl. toolkit, sada nástrojov).

2.1 OpenGL

OpenGL (Open Graphics Library) je softvérové rozhranie pre grafický hardvér. Toto rozhranie pozostáva z viac ako 250 odlišných príkazov (okolo 200 v jadre OpenGL a ďalších 50 v OpenGL Utility Library), ktoré môžeme použiť pre špecifikáciu objektov a operácií potrebných k vytvoreniu interaktívnej dvojrozmernej, alebo trojrozmernej aplikácie.[1]

Obrázok 2.1: OpenGL logo

OpenGL je stavový stroj. Čo to znamená? Môžeme si to predstaviť veľmi jednoducho, ako razítko. Keď razítko otlačíme do modrej farby, všetky otlačky tohto razítka budú modré, až do chvíle kým nezmeníme farbu razítka. Takéto razítko ma len jeden stav a tým je farba, OpenGL ma naopak mnoho stavov. Napríklad môžeme meniť spôsob vykresľovania polygónov, vlastnosť materiálu, vypnúť/zapnúť svetlá a textúrovanie, alebo meniť maticu modelu (takto zmeníme pozíciu, rotáciu, skreslenie, veľkosť objektu, ktorý sa chystáme vykresliť).

Jednotlivé stavy sa ovládajú pomocou príkazov `glEnable()` a `glDisable()`. Niektoré stavové premenné majú len dva stavy, vypnutý, alebo zapnutý. Iné ich majú viac, ako napríklad `glPolygonMode()`. U ostatných je stav reprezentovaný dátovou premennou (napríklad `glPointSize()`, pomocou ktorej sa dá špecifikovať priemer rasterizovaného bodu). Na získanie aktuálneho stavu stavovej premennej slúži príkaz `glGet*()` (kde `*` nahradíme dátovou premennou, napr. `GlGetDoublev()`). Každá stavová premenná má definovaný počiatočný stav.

Jednou z najdôležitejších vlastností OpenGL je prenositeľnosť, nezávislosť na platforme. Aby OpenGL mohlo dosiahnuť tejto vlastnosti, neobsahuje žiadne príkazy pre tvorbu okien, audia, textového výstupu, alebo spracovania vstupu z periférnych zariadení (myš, klávesnica, atď.). Taktiež neobsahuje príkazy pre tvorbu komplexných trojrozmerných geometrických modelov. Takéto modely si musíme vytvoriť zo základných geometrický útvarov sami, konkrétne nám OpenGL ponúka body, čiary a polygóny. Vymodelovať komplexný model, napríklad vesmírnej lode, z trojuholníkov teda nie

je úplne triviálna záležitosť. Touto problematikou sa budeme zaoberať neskôr, v podkapitole 3.4.6 „Modelovanie komplexných 3D objektov“.

Aké príkazy, okrem príkazov pre modelovanie jednouchej geometrie, teda OpenGL ponúka? Z tých najpotrebnejších sú to napríklad príkazy pre mapovanie súradníc v priestore do súradníc grafického kontextu, práca s maticami (rotácie, posuny, atď.), práca s farbami, práca so svetlom, alebo mapovanie textúr.

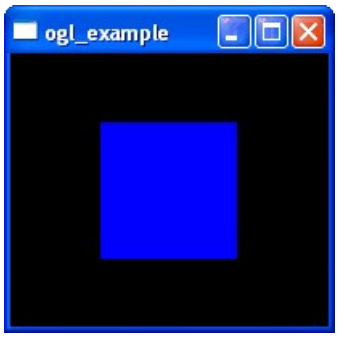
Príklad:

Jednoduchý príklad programovania v OpenGL. Vykreslenie modrého štvorca na čierne pozadie.

```
glClear(GL_COLOR_BUFFER_BIT);       /* zresetuje farbu */

glLoadIdentity();                   /* nahrá čistú maticu */
glMatrixMode(GL_PROJECTION);        /* prepne mód matice na maticu projekcie */
glOrtho( -1, 1, -1, 1, 1, 10 );     /* mapuje súradnice okna */

glMatrixMode(GL_MODELVIEW);         /* prepne mód matice na maticu modelu */
glTranslatef(0.0, 0.0, -5.0);       /* posunie maticu modelu o -5 po osi z */
glColor3f(0.0, 0.0, 1.0);           /* nastaví farbu na modrú */
glBegin(GL_QUADS);                  /* zapne vykresľovanie štvorcov */
      glVertex3f(-0.5, 0.5, 0.5);   /* ľavý-horný bod štvorca*/
      glVertex3f(0.5, 0.5, 0.5);    /* pravý-horný bod štvorca*/
      glVertex3f(0.5, -0.5, 0.5);   /* pravý-dolný bod štvorca*/
      glVertex3f(-0.5, -0.5, 0.5); /* ľavý-dolný bod štvorca*/
glEnd();
```



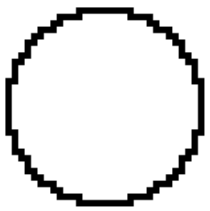
Obrázok 2.2: Výstup príkladu

Ale OpenGL neostáva len pri týchto jednoduchých príkazoch, ponúka nám techniky pre vylepšenia zobrazovania grafického výstupu. Jednou z týchto techník je aj anti-aliasing.

2.1.1 Anti-aliasing

V grafike označujeme pojmom „alias“ nežiadúci grafický artefakt, ktorý vznikol vplyvom diskrétneho prostredia. Existuje viac druhov aliasov, nás zaujíma plošný aliasing na hranách (aliasing ktorý sa dá pozorovať aj na jednom snímku).

Ako vlastne aliasing vzniká? Podstata problému je prevod vektorového objektu do rastru. Raster je v počítači typicky reprezentovaný ako mriežka pixlov. Problém je, že veľkosť plochy pixlu nie je zanedbateľná. Pri rasterizácii polygónu sa postupne vyfarbujú tie pixle ktorých stredy ležia v ploche polygónu. Obdobne je to aj s neplošnými útvarmi, kde sa zas berie v úvahu vzdialenosť od priamky. Takýmto vyfarbovaním pixlov dochádza k vzniku aliasingu, zúbkovaniu na hranách objektov.



Obrázok 2.3: Aliasing,na priblíženej kružnici.

Obrázok 2.4: Anti-aliasing, na priblíženej kružnici.

Anti-aliasing je proces, ktorý takto zúbkované hrany vyhladzuje. V OpenGL sa jedná o výpočet hodnoty pokrytia jednotlivých pixlov fragmentami z vykresľovaného objektu. Tieto hodnoty sa následne vynásobia s alfa hodnotami pixlov. Novo vzniknutá alfa hodnota sa potom použije na zmiešanie fragmentu s odpovedajúcimi pixlami vo framebufferi[1] a tým docielime vyhladenie objektu.

Anti-aliasing využívame v OpenGL aplikácii nasledovne:
Pomocou príkazu `glHint()` nastavujeme spôsob vyhladzovania (`GL_FAST`, `GL_DONT_CARE` a `GL_NICEST`) pre jednotlivé typy objektov (body, čiary, polygóny). `GL_FAST` reprezentuje najefektívnejšie, najrýchlejšie vyhladzovanie. Naopak pri nastavení `GL_NICEST` nám ide o čo najvyššiu kvalitu. Pri GL_DONT_CARE, ako už sám názov napovedá, nám nezáleží ktorá možnosť sa použije.

1 Framebuffer je obrazové výstupné zariadenie, ktoré prenáša obrazovú informáciu z vyrovnávacej pamäte. Informácie v tomto bufferi sú číselné hodnoty farieb pre každý pixel na obrazovke.

V OpenGL nestačí nastaviť anti-aliasing s `glHint()`, je potrebné tento stav ešte aktivovať a to pomocou `glEnable()` (napr. `glEnable(GL_LINE_SMOOTH)`). Ďalej treba zvoliť funkciu miešania a povoliť ju. V RGBA móde sa typicky nastavuje miešanie takto: `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`.

Príklad:
Nasledujúca sekvencia kódu patrí do inicializácie OpenGL kontextu. Jedna sa o nastavenie anti-aliasingu pre čiary a body.

```
glHint(GL_LINE_SMOOTH, GL_NICEST);
glHint(GL_POINT_SMOOTH, GL_NICEST);
glEnable(GL_LINE_SMOOTH);
glEnable(GL_POINT_SMOOTH);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_BLEND);
```

2.1.2 Multisampling

Multisampling je typ anti-aliasingu, pri ktorom sa jednotlivé pixle v scéne rozdelia na určitý počet subpixlov. Pokiaľ každý subpixel leží vo vnútri polygónu, potom aj pixel leží vo vnútri polygónu a jeho farba sa vzťahuje na všetky jeho subpixle. Ak však niektorý zo subpixlov skúmaného pixlu leží mimo polygónu, potom je jasné, že pixel musí byť okrajový a je potrebné ho vyladiť.
Údaje o subpixloch sú uložené mimo scénu v „multisampling bufferi“[2].
Pri multisamplingu nie je potrebné zoraďovať polygóny.

Implementácia multisamplingu v OpenGL je jednoduchá, stačí zabezpečiť OpenGL kontext s podporou multisamplingu a potom už len zavolať funkciu `glEnable(GL_MULTISAMPLE)`. Ako zabezpečiť OpenGL kontext s podporou multisamplingu v Qt si povieme neskôr, v sekcii „OpenGL v Qt“.

2.1.3 Zdvojená vyrovnávacia pamäť

V počítačovej grafike sa táto technika používa na odstránenie nepríjemného preblikávania scény. Anglicky „double buffering“, občas sa jej tiež hovorí „ping-pong buffering“. Ide o prostý princíp,

2 Špeciálna nadstavbu nad framebufferom, v ktorej sú uložené farba, hĺbka a stencil hodnota jednotlivých subpixlov.

OpenGL scéna sa vykresľuje do bufferu B1, buffer B2 sa vykreslí na obrazovku. Pri ďalšom priechode funkciou na vykreslenie scény sa naopak napĺňa buffer B2, zatiaľ čo sa vykreslí buffer B1.

Implementáciu zdvojeného bufferu v Qt si vysvetlíme v kapitole 2.3 „OpenGL v Qt“.

2.2 Qt

Qt je všestranný C++ toolkit (sada nástrojov) pre vývoj platformovo nezávislých aplikácií s konceptom „píš raz, prelož kdekoľvek“. S využitím jediného zdrojového stromu (angl. single source tree) a jednoduchej re-kompilácie sú aplikácie funkčné pre širokú škálu operačných systémov, vrátane vynikajúcej podpory 3D grafiky, multimédií a viacjazyčnosti. Qt podporuje vývoj nie len GUI[3], ale aj konzolových aplikácií (napr. konzolové nástroje, servery).

Obrázok 2.5: Qt logo

Najznámejšie aplikácie vyvíjané v Qt sú: KDE, Opera, Google Earth, Skype, Qt Extended, VirtualBox a OPIE. Qt nadobudlo reputáciu ako ultiplatformne orientovaný toolkit, avšak niektoré spoločnosti ho využívajú aj na vývoj pre konkrétnu platformu. Takýmto príkladom je aplikácia Adobe Photoshop Album, vyvíjaná pre platformu MS Windows.

Tvorcom a vývojárom Qt je nórska spoločnosť Qt Software (kedysi známa ako Trolltech), patriaca pod spoločnosť Nokia.[2][3]

Čo teda znamená „Qt“? Písmeno „Q“ bolo vybrané ako prefix tried, pre svoj pekný vzhľad v Haavardovom fonte pre Emacs (textový editor). Písmeno „t“ vyjadruje toolkit, inšpiráciou bol v tomto prípade Xt (X Toolkit). Taktiež sa skratka Qt hodila pre prvotný názov spoločnosti, Quasar Technologies.

Qt podporuje 32-bitové, ale aj 64-bitové platformy. Z tých najznámejších to sú napríklad Apple Mac OS X, Linux, Microsoft Windows XP/Vista, Solaris a Embedded Linux. Na väčšine podporovaných platformách môžeme Qt preložiť pomocou prekladača GCC.

2.3 OpenGL v Qt

V Qt toolkite je pre integráciu OpenGL vyčlenený samostatný modul, `QtOpenGL`. Jedna z najdôležitejších vecí ktorú nám tento modul ponúka je `QGLWidget`. Jedná sa o OpenGL okno, s ktorým môžeme v Qt pracovať obdobne ako napríklad s `QWidget`[4]. Využívame ho pre renderovanie OpenGL grafiky.

3 GUI - grafické užívateľské rozhranie, angl. graphical user interface

4 QWidget - Základná implementačná jednotka v Qt, základné okno

`QGLWidget` nám ponúka základné funkcie pre zobrazenie OpenGL grafiky, integrovanej do Qt aplikácie. Sú to virtuálne funkcie, ktoré vo svojej aplikácii prepíšeme.

- **`paintGL()`** - Renderovanie OpenGL scény, táto metóda sa volá vždy pri aktualizácii okna.
- **`resizeGL()`** - Volá sa pri zmene veľkosti okna a taktiež pri prvom zobrazení okna. V tejto metóde mapujeme súradnice okna do vykresľovanej scény.
- **`initializeGL()`** - K zavolaní tejto metódy dochádza len raz, a to pred prvým volaním `paintGL()`, alebo `resizeGL()`. Nastavujú sa tu základné parametre OpenGL, dochádza tu k definícii displej listov.

Pri zavolaní niektorej z týchto troch metód sa zároveň aktivuje OpenGL kontext (z dôvodu, aby Qt vedelo kam vykresľovať OpenGL príkazy). Ak chceme aktualizovať scénu nemali by sme volať metódu `paintGL()` priamo, ale pomocou `updateGL()`. Prekreslenie scény týmto zaradíme do fronty udalostí, ktoré okno vykonáva.

Pri hrách sa zvyčajne na prekresľovanie scény využíva `QTimer`. Jedná sa o časovač udalostí. Načasovaním `updateGL()` metódy pomocou `QTimer` s nastaveným časovaním na 0ms dosiahneme, aby sa scéna prekresľovala v každom priechode fronty udalostí (teda tak často ako sa dá).

Na platforme MS Windows pri zmene rodiča `QGLWidget` dochádza k znovu vytvoreniu OpenGL kontextu (tzn. opätovné volanie `initializeGL()`), na túto skutočnosť si musíme dať pozor najmä pri implementácii režimu celej obrazovky (angl. fullscreen).

Pred vytvorením OpenGL okna, je dobré si najprv nastaviť `QGLFormat`. Jedná sa o podporu rôznych techník, ktoré nám ponúka OpenGL.

- Zdvojený buffer
- Depth buffer
- RGBA
- Alfa kanál
- Akumulačný buffer
- Stencil buffer
- Stereo buffer
- Direct rendering
- prítomnosť vrstiev
- Multisampling

Ak chceme použiť techniku, ktorú OpenGL ovládač/hardvér nepodporuje, Qt vytvorí `QGLWidget` čo najviac podobný špecifikovaným požiadavkám.

2.4 Porovnanie Qt s inými toolkitmi

Z dôvodu lepšieho pochopenia výhod a nevýhod použitia Qt toolkitu, si ho porovnáme s inými toolkitmi z hľadiska efektívneho vývoja výkonnej a užívateľsky prívetivej GUI aplikácie. Qt si postupne porovnáme s týmito multiplatformovo orientovanými toolkitmi:

- AWT/Swing
- GTK+
- wxWidgets

Na konci každej podkapitoly zhrnieme porovnanie do niekoľkých stručných bodov. Znamienko pred každým bodom bude charakterizovať, či je skúmaná vlastnosť Qt toolkitu v danom bode výrazne lepšia „++“, lepšia „+“, zhodná „+/-“, horšia „-“, alebo výrazne horšia „--“.

2.4.1 AWT/Swing v porovnaní s Qt

Najväčší rozdiel medzi týmito dvoma toolkitmi je v ich implementačnom jazyku. Zatiaľ čo Qt je postavené na C++, AWT a Swing sú toolkity využívajúce Javu.

Obrázok 2.6: Java logo

„Abstract Windowing Toolkit“ (AWT) vznikol spolu s prvými verziami programovacieho jazyka Java. Pre vykresľovanie GUI komponentov využíva natívny kód danej platformy (napr. Win32 API pre Windows, Motif pre Unix). Takýto spôsob znamená, že program písaný v AWT sa bude chovať a aj vyzerať odlišne na rôznych platformách, keďže GUI komponenty spravuje a vykresľuje priamo konkrétna platforma.

Swing (rozšírenie AWT) touto limitáciou netrpí. Natívne knižnice už Swing využíva len pre tie najzákladnejšie potreby, ako napríklad vytvorenie okna, spracovanie udalostí a jednoduché vykresľovanie. Všetko ostatné už zvláda Swing samostatne, vrátane vykreslenia GUI komponentov. To rieši problém nejednotného správania a vzhľadu aplikácií na rôznych platformách.

Slabinou Swingu je prekvapujúco práve jeho implementačný jazyk. Java nie je známa ako príliš efektívny programovací jazyk a tak na slabšom hardvéri môžeme pozorovať pomalú odozvu a prekresľovanie užívateľského rozhrania. Java ako programovací jazyk má iné prednosti.

Napríklad lepšiu efektivitu programovania, nakoľko sa programátor nemusí starať o správu pamäte. Táto skutočnosť je však pri vykresľovaní GUI nepodstatná, a tak v tomto ohľade Java stráca na jazyky ako je C/C++.

Prečo je chod Java aplikácií pomalší oproti chodu C++ aplikácií? C++ aplikácie sú prekladané pomocou C++ prekladača priamo do binárneho formátu. Táto skutočnosť umožňuje C++ aplikácii, aby jej chod bol prevedený priamo na CPU. Celý chod aplikácie sa tak odohráva na hardvéri.

Oproti tomu Java aplikácie sú kompilované pomocou Java prekladača do bytového kódu. Takto preložený kód nie je spracovaný priamo na CPU, ale softvérovo pomocou Java Virtual Machine (JVM). JVM je softvér, ktorý beží na CPU. Takže chod Java aplikácie je vlastne softvérová emulácia, čo spôsobuje spomalenie.

Vráťme sa ale k porovnávaniu toolkitov. Čo sa týka vykresľovania GUI komponentov, tak Qt sa podobá viac Swingu ako AWT. Rovnako využíva natívne knižnice operačných systémov len pre tvorbu okien a o vykresľovanie GUI komponentov sa stará samo. Výhoda Qt je v efektívnosti C++, takže GUI netrpí pomalou odozvou. Qt aplikácie sú v niektorých prípadoch dokonca rýchlejšie ako aplikácie vytvorené pomocou natívnych knižníc, nakoľko Qt využíva špeciálnu techniku s využitím vyrovnávacej pamäte.

Jednou z povšimnutia hodných vecí je technika štýlovania aplikácií, ktorá umožňuje zobraziť GUI aplikáciu v niektorom z viacerých štýlov nezávisle od platformy. Túto techniku podporuje jednak Qt ako aj Swing. Je to možne vďaka tomu, že vykresľovanie sa nespolieha na natívne knižnice, ale oba toolkity ho implementujú samostatne. Zaujímavosťou je, že Qt ponúka štýl, ktorý imituje implicitný vzhľad Swing aplikácií. Mimo iné ponúka Qt aj štýly ímitujúce Win32, Motif, Macintosh, alebo aj MacOS X Aqua.

Programové paradigma (štýl programovania) je viacmenej osobitný pohľad každého programátora. Avšak niektoré toolkity ponúkajú elegantnejší a prehľadnejší kód aplikácie ako iné. Teraz uvidíme dva kúsy kódu, prvý písaný v Java/Swing a druhý v C++/Qt. Obe ukážky vkladajú niekoľko položiek do hierarchického stromového náhľadu (GUI komponenta).[4]

```
// Ukážka kódu v Swingu
DefaultMutableTreeNode root =
     new DefaultMutableTreeNode("Root");
DefaultMutableTreeNode child1 =
     new DefaultMutableTreeNode("Child 1");
```

```
DefaultMutableTreeNode child2 =
     new DefaultMutableTreeNode("Child 2");
DefaultTreeModel model =
     new DefaultTreeModel( root );
JTree tree = new JTree( model );
model.insertNodeInto(child1, root, 0);
model.insertNodeInto(child2, root, 1);

// Tá istá ukážka v Qt
QListView* tree = new QListView;
QListViewItem* root = new QListViewItem(tree, "Root" );
QListViewItem* child1 = new QListViewItem(root, "Child 1");
QListViewItem* child2 = new QListViewItem(root, "Child 2");
```

Ako môžeme vidieť, Qt kód je značne intuitívnejší. Je to z toho dôvodu, že Swing používa model-view-controller (MVC) architektúru. Jedná sa o softvérovú architektúru, ktorá rozdeľuje dátový model aplikácie, užívateľské rozhranie a riadiacu logiku do troch nezávislých komponent tak, že modifikácia niektorej z nich má minimálny vplyv na ostatné. Qt tento prístup podporuje, avšak ho nevnucuje.

Výhody a nevýhody Qt oproti AWT/Swing:

- +/- Qt ako aj AWT/Swing podporujú viacero platforiem
- ++ Java (Swing) je v porovnaní s C++ (Qt) menej vhodná pre tvorbu zložitejších GUI aplikácií, nakoľko je spúšťaná softvérovou emuláciou (cez Java Virtual Machine) a odozva GUI aplikácie tak môže byť nepríjemne spomalená.
- \+ Programové paradigma Qt je čitateľnejšie a elegantnejšie, keďže Qt nenúti používať „model-view-controller“ softvérovú architektúru.
- ++ Qt ponúka kvalitnejšie vývojárske nástroje, prepracovanejšiu dokumentáciu a množstvo užitočných príkladov.
- +/- Java/AWT/Swing sú vhodné pre menšie GUI aplikácie, nakoľko je osvojenie si programovania v tomto toolkite o niečo rýchlejšie aj napriek vynikajúcej Qt dokumentácii. Avšak pre rozsiahlejšie aplikácie je C++/Qt jednoducho kvalitnejšia a lepšia voľba.

2.4.2 GTK+ v porovnaní s Qt

„The GIMP Toolkit“ (GTK+), rovnako ako Qt podporuje viacero platforiem, vrátane MS Windows a Mac OSX. Primárna platforma však pre tento toolkit vždy bola a pravdepodobne aj bude Linux/Unix. GTK+ si oproti Qt toolkitu zachováva konzistentnejší dojem pri prechode z platformy na platformu, pretože vzhľad ostáva úplne rovnaký. Qt sa naopak snaží imitovať čo najviac natívnych prvkov ako sa dá. V tomto prípade sa len ťažko rozhoduje, čo je lepšie a čo horšie. Je to čisto záležitosť vkusu.

Obrázok 2.7: GTK+ logo

Čo sa týka portability do vstavaných zariadení, môžeme povedať, že Qt je o krôčik popredu. Qt beží priamo na hardvéri zariadenia bez nutnosti akéhokoľvek okenného systému a správcu. Naproti tomu GTK+ potrebuje na vstavanom zariadení minimálne X11 server a správcu okien.

Qt toolkit sa pýši svojou konzistenciou, stačí jednoducho pridať do kódu napríklad modul QHttp (*#include<QHttp>)* a môžeme využívať funkcie na sťahovanie http stránok a spracovávanie formulárov. GTK+ je naproti tomu čisto GUI toolkit, takže na podobné operácie by sme museli v našej aplikácii využiť Glib, GIO, GNet, alebo iné potrebné knižnice.

GTK+ je napísané kompletne v jazyku C. Táto skutočnosť dopomohla toolkitu získať previazanosť s mnohými známymi ale a menej známymi programovacími jazykmi. Z tých menej pravdepodobných sú to napríklad Pascal, Vala, JavaScript, D a mnohé ďalšie. Zaujímavé je, že GTK+ podporuje aj tvorbu v C++ a to pomocou gtkmm.

Ďalšou výhodou jazyka C je spoľahlivý a hlavne rýchly kód. Na druhú stranu to moc neprospieva celkovej prehľadnosti implementovanej aplikácie, kde je treba tak trochu viac „námahy“ (viac riadkov kódu) oproti Qt. V Qt jednoduchý a prehľadný paradigma signálov a slotov je v GTK+ nutné programovať pomocou spätných volaní. Dôsledok tohto prístupu je pomerne obtiažne debugovanie programu[5].

Dokumentácia, príklady a nástroje pre vývoj softvéru sú silnou stránkou Qt toolkitu, a tak sa nemusíme diviť, že v tomto ohľade GTK+ ako aj väčšina ostatných toolkitov za Qt zaostávajú. Za zmienku stojí obstojný a užívateľmi často vychvaľovaný Glade, nástroj pre vizuálnu tvorbu GUI štruktúry a spätných volaní pre GUI komponenty. Obdobný nástroj však môžeme pozorovať aj z dielne Qt Software, Qt Designer.

5 Proces hľadania a odstraňovania chýb v programe.

Jednou z vecí, ktoré Qt toolkit dlho postrádal bola licencia, ktorá by nám umožňovala komerčné využitie aplikácie programovanej pomocou voľne dostupnej verzie Qt. Takáto licencia (LGPL) bola prijatá v januári 2009 spoločnosťou Nokia. Dovtedy sa dala voľne dostupná verzia Qt využiť len pod GPL licenciou na „open source“ (otvorený kód) využitie. Platená verzia sa samozrejme dala využiť aj pre komerčné aplikácie.

GTK+ je distribuované pod licenciou LGPL ktorá, ako už vieme, umožňuje aj komerčné šírenie aplikácie so zatvoreným kódom.

Výhody a nevýhody Qt oproti GTK+:

- \+ Oba toolkity sú multiplatformovo orientované, Qt má miernu prevahu na poli vstavaných zariadení.
- +/- Rýchlosť odozvy aplikácie je v oboch toolkitoch výborná vďaka prog. jazyku C a C++.
- +/- GTK+ toolkit je zameraný čisto na tvorbu GUI, naproti tomu Qt je komplexnejšie. Z hľadiska pohodlia užívateľa je táto skutočnosť výhodou pre Qt, avšak aplikácie sú takto náchylné zaberať viac pamäte ako je skutočne potrebné.
- \+ Spätné volania (GTK+) oproti signálom a slotom (Qt).
- +/- Konzistencia GTK+ aplikácií pri prechode z platformy na platformu. Qt sa snaží imitovať natívny vzhľad vlastnou implementáciou GUI.
- \+ Dokumentácia, príklady a nástroje sú opäť v prospech Qt toolkitu, avšak Glade sa plnohodnotne vyrovná nástroju Qt Designer.
- +/- Licenčné podmienky sú pre oba toolkity prakticky rovnaké (LGPL). Umožňujú komerčne využitie, ako aj uzatvorený zdrojový kód aplikácie.

2.4.3 wxWidgets v porovnaní s Qt

Pôvodný názov toolkitu wxWidgets bol wxWindows, tento projekt odštartoval Julian Smart, ktorý je dodnes jedným z hlavných vývojárov. Tento toolkit patrí medzi „free software“ (voľne šíriteľný a dostupný softvér).

Obrázok 2.8: wxWidgets logo

Multiplatformovo orientovaný toolkit wxWidgets ponúka podporu pre bežne používané platformy, ako je MS Windows, Mac OS X, Linux/Unix. Podporovaná sú aj napríklad platformy AmigaOS, OS/2, alebo OpenVMS. Podpora pre vstavané systémy však chýba.

Implementačný jazyk je rovnaký ako pre Qt toolkit. Avšak wxWidgets neostávajú len pri C++, pripravená je aj väzba pre široké spektrum bežne využívaných programovacích jazykov. Z tých menej bežných sú to napríklad Lua, Lisp, alebo aj Smalltalk.

Ako už bolo spomenuté wxWidgets chýba podpora pre vstavané zariadenia. Qt je v tomto ohľade o dva kroky dopredu. Jediné čo Qt potrebuje, je základný linux s podporou framebufferu (takže stačí „/dev/fb“).

Ako wxWidgets zvládajú portabilitu GUI pre rôzne platformy? Tento toolkit skutočne využíva natívne GUI knižnice operačných systémov. To znamená, že ak preložíme program pod linuxom, bude mať vzhľad z knižnice Motif a ak bude preložený pre MS Windows, osvojí si program vzhľad z natívnej knižnice Win32 API. Qt ako už vieme implementuje vlastné GUI, ktoré sa snaží imitovať vzhľad platformy. Oba spôsoby majú svoje pre a proti, ktoré sú založené skôr na subjektívnom názore programátorov a užívateľov aplikácie.

Základná trieda Qt toolkitu QWidget využíva vo verzii 4.5 celkom 37 virtuálnych funkcií. Týmto získava Qt viac objektovo-orientovaný dizajn, jednoduchšiu implementáciu aplikácie pre programátora, avšak môže to znamenať o niečo pomalší chod programu. wxWidgets využívajú viac makier, čo znamená síce rýchlejší beh aplikácie avšak väčšie nároky spustiteľného súboru na pamäť. Muselo by sa však jednať o väčšie množstvo virtuálnych funkcií, alebo makier, aby sa táto skutočnosť citeľne prejavila.

Dokumentácia pre wxWidgets je síce rozsiahla, avšak trochu roztrúsená a neorientuje sa v nej s takou ľahkosťou ako v Qt dokumentácii. Navyše chýba značný počet príkladov pre rôzne druhy problémov. Čo sa týka nástrojov pre vývoj softvéru je na tom opäť o niečo lepšie toolkit Qt.

Výhody a nevýhody Qt oproti wxWidgets:

- \+ Qt ako aj wxWidgets podporujú viacero platforiem, avšak wxWidgets chýba podpora pre vstavané systémy.
- +/- Rýchlosť odozvy aplikácie je v oboch toolkitoch výborná vďaka prog. jazyku C++.
- +/- Komplexnosť oboch toolkitov je zrovnateľná.
- +/- wxWidgets sa snažia o čisto natívnu implementáciu GUI pri prechode z platformy na platformu. Qt sa snaží imitovať natívny vzhľad vlastnou implementáciou GUI.

+ Vďaka pomerne neucelenej dokumentácii wxWidgets a chýbajúcim príkladom implementácie má v tomto ohľade Qt opäť prevahu. wxWidgets citeľne chýba nástroj podobný qmake, ktorý by rádovo zjednodušil preklad programov.
- wxWidgets patria do rodiny „free software“ a tak si môžeme určiť vlastné podmienky distribúcie aplikácie napísanej vo wxWidgets. Qt ako už vieme tento rok prijalo licenciu LGPL, ktorá mu poskytla potrebnú voľnosť pre komerčné využitie programov písaných s pomocou voľne šíriteľného Qt toolkitu. V tomto ohľade je toolkit wxWidgets o kúsok popredu.

3 Návrh a implementácia aplikácie

V tejto kapitole sa oboznámime s požiadavkami na aplikáciu a návrhom. Priblížime si implementáciu so zameraním na riešenie niektorých zaujímavých problémov. Nakoniec zhodnotíme výsledky a ukážeme si obrázky finálneho produktu.

3.1 Požiadavky

Aplikácia je zameraná na ukážku skĺbenia OpenGL spolu s Qt toolkitom. To znamená, že aplikácia bude využívať Qt GUI prvky a zároveň OpenGL techniky trojrozmerného zobrazenia scény. OpenGL scéna a GUI budú vzájomne komunikovať.

Čo sa týka OpenGL, aplikácia by mala byť dostatočne dynamická a nie príliš jednoduchá. Takéto správanie aplikácií najlepšie odzrkadľujú hry. Dynamičnosť sa v nich dá dosiahnuť pohybom, do ktorého zasahuje užívateľ jednoduchým ovládaním. Zložitosť aplikácie dosiahneme implementáciou rôznych OpenGL techník. Aplikácia by teda mala využívať textúry, displej listy, komplexné modely, multisampling, anti-aliasing.

Z pohľadu Qt, bude grafické užívateľské rozhranie prispôsobené OpenGL scéne. Nie je potrebné demonštrovať plnú silu Qt toolkitu, práve naopak. Treba vkusne vybrať GUI komponenty ktoré sa budú hodiť pre prácu s OpenGL, netreba zabúdať, že sa jedná o ukážu skĺbenia týchto dvoch živlov.

Aplikácia bude určená pre programátora, ktorý chce vidieť funkčný výsledok spolupráce OpenGL a Qt. To mu pomôže rozhodnúť sa, či je pre neho výhodné oboznámiť sa bližšie s Qt toolkitom. Inak povedané, či sa pre budúcu tvorbu OpenGL programu, rozhodne použiť Qt.

3.2 Návrh

Aplikáciu navrhneme z ohľadom na požiadavky, ktoré sme si priblížili v predchádzajúcej podkapitole. Aplikácia bude objektovo orientovaná a jednotlivé triedy budú zabezpečovať jej funkčnosť.

Ako už bolo spomínané v požiadavkoch, budeme implementovať hru. Bude sa jednať o dynamickú hru, simulátor letu vesmírnej lode cez zakrivený, oválny objekt podobajúci sa červej diere.

V prvom rade potrebujeme hlavé okno aplikácie. Triedu tohto okna nazveme `cMainWindow`. Všetky ďalšie triedy budú pracovať vrámci tohto hlavného okna. Funkcia okna je teda jednoznačná, bude tvoriť kostru aplikácie. Na jednej strane bude nositeľom GUI, na druhej strane bude obsahovať OpenGL scénu. Značná časť GUI bude implementovaná práve v tejto triede.

Ostatnú časť GUI budeme implementovať do triedy `cDSettings`. Táto trieda bude reprezentovať nastavenia aplikácie a pokúsime sa ju implementovať pomocou QtDesigneru (nástroja pre vizuálnu tvorbu GUI v Qt).

Pre správu OpenGL navrhneme triedu `cGLWidget`. Metódy pre inicializáciu, zmenu veľkosti a samotné vykreslenie OpenGL kontextu budeme implementovať práve v tejto triede. Triedu budeme ďalej využívať na spracovanie vstupu od užívateľa pri ovládaní hry (o spracovanie vstupu užívateľa vrámci GUI sa postará Qt).

Ďalej potrebujeme triedu pre objekty v OpenGL scéne. Na to nám poslúži trieda `cGLObject`.. Táto trieda zaistí metódy pre vytvorenie objektov a poskytne nástroj pre vytvorenie objektov priamo z „obj“ súborov. Jedná sa o exportované 3D modely.

Tento nástroj budeme implementovať v triede `cObj2OGL`. Trieda bude obsahovať metódy pre spracovanie dát z obj súborov.

Z triedy `cGLObject` si odvodíme triedy `cUfo` a `cWormhole`. Trieda `cUfo` bude reprezentovať lietajúci objekt v scéne a trieda *`cWormhole`* zase objekt, cez ktorý let prebieha.

3.3 Implementačné nástroje

Aplikáciu budeme implementovať v objektovo orientovanom jazyku C++ s využitím Qt toolkitu (verzia 4.4.3.0).

Qt ponúka pre vývoj užitočné multiplatformovo orientované nástroje a niektoré z nich pri tvorbe tejto aplikácie využijeme. Konkrétne sa bude jednať o vývojové prostredie „Qt Creator“, grafický dizajnér pre GUI „Qt Designer“ a prehliadač online dokumentácie „Qt Assistant“. Pre preklad aplikácie použijeme „qmake“, jedná sa o ďalší nástroj z dielne Qt.

Na vygenerovanie programovej dokumentácie použijeme voľne dostupný nástroj „Doxygen“, ktorý spracováva špeciálne upravené komentáre v kóde.

3.4 Implementácia

V tejto podkapitole sa budeme zaoberať implementáciou jednotlivých tried. Uvedieme si najskôr stručný prehľad všetkých tried, a tie najzaujímavejšie si potom popíšeme podrobnejšie.

Pri implementácii si vystačíme s prostriedkami Qt toolkitu a OpenGL, v tomto nám pomôžu online dokumentácie. [5][6]

Triedy a metódy sú pomenované v anglickom jazyku. Rovnako aj komentáre v kóde. Každá trieda má vlastný zdrojový kód a obsahuje hlavičku, v ktorej je uvedený autor, dátum a stručná charakteristika triedy.

Zdrojové kódy a programová dokumentácia sa nachádzajú na CD nosiči uvedenom v prílohách.

3.4.1 Prehľad tried

cMainWindow - Hlavné okno aplikácie, obsahuje značnú časť GUI.

cDSettings - Dialog nastavení s ostatnou časťou GUI.

cGLWidget - Okno pre OpenGL kontext.

cGLObject - Základná trieda objektov.

cWormhole - Trieda reprezentujúca červiu dieru.

cUfo - Trieda reprezentujúca vesmírnu loď.

cObj2OGL - Obsahuje metódy pre spracovanie súborov typu „obj“.

3.4.2 cMainWindow

Keď hovoríme o triede `cMainWindow`, hovoríme o základnej triede celej aplikácie. Táto trieda totiž obsahuje objekty tried `cGLWidget` a `cDSettings`. Poskytuje komunikačné rozhranie medzi Qt GUI prvkami a scénou OpenGL. Je odvodená od Qt triedy `QMainWindow`.

Za zmienku stojí implementácia nastavení aplikácie. Qt poskytuje prvok pre elegantné zvládnutie problému ukladania a nahrávania nastavení. Je ním `QSettings`. Stačí keď v konštruktore `cMainWindow` špecifikujeme základné informácie o aplikácii a vytvoríme objekt triedy `QSettings`.

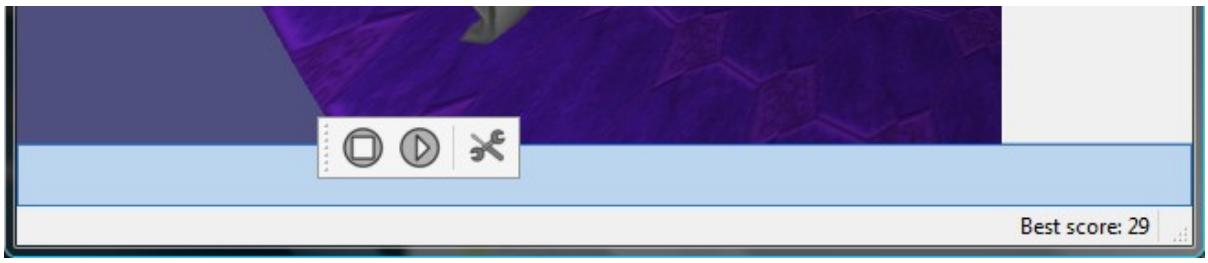
```
QCoreApplication::setOrganizationName("FIT");
QCoreApplication::setOrganizationDomain("fit.vutbr.com");
QCoreApplication::setApplicationName("Wormhole X-treme");
settings = new Qsettings();
```

Následne implementujeme dve metódy. Jednu pre čítanie a druhú pre ukladanie nastavení.

```
void cMainWindow::writeSettings() {
      settings->beginGroup("MainWindow");
            settings->setValue("size", size());
            settings->setValue("pos", pos());
      settings->endGroup();
}
void cMainWindow::readSettings() {
      settings->beginGroup("MainWindow");
      resize(settings->value("size", QSize(500, 400)).toSize());
      move(settings->value("pos", QPoint(100, 100)).toPoint());
      settings->endGroup();
}
```

Výhodou je, že nemusíme použiť žiaden XML parser. Takto riešené ukladanie a obnova nastavení je samozrejme multiplatformovo orientované.

Ďalšou povšimnutia hodnou vecou v triede `cMainWindow` je dokovateľný `QToolBar`. Ak ho uchopíme a presunieme ho napríklad pod OpenGL kontext, tento sa akoby nadvihne a toolbar sa uchytí na spodok aplikácie.



Obrázok 3.1: Demonštrácia uchytenia toolbaru

3.4.3 cGLWidget

Trieda `cGLWidget` je odvodená od Qt triedy `QGLWidget`. Ponúka štandardné funkcie OpenGL. Práca OpenGL sa odohráva čisto vrámci tejto triedy. Obsahuje objekty cWormhole a cUfo. Komunikuje s nimi medzi GUI a spracováva vstupy od užívateľa cez zdedené, virtuálne metódy (`keyPressEvent()`, `keyReleaseEvent()`, `mousePressEvent()`, `wheelEvent()`, `mouseMoveEvent()`, `mouseDoubleClickEvent()`).

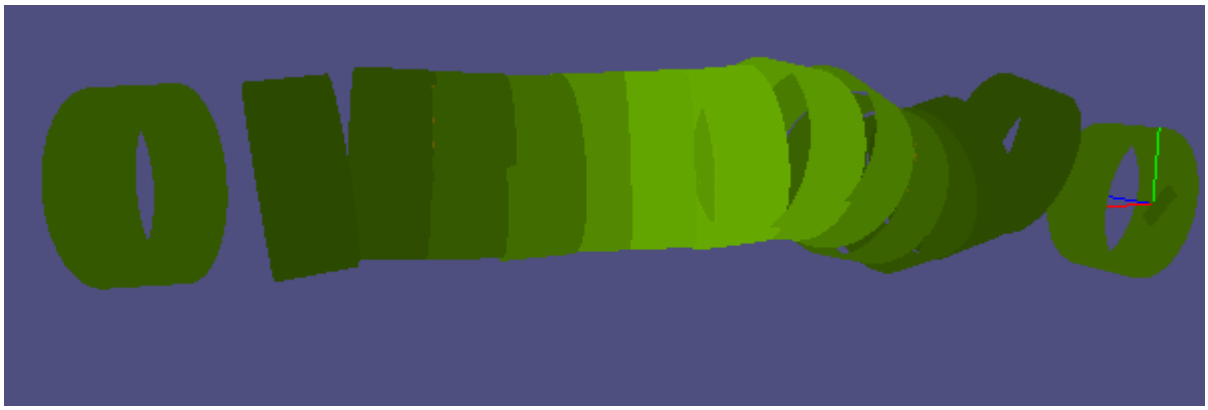
Zaujímavá je implementácia režimu celej obrazovky (angl. fullscreen). Qt síce poskytuje metódu `showFullScreen()`, avšak tú v tomto prípade nemôžme použiť. Dôvod je, že `cGLWidget` nie je samostatné okno, ale patrí pod `cMainWindow`. Metóda na prepínanie medzi normálnym režimom a režimom celej obrazovky preto musela byť implementovaná samostatne. Funguje spoľahlivo, ba dokonca je optimalizovaná aj pre viacobrazovkový režim.

3.4.4 cWormhole

Trieda `cWormhole` je odvodená od triedy `cGLObject`. V tejto triede je implementovaná červia diera, ktorou má letieť objekt triedy `cUfo`.

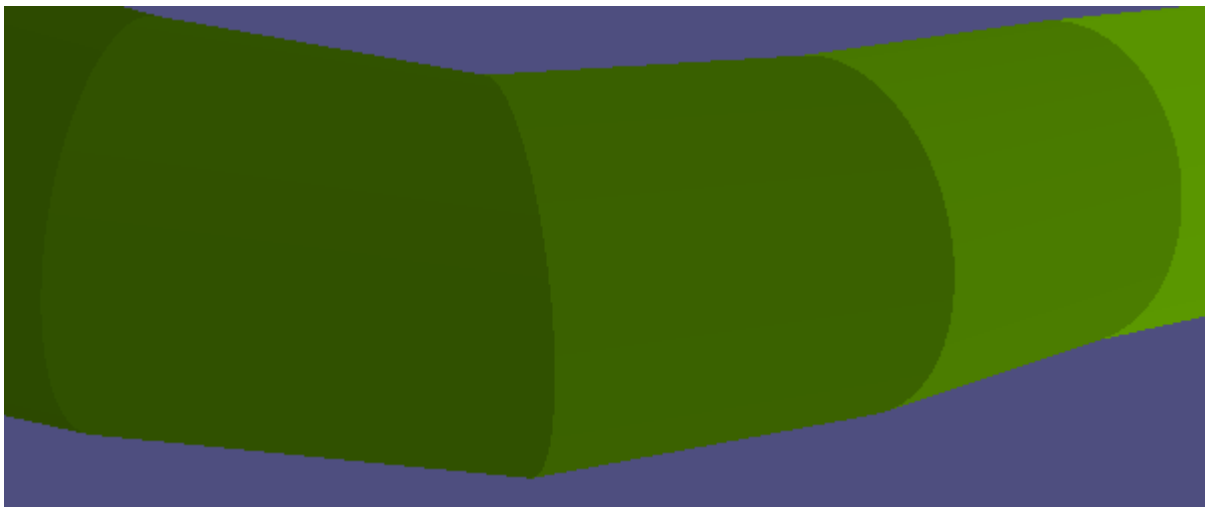
Implementácia spojitého a zakriveného valca, ktorý reprezentuje štruktúru červej diery, v OpenGL nie je tak úplne triviálna.[7]

Predpokladajme, že máme k dispozícii krivku reprezentovanú bodmi. Môže sa zdať, že nám stačí vykresliť okolo týchto bodov jednotlivé válce s výškou rozdielu vzdialenosti bodov, avšak nie je tomu tak. Týmto postupom dostaneme valce, ktoré sa prekrývajú (viď. Obrázok 3.2).



Obrázok 3.2: Chybná implementácia zakriveného valca

Správne riešenie je, že si vypočítame body tvoriace kružnice okolo jednotlivých bodov. Následne budeme spájať body jednej kružnice s nasledujúcou. Pri opakovaní tohto postupu pre všetky body krivky, dostaneme požadovanú spojitú štruktúru valca (viď. Obrázok 3.3).



Obrázok 3.3: Správna implementácia zakriveného valca

3.4.5 cObj2OGL

Pre pomoc s modelovaním komplexných 3D objektov sme si v našej aplikácii implementovali triedu `cObj2OGL`. Táto trieda ponúka metódy pre parsovanie súborov typu obj, v ktorých sú definované body a normály komplexných 3D objektov.

V prípade, že 3D objekt v súbore nemá definované normály, budú vypočítane, postará sa o to metóda makeObjectFromObjFile(). Táto metóda prijíma ako parameter názov súboru, ktorý obsahuje 3D objekt vo formáte obj, a vracia identifikačné číslo displej listu.

3.4.6 Modelovanie komplexných 3D objektov

V našej aplikácii sme vyriešili modelovanie komplexných 3D objektov elegantne, s využitím voľne dostupného nástroja **Blender**[6] v kombinácii s vlastne implementovaným parserom obj súborov.

V nástroji ako je Blender objekt jednoducho nakreslíme a exportujeme do súboru vo formátu obj. V aplikácii potom použijeme spomínaný parser ktorý nám súbor spracuje do podoby vhodnej pre implementáciu v OpenGL.

V aplikácii môžete vidieť jeden model vymodelovaný v programe Blender (model „own ship“) a dva stiahnuté z voľne dostupných modelov na „Google 3D Warehouse“[7] („small ship“ a „alien ship“).

6 Blender je voľne dostupný nástroj pre tvorbu 3D modelov a animácií.

7 Google 3D Warehouse: http://sketchup.google.com/3dwarehouse/

4 Výsledky

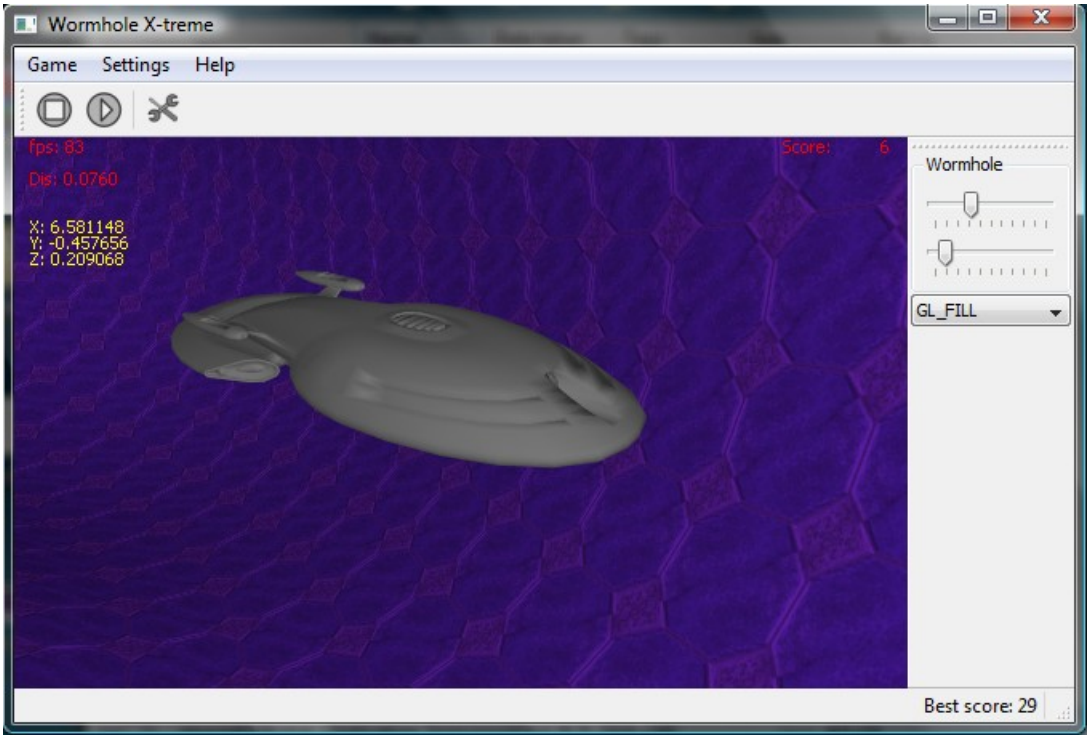
Hra nie je príliš zábavná, ale vzhľadom na zameranie aplikácie to nie je podstatné. Vývoj aplikácie prebiehal na platforme MS Windows XP. Testy prebiehali na platformách MS Windows XP a MS Windows Vista. Vďaka Qt toolkitu, môžeme aplikáciu preložiť aj na iných platformách pomocou nástroja qmake. Spustiteľná a plne funkčná verzia aplikácie pre MS Windows sa nachádza na CD nosiči v prílohách.

4.1 Štatistiky

Počet riadkov kódu: **3920**
Veľkosť zdrojových kódov: **168 kB**
Veľkosť spustiteľného súboru (MS Windows): **405 kB**
Veľkosť celej aplikácie (MS Windows): **16 MB**

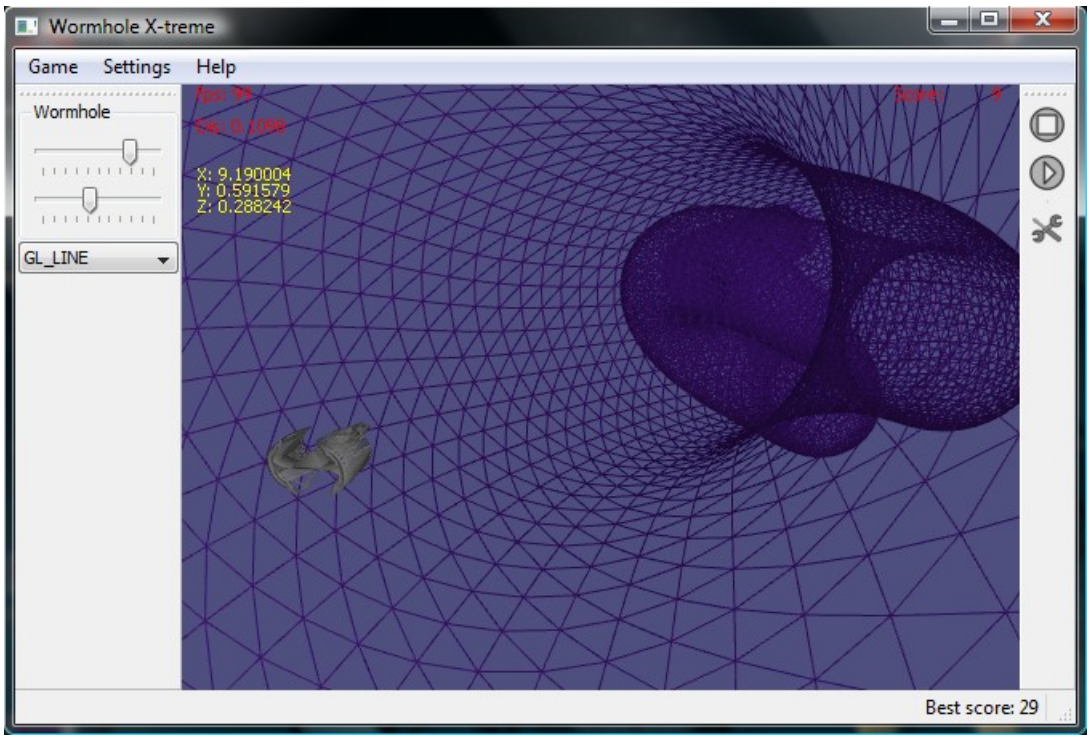
4.2 Ukážky

Pre lepšiu predstavu dosiahnutého výsledku si ukážeme zopár obrázkov implementovanej aplikácie na platforme MS Windows Vista.



Obrázok 4.1: Pohľad na model vesmírnej lode

Na nasledujúcom obrázku si môžeme všimnúť iné usporiadanie toolbarov, zmenu modelu vesmírnej lode a zmenu módu vykresľovania polygónov na čiary.



Obrázok 4.2: Pohľad na červiu dieru v čiarovom móde

5 Záver

Táto práca poskytuje čitateľovi porovnanie rôznych multiplatformovo orientovaných toolkitov. Prezentuje spoluprácu knižnice OpenGL s Qt toolkitom. Výsledná aplikácia slúži ako ukážka tohto spojenia. A má pomôcť čitateľovi rozhodnúť sa , či použije Qt pre vývoj svojho budúceho OpenGL projektu. Aplikáciu sa podarilo implementovať podľa návrhu spĺňajúc všetky požiadavky. Grafické užívateľské rozhranie je prehľadné a základné ovládacie prvky sú intuitívne.

Osobitným prínosom bola pre mňa hlavne práca v Qt a OpenGL. V Qt toolkite som programoval prvýkrát a dovolím si tvrdiť, že táto skúsenosť je pre mňa veľkým prínosom. Rovnako som sa lepšie zorientoval v multiplatformovo orientovaných toolkitoch pre tvorbu GUI aplikácií.

Aplikácia je pomerne nezáživná. Ako oblasť budúceho vývoja preto odporúčam implementáciu audio-vizuálnych efektov na zlepšenie celkového dojmu z programu.

Literatúra

[1] Dave Shreiner, Mason Woo, Jackie Neider, Tom Davis. OpenGL Programming Guide Fifth Edition. Addison-Wesley. 2006. 0-321-33573-2

[2] WWW stránky. Wikipedia. Qt (toolkit). http://en.wikipedia.org/wiki/Qt_(toolkit)

[3] WWW stránky. Qt Software. Qt White Paper. http://www.qtsoftware.com/products/files/pdf/qt-4.4-whitepaper

[4] WWW stránky. Matthias Kalle Dalheimer. Qt vs. Java. http://turing.iimas.unam.mx/~elena/PDI-Lic/qt-vs-java-whitepaper.pdf

[5] WWW stránky. Nokia. Qt Online Reference Documentation. http://doc.trolltech.com/

[6] WWW stránky. OpenGL.org. OpenGL Reference Pages. http://www.opengl.org/sdk/docs/man/

[7] Jiří Žára, Bedřich Beneš, Jiří Socher, Petr Felkel. Moderní počítačová grafika. Compuer Press. 2004. 80-251-0454-0

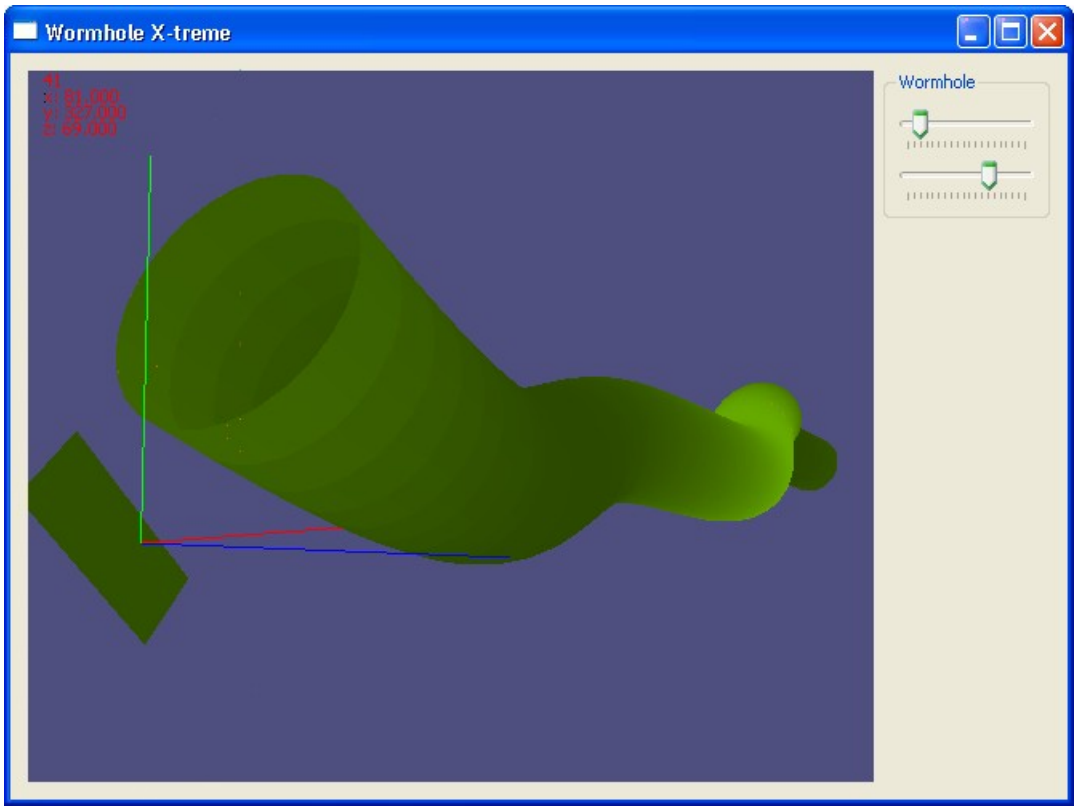
Prílohy

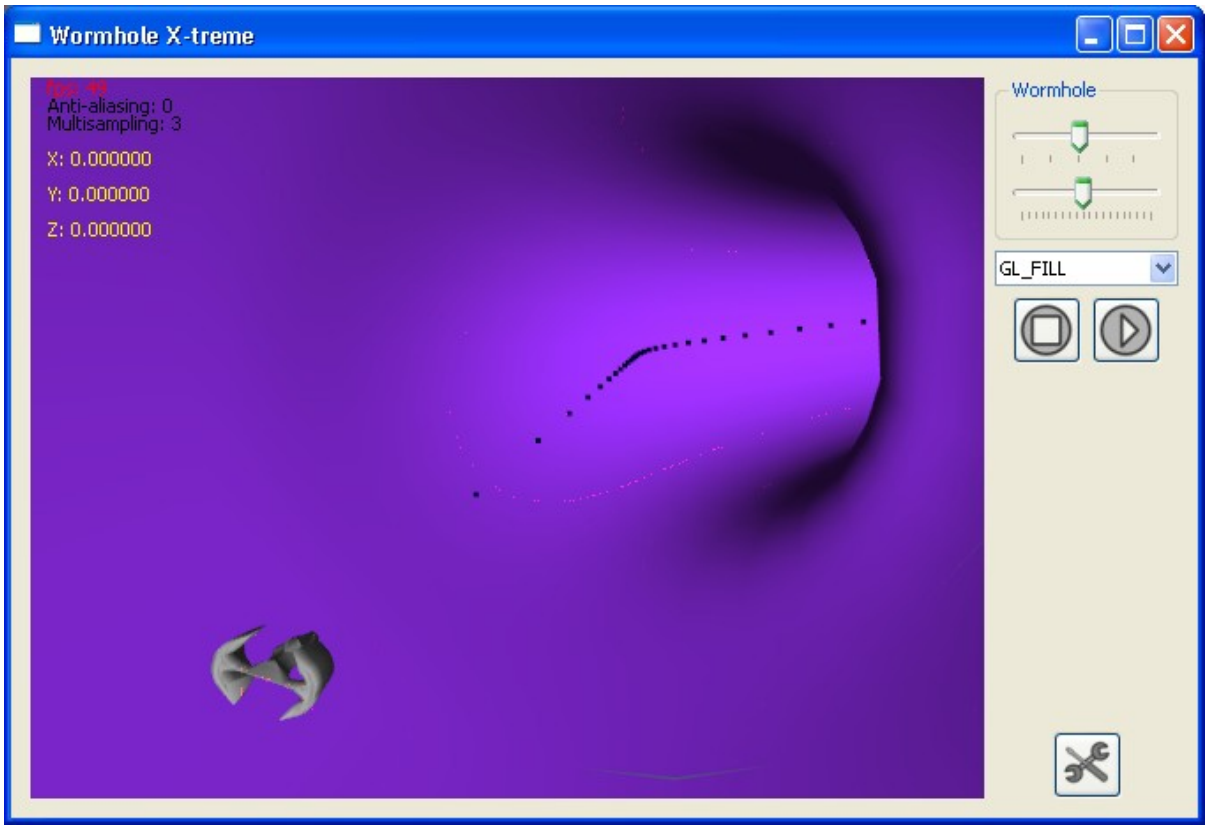
Príloha 1. Obrázky z vývoja aplikácie

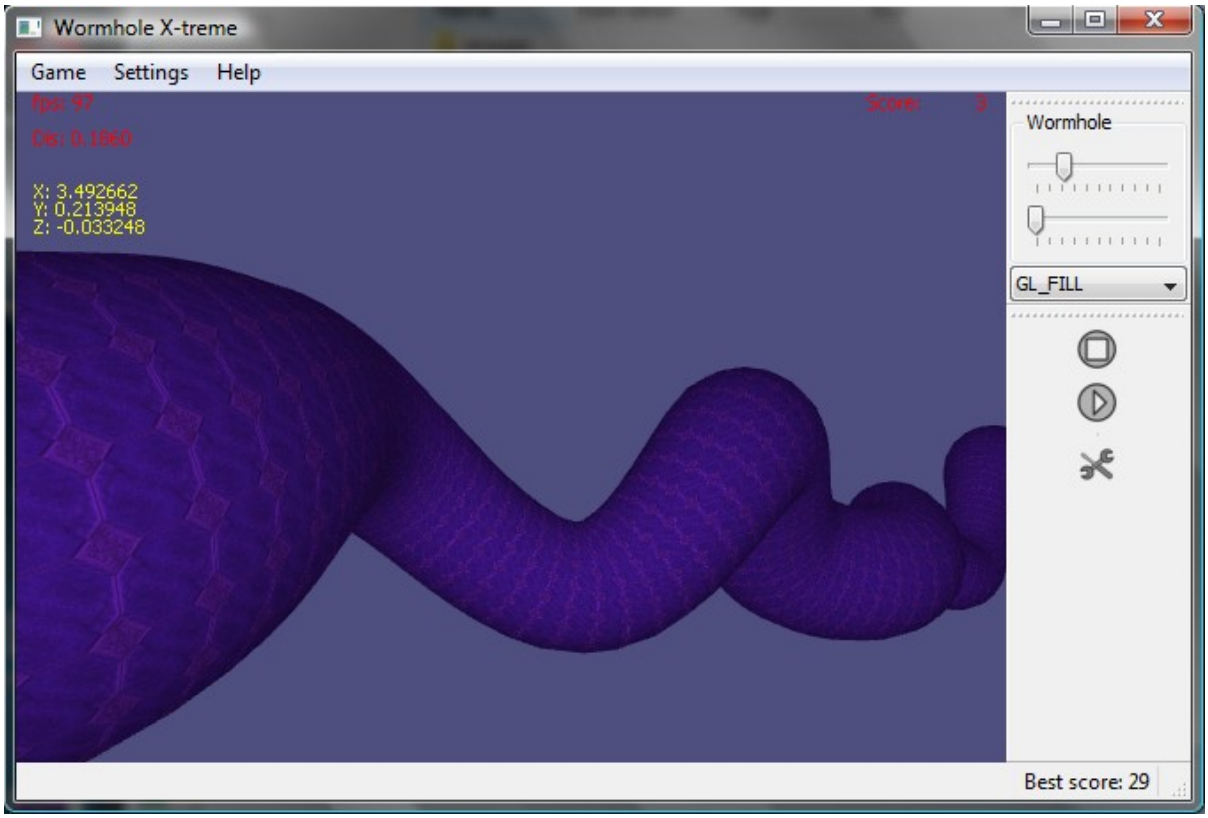
Príloha 2. CD nosič

- zdrojové kódy (v zložke „src”)
- programová dokumentácia (v zložke „doc“)
- manuál k programu (manual.pdf)
- technická správa (technicka_sprava.pdf)
- program (v zložke „wormhole“)

Príloha 1. : Obrázky z vývoja aplikácie






Wormhole X-treme
Game
Settings
Help
fps: 97
Score: 3
Dis: 0.1860
X: 3.492662
Y: 0.213948
Z: -0.033248
Wormhole
GL_FILL
Best score: 29