Vizualizace pracovního prostoru pro obráběcí stroje s využitím knihovny OpenGL a aplikační platformy Qt

Adam Říha

Bakalářská práce
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Adam Říha**
Osobní číslo:	**A14135**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**prezenční**
Téma práce:	**Vizualizace pracovního prostoru obráběcího stroje s využitím knihovny OpenGL a aplikační platformy Qt**
Téma anglicky:	**A Visualisation of a Machine Tool's Working Space Using an OpenGL Library and the Qt Application Platform**

Zásady pro vypracování:

1. **Popište metody související s popisem, analýzou a zobrazováním trojrozměrných objektů.**
2. **Popište knihovnu OpenGL, aplikační platformu Qt a formát STL.**
3. **Navrhněte metody vizualizace pracovního prostoru obráběcího stroje ve formátu STL s využitím knihovny OpenGL a aplikační platformy Qt.**
4. **Vytvořte ukázkovou aplikaci demonstrující základní principy a popište její klíčové části.**
5. **Demonstrujte výsledky a formulujte závěr.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **ŽÁRA, Jiří, BENEŠ, Bedřich, FELKEL, Petr. Moderní počítačová grafika. 1. vyd. Praha : Computer Press, 2005. 448 s. ISBN 80-7226-049-9.**
2. **PRATA, Stephen. Mistrovství v C++. 3., aktualiz. vyd. Brno: Computer Press, 2007. Bestseller (Computer Press). ISBN 978-80-251-1749-1.**
3. **Qt Documentation: Qt 4.8. Qt Documentation [online]. Finland: The Qt Company, 2016 [cit. 2017-11-26]. Dostupné z: http://doc.qt.io/qt-4.8/**
4. **OpenGL 2.1, GLX, and GLU Reference Pages. OpenGL 2.1, GLX, and GLU Reference Pages [online]. Silicon Graphics, 1991-2006 [cit. 2017-11-26]. Dostupné z: https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/**
5. **KESSENICH, John., Graham SELLERS a Dave SHREINER. Opengl programming guide: the official guide to learning opengl, version 4.5 with spir-v. 9th edition. Boston, MA: Addison-Wesley, 2016. ISBN 978-0134495491.**

Vedoucí bakalářské práce: **Ing. Erik Král, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **15. prosince 2017**

Termín odevzdání bakalářské práce: **25. května 2018**

Ve Zlíně dne 15. prosince 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 22. 5. 2018

.....
podpis diplomanta

ABSTRAKT

Cílem práce je vytvoření vzorové aplikace demonstrující vizualizaci pracovního prostoru obráběcího stroje s využitím knihovny OpenGL a aplikační platformy Qt. V teoretické části je popsána knihovna OpenGL, aplikační platforma Qt, souborový systém STL a metody související s popisem, analýzou a zobrazováním trojrozměrných objektů. V praktické části je vytvořena vzorová aplikace demonstrující základní principy. Dále jsou popsány klíčové části aplikace.

Klíčová slova: OpenGL, Qt, C++, počítačová grafika, kolize, programování, STL souborový formát, vizualizace, 3D prostor

ABSTRACT

The purpose of this bachelor thesis is to create a demo application that demonstrates visualization of machine work space using OpenGL library and platform application framework Qt. In the theoretical part of this work are described the OpenGL library, framework Qt, STL file system and methods related to describing, analyzing and displaying three-dimensional objects. In the practical part is created the demo application that demonstrates basic principles. The key parts of the application are also described.

Keywords: OpenGL, Qt, C++, computer graphic, collision, programming, STL file format, visualization, 3D space

Úvodem bych rád poděkoval zejména vedoucímu bakalářské práce panu Ing. et Ing. Eriku Králi, Ph.D., za odborné rady, cenné připomínky a čas strávený nad prací a konzultacemi.

Dále bych chtěl poděkovat konzultantovi panu Ing. Petru Lukaštíkovi, Ph.D. za odborné konzultace a připomínky. Děkuji také firmě TAJMAC-ZPS, a.s. za umožnění stáže a vedoucímu mé stáže Ing. Zdeňku Slovákovi.

Také bych chtěl poděkovat mé rodině, za podporu v průběhu studia a při psaní bakalářské práce.

„Todo lo que puedas imaginar es real.“

Pablo Picasso

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Tato bakalářská práce se zabývá návrhem, vytvořením a popisem klíčových částí aplikace, která umožňuje vizualizaci pracovního prostoru obráběcího stroje. Vizualizace bude rozšířena vytvořením nástroje, který pomáhá předcházet vzniku kolizím v reálném cyklu obrábění. Testování kolizí probíhá mezi jednotlivými objekty v pracovním prostoru obráběcího stroje. Aplikace obsahuje principy pro zobrazení a manipulaci nahraných modelů v souborovém formátu STL. Z jednotlivých objektů je možné vytvořit prostorový model obráběcího stroje.
Důvod pro výběr tohoto tématu práce byla několikaletá stáž ve firmě TAJMAC-ZPS, a.s. Práce je rozložena do dvou hlavních částí, teoretické části a praktické části.

Teoretická část této práce přiblíží čtenářovi technologie, které byly využity pro tvorbu výsledné aplikace. První kapitola popíše knihovnu OpenGL a také základní informace o vykreslování a budou popsány praktiky, které umožní jeho zrychlení. Další kapitola seznámí čtenáře s aplikační platformou Qt, jeho nástroji a technologií signálů a slotů. Kapitola dále nabídne stručný přehled možností pro tvorbu grafických uživatelských rozhraní s využitím platformy Qt. V následující části budou rozepsány informace o souborovém formátu STL a jeho struktuře. Poslední kapitola teoretické části se zaměří na popis metod a technik, související se zobrazováním a analýzou trojrozměrných objektů v grafické scéně.

V praktické části práce bude vysvětlena struktura vytvořené aplikace, vysvětlen termín pracovní poloha stroje a popsány jednotlivé části uživatelské rozhraní aplikace. Dále budou popsány důležité metody a klíčové části jako vykreslení modelů, kolizní algoritmus, identifikace modelu ve scéně a výpočetní úpravy nad modelem.

I. TEORETICKÁ ČÁST

1 KNIHOVNA OPENGL

1.1 Stručná historie

Knihovna OpenGL (Open Graphics Library) byla vytvořena společností Silicon Graphics Inc. v roce 1992. Jedná se o multiplatformní aplikační programové rozhraní (Application Programming Interface – API) pro vykreslování 2D a 3D grafiky. Knihovna je napsána v programovacím jazyce C. Za přímého předchůdce se dá určit knihovna IRIS GL, která však pracovala pouze s konkrétními typy grafických karet. U OpenGL byl od začátku vývoje kladen důraz na rozšířenou použitelnost u grafických karet. [1]

1.2 Základní informace

V dnešní době spravuje specifikace OpenGL průmyslové konsorcium Khronos Group, členy jsou společnosti zabývající se návrhem operačních systémů a výrobou grafických karet. Knihovna OpenGL není závislá na operačním systému nebo hardwarovém vybavením. OpenGL se soustředí pouze na vykreslování grafiky. Neřeší správu a manipulaci s okny grafického uživatelského prostředí, vstupy od uživatele nebo audio. Knihovna popisuje sadu procedur a funkcí (cca 250) pro vykreslování grafiky. [2]

Architektura OpenGL je založená na typu klient-server. V síťovém prostředí počítač, na kterém běží program vydávající příkazy pro kreslení se nazývá klient a počítač, který přijímá tyto příkazy a zprostředkovává vykreslování se nazývá server. Pokud OpenGL program běží pouze na jednom počítači, klientem se označuje běžící program a grafický akcelerátor interpretující vykreslování se označuje jako server. [2]

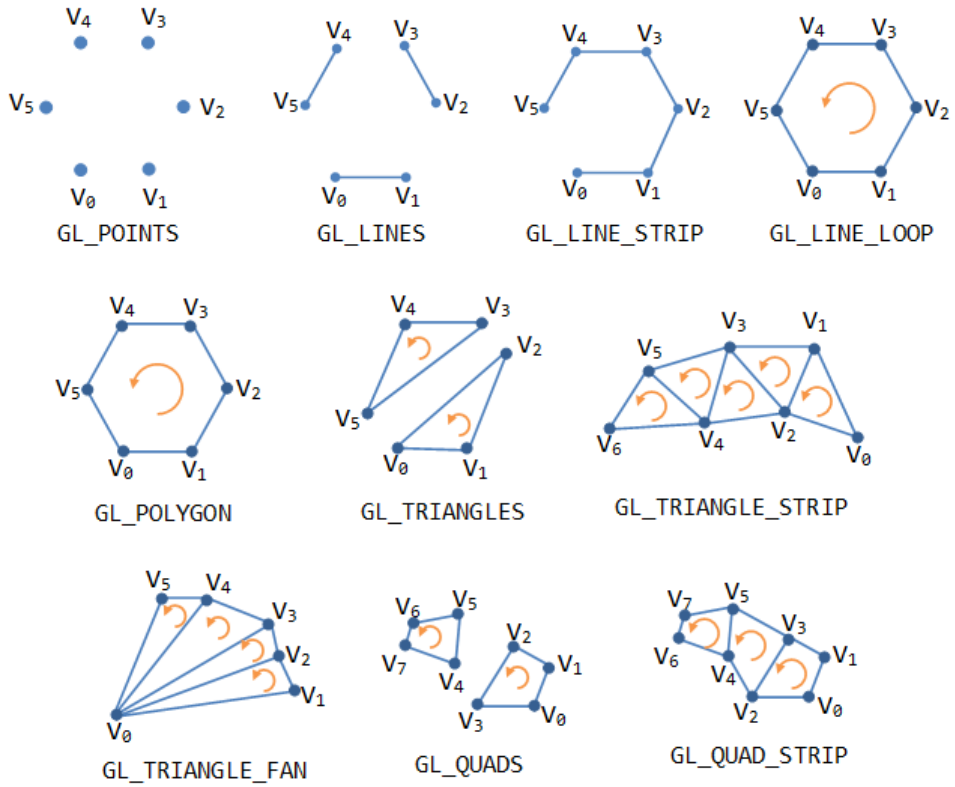
Příkazová syntaxe – OpenGL využívá u svých funkcí předponu „gl“, následuje počáteční velké písmeno názvu funkce (například *glClearColor*). Obdobně konstanty, které definuje OpenGL, začínají předponou „GL_“, v tomto případě je název psaný velkými písmeny.

Mnohé funkce také obsahují příponu, která udává přijímaný datový typ v argumentu funkce. Například funkce *glVertex3f*, písmeno „f“ na konci značí, že parametry funkce jsou ve tvaru desetinného čísla s pohyblivou čárkou. Písmeno přípony „i“ indikuje celočíselný formát. Jako vstupní parametr některé funkce umožňují zadat ukazatel na vektor nebo pole hodnot.

Některé funkce umožnují přijímat jako parametr ukazatel na vektor nebo pole, kde jsou uložena potřebná data, funkce poté obsahuje příponu „v“. [2]

1.3 Informace o vykreslování

Funkce *glBegin* s patřičným parametrem informujeme OpenGL, že chceme zahájit vykreslování. Akceptované parametry jsou zobrazeny na obrázku č. 1. OpenGL po zavolání funkce *glBegin* očekává definici vrcholů. Definice je provedena voláním příkazu *glVertex** s příslušnými souřadnicemi. Ukončení vykreslování zajistí funkce *glEnd*. [2]



Obrázek 1: OpenGL geometrická primitiva [3]

OpenGL využívá transformační matice o rozměru 4x4. Matice *GL_PROJECTION* definuje zobrazovaný prostor, podle stylu použitého promítání. Matice *GL_MODELVIEW* uchovává hodnoty rotací, translací nebo změny měřítka vykreslených objektů. Operace změny polohy a natočení kamery také využívají matici *ModelView*. Zavoláním funkce *glMatrixMode* s názvem matice v argumentu aktivujeme její využívání. Funkce *glLoadIdentity* transformuje právě využívanou matici do tvaru jednotkové matice. [2]

Pokud vyžadujeme provést transformaci pouze nad konkrétním modelem, uložíme aktuální tvar matice funkcí *glPushMatrix* do zásobníku. Matici nastavíme na jednotkovou, provedeme úpravu a zavoláním funkce *glPopMatrix* navrátíme původní tvar matice. [2]

1.4 Zrychlení vykreslování grafické scény

Každé volání funkce, která zajišťuje nebo specifikuje vykreslování objektu poměrně značně zatěžuje procesor. Zatížení zejména způsobuje přesun dat mezi registry procesoru, operační pamětí a zásobníkem. Proto je vhodné snížit počet těchto volaných funkcí na minimum, níže jsou uvedeny příklady možností. [4]

OpenGL jako stavový automat
Nastavením libovolné vlastnosti vykreslování, zůstane konkrétní vlastnost platná do doby, než ji přenastavíme na jinou hodnotu. Například při vykreslování jednobarevného objektu nemusíme opakovaně definovat barvu pro jednotlivé trojúhelníky. Hodnotu barvy definujeme pouze jednou před voláním funkce *glBegin*. [2]

Grafická primitiva redukující počet vrcholů

Trs trojúhelníků (*GL_TRIANGLE_FAN*), pás trojúhelníků (*GL_TRIANGLE_STRIP*) a pás čtyřúhelníků (*GL_QUADS_STRIP*) redukují počet vrcholů potřebných k vykreslení objektu. Společné vrcholy sousedících ploch jsou přenášeny pouze jednou. [4]

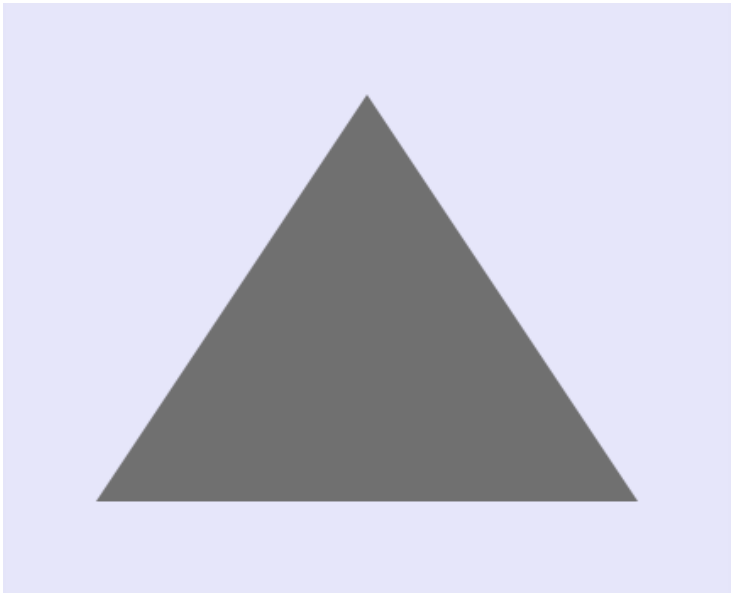
Display list
Pokud opakovaně v grafické scéně vytváříme a vykreslujeme totožné objekty, je vhodné využít techniku display listu. Příkazy popisující vlastnosti a vytvoření objektu zapíšeme pouze jednou při vytvoření listu. Objekt vykreslíme zavoláním daného listu. Data jsou uložena přímo v paměti grafického akcelerátoru, nepřenáší se tedy s každým vykreslením po sběrnici. Změna vlastností nebo změna pozice vrcholů objektu, však vyžaduje vytvoření nového listu. [2]

Pole vrcholů (vertex arrays)
Data popisující vrcholy objektu jsou uložena v poli na straně klienta. Při vyžadovaném vykreslení specifikujeme ukazatele na adresu výskytu prvního vrcholu a normály. Prvotní přístup k datům zajistí přenesení dat do paměti grafického akcelerátoru, tedy na stranu serveru. [4]

Vykreslování objektů voláním funkcí *glBegin* a *glEnd* je označováno jako zastaralé a bude zřejmě v následujících verzí OpenGL nepodporované. K vykreslování objektů je tedy doporučeno využít Display listy a pole vrcholů. [5]

1.5 Ukázka použití

Následující kapitola znázorňuje postup vykreslení triviálního trojúhelníku na obrazovku.



Obrázek 2: Vykreslení jednoduchého trojúhelníku

```
glClearColor(0.902f, 0.902f, 0.98f);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.412f, 0.412f, 0.412f);          // barva trojuhleniku
glBegin(GL_TRIANGLES);
      glVertex3f(-50, 0, 0);                // levý dolní vrchol
      glVertex3f(50, 0, 0);                 // pravý dolní vrchol
      glVertex3f(0, 50, 0);                 // horní vrchol
glEnd();
```

Funkce *glClearColor* definuje barvu pozadí scény, která bude nastavena po vyčistění obrazovky. Vymazání umožní zavolání funkce *glClear.* Barva plochy trojúhelníku je specifikována funkcí *glColor3f*, ve formátu RGB s rozsahem hodnot 0 až 1. Pro možnost specifikování RGB barvy hodnotami v rozsahu 0 až 255 (8 bitů) zvolíme funkci *glColor3ub*. OpenGL po zvolání funkce *glBegin* s parametrem *GL_TRIANGLES* očekává zadaní trojúhelníku po 3 vrcholech. Výsledné vykreslení trojúhelníku znázorňuje obrázek č. 2. [2]

2 APLIKAČNÍ PLATFORMA QT

2.1 Stručná historie

V roce 1990 Haavard Nord a Eirik Chambe-Eng přišli s první myšlenkou vytvoření multiplatformního grafického uživatelského prostředí. První veřejné vydání v roce 1995 se zdrojovými kódy a pod licencí Qt Free Edition License společností Troll Tech.

Při vzniku bylo cíleno na snadnou tvorbu jednotného multiplatformního grafického rozhraní. V dnešní době však představuje vysoce komplexní softwarovou sadu nejrůznějších modulů k vytváření softwarového obsahu.

V roce 2008 firma Nokia zakoupila od společnosti Trolltech Qt toolkit. Na začátku roku 2011 Nokia prodala práva na provoz podpůrných služeb a prodej licencí QT pro komerční účely firmě Digia. Od roku 2014 do současnosti vlastní práva společnost The Qt Company, dceřiná společnost firmy Digia. [6]

2.2 Obecné informace

Qt je multiplatformní aplikační platforma pro usnadnění vývoje aplikačního softwaru a grafického uživatelského prostředí. Mezi podporované počítačové platformy patří například Windows, MacOS, Linux a mobilní platformy iOS a Android. Platforma je napsaná v programovacím jazyce C++. Celosvětové použití dokazuje vysokou oblibu platformy Qt. Mezi velké firmy používající technologii Qt patří AMD, Intel, Samsung, Autodesk, Siemens, Panasonic, LG a další. [7]

Aktuální verze Qt 5.1 rozšiřuje integraci grafické knihovny Vulkan (nástupce knihovny OpenGL) pro operační systémy Windows, Linux a Android. Během této práce byla využita verze 4.8 pro vytvoření aplikace. [7]

2.2.1 Meta-objektový systém MOC

Jádro aplikační platformy obsahuje vlastní preprocesor Meta-Object Compiler (MOC), který před vlastní kompilací upravuje zdrojové kódy, obsahující rozšiřující zápis Qt a generuje z nich standartní kompilovatelné zdrojové kódy jazyka C++. [8]

Pro vlastní kompilaci může být poté použit libovolný standartní kompilátor jazyka C++ jako jsou GCC, MinGW, Clang a další. [7]

2.2.2 Podporované programovací jazyky

Qt je typicky využíváno v aplikacích, které jsou psané v programovacím jazyce C++. Flexibilita a modularita ovšem zajišťuje i použití jiných programovacích jazyků díky jazykovým vazbám (language bindings). [9]

Část podporovaných jazyků:

- Java (QtJambi)
- C#/Mono/.Net (QtSharp)
- Python (PyQt)

2.3 Licencování

Platforma je dostupná pod rozmanitou várkou licencí. The Qt Company prodává komerční licence, které se v současné době dělí na dvě odnože pro aplikační vývoj a vývoj aplikací pro vestavěné systémy tzv. Embedded Systems. K dispozici je také bezplatná varianta pod licencemi GPL (verze 2 a 3) a LGPL verze 3. [10]

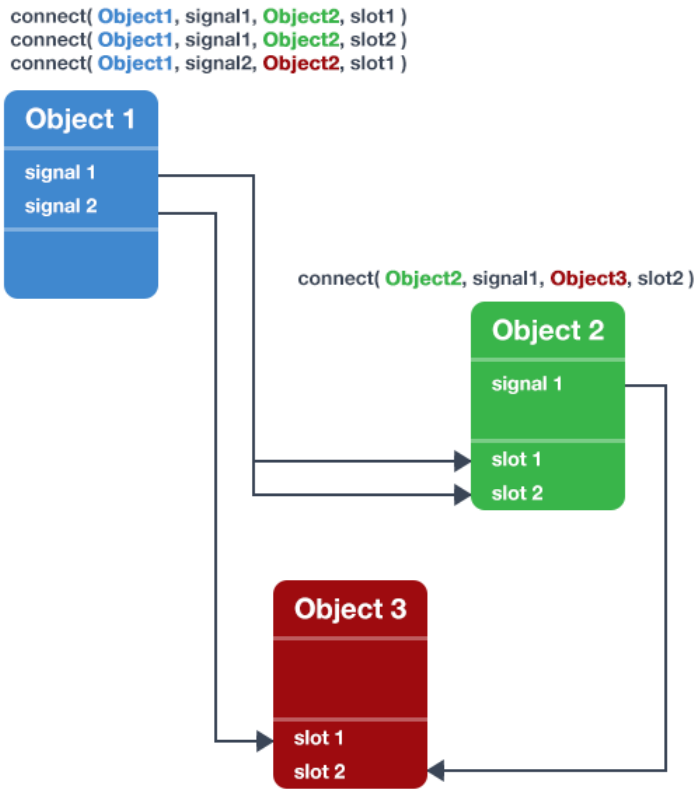
Při použití komerční verze lze aplikační platformu linkovat staticky, oproti opensource verzi, kde musí být knihovna linkována k projektu dynamicky. Některé moduly mohou mít odlišnou dostupnost pod opensource licencí. [11]

2.4 Signály a sloty

MOC umožnuje rozšíření programovacího jazyka C++ o další velmi užitečné funkce, jako jsou například Signály a Sloty, poskytující možnost komunikace mezi objekty. Například pokud uživatel klikne na tlačítko zavřít, vyžadujeme, aby bylo okno upozorněno a zavolalo funkci na zavření. Toto rozšíření slouží pro zachycení událostí a na jejich základě provedení požadovaných operací. Pro použití vlastních signálů a slotů ve třídě, musí být tato třída přímo či nepřímo odvozená od třídy QObject. Také je nezbytné uvést klíčové slovo *Q_OBJECT* na začátku její deklarace. [12]

Signály – jsou vyvolány objektem, když se změní jeho vnitřní stav nebo nastane konkrétní situace. Také můžeme signály vyvolat manuálně pomocí klíčového slova *emit*. Příkladem signálu je například kliknutí myší na tlačítko nebo pohyb kurzoru myši. Vlastní signály musíme definovat pod návěštím *public slots*. [12]

Sloty – jsou metody, které jsou volány v reakci na určitý signál. Sloty můžeme v programu normálně volat napřímo, jako klasické metody. Ve verzi Qt 4.8 musí být deklarovány pod návěštím *signals*, od vyšší verze 5 lze signál připojit také ke klasické metodě. [12]



Obrázek 3: Ilustrace signálů a slotů [12]

Obrázek č. 3 znázorňuje vzájemné propojení mezi objekty. Spojení signálů a slotů zajišťuje statická metoda *connect*. Obsahuje vždy čtyři parametry:

1. Ukazatel na objekt, který vyvolá očekávaný signál.
2. Název vlastního signálu a případně typy jeho parametrů. Uzavřeno v konstrukci *SIGNAL()*.
3. Ukazatel na objekt, který vlastní slot reagující na signál
4. Název vlastního slotu a případně typy jeho parametrů. Uzavřeno v konstrukci *SLOT()*.

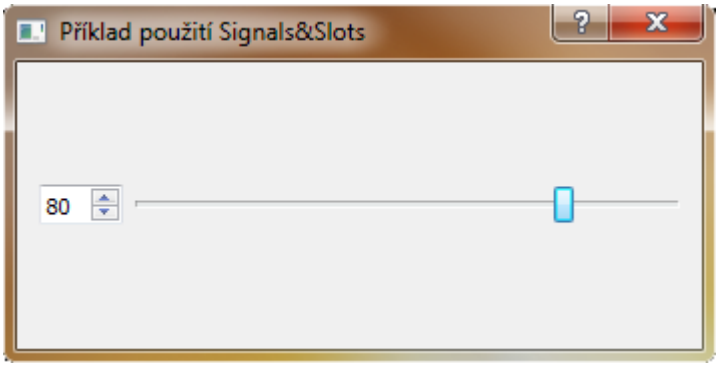
Signály a sloty mohou mít libovolný počet parametrů jakéhokoliv typu. Pro vzájemné propojení však musí mít souhlasné parametry. Qt komponenty vlastní spoustu předdefinovaných signálů a také slotů. V běžné praxi však definujeme vlastní signály a sloty, dle vlastních potřeb.

- Jeden signál může být propojen s více sloty
- Více signálů může být propojeno se stejným slotem
- Signál může být propojen s ostatním signálem
- Propojení může být odstraněno

Pokud je na jeden signál připojeno více slotů, sloty jsou zavolány postupně jeden po druhém v takovém pořadí, v jakém byly spojeny funkcí *connect*. [12]

2.4.1 Příklad použití

Následující triviální příklad ilustruje využití signálů a slotů pro vzájemné předávání celočíselné hodnoty mezi dvěma objekty. Příklad obsahuje dva objekty, číselník (spinbox) a posuvník (slider). Jestliže změníme hodnotu na jednom objektu, nastaví se daná hodnota také na druhém objektu.



Obrázek 4: Ukázka funkce Signály a Sloty

Zdrojový kód příkladu:

```
QSpinBox* spinBox = new QSpinBox();
QSlider* slider = new QSlider(Qt::Horizontal);
spinBox->setRange(0,100);
slider->setRange(0,100);

connect(spinBox, SIGNAL(valueChanged(int)), slider, SLOT(setValue(int)));
connect(slider, SIGNAL(valueChanged(int)), spinBox, SLOT(setValue(int)));

slider->setValue(80);
```

V prvním části si vytvoříme ukazatele na nové objekty a nastavíme povolený rozsah hodnot. Pokud se změní vnitřní stav objektu je vyvolán signál *valueChanged*, který obsahuje údaj o nové hodnotě. Slot *setValue* nastaví vnitřní hodnotu objektu podle zadané hodnoty. Vytvoříme vzájemné propojení objektů pomocí funkce *connect* a nastavíme výchozí hodnotu. Výsledek zdrojového kódu reprezentuje obrázek č. 4.

V příkladu si můžeme také povšimnout, že během předávání hodnot u propojení uvádíme pouze datový typ parametru beze jména.

2.5 Další obsah platformy

Qt dále poskytuje širokou škálu rozšiřujících knihoven tříd, které můžeme využít pro usnadnění vývoje jednoduchých, komplexních, ale i profesionálních aplikací. Zahrnuta je například podpora pro grafiku, webové technologie, sítě, databáze, multimédia, komunikační protokoly jako jsou NFC a Bluetooth. Podporované technologické standarty XML, SVG, JSON a mnoho dalších, které lze snadno integrovat do aplikace. S neustálým rozšiřováním knihovny tříd a modulů o nové užitečné technologie, funkce a třídy roste uplatnění a využitelnost této aplikační platformy. [7] [9]

2.5.1 Nástroje

Platforma Qt také nabízí vlastní multiplatformní vývojové nástroje pro zvýšení produktivity a urychlení programování s touto technologií. Součástí je také nástroj pro zobrazení dokumentace, nápovědy a ukázka využití konkrétní technologie v reálné aplikaci. [7]

Visual Studio Add-in – Pro vývojáře pracující na operačním systému Windows je k dispozici integrující rozšíření do vývojového prostředí Microsoft Visual Studio verze 2013 a vyšší. [9] [13]

Qt Creator – Integrované vývojové prostředí vytvořené na míru pro vývojáře pracující s platformou Qt. Toto prostředí disponuje všemi potřebnými požadavky na kompletní vývoj aplikací. Mezi podporující operační systémy patří Windows, Linux i macOS. [9] [14]

Qt Designer – Nástroj pro rychlý návrh a tvorbu grafických uživatelských prostředí, pomocí přetahování jednotlivých grafických komponentů Qt do pracovního prostoru. Komponentům lze poté nastavit vlastnosti a vzájemné propojení signálů a slotů. [7]

Qt Linquist – Poskytuje způsob zajištění lokalizace, multijazyčnosti a snadného překladu do více jazyků, výsledné softwarové aplikace. [15]

2.6 Tvorba grafického rozhraní

Následující kapitola nabídne stručný přehled možností vývoje grafických rozhraní pro aplikace. Tento přehled neobsahuje veškeré možnosti, pouze část možností. Aplikační platforma Qt poskytuje více způsobů pro tvorbu grafického uživatelského rozhraní neboli GUI. První možností je využít nástroj *Qt Designer* pro tvorbu a stylování grafického prostředí pomocí skládání a přetahování grafických komponentů Qt. [9]

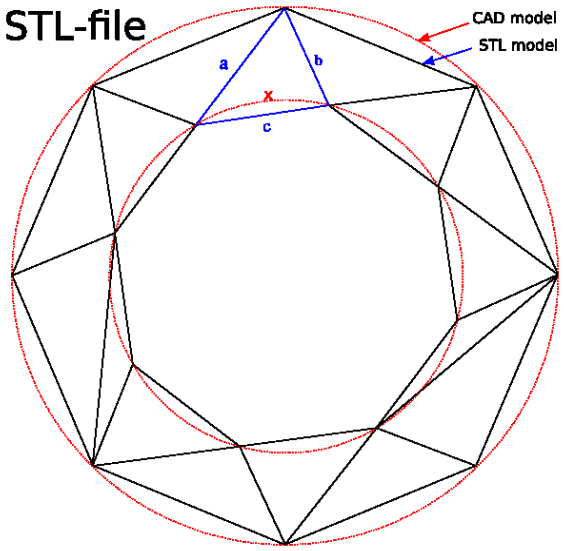
Další možností je využít standartní knihovnu QtQuick pro psaní aplikací v jazyku QML, deklarativní značkovací jazyk, velmi podobný konstrukci CSS a JSON. Programování se provádí v kombinaci s JavaScriptem. Výkonově náročnější operace lze psát v jazyku C++ a následně propojit. [9] [16]

Veškeré grafické komponenty jsou reprezentovány vlastní třídou. Využitím těchto tříd lze grafické prostředí vytvořit pouze vlastně psaným zdrojovým kódem. Tento způsob byl také zvolen pro vytvoření grafického prostředí demonstrující aplikace.

3 SOUBOROVÝ FORMÁT STL

3.1 Základní informace

STL soubor popisuje povrch trojrozměrného objektu za pomocí trojúhelníků, viz obrázek č. 5. Velmi využívaný v oblasti průmyslu a jako tiskový podklad pro 3D tiskárny. Vyvinuto firmou Albert Consulting Group pro společnost 3D Systems v roce 1987. Původní specifikace formátu vyžadovala vyjádření všech pozičních hodnot jako kladná čísla, v dnešní době již není vyžadováno a v souboru mohou být uloženy i negativní souřadnice. Existují dva druhy formátu, rozlišující se svou datovou reprezentací. Jednotlivé datové formáty jsou popsány níže v samostatných kapitolách. [17]



Obrázek 5: Ilustrace STL formátu [18]

3.1.1 Struktura

Jednotlivé trojúhelníky jsou popsány normálou a třemi vertexy v trojrozměrném kartézském prostoru. Každý vertex je reprezentován třemi souřadnicemi, číselné hodnoty souřadnic jsou vyjádřeny datovým typem Float (desetinné číslo s plovoucí čárkou). Trojúhelníky popisující povrch objektu nejsou žádným způsobem řazeny, výsledný popis je tedy uložen nestrukturovaně. Soubor STL neobsahuje žádné podrobnosti o použitých jednotkách nebo měřítku. Dále také neobsahuje informace o barvě objektu či trojúhelníku. [19]

3.1.2 Normála

V obou dostupných datových verzí by měla být normála jednotkový vektor směřující od plochy trojúhelníku směrem ven od povrchu objektu. Velké množství programů pracující s daným formátem při exportování objektu ponechává nulovou normálu, teda souřadnice (0,0,0). Software poté při importování objektu automaticky vypočítá správné normály pomocí pravidla pravé ruky. Normály jsou využívány například při vykreslování odlesků a stínů. [19]

3.2 ASCII formát

Souborový formát začíná řádkem s klíčovým slovem solid a parametrem name, který je reprezentován jako libovolný řetěz znaků. Obvykle využíván pro pojmenování objektu, který je popsán daným textovým dokumentem.

```
solid name
facet normal 0.0 0.0 1.0

outer loop
vertex  1.0  1.0  0.0
vertex -1.0  1.0  0.0
vertex  0.0 -1.0  0.0
endloop
   endfacet
endsolid name
```

Struktura souboru již dále pokračuje zápisem hodnot pro jednotlivé souřadnice trojúhelníků. Poslední řádek musí obsahovat ukončovací klíčové slovo endsolid name. Kde name je párový řetězec zadán při začátku struktury u příkazu solid. [19]

3.3 Binární formát

Binární formát je razantně úspornější než textový formát ASCII, jelikož neobsahuje popisující klíčové slova a nadbytečné mezery a prázdné znaky.

```
UINT8[80] – Hlavička
UINT32 – Počet trojúhelníků

Pro každý trojúhelník

REAL32[3] – Normálový vektor
REAL32[3] – Vrchol č.1
REAL32[3] – Vrchol č.2
REAL32[3] – Vrchol č.3

UINT16 – Kontrolní atribut
end
```

Začátek souboru je tvořen popisující hlavičkou o velikosti 80 znaků, která popisuje název objektu a základní informace. Hlavnička v žádném případě nesmí začínat slovem solid, jelikož by většina programů poté soubor při importaci požadovala jako datový formát ASCII. Po hlavičce následuje 32 bitový bezznaménkový integer, jehož hodnota odpovídá počtu trojúhelníkům v daném souboru.

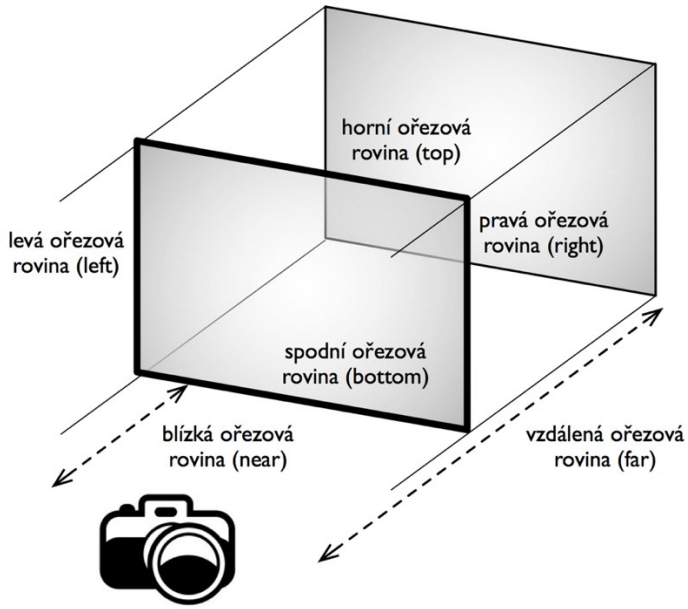
Poté již následuje popis dílčích trojúhelníků. Každý trojúhelník je popsán 3 čísly pro normálu a 12 čísly pro kartézské souřadnice. Poslední číselná hodnota bezznaménkový integer o velikosti 16 bitů je tvořen znakem 0 a jeho význam je kontrolní. [19]

4 SOUVISEJÍCÍ METODY

V této části práce jsou teoreticky popsány technické metody, které jsou využity v praktické části. Aplikace pracující s trojrozměrnou grafikou, využívají nejčastěji perspektivní typ projekce, která definuje prostor scény jako komolý jehlan. Perspektivní projekce deformuje tvary objektů, proto využití této projekce není vhodné pro vytvořenou aplikaci.

4.1 Ortografická projekce

Ortografická projekce se vyznačuje vlastností, že objekty s rostoucí vzdáleností od pozorovací kamery nezkreslují svou velikost. Tento typ projekce využívají aplikace k projektování architektových návrhů nebo aplikace systémů CAD. V těchto aplikací je důležité zachovat skutečné rozměry objektů a úhly promítání mezi nimi. Popsání nastavení ortografické projekce popisuje kapitola 7.1 v praktické části. [20]



Obrázek 6: Znázornění ortografické projekce [21]

Vykreslovací prostor scény je reprezentován jako osově orientovaný pravoúhlý kvádr, vytvořený podle šesti ořezávacích rovin. Vrcholy ležící mimo oblast kvádru nejsou zobrazeny.

Funkce *glOrtho* vytvoří matici pro odpovídající ortografické promítání (zadané parametry). Vytvořenou maticí vynásobí právě používanou matici, proto je důležité před zavoláním funkce využívat projekční matici (*GL_PROJECTION*), nastavenou na jednotkovou.

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,
             GLdouble top, GLdouble near, GLdouble far)
```

Význam jednotlivých parametrů funkce znázorňuje obrázek č. 6 výše. Parametry *near* a *far* určují vzdálenost, nikoliv absolutní pozici v souřadném systému! Mohou nabývat kladných, záporných i nultých hodnot, avšak nesmí obsahovat stejné hodnoty. [2] [21]

4.1.1 Pohledová transformace

Po nastavení ortografického promítání je nezbytné nadefinovat ještě pozici pozorovací kamery neboli pozorovatele, ze které získáme dobrý pohled na scénu. Techniku popisuje tato kapitola. Nastavení pozice a směru pohledu se provádí nad transformační maticí *ModelView* (*GL_MODELVIEW*), kterou při aktivovaní nastavíme na jednotkovou. [20]

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

Funkce *gluLookAt* je vytvořena přesně pro tento účel. Vyžaduje tři sady parametrů, které specifikují umístění pohledu kamery, definují vztažný bod, ke kterému je kamera zaměřena a udávají vektorem, který směr je „nahoru“.

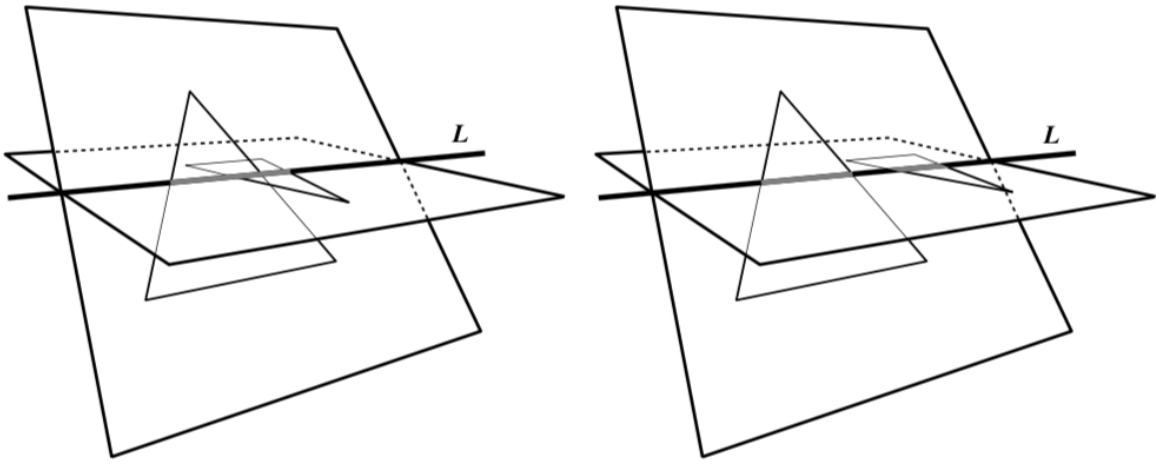
```
void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,
               GLdouble centerx, GLdouble centery, GLdouble centerz,
               GLdouble upx, GLdouble upy, GLdouble upz)
```

Po použití této funkce se většinou zavolají ostatní funkce, které ovlivňují matici *ModelView,* například funkce pro nastavení, velikosti objektu *glScale* nebo rotaci scény *glRotate*.

Lze si povšimnout, že funkce *gluLookAt* má odlišnou předponu než většina OpenGL funkcí. Tato funkce není součástí klasické OpenGL knihovny, ale knihovny Utility. Hlavní důvod tohoto oddělení je, že funkce sama zapouzdřuje řadu klasických OpenGL příkazů, přesněji funkce *glTranslate* a *glRotate*. [2]

4.2 Algoritmus na testování průniku dvou trojúhelníků

Tato kapitola teoreticky popisuje algoritmus, který je součástí detekce kolize mezi dvěma modely. Algoritmus testuje, zda dochází k průnikům mezi dvěma trojúhelníky a jeho využití v aplikaci popisuje kapitola č. 8.2. Využití tohoto algoritmu bylo zvoleno na základě odpovídajících požadavků a vysoké rychlosti výpočtu, je tedy vhodný pro použití v dalších algoritmech na detekci kolize modelů.



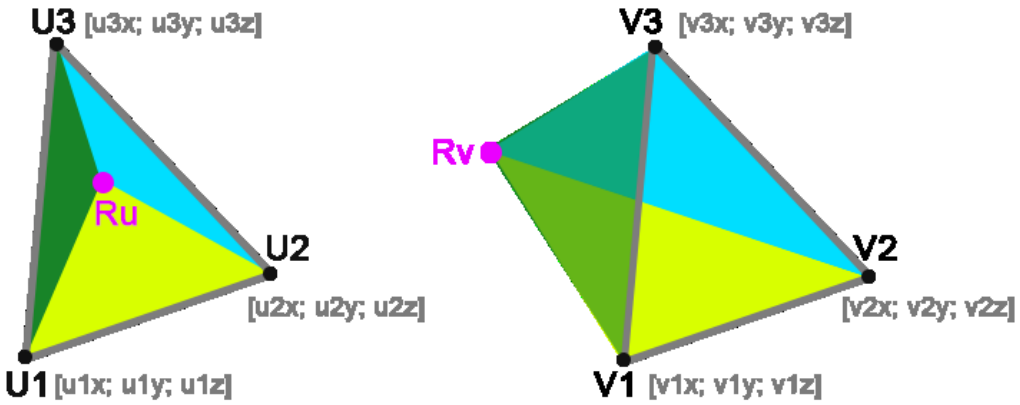
Obrázek 7: Ilustrace trojúhelníků a rovin na kterých leží [22]

Definujme dva trojúhelníky, podle obrázku č. 7, jako trojúhelník č. 1 a trojúhelník č. 2. Prvním krokem je vypočítána rovnice roviny pro trojúhelník č. 2. Kolize není, pokud jsou všechny body trojúhelníku č. 1 na stejné straně vypočítané roviny. Následuje výpočet rovnice roviny, tentokrát pro trojúhelník č. 1. Kolize nenastává, pokud jsou všechny body trojúhelníku č. 2 na stejné straně vypočítané roviny. Vypočteme průsečnici rovin, na kterých leží trojúhelníky. Tuto průsečnici označíme jako přímku *L*. Díky předešlým výpočtům můžeme garantovat, že oba trojúhelníky protnou přímku *L*. Body vzniklé protnutím vytvářejí intervaly na přímce *L*, pokud se vzniklé intervaly protínají, trojúhelníky se také protínají a vzniká mezi nimi kolize. Podrobnější popis techniky, lze nalézt v článku literatury [22].

4.3 Bod v trojúhelníku

Následující kapitola popisuje teoretický a technický princip metody *ListOP::IsPointInTriangle*. Metoda je součástí algoritmu na identifikaci konkrétního modelu pomocí kurzoru myši v grafické scéně programu, viz kapitola č. 8.3 v praktické části této práce.

Metoda kontroluje, zda se referenční bod (v práci reprezentuje pozici kurzoru myši) nachází uvnitř trojúhelníku (v kapitole uváděn jako hlavní) či nikoliv. Referenční bod a trojúhelník se nachází v trojrozměrném kartézském prostoru. Bod a vrcholy trojúhelníku jsou popsány souřadnicemi x, y a z.



Obrázek 8: Ilustrace referenčního bodu uvnitř a mimo trojúhelník

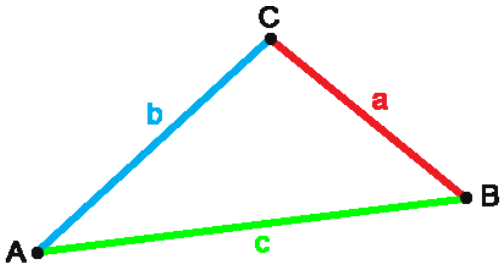
Obrázek č. 8 ilustruje dvě situace. V první variantě se nachází referenční bod uvnitř hlavního trojúhelníku. Druhá varianta zobrazuje opačnou situaci.

Pomocí referenčního bodu a vrcholů hlavního trojúhelníku lze sestrojit 3 nové referenční trojúhelníky. Na obrázku č. 8 jsou referenční trojúhelníky znázorněny zelenou, žlutou a modrou barvou. Referenční trojúhelník je sestrojen dvěma vrcholy hlavního trojúhelníku a referenčním bodem.

Technika výpočtu pracuje na principu porovnání součtu obsahu referenčních trojúhelníků vůči obsahu hlavního trojúhelníku. Pokud je součet obsahu referenčních trojúhelníků menší nebo roven obsahu hlavního trojúhelníku, referenční bod se nachází uvnitř. V opačném případě, bod leží vně trojúhelníku. Popis výpočtu obsahu trojúhelníku je uveden níže.

Výpočet obsahu (plochy) trojúhelníku, zadaný souřadnicemi vrcholů:

Tato strana popisuje techniku výpočtu obsahu trojúhelníku, která je součástí metody pro identifikaci modelu ve scéně.



Obrázek 9: Obecný trojúhelník

$$|AB| = c = \sqrt{(B_x - A_x)^2 + (B_y - A_y)^2 + (B_z - A_z)^2} \quad (1)$$

Rovnice (1) výpočtu vzdálenosti dvou bodů (délka strany trojúhelníku) v trojrozměrném prostoru, pomocí aplikace Pythagorovy věty. Jelikož jsou rozdíly souřadnic bodů ve druhé mocnině, nezáleží na jejich pořadí. Pro výpočet obsahu musíme znát délky všech stran. [23]

$$t = \frac{a + b + c}{2} \quad (2)$$

V rovnici (2) ***t*** značí poloviční obvod trojúhelníku, který je zobrazen na obrázku č. 9.

$$S = \sqrt{t(t - a)(t - b)(t - c)} \quad (3)$$

Pro výpočet obsahu obecného trojúhelníku je využit Heronův vzorec (3). Kde a, b, c značí délky jednotlivých stran trojúhelníku vypočítané ze zadaných souřadnic pomocí rovnice (1). [24]

Tento postup výpočtu obsahu musí být proveden pro všechny 4 trojúhelníky (1 hlavní a 3 referenční), viz předcházející strana a obrázek č. 8.

II. PRAKTICKÁ ČÁST

5 POPIS APLIKACE

Následující část bakalářské práce popisuje strukturu vytvořené aplikace a postupy při jejím vytvářením. Jednotlivé části aplikace jsou popsány pohledem programátora a také uživatele. Dále práce čtenářovi přibližuje klíčové části aplikace. Vzniklá aplikace nese jméno VirtualM, zkratka pro termín Virtual Machine.

Pro vývoj aplikace byl zvolen programovací jazyk C++, jelikož aplikační platforma Qt nativně podporuje tento jazyk. Dále byl zvolen kvůli své vysoké rychlosti a četnému výskytu v návodech a publikacích zabývající se tématikou počítačové grafiky. V neposlední řade také důvod osobní zkušenosti s tímto jazykem. Aplikační platforma Qt byla zvolena z důvodu tvorby grafického rozhraní, usnadnění vývoje s programovacím jazykem C++ a také začlenění knihovny OpenGL do této platformy. Souborový formát STL byl zvolen z důvodu velkého zastoupení v modelovacích programech a jeho stručné struktuře. Aplikace byla vytvořena ve vývojovém prostředí Microsoft Visual Studio s rozšiřujícím modulem Qt.

Vizualizované modely objektů může uživatel barevně odlišit, nastavit rozdílné úrovně průhlednosti nebo vykreslit v drátěném formátu. Dále je možné modely dočasně zneviditelnit a následně opět zobrazit. Pro aplikaci byly také vytvořeny manipulační a transformační metody pracující s modelem ve scéně. Například posunutí, rotace, zrcadlení a změna polohy modelu. Provedené úpravy jsou zaznamenány a uloženy do logovacího souboru. Metody jsou popsány v samostatných kapitolách. Aplikace obsahuje nástroj k vytvoření bitmapového obrázku vykreslovacího okna OpenGL. Akce k záznamu okna se nachází v horním menu pod položkou „Scéna“.

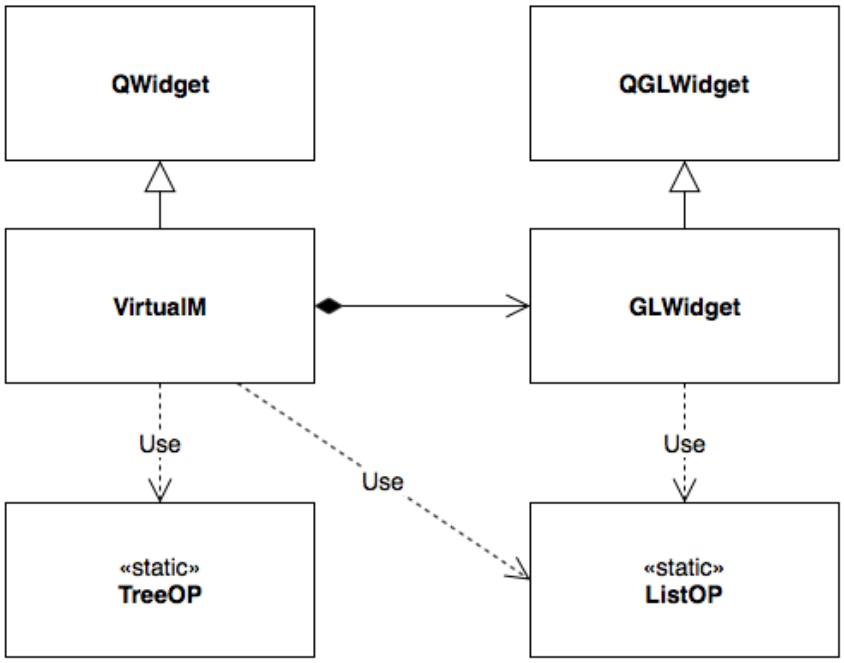
Výslednou scénu složenou z modelů je možné uložit do souboru a následně opětovně načíst. Pro tuto akci byl navržen strukturovaný soubor Virtual Machine Scene, umožňující popis celkové scény.

Jelikož zdrojový kód vytvořené aplikace obsahuje několik desítek tisíc řádků kódu, nejsou v práci uvedeny veškeré zdrojové kódy metod, ale pouze nejdůležitější části. Každá uvedená část zdrojového kódu uvádí, kde přesně se nachází. Pro případ zájmu jejího využití nebo získání více informací.

6 STRUKTURA APLIKACE

Hlavní část celé aplikace zastupuje třída *VirtualM*, která dědí vlastnosti ze třídy *QWidget.* Třída *VirtualM* obstarává inicializaci a obsluhu grafického prostředí aplikace. Inicializaci grafického prostředí umožnuje metoda *createControls*. V neposlední řadě zapouzdřuje třídu *GLWidget*, zajišťující vykreslovaní hlavní a vedlejší grafické scény OpenGL.

Třída *TreeOP* obsahuje statické metody pro usnadnění práce se stromovým listem a jeho položkami. Třída *ListOP* také poskytuje statické metody. Metody však slouží pro práci se strukturou obsahující data a vlastnosti modelu nebo s datovým polem, které ukládá jednotlivé struktury modelů. Příkladem jsou metody pro importovaní, odstraňování, detekci kolizí nebo vlastní rotace, translace a zrcadlení modelu. Strukturu aplikace také znázorňuje obrázek č. 10.



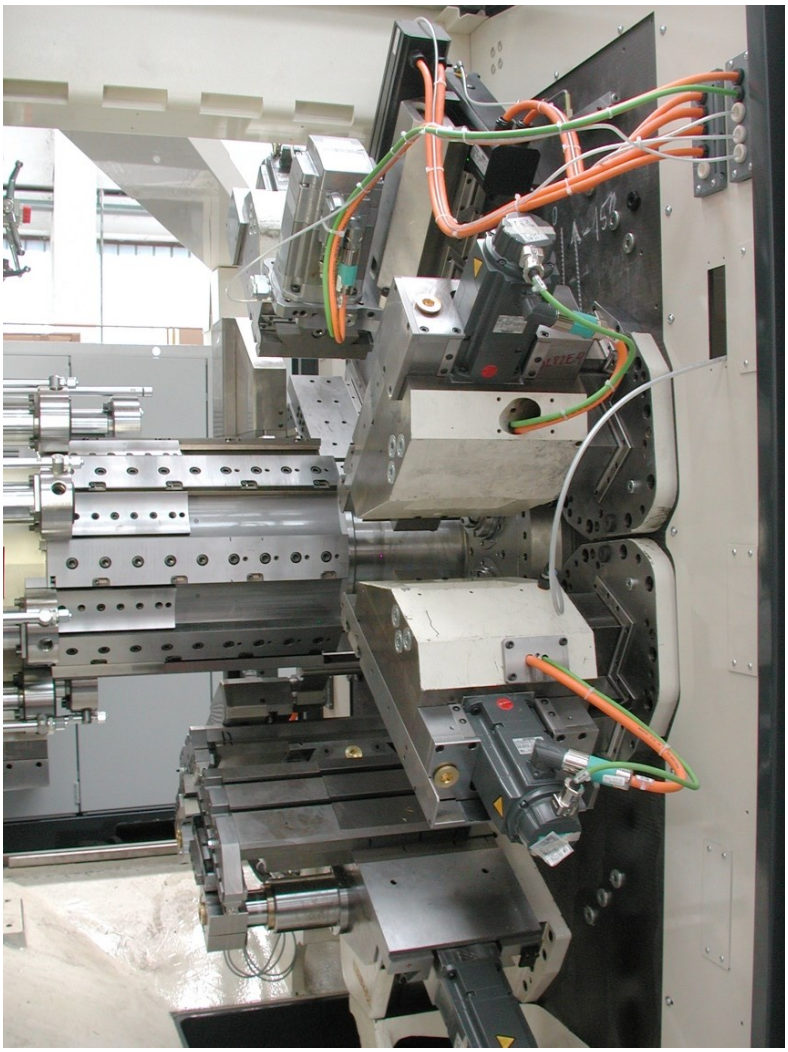
Obrázek 10: UML digram hlavních částí

V práci se objevují výrazy metoda a funkce. Oba termíny mají stejný význam, jako podprogram. Avšak metodou označujeme podprogram v objektově orientovaném jazyce. Jelikož knihovna OpenGL není napsaná v OOP její podprogramy jsou označeny jako funkce. Naopak podprogramy vytvořené aplikace jsou popsány jako metody.

6.1 Pracovní polohy stroje

Na jednotlivých souřadnicích pracovních poloh stroje se nachází supporty nesoucí obráběcí nástroje, zachyceno na obrázku č. 11. V aplikaci jsou pracovní polohy reprezentovány jako uzlové položky ve stromovém listu, s definovanými souřadnicemi počátečního bodu ve scéně. Například v okně stromu označíme konkrétní polohu a nahrajeme STL model nástroje do aplikace. Model je automaticky posunut na souřadnice polohy a ve stromě se vytvoří jeho záznam pod položkou uzlu dané polohy.

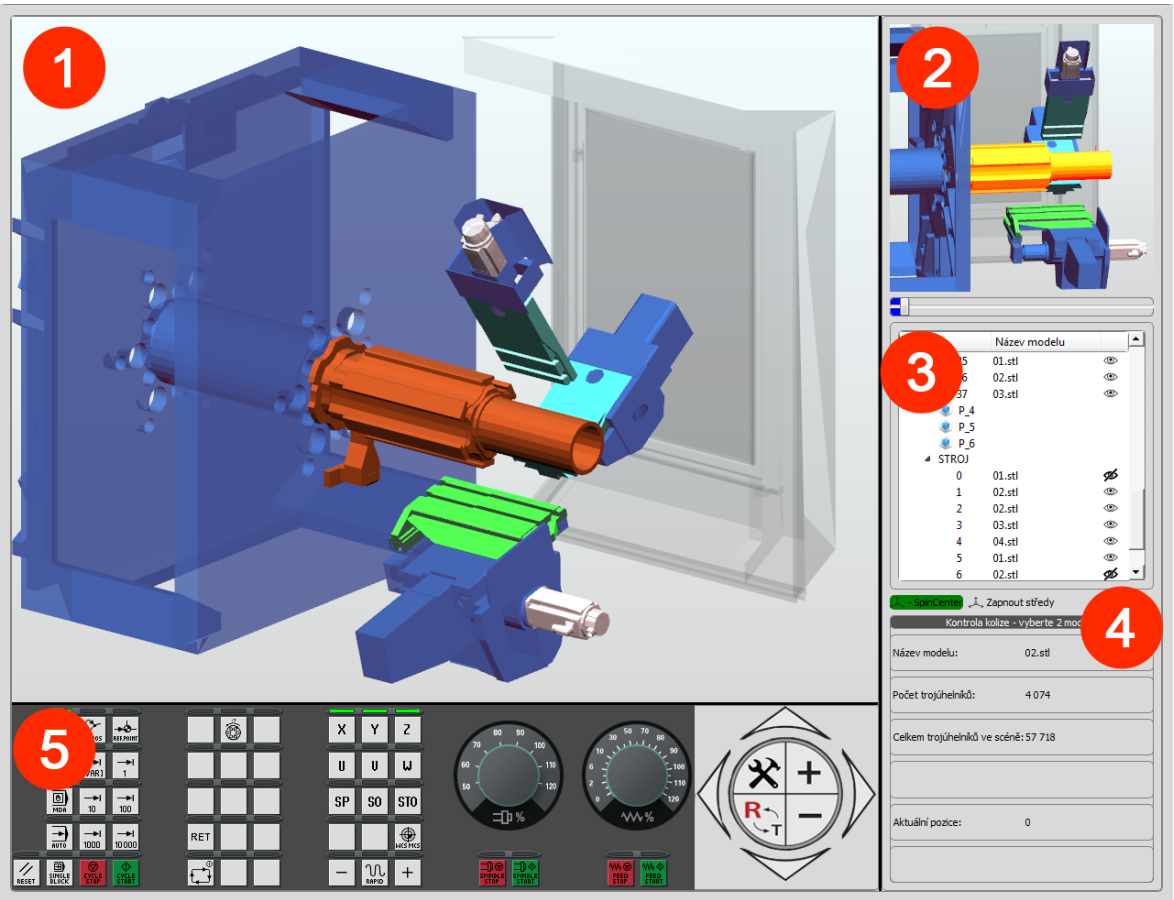
Aplikace také obsahuje speciální polohu s názvem STROJ, která se vyznačuje svými souřadnicemi ve středu scény, tedy [0,0,0]. Na tuto polohu jsou nahrány modely objektů nacházející se mimo pracovní polohy, například kostra stroje.



Obrázek 11: Pracovní prostor reálného stroje TMZ-642

6.2 Uživatelské rozhraní

Grafické uživatelské rozhraní vytvořené aplikace je rozděleno do více částí. Jednotlivé části jsou znázorněny na obrázku č. 12 a popsány v jednotlivých kapitolách níže. Rozhraní obsahuje dva posouvatelné rozdělovače třídy *QSplitter*. Rozdělovače umožní uživatelovi tažením skrýt nebo změnit velikost spodního a bočního panelu.



Obrázek 12: Prostředí vytvořené aplikace

6.2.1 Hlavní zobrazovací část

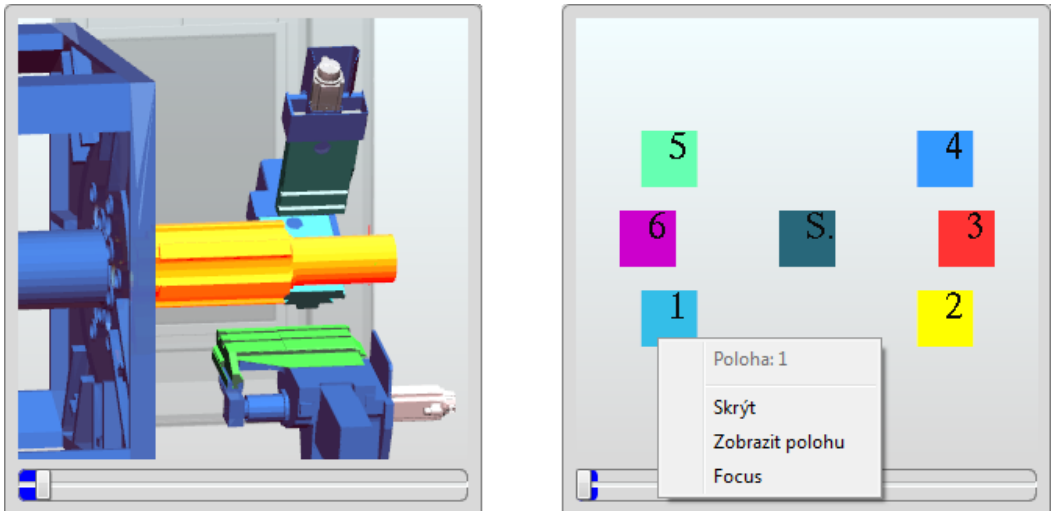
Hlavní část je reprezentována grafickým oknem OpenGL, které vykresluje modely nahrané do aplikace. Uživatelské ovládání a manipulaci ve scéně popisuje samostatná kapitola 8.5.

Klávesovou zkratkou CTRL + N, lze rychle smazat veškeré modely v aplikaci a nastavit scénu na počáteční stav.

6.2.2 Vedlejší zobrazovací část

Vedlejší zobrazovací část slouží k dalšímu zobrazení scény z jiného úhlu. Po přepnutí do zjednodušeného módu, dále slouží jako prostředek k rychlému výběru aktuální polohy.

Manipulace s vedlejším zobrazovacím oknem je obdobná jako s hlavním oknem scény. Vykreslená scéna je složena ze stejných dat. Pod oknem se nachází posuvný jezdec pomocí, kterého lze měnit přiblížení modelů.



Obrázek 13: Vedlejší grafické okno

Přetáhnutím posuvného jezdce do vyznačené krajní polohy na obrázku č. 13 se okno přepne do zjednodušeného módu, který zobrazí pouze polohy stroje. Jednotlivé polohy jsou symbolizovány objektem ve tvaru kostky s číslem reprezentující pořadí dané polohy.

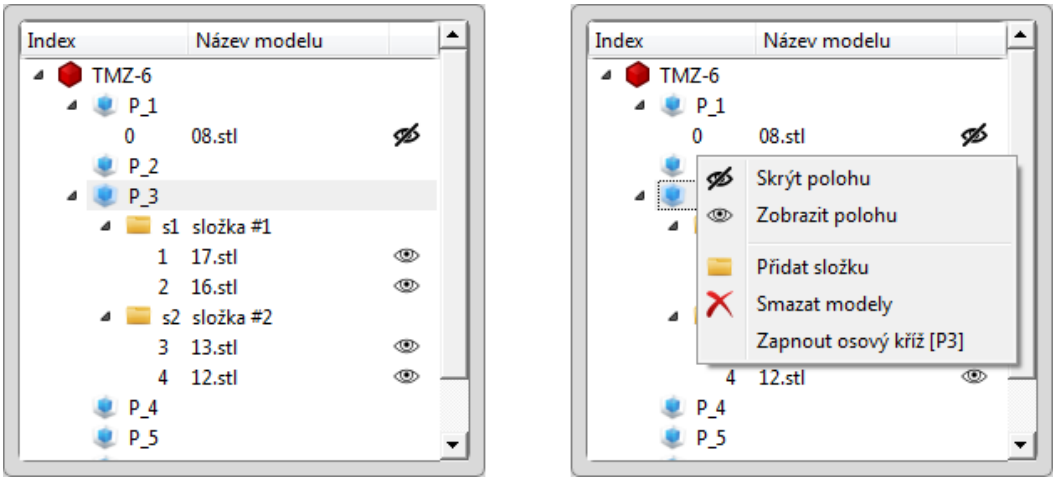
Kliknutím levým tlačítkem myši na konkrétní kostku se nastaví výchozí středový bod hlavní zobrazovací scény na výchozí souřadnice dané polohy. Kliknutím do prázdného prostoru nebo na středovou kostku se výchozí bod resetuje, to znamená nastavení souřadnic (0,0,0).

Po stisknutí pravého tlačítka myši se zobrazí kontextová nabídka. Možnost „Skrýt“ skryje všechny modely, která scéna obsahuje. Akce „Zobrazit polohu“ naopak veškeré modely zobrazí. Poslední položka „Focus“ ponechá ve scéně viditelné pouze modely v dané poloze a nastaví výchozí bod polohy jako střed hlavní zobrazovací scény.

6.2.3 Strom modelů

Stromový list zobrazuje veškeré nahrané modely do aplikace pod vlastní položkou. Položka modelu obsahuje číselný index, pod kterým je struktura, která reprezentuje data modelu uložena v datovém poli aplikace. Dále položka nese informace o pojmenování modelu a stavu viditelnosti v hlavním grafickém okně scény.

Počáteční kořen (uzel nejvyšší úrovně) stromu tvoří položka nesoucí název stroje. Kořen vlastní uzly představující konkrétní pracovní polohy stroje, pojmenované ve formátu „P_X“, kde písmeno X značí číselné označení polohy. Poloha „STROJ“ reprezentuje počáteční body scény [0,0,0].



Obrázek 14: Stromová struktura modelů

Nabídka možností se zobrazí kliknutím pravým tlačítkem myši na položku, viz obrázek č. 14. Stromové zobrazení je vytvořeno pomocí třídy *QTreeWidget* a jednotlivé položky třídou *QTreeWidgetItem*.

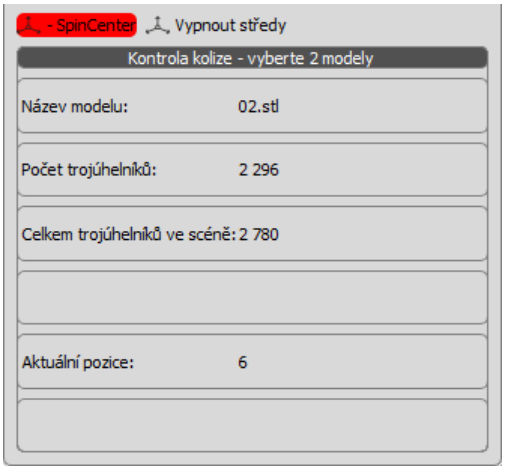
Ve stromě modelů lze vytvořit položku, která zastupuje složku, pro zvýšení přehlednosti a lepší uspořádání ve stromě. Položkám modelům a složek mohou být přiřazena popisující jména. Strom dovoluje smazat, konkrétní model, složku obsahující modely, veškeré modely na dané poloze nebo kompletně celou scénu. Kliknutím na ikonu oka se daný model zobrazí nebo skryje. Zobrazit a skrýt lze také složka modelů nebo celá poloha. Na počátečních souřadnicích poloh je možné vykreslit osový kříž.

Veškeré usnadňující metody pro práci se stromem byly vytvořeny pro tuto aplikaci a nachází se ve třídě *TreeOP*.

6.2.4 Informace o modelu a akční oblast

Oblast pod stromovou strukturou zobrazuje informace o aktuálně vybraném modelu, znázorněno na obrázku č. 15. Zobrazuje se také například celkový počet trojúhelníků, ze kterých se skládají modely ve scéně.

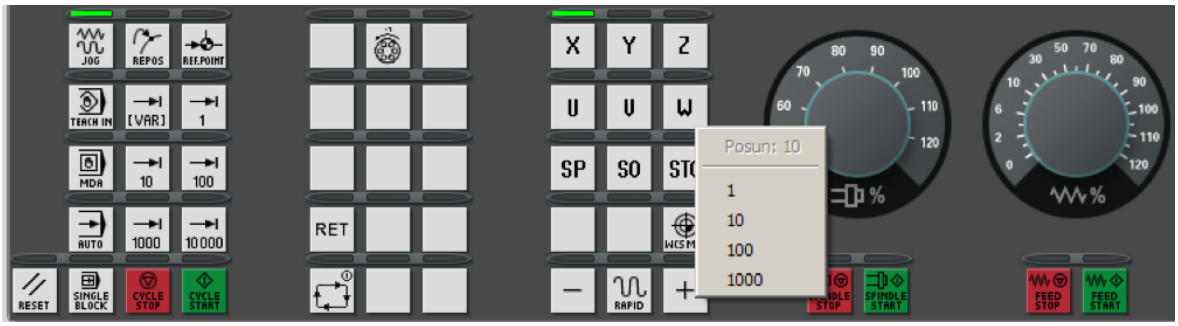
Dále se v oblasti nachází 3 akční tlačítka. První tlačítko *SpinCenter* po zapnutí nastaví výchozí bod pro rotaci scény na souřadnice středu aktuálně vybraného modelu. Druhé tlačítko „Vypnout/Zapnout středy“ zobrazí nebo skryje osové kříže na počátečních souřadnicích poloh ve scéně. Aktivací posledního tlačítka se provede kontrola kolize mezi dvěma modely, viz kapitola 8.2.



Obrázek 15: Informace o modelu

6.2.5 Virtuální panel stroje

Aplikace ve spodní části obsahuje texturu ovládacího panelu řídícího systému Sinumerik od společnosti Siemens. Ovládací panel je zobrazen na obrázku č. 16. Aplikace implementuje možnost posunutí (viz obrázek č. 24) vybraných modelů ve scéně, aktivací tlačítka „JOG“ společně s určením osy pro posun a stiskem tlačítka s popiskem „+“ nebo „-“.



Obrázek 16: Virtuální panel stroje

6.3 Kreslící plátno OpenGL

Pro možnost využití knihovny OpenGL v projektu Qt je nutné přidat daný modul do projektového souboru s koncovkou „.pro“. Přidání je realizováno přidáním řádku s klíčovým slovem „+= opengl“.

V případě využití přídavného modulu pro vývojové prostředí Microsoft Visual Studio, lze moduly přidávat v grafickém okně. V konkrétním souboru, kde budeme využívat technologii musíme také přidat hlavičkové soubory (<QGL> nebo <QGLWidget>) pro zpřístupnění.

6.3.1 Využití třídy QGLWidget

GLWidget je nejdůležitější třída pro vykreslování grafiky v aplikaci. Dědí vlastnosti od třídy QGLWidget, která komplexně zapouzdřuje objekt reprezentující kreslící plátno knihovny OpenGL. Odvozená třída dále přepisuje implementaci vysokoúrovňových metod, zajišťující prvotní nastavení, definice vlastností při změně velikosti a vykreslování grafické scény. [9]

```
[opengl.h]
class GLWidget : public QGLWidget{
      Q_OBJECT
protected:
      void initializeGL();
      void resizeGL(int width, int height);
      void paintGL();
};
```

initializeGL – Metoda pro nastavení prvotních vlastností a inicializaci proměnných kreslícího plátna OpenGL. Zavolána pouze jednou před prvním volání metody *resizeGL* nebo *paintGL*.

resizeGL – Metoda je zavolána v případě změny velikosti vykreslovacího okna.

paintGL – Hlavní část, zodpovědná za vykreslování obsahu. Metoda je opakovaně volaná pro každé jednotlivé překreslení scény.

7 RENDEROVÁNÍ

Před vykreslením prvního snímku obrazovky se zavolá metoda *initializeGL*, která nastaví výchozí vlastnosti vykreslování. Například hladké stínování barvy u modelu nebo specifikaci pozice světla ve scéně.

7.1 Nastavení scény

Nejdůležitější nastavení scény provádí metoda *paintGL* ve třídě *GLWidget*, metoda je zavolána s každým překreslením scény. Dílčí nastavení vykreslování scény probíhá i v ostatních metodách.

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Před vykreslením každého nového snímku scény je nutné smazat obsah původní obrazovky. Funkce *glClear* vymaže obsah bufferu, který byl specifikován v parametru pomocí konstanty. V parametru lze prostřednictvím logického součtu zadat více bufferů najednou. První konstanta ve funkci níže značí framebuffer a druhá hloubkový buffer.

V aplikaci je použito gradientní pozadí scény, pro tento typ pozadí není v OpenGL definovaná konkrétní funkce. Je nutné vykreslit čtyřúhelník s vyžadovanou barvou přes celou plochu pozorovací kamery. Před vykreslením musí být zavolána funkce *glDisable* s parametrem *GL_DEPTH_TEST*, abychom nezaznamenali vykreslený objekt do hloubkového bufferu.

```
glViewport(0, 0, rOkno – lOkno, bOkno - tOkno);
glMatrixMode(GL_PROJECTION);
glLoadIndentity();
glOrtho(lOkno, rOkno, tOkno, bOkno, 8000, -8000);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

Funkce *glViewport* definuje obdélníkovou oblast v okně, na kterou je namapován výsledný snímek. První dva parametry definují pozici levého spodního rohu oblasti v souřadnicích x a y. Následující parametry popisují šířku a výšku oblasti. Projekční matice je obnovena na jednotkovou matici a následně nastavena na konkrétní ortografické promítání. Matice *GL_MODELVIEW* je také nastavena na jednotkovou matici.

```
gluLookAt(aktualniPoloha.x, aktualniPoloha.y, 100, aktualniPoloha.x,
          aktualniPoloha.y, 0, 0, 1, 0);
glTranslatef(aktualniPoloha.x, aktualniPoloha.y, 0);
glScalef(meritko, meritko, 1);
glRotated(rotLx, 1.0, 0.0, 0.0);
glRotated(rotLy, 0.0, 1.0, 0.0);
glTranslatef(-aktualniPoloha.x, -aktualniPoloha.y, 0);
```

Funkcí *gluLookAt* nastavíme pozici pozorovatele neboli pozici kamery. Pozorovací kamera při oddalování a přibližování scény nemění svou pozici, nýbrž vykreslené objekty mění svou velikost. Této situace je docíleno vynásobením aktuálně využívané matice maticí nastaveného měřítka. Rotace a změna měřítka scény probíhá vůči středu aktuálně zvolené poloze. Toho docílíme posunutím na požadované souřadnice středu, provedením operací rotace, změna měřítka a opětovného posunutí na původní souřadnice.

V aplikaci je možné zapnout možnost otáčení scény vůči středu aktuálně vybraného modelu tzv. „SpinCenter“. Nejprve je vypočítán střed modelu v souřadnicích scény pomocí metody *ListOP::minMaxMID*. Vypočítané souřadnice jsou použity namísto souřadnic aktuální polohy ve výše uvedeném zdrojovém kódu.

Použitím funkce *glTranslate* se neposouváme z výchozího bodu scény na nový, ale z původního místa, kam jsme se dostali předchozím použitím této funkce.

7.2 Vykreslování modelů

Zobrazení modelů v grafickém okně OpenGL zajišťuje metoda *drawSTL,* která pomocí smyčky prochází všechny struktury nahraných modelů v datovém poli aplikace. Metoda vyžaduje předání ukazatele na datové pole ve svém parametru.

```
void GLWidget::drawSTL(MODEL_DATA *listOfModels_ZDROJ)
```

Před vlastním vykreslením konkrétního modelu na obrazovku jsou nejprve nastaveny vlastnosti vykreslovaní pro daný model, dle jeho vlastností. Například barva a úroveň průhlednosti.

V OpenGL knihovně se využívá Phongův osvětlovací model pro stínování. Barva modelů je v programu specifikována pomocí 5 složek, aby získala metalický vzhled. Funkcí *glMaterialfv* charakterizujeme hodnoty pro jednotlivé složky:

- ambientní (GL_AMBIENT)
- difuzní (GL_DIFFUSE)
- zrcadlová (GL_SPECULAR)
- emisní (GL_EMISSION)
- lesk (GL_SHININESS)

Aplikace obsahuje paletu 30 barev, které lze použít pro nastavení barvy. V situacích přidání modelu do výběru nebo přejetí kurzorem myši se model vybarví odlišnou barvou pro zvýraznění. [2]

```
if(listOfModels_ZDROJ[i].properties.wirefreme == true)
      glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
else
      glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
```

Výše uvedený blok kód rozhodne, zda bude vykreslený model plně vyplněn nebo bude vykreslen pouze drátěný model.

```
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, &listOfModels_ZDROJ[i].Vertex[0]);
glNormalPointer(GL_FLOAT, 0, &listOfModels_ZDROJ[i].NormalVector[0]);
glDrawArrays(GL_TRIANGLES, 0, listOfModels_ZDROJ[i].pocetVertexu / 3);
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_NORMAL_ARRAY);
```

Aplikace k vykreslení modelů na obrazovku využívá techniky pole vrcholů. Data popisující trojúhelníky jsou uložena ve dvou polích. První pole popisuje souřadnice jednotlivých vrcholů a druhé normálové vektory. V prvním kroku je tedy zapotřebí povolit využívání těchto polí, za pomocí volání funkce *glEnableClientState* s příslušnými parametry identifikující konkrétní pole. Druhým krokem je předání ukazatelů na pole knihovně OpenGL. Pro každý typ pole je vytvořena samostatná funkce.

Funkce *glVertexPointer* v prvním parametru očekává číslo udávající počet souřadnic, které tvoří jeden vrchol. Druhý parametr popisuje datový typ souřadnic. Třetí udává mezeru mezi jednotlivými položkami v bytech. Poslední parametr očekává výsledný ukazatel na začátek pole, které obsahuje souřadnice vrcholů.

První parametr funkce *glNormalPointer* udává datový typ položky v poli. Druhý mezeru v bytech mezi položkami a poslední ukazatel na začátek daného pole. [4]

Vykreslení modelu je poté velmi snadné pouhým jedním zavoláním funkce *glDrawArrays* se vykreslí všechny dílčí trojúhelníky modelu. V prvním parametru informujeme OpenGL, aby vykreslilo trojúhelníky ve formátu *GL_TRIANGLES*. Druhý parametr funkce udává počáteční index vrcholu v poli, třetí specifikuje celkový počet vrcholů k vykreslení.

Po vykreslení modelu vypneme využívané pole zavoláním funkce *glDisableClientState* se stejnými identifikátory v parametru jako při zapnutí, před vykreslováním.

Použitím této techniky je vykreslování zrychleno a je razantně snížen počet volaných funkcí. Oproti technice s funkcemi *glBegin*, *glEnd* a definování normál a vrcholů.

7.2.1 Struktura modelu

Data nahraného modelu jsou v programu reprezentovány vytvořenou strukturou *MODEL_DATA*. Třída *GLWidget* obsahuje datové pole (*listOfModels*), kde jsou uloženy jednotlivé struktury.

```
[opengl.h]
struct MODEL_DATA{
	UINT8			headerBackup[80];
	float			*NormalVector;
	float			*Vertex;
	int			pocetNormal;
	int			pocetVertexu;
	quint16			byte;
	float			modelPosition[3];
	float			modelMatrix[16];		/// hlavni
	MATRIX_TEMP		matrix_temp;			/// dilci matice
	bool			using_structure;
	bool			visible;
	bool			active;
	bool			hover;
	MODEL_PROPERTIES	properties;
	TCHAR			modelName[256];
	TCHAR			alias[ALIAS_LENGHT];

	RV6_VECTOR3		historyOfTransformation[MAX_UNDO_COUNT];
	RV6_VECTOR3		historyOfRotation[MAX_UNDO_COUNT];
	RV6_VECTOR3		total_Transformation;
	RV6_VECTOR3		total_Rotation;
	RV6_VECTOR3		pocatekModelu;

	int			actualPosition;
	int			structParent;
	float			transparency;
};
struct MODEL_PROPERTIES{
	COLOR_STRUCT		ambient;
	COLOR_STRUCT		diffuse;
	COLOR_STRUCT		specular;
	COLOR_STRUCT		emission;
	float			shininess;
	bool			wireframe;
}
```

Struktura *pocatekModelu* obsahuje souřadnice počátečního bodu v prostoru, ke kterém byl model vytvořen. Datový prostor pro souřadnice vrcholů *(*Vertex*) a normál *(*NormalVector*) je alokován dynamicky. Struktura *MODEL_DATA* udržuje informace modelu jako barva, úroveň průhlednosti, aktuální poloha, cesta k originálnímu STL souboru a ostatní.

8 POPIS DŮLEŽITÝCH METOD

V následujících podkapitolách jsou podrobně popsány klíčové části vytvořené aplikace.

8.1 Importování modelu do programu

Ve stromové struktuře programu musíme nejprve označit cílovou požadovanou polohu, kam chceme nahrát model. Označíme jeden nebo více souborů formátu STL a přetáhneme je do okna aplikace (drag and drop). Aby bylo možné přijmout přetažená data, je nutné operaci přetažení aktivovat a implementovat metodu pro zpracování dat.

```
setAcceptDrops(true);
```

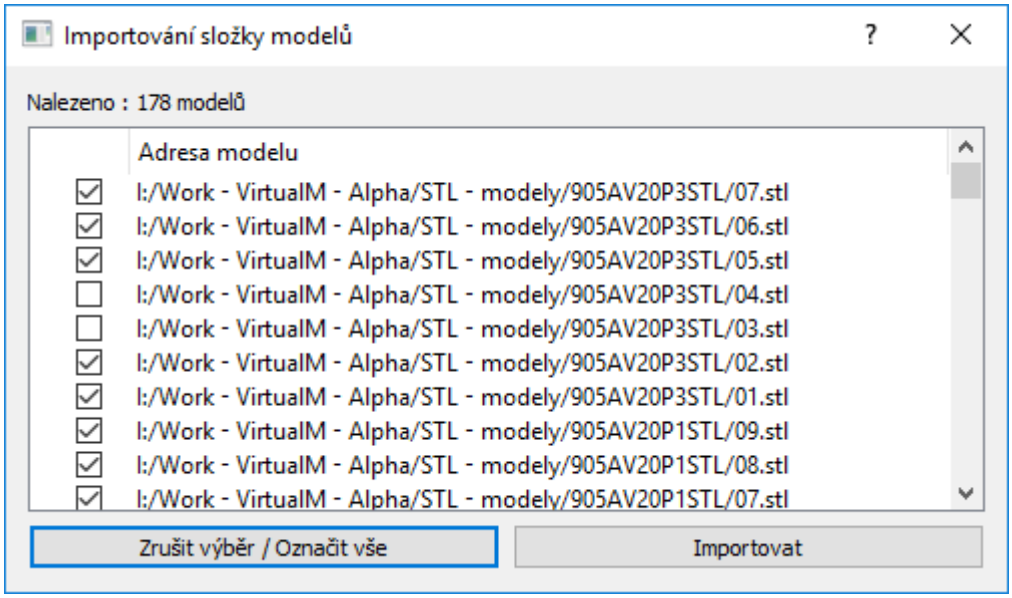
Třída VirtualM obsahuje metodu *dropEvent*, která se zavolá, pokud uživatel operaci přetažení dokončí a uvolní tlačítko myši. Parametr třídy *QDropEvent* obsahuje adresy, k souborům, které byly přetaženy do okna aplikace. [25]

```
void VirtualM::dropEvent(QDropEvent *ev){
      QList<QUrl> urls = ev->mineData()->urls();
```

Program smyčkou *foreach* ověří formát a ostatní podmínky u všech jednotlivých vstupních souborů. Pokud proběhne vše bez problémů, zavolá se metoda *tree_AddChild*, která přidá nový záznam do okna stromové struktury pod označenou polohu. Metoda také importuje data ze souboru do datové struktury aplikace, zavoláním další metody *ListOP::importSTL*.

```
int ListOP::importSTL(MODEL_DATA *list, QString url, QString alias)
```

Metoda otevře vstupní soubor a postupně čte ze souboru bloky o pevné datové velikosti. Viz kapitola 3.3, v teoretické části o struktuře binárního datového formátu souboru STL. Data ze souboru se uloží do struktury *MODEL_DATA*, která reprezentuje model. Pro data reprezentující normály a vrcholy trojúhelníků jsou dynamicky alokovány spojité bloky v paměti počítače. V případě nahrání modelu na jinou polohu než středovou, je nutné provést translaci modelu na výchozí souřadnice nové polohy.

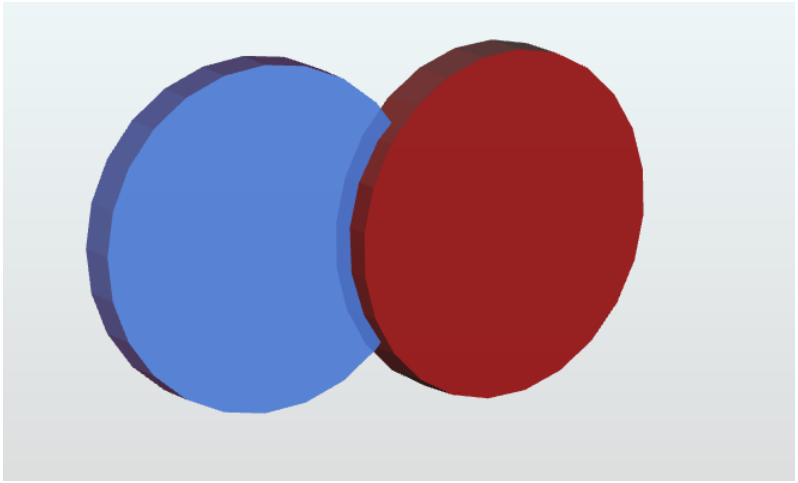


Obrázek 17: Hromadné importování modelů, pomocí složky

Přetažením složky, která obsahuje modely uložené v souborovém formátu STL, lze dosáhnout hromadného importování modelů. Program rekurzivně prohledá celou složku a vyhledá soubory vhodné k importaci. Poté zobrazí okno, viz obrázek č. 17, s možností vybrání specifických souborů, které se importují do aplikace.

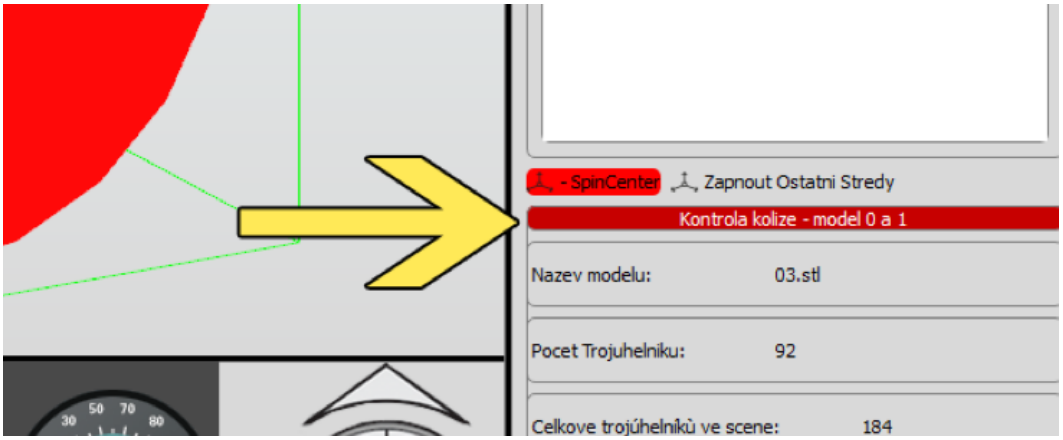
8.2 Kolize modelů

V aplikaci je možné ověřit, zda nevzniká kolize mezi dvěma vybranými modely ve scéně. Obrázek č. 18 zachycuje dva modely ve scéně.



Obrázek 18: Modely připravené na kontrolu kolize

Dvojstiskem klávesy Shift + kliknutím levým tlačítkem myši na vykreslený model, se vybraný model přidá do vícenásobného výběru. Po přidání dvou modelů do výběru se zpřístupní tlačítko „Kontrola kolize“. Tlačítko také zobrazuje indexy označených modelů.

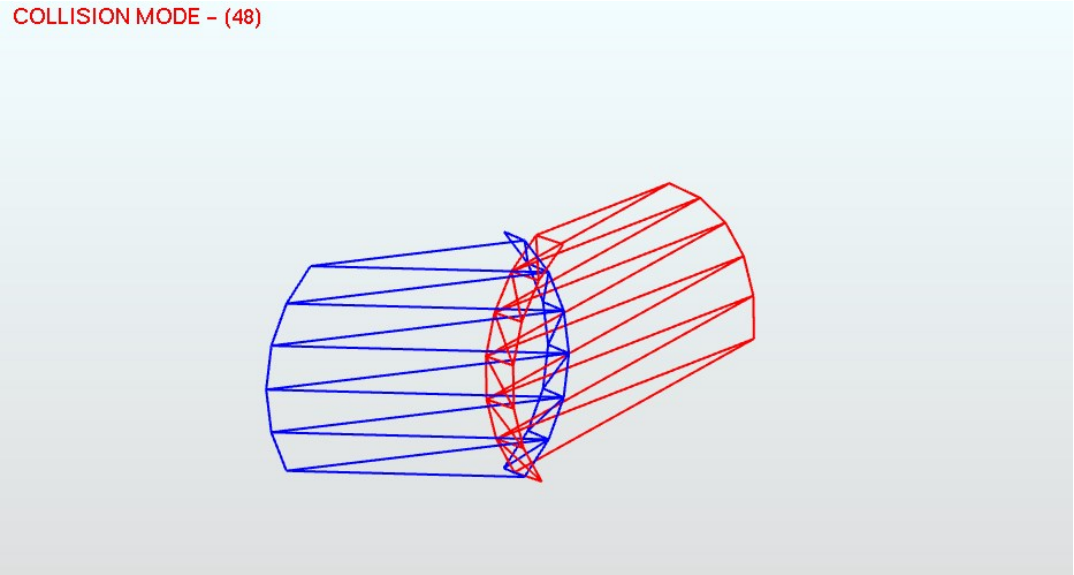


Obrázek 19: Znázornění aktivace výpočtu kolize

Klikem na tlačítko, zvýrazněné na obrázku č. 19, se vykoná kolizní algoritmus, jeho technika je popsána v samostatné kapitole níže. Po dokončení výpočtu se zobrazí okno udávající celkový počet kolizních trojúhelníků.

8.2.1 Vykreslení kolizních trojúhelníků

Provedením kontroly se uloží indexy konkrétních kolizních trojúhelníků. Tyto trojúhelníky lze vykreslit v kolizním módu, který je vyobrazen na obrázku č. 20. Mód se aktivuje po stisknutí klávesy „C“, po opakovaném stisknutí se mód deaktivuje. Kompletní zdrojový kód se nachází v souboru opengl.cpp, v metodě *drawSTL*.



Obrázek 20: Vykreslení kolizních trojúhelníků

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
```

Tato funkce určuje vykreslovací režim grafických primitiv. Pomocí parametru *GL_LINE* vykreslíme pouze hrany tělesa tzn. „drátěný model“.

```
glLineWidth(2);
glColor3f(0,0,1);
glBegin(GL_TRIANGLES);
```

Prvním řádkem nastavíme šířku vykreslené čáry na hodnotu 2 pixely. Funkce *glBegin* informuje OpenGL, že následuje zadání souřadnic vrcholů, které chceme vykreslit. Následuje smyčka *for*, procházející všechny trojúhelníky a pomocí funkcí *glNormal3f* a *glVertex3f* definujeme normálu a souřadnice vrcholů.

```
glEnd();
```

Funkcí *glEnd* informujeme OpenGL o ukončení vykreslování.

8.2.2 Kolizní algoritmus

Metoda vyžaduje ukazatel na místo, kde se nachází data o modelech a indexy dvou modelů na kterých se bude testovat, zda jsou modely v kolizi. Metoda vrací výsledek ve formě struktury.

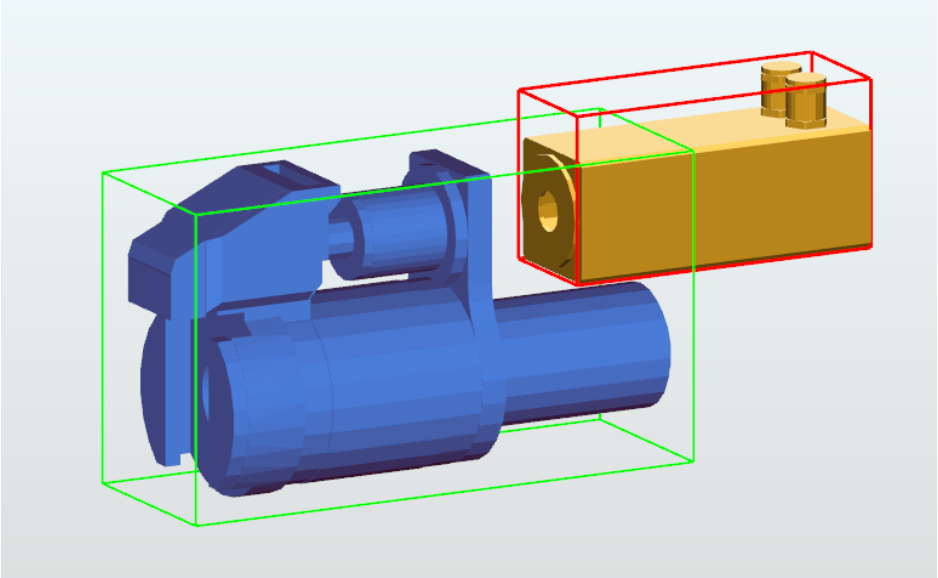
```
ListOP::checkCollision(MODEL_DATA *listOfModels, int first, int second)
[opengl.h]
struct DOUBLE_QLIST {
      QList<int>  first;
      QList<int>  second;

      int first_model_id;
      int second_model_id;
};
```

Struktura obsahuje indexy modelů, na kterých se prováděla kontrola kolize. Dále obsahuje dva datové kontejnery v provedení listu pro každý model, kde jsou uloženy indexy konkrétních kolizních trojúhelníků.

Nejprve u kontrolovaných modelů projdeme všechny souřadnice vrcholů trojúhelníků a získáme minimální a maximální hodnoty na všech osách, tedy x, y a z. Za pomocí těchto hodnot, lze objekt ohraničit a uzavřít do takzvané krabice. Krabice má nejmenší možnou velikost. Kontrola kolize je nejprve testována na vyhraničených oblastech pomocí metody *ListOP::intersection_minmax*. Jedná se o hojně využívanou techniku v prostorových grafických aplikací pod názvem Bounding box. Pokud je mezi ohraničenými objekty průnik, pokračuje se dále ve výpočtu.

Ne vždy však průnik ohraničených oblastí značí skutečnou kolizi mezi modely, jak ilustruje obrázek č. 21.



Obrázek 21: Kolize hraničních oblastí modelu

V dalším kroku jsou ověřeny všechny trojúhelníky mezi sebou pomocí metody *TriTri_Intersection::NoDivTriTriIsect*. Technika je popsána v samostatné kapitole 4.2 v teoretické části této práce. Metoda prověřuje, zda je mezi dvěma trojúhelníky kolize. Pokud ano, indexy kolizních trojúhelníků jsou uloženy do datových kontejnerů.

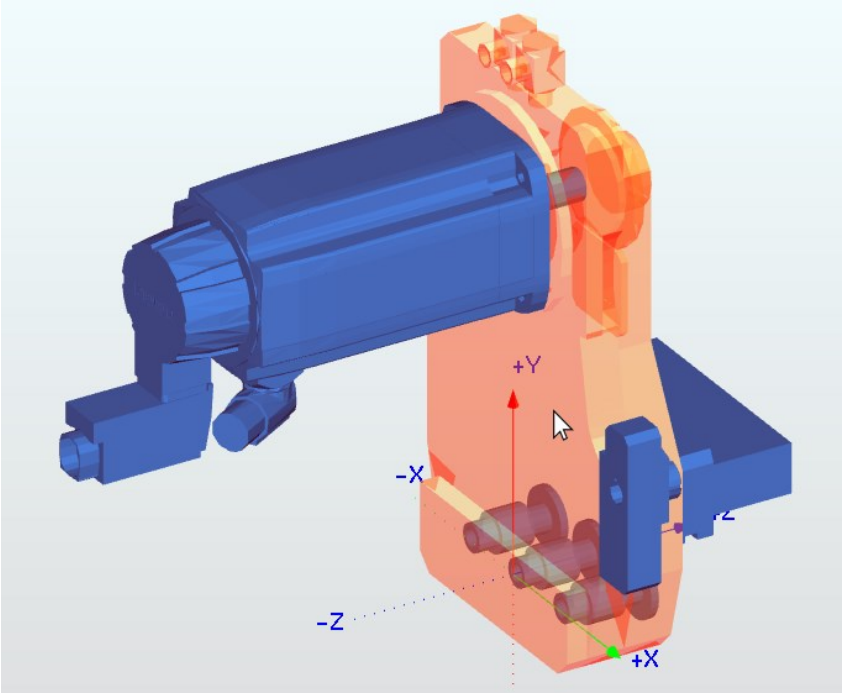
Zvýšení rychlosti výpočtu

Postupné procházení všech trojúhelníku a kontrolování, zda nedochází ke kolizi je velice náročné z hlediska výpočetního času. Využitím vícejádrového procesoru a rozdělením výpočtu na jednotlivá vlákna pomocí QThreads bychom urychlili výpočet. Rychlost výpočtu by se odvíjela od nejpomalejšího vlákna, tudíž je důležité správné rozložení zátěže.

Konkrétní trojúhelník prvního modelu by se nekontroloval lineárně se všemi trojúhelníky druhého modelu. Nýbrž by byla vytvořena například 4 vlákna a trojúhelníky druhého modelu by byly rozděleny na 4 části. Každé vlákno by tedy kontrolovalo kolizi trojúhelníku prvního modelu, s čtvrtinou trojúhelníků druhého modelu.

8.3 Výběr modelu ve scéně

Vykreslené modely v grafickém okně jsou interaktivní na pohyb kurzoru nebo kliknutí myší. Pokud například kurzor myši přejede přes vykreslený model, je vhodné konkrétní model odlišit jinou barvou od ostatních. Obrázek č. 22 zobrazuje odlišení modelu. V programu je také při téhle situaci zvýšena průhlednost skrz model. Další odlišení modelu nastane, když uživatel model označí do výběru pomocí kliknutí tlačítka myši na model.



Obrázek 22: Zvýraznění modelu na pozici kurzoru myši

Pro provádění výše uvedených operací je nutné identifikovat konkrétní model v grafické scéně programu. Vysvětlení techniky je popsáno v této kapitole. Při kliknutí tlačítka myši nebo při pohybu kurzoru v grafickém okně OpenGL je zavolána metoda *GLWidget::model-Picking*.

```
int GLWidget::modelPicking(MODEL_DATA *listModelu, QPoint mousePosition)
```

Metoda vyžaduje dva vstupní argumenty. Prvním argumentem je ukazatel na první prvek datového pole, reprezentující data modelu. Druhý vstupující argument je třída QPoint, která definuje pozici bodu ve dvourozměrném prostoru za pomocí souřadnic x a y v celočíselných hodnotách. Bod reprezentuje pozici kurzoru myši v oblasti vykreslovacího plátna OpenGL.

V případě úspěšné identifikace modelu, metoda vrací kladnou hodnotu reprezentující index modelu, který byl vybrán. Pokud nastane chyba metoda vrací zápornou hodnotu. Číslo -1

odpovídá neúspěšné identifikaci. Hodnota -2 je navrácena v případě neexistujících vstupních dat v prvním parametru.

8.3.1 Popis techniky identifikace

Technika identifikování modelu se dělí na tři dílčí části:

- Určení, zda se na daných souřadnicích nachází libovolný viditelný objekt.
- Vybrání modelů, u kterých jejich krabicová oblast zasahuje do souřadnic kurzoru. V případě pouze jednoho zasahujícího modelu, můžeme jednoznačně konstatovat správnost identifikace a navrátit index modelu.
- Více modelů zasahuje do dané pozice, musíme tedy konkrétně určit o jaký se jedná model.

8.3.2 Podrobné vysvětlení techniky

Identifikace modelu vyžaduje zapnutí paměti hloubky, toho lze dosáhnout pomocí funkce *glEnable(GL_DEPTH_TEST)*.

Test přítomnosti modelu

V první části využijeme volání funkce *glReadPixels*. Funkce umožňuje čtení a získání hodnoty daného obrazového bodu z vyrovnávací paměti zásobníku. V tomto případě z paměti hloubky. Funkce vyžaduje 7 parametrů:

1. Souřadnice osy x požadovaného pixelu.
2. Souřadnice osy y požadovaného pixelu.
3. Šírka výběrové oblasti. Hodnota 1 odpovídá jednomu pixelu.
4. Výška výběrové oblasti. Hodnota 1 odpovídá jednomu pixelu.
5. Určení, z jakého zásobníku se mají číst data.
6. Datový typ výsledných dat, v našem případě GL_FLOAT.
7. Ukazatel na proměnnou, kam chceme uložit výsledná data.

Data pro parametr 2 musí být upraveny. Z důvodu, že technologie OpenGL má nastaven počátek kreslícího plátna na pozici vrchního levého rohu. Kdežto vstupní souřadnice jsou počítány k počátku na pozici spodního levého rohu. Odečtením vstupní hodnoty souřadnice y od výšky kreslícího plátna získáme správný tvar parametru.

Návratová hodnota funkce je desetinné číslo v rozsahu 0 až 1. Čím menší číslo, tím je objekt blíže k pozorovacím bodu ve scéně, tedy obrazovce. Hodnota 1 odpovídá pixelu na jehož pozici se nenachází žádný vykreslený objekt, tudíž se jedná o prázdný prostor.

Získání hodnoty z hloubkové zásobníku je velice rychlé a málo náročné na výpočetní výkon počítače. Použitím této funkce si zajistíme, že se nebudou provádět náročnější výpočty na určení modelu, jestliže na pozici kurzoru myši není viditelný model či modely.

Detekce modelů v oblasti bodu

Pokud se tedy návratová hodnota z funkce nerovná hodnotě 1, na daných souřadnicích je vykreslen model nebo více modelů. Vstupní data reprezentující souřadnice ve dvojrozměrném prostoru musíme přepočítat na souřadnice trojrozměrného prostoru odpovídající vykreslené scéně. Převod realizujeme pomocí funkce *getOGLPost.*

U všech modelů vypočítáme souřadnice hraničních vrcholů oblasti, ve kterých se nachází. Porovnáním souřadnic zjistíme, zda požadovaný bod zasahuje do oblasti z některého modelu. V kladném případě uložíme index modelu do datového kontejneru vektor. Jestliže datový kontejner obsahuje pouze jeden člen, identifikovali jsme model na daných souřadnicích.

Identifikace přesného modelu

Za podmínky že vektor obsahuje více indexů je nutné pokračovat v přesnější identifikaci. Prostřednictvím dvojité smyčky procházíme konkrétní modely uložené v kontejneru a kontrolujeme všechny jejich trojúhelníky, dokud nezískáme výsledek.

Metoda *ListOP::IsPointInTriangle* přesně určí, zda bod nacházející se na daných souřadnicích zasahuje do oblasti trojúhelníku. Princip a teorie metody je popsána v samostatné kapitole (4.4 Bod v trojúhelníku) v teoretické části. Zdrojový kód je uveden níže.

Se zvyšováním vzdálenosti modelů od obrazovky, roste nepřesnost identifikace modelů. Jestliže po projití všech trojúhelníků nezískáme shodu, musíme snížit přesnost a výpočet opakovat.

Níže uvedené zdrojové kódy metod se nachází v souboru ListOP.cpp.

```
bool ListOP::IsPointInTriangle(RV6_VECTOR3 *pt, RV6_VECTOR3 *v1, RV6_VECTOR3
 *v2, RV6_VECTOR3 *v3, double accuracy){

      float TotalArea    = CalcTriangleArea(v1, v2, v3);
      float Area1        = CalcTriangleArea(pt, v2, v3);
      float Area2        = CalcTriangleArea(pt, v1, v3);
      float Area3        = CalcTriangleArea(pt, v1, v2);
      float tempSoucet   = std::floorf(Area1 + Area2 + Area3);
      tempSoucet -= 0.3;

      if((tempSoucet + accuracy) > TotalArea)

            return false;      // bod neleží v trojúhelníku

      else

            return true;       // bod leží v trojúhelníku

}
float ListOP::CalcTriangleArea(RV6_VECTOR3 *v1, RV6_VECTOR3 *v2, RV6_VECTOR3
 *v3){

      float a,b,c,t,area;
      area=a=b=c=t=0;

      a = sqrt(pow((v2->x - v1->x), 2) + pow((v2->y - v1->y), 2)
            + pow((v2->z - v1->z), 2));
      b = sqrt(pow((v3->x - v2->x), 2) + pow((v3->y - v2->y), 2)
            + pow((v3->z - v2->z), 2));
      c = sqrt(pow((v3->x - v1->x), 2) + pow((v3->y - v1->y), 2)
            + pow((v3->z - v1->z), 2));

      t = (a+b+c)/2;
      area = sqrt(t*(t-a)*(t-b)*(t-c));
      return area;

}
```

8.4 Výpočetní úpravy

Metody pro výpočetní transformace nad modelem se nachází v souboru a třídě ListOP. Úpravy se neprovádí pouze nad maticemi pohledu, ale provádí se přímo na vlastních souřadnicích vrcholů vybraného modelu nebo více modelech. Nabídka pro transformace se vyvolá kliknutím pravého tlačítka myši na položku ve stromové struktuře nebo na kliknutím na označený model v grafické scéně.

8.4.1 Změna polohy modelu

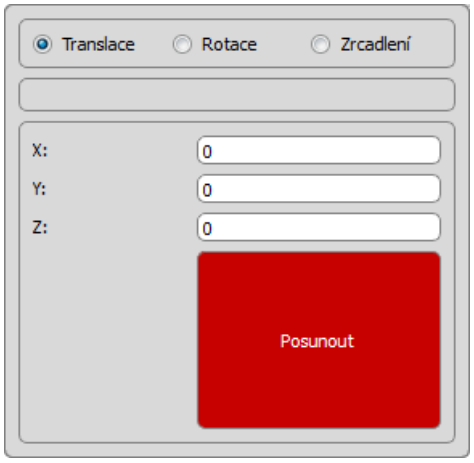
Model ve scéně je přemístěn na souřadnice nově zvolené polohy stroje. Položka modelu ve stromu je také přemístěna pod uzel reprezentující novou polohu. Transformační metody jsou použity také při změně polohy modelu. S tím rozdílem, že podpůrné transformace nejsou poté zahrnuty do historie provedených transformací. Záložka pro změnu polohy obsahuje možnost resetování všech dříve provedených transformací.

```
ListOP::model_transform(model, -old.x, -old.y, -old.z, false, true);
ListOP::model_transform(model, new.x, new.y, new.z, false, true);
model->actualPosition = novaPozice;
```

Model nejprve posuneme na počáteční souřadnice [0, 0, 0], toho docílíme odečtením vektoru souřadnic předešlé polohy. Následně je možné model posunout na souřadnice nové polohy. Struktura uchovává údaj o aktuálně nastavené poloze.

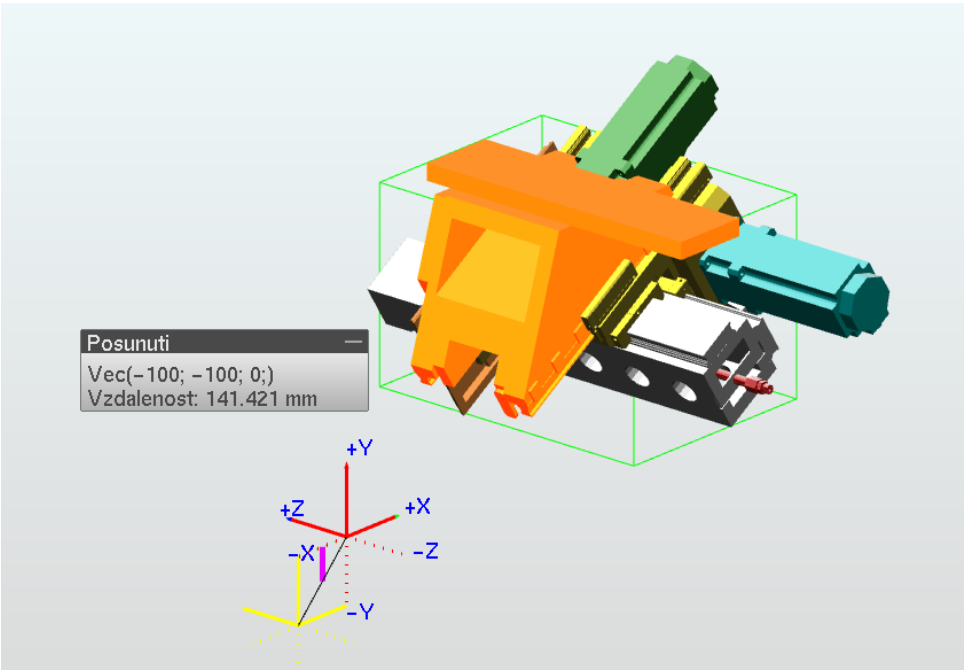
8.4.2 Translace (posunutí)

Posunutí je provedeno zadáním třemi body vektoru posunutí. Ke všem souřadnicím vrcholů modelu je přičtena hodnota vektoru pro danou osu. Dále je nutné přičíst vektor posunutí ke struktuře, která udává souřadnice počátku modelu. Model si udržuje výsledný vektor, složený ze všech předešlých translací pro možnost resetování na původní souřadnice. Obrázek č. 23 ilustruje uživatelské rozhraní pro translaci.



Obrázek 23: Translace modelu

Pokud provedeme posunutí modelu za pomocí virtuálního panelu stroje, přes grafickou scénu se vykreslí HUD okno, zobrazující vektor posunutí a vypočtenou celkovou vzdálenost. Okno lze také libovolně posouvat ve scéně. Virtuální panel stroje přibližuje kapitola č. 6.2.5.



Obrázek 24: Translace modelu za pomocí virtuálního panelu stroje

8.4.3 Rotace

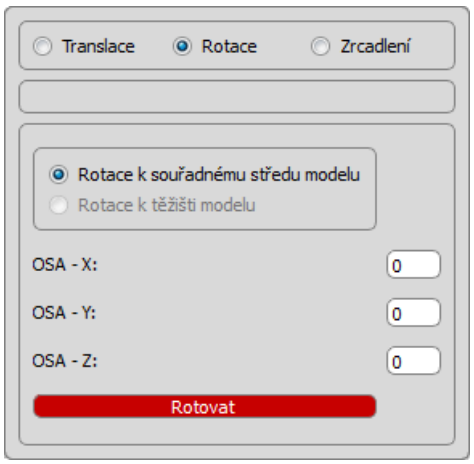
Hodnoty úhlu rotace podél jednotlivých os jsou zadávány uživatelem ve stupních, viz obrázek č. 25. Program nadále pracuje s radiány. Struktura reprezentující model obsahuje pomocnou matici rotací.

```
if(!resetMatice(model->modelMatrix, MatrixSize))
      return false;
Rotate(model->modelMatrix, x, y, z);
```

Matice je nejprve inicializována na jednotkovou matici metodou *resetMatice* a poté nastavena na hodnoty rotace odpovídající zadaným úhlům. V tomto bodě již metoda *Rotate* pracuje s radiány. Metoda dále obsahuje metody (*RotateX*, *RotateY*, *RotateZ*) pro rotace na jednotlivých osách. Sestavení matice se tedy realizuje složením několika dílčími kroky.

```
ListOP::model_transform(model, -poziceModelu[0], -poziceModelu[1],
                        -poziceModelu[2], false, true);
```

Jelikož se vlastní rotace provádí vůči počátečnímu bodu modelu, je nutné model na souřadnice tohoto bodu nejprve posunout. Poté jsou souřadnice vrcholů přepočítány za pomocí podpůrné matice rotací. S dokončením výpočtu nových souřadnic se model zpětně posune na původní pozici. Datová struktura modelu uchovává celkový součet hodnot úhlů rotací pro jednotlivé osy. Se zápisem těchto dat je provedeno modulo 360, pro snížení datové náročnosti programu.



Obrázek 25: Rotace modelu

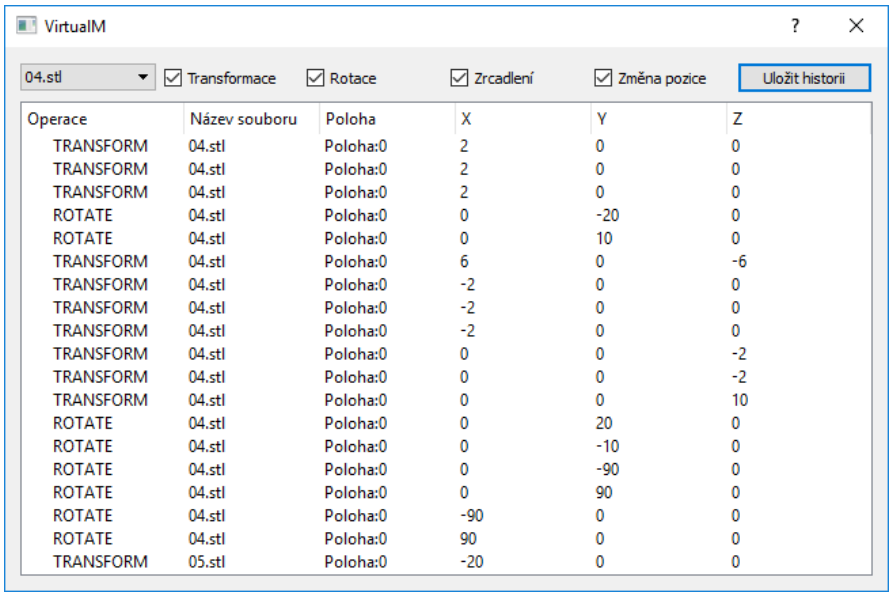
8.4.4 Zrcadlení

Vykreslený model ve scéně je možné zrcadlit podle konkrétní osy, také někdy označováno jako rovinná souměrnost podle osy. Například zrcadlením modelu podle osy X, jsou všechny souřadnice vrcholů na osách Y a Z vynásobeny hodnotou -1. Uživatel může zaškrtnout více os, podle kterých bude model zrcadlen, výsledná operace je složena z dílčích transformací. Obrázek č. 26 ilustruje uživatelské rozhraní pro translaci.

Obrázek 26: Zrcadlení modelu

8.4.5 Historie úprav

Provedené transformační metody jsou zaznamenány pomocí třídy *LOG* do dočasného logovacího souboru. V aplikaci je možné zobrazit seznam všech provedených úprav, zřetelné na obrázku č. 27. Akce pro otevření okna se nachází v menu aplikace pod záložkou scéna.



Operace	Název souboru	Poloha	X	Y	Z
TRANSFORM	04.stl	Poloha:0	2	0	0
TRANSFORM	04.stl	Poloha:0	2	0	0
TRANSFORM	04.stl	Poloha:0	2	0	0
ROTATE	04.stl	Poloha:0	0	-20	0
ROTATE	04.stl	Poloha:0	0	10	0
TRANSFORM	04.stl	Poloha:0	6	0	-6
TRANSFORM	04.stl	Poloha:0	-2	0	0
TRANSFORM	04.stl	Poloha:0	-2	0	0
TRANSFORM	04.stl	Poloha:0	-2	0	0
TRANSFORM	04.stl	Poloha:0	0	0	-2
TRANSFORM	04.stl	Poloha:0	0	0	-2
TRANSFORM	04.stl	Poloha:0	0	0	10
ROTATE	04.stl	Poloha:0	0	20	0
ROTATE	04.stl	Poloha:0	0	-10	0
ROTATE	04.stl	Poloha:0	0	-90	0
ROTATE	04.stl	Poloha:0	0	90	0
ROTATE	04.stl	Poloha:0	-90	0	0
ROTATE	04.stl	Poloha:0	90	0	0
TRANSFORM	05.stl	Poloha:0	-20	0	0

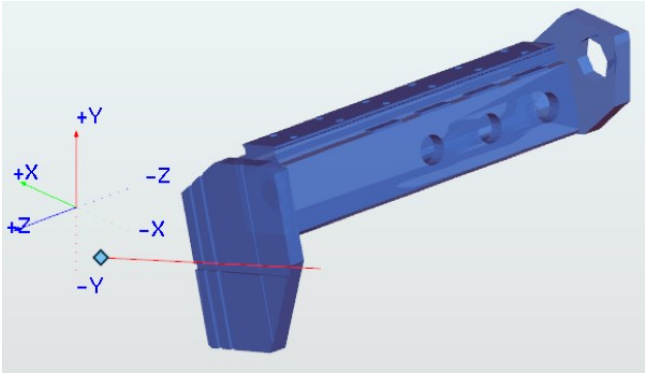
Obrázek 27: Seznam provedených transformací

8.5 Manipulace ve scéně

V průběhu rotace nebo posouvání scény se přes trojrozměrnou scénou vykreslí dvojrozměrná grafika. Tento styl vykreslování informací na obrazovku se označuje jako HUD.

8.5.1 Posouvání

Ve vykreslovacím okně je možné posouvat scénu pomocí stisknutí a držení klávesy Shift se souběžným stiskem středového tlačítka a pohybu myši pro výsledný posun.

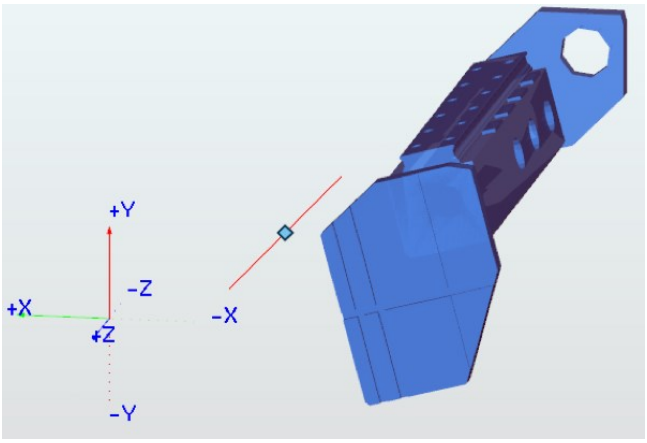


Obrázek 28: Posouvání scény

V průběhu posouvání scény je na obrazovku vykreslena grafika znázorňující počáteční bod propojený přímkou s aktuální pozicí kurzoru myši. Zobrazení grafiky zajišťuje metoda *GLWidget::drawDragMouse*. Obrázek č. 28 reprezentuje posunutí scény.

8.5.2 Rotování

Stiskem středového tlačítka a pohybu myši lze scénu otáčet okolo vztažného rotačního bodu. Rotační bod je nastaven na souřadnice aktivní polohy nebo středový bod modelu v případě aktivace módu „SpinCenter“. Grafiku vykresluje metoda *GLWidget::drawRotateMouse*.



Obrázek 29: Rotace scény

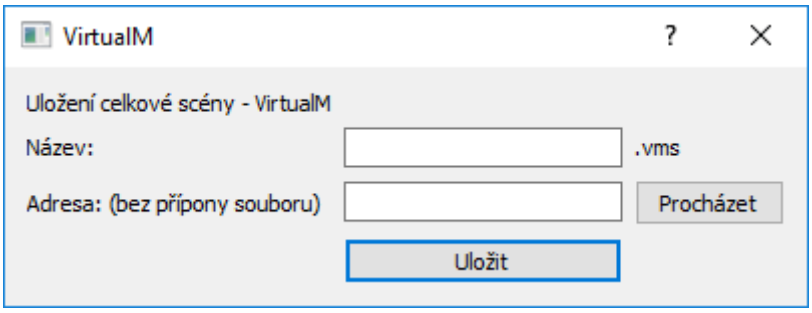
8.5.3 Změna měřítka scény

Rolování kolečka myši upravuje měřítko velikost scény. Při zahájení rolování je zavolána metoda *wheelEvent,* která předá instancí třídy *QWheelEvent* jako argument. Třída obsahuje funkci *delta()*, která vrací číslo udávající vzdálenost pootočení. Kladné číslo indikuje pohyb kolečka od uživatele, tedy přiblížení scény. Záporná hodnota čísla popisuje opačnou situaci.

Dle směru potočení kolečka se upravuje hodnota proměnné *meritko*. Tato proměnná je vstupním parametrem OpenGL funkce *glScale*, při vykreslení nového snímku. Se změnou velikosti scény se také decentně posouvá scéna vůči aktuální pozici kurzoru myši.

8.6 Ukládání a načítaní scény

Rozmístění a provedené úpravy nahraných modelů ve scéně je možné uložit do souboru s příponou „***vms***“ (Virtual Machine Scene). Otevřením tohoto souboru se scéna nastaví do stavu před uložením. Na obrázku č. 30 je dialogové okno, umožňující zadat název a adresu výstupního souboru.



Obrázek 30: Okno umožňující uložení scény

Vytvořený soubor popisující uloženou scénu byl navržen pro použití v této aplikaci. Soubor je v textovém formátu. Soubor neobsahuje data popisující souřadnice vrcholů modelu, ale obsahuje cestu k originálnímu souboru a hodnoty provedených transformací nad každým modelem. Dále obsahuje parametry pro nastavení scény.

ZÁVĚR

Cílem této práce bylo vytvoření aplikace, která demonstruje vizualizaci pracovního prostoru obráběcího stroje. Práce byla rozdělena do dvou částí.

Teoretická část popisovala technologie, které byly použity ke vzniku aplikace. První kapitola popsala knihovnu OpenGL, přiblížila čtenářovi vykreslování grafických primitiv a možnosti pro zrychlení vykreslování objektů. Kapitola také popsala transformační matice a základní operace s těmito maticemi. Další kapitola se zabývala aplikační platformou Qt a nástroji, které jsou součástí platformy. Technologie signálů a slotů byla podrobněji rozebrána a také byl uveden příklad použití. Následně byl popsán souborový formát STL, zejména byl vysvětlen binární formát a jeho struktura, která je důležitá při načítání modelu do programu. Následující kapitola popsala projekci a pohledovou transformaci, zde byla rozepsána ortografická projekce, z důvodu implementace v aplikaci. Závěr teoretické části se věnoval matematickým algoritmům, které byly použity pro identifikaci modelu ve scéně a detekci kolizí.

Praktická část úvodem zdůvodnila použité technologie a vysvětlila k jakému účelu byly použity. Byly také zmíněny principy a možnosti aplikace. Následující kapitola se zabývala strukturou aplikace, popis aplikace z pohledu programátora byl doplněn UML diagramem. Pojem pracovní poloha stroje vysvětlila samostatná kapitola. Grafické rozhraní bylo popsáno z pohledu uživatele. Následně bylo probráno kreslící plátno OpenGL a jeho důležité metody. Další kapitola podrobně objasnila nastavení scény a vykreslování modelů, které v aplikaci reprezentují struktury obsahující veškeré data a vlastnosti. Dále byly popsány metody jako importování modelů do aplikace, kolizní algoritmus, výběr modelu ve scéně a výpočetní úpravy modelu. Závěr praktické části popisoval ovládání, ukládání a načítání scény.

Kapitoly zabývající se klíčovými částmi aplikace byly doplněny útržky zdrojových kódů a obrázky ilustrující konkrétní část.

Aplikace byla primárně vyvíjena pro operační systém Windows, ale například s použitím softwaru Wine, který vytváří aplikační rozhraní, je možné aplikaci provozovat na systémech Linux a macOS. Program nebo jeho část však v tomto případě nemusí pracovat kompletně správně.

Za pomocí nahraných modelů, aplikace umožňuje vizualizovat scénu, která reprezentuje podobu reálného pracovního prostoru obráběcího stroje. Po provedení transformačních úprav nad modelem v této scéně, můžeme kontrolovat, zda provedení těchto konkrétních úprav nezpůsobuje vznik kolizí mezi jednotlivými částmi stroje. Vizualizací se tedy dá předejít vzniku kolizí, které by měly velké následky, v pracovním prostoru obráběcího stroje.

Upravená verze aplikace by mohla být napojena na reálný řídící systém obráběcího stroje a díky vstupním datům by umožňovala simulaci pohybu objektů v pracovním prostoru v reálném čase.

Dalším možným rozšířením aplikace by mohlo být implementace techniky konstruktivní geometrie těles, zkráceně CSG. Tato technika by umožnila provádět operace rozdílu mezi CSG primitivem a objektem reprezentující materiál. CSG primitiv by reprezentoval obráběcí nůž. Jeho postupné posouvání by vytvořilo plynulou animaci úběru materiálu.

SEZNAM POUŽITÉ LITERATURY

[1] OpenGL Overview: About OpenGL. *OpenGL* [online]. Beaverton (Oregon): The Khronos™ Group, ©1997-2018 [cit. 2018-02-10]. Dostupné z: https://www.opengl.org/about

[2] KESSENICH, John., Graham SELLERS a Dave SHREINER. Opengl programming guide: the official guide to learning opengl, version 4.5 with spir-v. 9th edition. Boston, MA: Addison-Wesley, 2016. ISBN 978-0134495491.

[3] OpenGL Tutorial: An Introduction on OpenGL with 2D Graphics. *Programming notes*[online]. Chua Hock-Chuan, 2012 [cit. 2018-02-10]. Dostupné z: https://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_Introduction.html

[4] TIŠNOVSKÝ, Pavel. Grafická knihovna OpenGL: Pole vrcholů. *Root.cz: Vývojářský software* [online]. Praha: Root.cz, 2003 [cit. 2018-02-11]. Dostupné z: https://www.root.cz/clanky/opengl-16-pole-vrcholu-vertex-arrays

[5] OpenGL: vykreslování objektů. *Elektronické studijní materiály* [online]. Brno: Mendelova univerzita [cit. 2018-02-11]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=32516

[6] WIELAND, WELBOURNE, Edward, ed. Qt History. *Qt wiki* [online]. 2015, Date of latest edit, November 2016 [cit. 2018-02-13]. Dostupné z: https://wiki.qt.io/Qt_History

[7] WIELAND, JONATANPC8, ed. About Qt. Qt wiki [online]. 2015, Date of latest edit, March 2018 [cit. 2018-02-13]. Dostupné z: https://wiki.qt.io/About_Qt

[8] Qt 4.8: Using the Meta-Object Compiler (moc). *Qt Documentation Archives*[online]. Finland: The Qt Company, ©2016 [cit. 2018-02-13]. Dostupné z: http://doc.qt.io/archives/qt-4.8/moc.html

[9] CHROBOCZEK, Martin. *Grafická uživatelská rozhraní v Qt a C++: [plně kompatibilní s Qt 5]*. Brno: Computer Press, 2013. ISBN 978-802-5141-243.

[10] Licensing. *Qt* [online]. Finland: The Qt Company, ©2018 [cit. 2018-02-13]. Dostupné z: https://www1.qt.io/licensing/

[10] Licensing comparison. *Qt* [online]. Finland: The Qt Company, ©2018 [cit. 2018-02-13]. Dostupné z: https://www1.qt.io/licensing-comparison/

[11] Qt 4.8: Signals & Slots. *Qt Documentation Archives* [online]. Finland: The Qt Company, ©2016 [cit. 2018-02-14]. Dostupné z: http://doc.qt.io/archives/qt-4.8/signalsandslots.html

[12] Qt Visual Studio Add-in. *Qt Documentation Archives* [online]. Finland: The Qt Company, ©2016 [cit. 2018-02-14]. Dostupné z: http://doc.qt.io/archives/vs-addin/index.html

[13] Qt Creator Manual. *Qt Documentation* [online]. Finland: The Qt Company, ©2018 [cit. 2018-02-14]. Dostupné z: http://doc.qt.io/qtcreator/

[14] Qt Linguist Manual. *Qt Documentation* [online]. Finland: The Qt Company, ©2018 [cit. 2018-02-13]. Dostupné z: http://doc.qt.io/qt-5/qtlinguist-index.html

[15] Qt 4.8: Introduction to the QML Language. *Qt Documentation Archives* [online]. Finland: The Qt Company, ©2016 [cit. 2018-05-19]. Dostupné z: http://doc.qt.io/archives/qt-4.8/qdeclarativeintroduction.html

[16] What Is An STL File?. *3D Systems* [online]. Rock Hill (South Carolina): 3D Systems, ©2018 [cit. 2018-01-15]. Dostupné z: https://www.3dsystems.com/quickparts/learning-center/what-is-stl-file

[17] VAN LIESHOUT, Laurens. STL-file: Simple description of a STL-file. The drawing is used to explain the accuracy of a STL file. In: *Wikipedia* [online]. San Francisco: Wikimedia Foundation, 2009, 7 March 2009 [cit. 2018-04-10]. Dostupné z: https://commons.wikimedia.org/wiki/File:STL-file.jpg

[18] CHAKRAVORTY, Dibya. STL File Format for 3D Printing – Simply Explained. *All3DP*[online]. Mnichov: All3DP, 2017, Jun 13, 2017 [cit. 2018-01-15]. Dostupné z: https://all3dp.com/what-is-stl-file-format-extension-3d-printing/

[19] TIŠNOVSKÝ, Pavel. Grafická knihovna OpenGL: Zobrazení prostorové scény s ortografickou kamerou, double-buffering. *Root.cz* [online]. Praha: Root.cz, 2003 [cit. 2018-04-10]. Dostupné z: https://www.root.cz/clanky/opengl-13-double-buffering/

[20] PROCHÁZKA, David, Tomáš KOUBEK a Jana ANDRÝSKOVÁ. Programování grafických aplikací s využitím OpenGL a OpenCV: Nastavení projekční matice v OpenGL. *Elektronické studijní materiály* [online]. Brno: Mendelova univerzita

[cit. 2018-04-10]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=22411

[21] MÖLLER, Tomas. A Fast Triangle-Triangle Intersection Test. *Journal of Graphics Tools*. Natick (Massachusetts): A. K. Peters, 1997, **1997**(2), 25-30. ISSN 1086-7651. Dostupné také z: http://fileadmin.cs.lth.se/cs/Personal/Tomas_Akenine-Moller/code/tritri_tam.pdf

[22] ŽÁRA, Jiří, BENEŠ, Bedřich, FELKEL, Petr. Moderní počítačová grafika. 1. vyd. Praha: Computer Press, 2005. 448 s. ISBN 80-7226-049-9.

[23] Heron's Formula: Triangle Properties. *Wolfram MathWorld* [online]. Champaign (Illinois): Wolfram Research, ©1999-2018 [cit. 2018-04-12]. Dostupné z: http://mathworld.wolfram.com/HeronsFormula.html

[24] Qt 4.8: Drag and Drop. *Qt Documentation Archives* [online]. Finland: The Qt Company, ©2016 [cit. 2018-03-15]. Dostupné z: http://doc.qt.io/archives/qt-4.8/dnd.html

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

CAD	Computer-aided design (Počítačem podporované projektování)
CSG	Constructive Solid Geometry (Konstruktivní geometrie těles)
CSS	Cascading Style Sheets (Kaskádové styly)
GUI	Graphical User Interface (Grafické uživatelské rozhraní)
HUD	Heads-up display
JSON	JavaScript Object Notation (JavaScriptový objektový zápis)
MOC	Meta-Object Compiler
OOP	Objektově orientované programování
OpenGL	Open Graphics Library
QML	Qt Modeling/Markup Language
STL	Stereolithography

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH