Využití OpenGL při vizualizaci řídících procesů

Petr Mahdalíček

Bakalářská práce
2007

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2006/2007

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Petr MAHDALÍČEK**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Využití OpenGL pro simulaci a vizualizaci řídicích procesů**

Zásady pro vypracování:

1. Vypracujte literární rešerši na zadané téma. Ta bude obsahovat charakteristiky vizualizačních programů a požadavky na ně. Popisy založte na vizualizačních řídicích programech Intouch a ControlWeb.
2. V teoretické části práce dále charakterizujte OpenGL a blíže se seznamte s nástroji, které by se daly využít při vizualizaci.
3. Vypracujte návrh vizualizační aplikace založené na OpenGL. Vytvořte vhodný konkrétní model, na kterém by byla vizualizace dobře patrná.
4. Na základě návrhu uvedeného v předchozím bodě tento program vytvořte. Doplňte ho o vhodné vlastnosti, které se při vizualizaci běžně používají.
5. Seznamte se se základními algoritmy řízení. Vhodné typy regulátorů implentujte do Vámi vytvořeného programu a následně ověřte jejich funkčnost.

Rozsah práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. Shreiner, D. a kol.: OpenGL - průvodce programátora. Computer Press, 2006.
2. balátě, J.: Vybrané statě z automatického řízení. Skripta FT, VUT Brno, 1996.
3. http://www.opengl.org - domovská stránka OpenGL.
4. http://nehe.gamedev.net - www stránka s mnoha OpenGL projekty.

Vedoucí bakalářské práce: **Ing. Pavel Pokorný, Ph.D.**
Ústav aplikované informatiky

Datum zadání bakalářské práce: **13. února 2007**

Termín odevzdání bakalářské práce: **24. května 2007**

Ve Zlíně dne 13. února 2007

prof. Ing. Vladimír Vašek, CSc.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Cílem mé bakalářské práce je vytvořit program, který bude pomocí OpenGL zobrazovat průběh řízení různých systémů. Program se bude zaměřovat na visuální stránku. Pro jednoduchost budou podporovány pouze spojité dvoupolohové regulátory s penalizací. Nejedná se tedy o program určený pro laboratorní účely, ale spíše o program určený k prezentovaní výrobků, řídících systémů, různých simulací. Vykreslování scény bude postaveno na vlastním vykreslovacím enginu. K vytvoření aplikace bude použita knihovna SDL, celý program díky tomu bude spustitelný pod operačními systémy windows i linux.

Klíčová slova: opengl, řídící systém, simulace, vykreslování

ABSTRACT

Main goal of this bachelor thesis is create programme, which will use OpenGL to draw process of regulation. Programme will be focused on visual part. To keep it simple, programme will support just high-low regulators with penalization. So, this programme is not determined to use in the laboratory, but rather to present products, control systems and different simulations. Rendering will be made by my own render engine. Application will be made with library SDL. Whole programme will run under operating systems windows and linux.

Keywords: OpenGL, regulation, simulation, rendering

Tímto bych chtěl poděkovat Ing. Pavlu Pokornému, Ph.D. za vedení při tvorbě této práce. Děkuji mu za rady, podněty, připomínky a čas, který mi věnoval.

Děkuji také Martinu Šrámkovi za tvorbu všech modelů obsažených v programu.

Ve Zlíně …….

Podpis diplomanta

OBSAH

ÚVOD

V současné době výpočetní technika významně zasahuje do všech odvětví lidského snažení. Je také důležitou pomůckou a hybnou silou v komplexním vývoji informačních technologií. Využívá se hlavně tam, kde by běžné metody zpracovávání dat byly časově a technicky velmi náročné a díky tomu i značně nákladné. Jsou to zejména grafické simulace průběhu procesu na základě vložených informací, složité výpočtové operace, rozsáhlé databáze dat a jiné operace, které by bez pomocí programů navržených pro tyto účely byly dnes již snad neřešitelné. Vývoj softwarového vybavení výpočetní techniky je již dnes na takové úrovni, že nám neslouží jen jako pomocné nástroje, ale jsou schopny samy zadané úlohy řešit nebo kontrolovat jejich navrhování, provoz, řízení a ovládání.

Automatizace a řízení je jeden z oborů, kde zastávají zmíněné technologie jednu z klíčových úloh, protože zde člověka v některých fázích zcela úplně nahrazují. Usnadňují navrhování soustav a jejich parametrů, jako je volba typu regulátoru a jeho parametrů, struktury regulátoru atd. Je důležitá také při průběžné analýze, simulaci, monitorování, atd.

V tomto odvětví se používá velké množství programů, které provádějí činnosti zmiňované v předchozím odstavci např. regulaci měřené soustavy, simulaci regulačního pochodu, atd. Mezi nejpoužívanější programy v této oblasti se jistě řadí ControlWeb a InTouch.

Tato práce se podrobněji zabývá problematikou vizualizace regulačního pochodu. Při návrhu celého systému jsem se řídil zkušenostmi získanými při práci s programem ControlWeb. Realizovat jsem se ovšem pokusil jen zlomek z toho co nabízí tento komplexní program. Cílem je vytvořit program, který by umožňoval jednoduché vytvoření scény z hotových modelů. Následné propojení modelů do funkčního celku, a efektní presentaci. Vzhledem k rozsáhlosti, nebude součástí práce IDE ve kterém by bylo možné tyto scény vytvářet. Tvorba IDE by byla vhodným pokračováním projektu, například jako diplomová práce. V ukázkovém programu budou dvě vizualizace, a sice regulace teploty a regulace výšky vodní hladiny.

I. TEORETICKÁ ČÁST

1 CONTROL WEB

Obrázek 1 : Logo aplikace Control Web

Definovat co je Control Web nebo vyjmenovat všechny jeho vlastnosti je na omezeném prostoru prakticky nemožné. Pro někoho

je Control Web přístupný nástroj, který umožní levně realizovat řízení např. malé vodní elektrárny. Pro někoho jiného je to prostředek tvorby rozsáhlé podnikové distribuované aplikace s desítkami tisíc měřených bodů a obsahující stovky operátorských obrazovek, pracující na řadě počítačů zapojených do sítě. Nebo může Control Web pracovat jako programový most mezi SQL databází, WWW prohlížeči a GSM sítí. Pro řadu studentů je to nástroj, který jim ušetří spoustu práce s laboratorními pracemi, neboť automatizovaně provádí měření a tvoří protokoly.

1.1 Co je to Control Web?

- programový systém rychlého vývoje aplikací pro průmysl, laboratoře, školy, . . .
- vizualizace a řízení technologických procesů v reálném čase.
- most mezi technologií a informačním systémem pod- niku.
- rozhraní člověk-stroj.
- přímé řízení strojů a technologií.
- simulace, výzkum, vývoj a výuka (třeba LF UK).

1.2 Podpora Hardware

- Control Web je důsledně navrhován jako systém nezávislý na hardware.
- s patřičným ovladačem komunikuje s jakýmkoliv průmyslovým zařízením:
  - PLC (Siemens, Mitsubishi, Omron, Teco, Allen-Bradley, ABB, Honeywell, . . .).
  - I/O moduly (DataLab IO, ELSACO, ADAM, . . .).
  - měřicí karty (Advantech, Axiom, Tedia, . . .).
  - „virtuální“ zařízení, např. WWW server apod.
- architektura ovladačů je otevřená a pečlivě dokumentovaná, každý může implementovat vlastní

ovladač.

1.3 Podpora otevřených protokolů

- ASCII komunikace po sériové lince.
  - Znakový protokol využívá velké množství jednoduchých zařízení . . .
- OPC Data Access.
  - Stále vzrůstající množství OPC serverů.
- DDE / NetDDE, FastDDE.
  - Zachování zpětné kompatibility s DDE servery.

- GSM modemy, SMS zprávy.
- HTTP přístup k WWW serverům.
- Modicon Modbus, Modbus/TCP.

1.4 Podpora otevřených standardů

- široká interoperabilita díky podpoře standardních protokolů a formátů dat.
- TCP/IP, HTTP, HTML (Ethernet, WiFi, dial-up, . . .).
- ODBC / SQL
- COM / ActiveX
- OPC (OLE for Process Control)
- GSM / GPRS
- DDE, NetDDE

1.5 Schopnost práce v distribuovaném prostředí

1.5.1 Control Web Runtime („tlustý klient“)

- aplikace Control Web dokáží sdílet data po síti, volat vzdálené metody a podobně.
- data mohou být sdílena za účelem zálohování (synchronizace dat).
- nebo je možné přistupovat na vzdálené data (vzdálený přístup).
- oba způsoby je možno libovolně kombinovat a tvořit tak aplikace klient / server nebo peer-to-peer.

1.5.2 Přístup k aplikaci přes WWW prohlížeč („tenký klient“)

- Control Web obsahuje zabudovaný HTTP server a do- káže vytvářet dynamické aplikace založené na WWW technologiích, zpřístupňované prostřednictvím standardních WWW prohlížečů.
- je možné vytvářet serverové aplikace pro klienty na plnohodnotných PC i na mobilních telefonech.
- bohatost aplikace lze nastavovat podle požadavků na přístup z různých klientů (čisté HTML,

  DHTML/CSS, Java, ActiveX, . . .).

1.6 Podpora platforem

Control Web podporuje všechny Win32 platformy:

- Windows 9x/Me (dožívající platforma)
- Windows XP Embedded (možnost práce z CF karty, bez HDD)
- Windows 2000 Advanced Server Clusters
- Windows CE na standardním x86 PC (CEPC)
- Windows CE na RISC systémech (verze pro procesory ARM, MIPS, SH3/4)

Obrázek 2 : Popis systémů Control Web

1.7 Podpora jazyků a kódování

1.7.1 Podpora kódování

- Control Web ANSI (8bitové znaky pro Evropu a USA).
- Control Web UNICODE (16bitové kódování obsahu- jící znaky všech abeced).
- UNICODE verze je nutná pro podporu východních jazyků.

1.7.2 Podpora jazyků

- Vývojová i runtime verze v češtině, angličtině, němčině a japonštině.

- možnost upravit texty v runtime verzi pro jakýkoliv jazyk.
- runtime ve slovenštině, ruštině, . . .

1.8 Neomezená programovatelnost

- zabudovaný programovací jazyk s real-time rozšíření- mi dovoluje realizovat zcela libovolné řídicí sekvence a algoritmy.
- programové API jednotlivých komponent dovoluje jejich plně programové řízení
  − např. archivace dat, SQL dotazy apod.
- událostní rozhraní komponent umožní reagovat na rozličné stavy aplikace.
- atributy kanálů umožňují precizně řídit komunikaci a časování (QoS, Round-trip-time, timeout).

1.9 Bezpečný programový model

- aplikační program nemá přímý přístup k paměti.
- eliminuje problémy s neplatnými ukazateli.
- zabraňuje nestabilitám způsobeným nevracením paměti (memory leaks).
- ošetření chyb ve výrazech (dělení nulou, indexování mimo rozsah pole, přetečení, podtečení, ztráta přesnosti, . . .).
- možnost testování výskytu chyby a její programové ošetření.

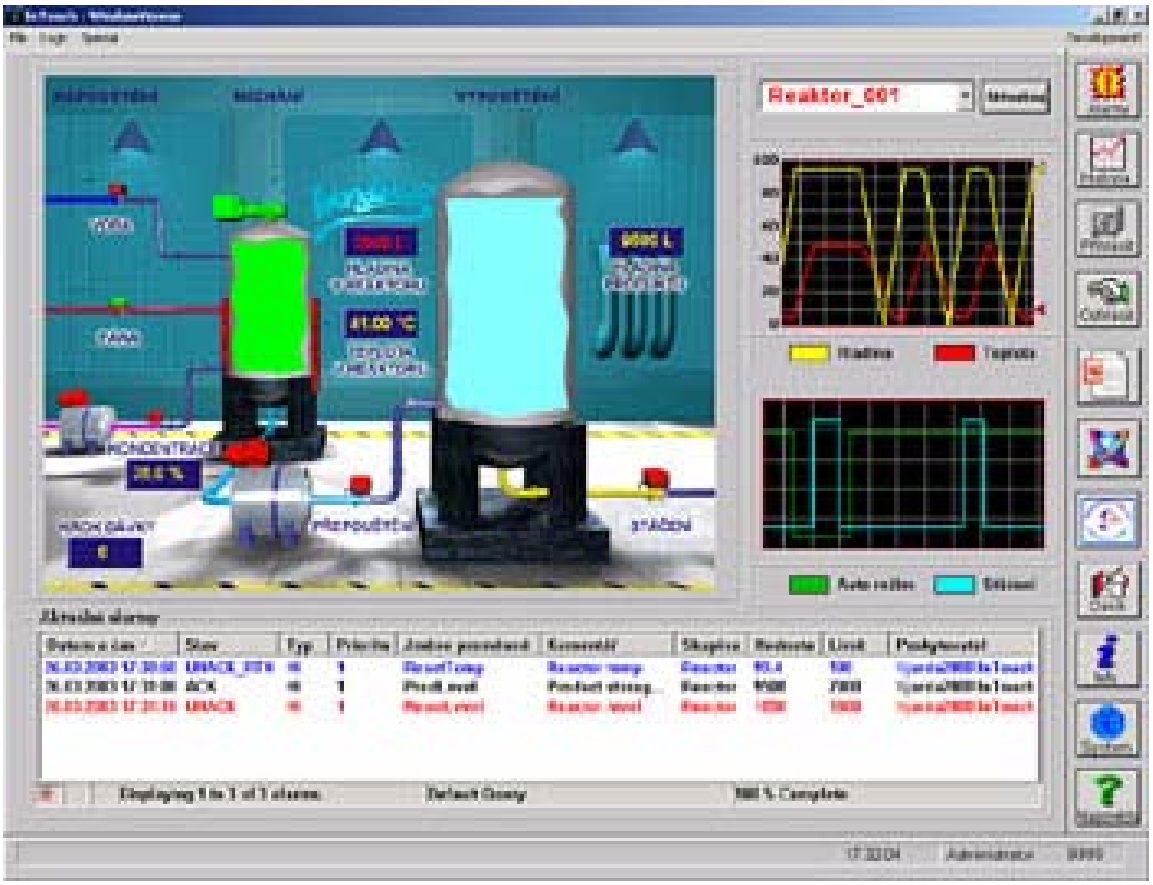
1.10 Integrované vývojové prostředí

- tvorba aplikace drag-and-drop.
- komponenty (virtuální přístroje) aplikace přetahovány z palety.
- modifikace parametrů v dialogových oknech.
- přístrojový inspektor modifikuje specifické parametry daného virtuálního přístroje.
- k tvorbě aplikace není zapotřebí znalost programování.
- programovací jazyk je k dispozici pro řešení náročnějších požadavků zákazníků.
- grafický vývoj aplikací.

2 INTOUCH

Wonderware InTouch je světově nejpoužívanější softwarový produkt kategorie SCADA/HMI (Supervisory Control and Data Acquisition / Human-Machine Interface), který poskytne jednotný a integrovaný pohled na všechny Vaše řízené technologie a výrobní procesy.

InTouch umožňuje operátorům, technologům, kontrolorům i manažerům v reálném čase sledovat a reagovat na průběhy veškerých výrobních operací prostřednictvím názorného grafického znázornění libovolných technologických procesů.



Obrázek 3 : Ukázka z programu InTouch

2.1 Základní vlastnosti

2.1.1 Podporované operační systémy

Aplikace InTouch lze provozovat na operačních systémech MS Windows 2000 (včetně podpory terminálových služeb), Windows XP nebo Windows NT 4.0.

2.1.2 Pracovní nástroje

Pro sběr dat z technologických procesů je k dispozici rozsáhlá nabídka komunikačních I/O Serverů přímo od Wonderware nebo od nezávislých softwarových firem, podporována je samozřejmě i komunikace s OPC Servery od libovolných dodavatelů.

Kromě nástrojů pro snadné vytvoření grafických obrazovek zobrazujících aktuální stavy provozovaných technologií je součástí systému InTouch i správa distribuovaných historických dat umožňující i spolupráci s výkonnou procesní databází Wonderware Industrial-SQL Server a správa distribuovaných alarmů (výstrah), které lze ukládat do databáze MS SQL Server 2000 nebo MSDE 2000.

Pro dosažení libovolných funkčností aplikací je k dispozici výkonný událostně orientovaný skriptový jazyk (QuickScripts) s množstvím vestavěných funkcí. Speciální požadavky lze splnit i s využitím jiných technologických standardů, např. pomocí objektů ActiveX, komunikace s relačními databázemi s využitím rozhraní ADO/ODBC apod.

Standardní součástí systému InTouch jsou také rozšiřující moduly Recipe Manager, SQL Access, SPC (Statistical Process Control) a sada rozšiřujících nástrojů zahrnující mj. užitečnou knihovnu SymbolFactory, obsahující více než 2000 objektů s rozmanitou funkčností, díky nimž je vývoj aplikací ještě snadnější než kdykoliv předtím.

3 OPENGL

Obrázek 4 : Logo OpenGL

3.1 Historie OpenGL

Na počátku osmdesátých let minulého století, v době, kdy výpočetní technice kraloval textový režim, se u společnosti Silicon Graphics Inc. (SGI) začaly vyvíjet grafické stanice a knihovna IRIS GL, kterou lze považovat za přímého předka OpenGL. Jak šel vývoj dopředu, zbavovala se některých problematických rysů a dá se říct, že její poslední verze je s OpenGL téměř kompatibilní. Standard OpenGL vznikl na začátku devadesátých let (uvádí se rok 1992) a narozdíl od IRIS GL byl od počátku koncipován jako nezávislý na hardwaru, operačním systému a programovacím jazyce. Od té doby se změnil pouze minimálně, nejnovější verze je 2.1 .

3.2 Základní informace o OpenGL

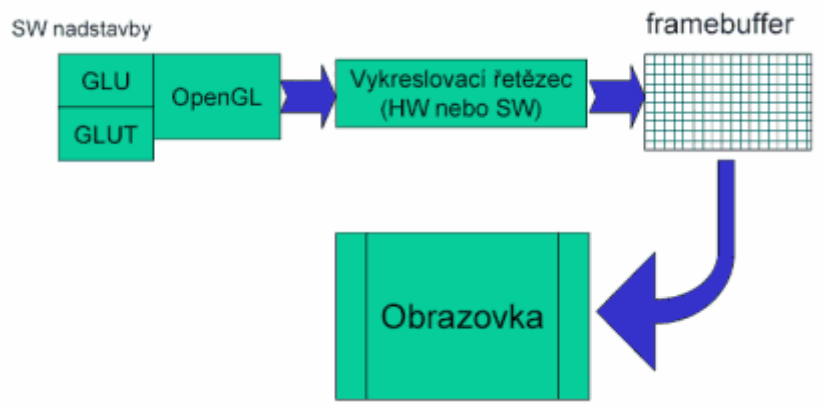
Knihovna OpenGLTM (Open Graphics Library) byla navržena firmou SGI (Silicon Graphics Inc.) jako aplikační programové rozhraní (Application Programming Interface - API) k akcelerovaným grafickým kartám resp. celým grafickým subsystémům. Předchůdcem této knihovny byla programová knihovna IRIS GL (Silicon Graphics IRIS Graphics Library). OpenGL byla navržena s důrazem na to, aby byla použitelná na různých typech grafických akcelerátorů a aby ji bylo možno použít i v případě, že na určité platformě žádný grafický akcelerátor není nainstalován - v tom případě se použije softwarová simulace. V současné době lze knihovnu OpenGL použít na různých verzích unixových systémů (včetně Linuxu a samozřejmě IRIXu), OS/2 a na platformách Microsoft Windows.

Logo OpenGL a název OpenGLTM je registrovaná známka firmy Silicon Graphics Inc.

Na některých platformách je možné rozdělení aplikace na dvě relativně samostatné části - serverovou a klientskou. Při vykreslování se potom jednotlivé příkazy (což jsou většinou parametry funkcí OpenGL) přenášejí přes síťové rozhraní. Knihovna OpenGL [3] (narozdíl od IRIS GL nebo Direct 3D) byla vytvořena tak, aby byla nezávislá na použitém operačním systému, grafických ovladačích a správcích oken (Window Managers). Proto také neobsahuje žádné funkce pro práci s okny (otevírání, zrušení, změnu velikosti), pro vytváření grafického uživatelského rozhraní (Graphical User Interface - GUI) ani pro zpracování událostí. Tyto funkce lze zajistit buď přímo voláním funkcí příslušného správce oken, nebo lze použít některou z nadstaveb, například knihovnu GLUT [3] (OpenGL Utility Toolkit). Pro dosažení co největší nezávislosti na použité platformě zavádí knihovna OpenGL vlastní primitivní datové typy, například GLbyte, GLint nebo GLdouble.

Programátorské rozhraní knihovny OpenGL je vytvořeno tak, aby knihovna byla použitelná v téměř libovolném programovacím jazyce. Primárně je k dispozici hlavičkový soubor pro jazyky C a C++. V tomto souboru jsou deklarovány nové datové typy používané knihovnou, některé symbolické konstanty (např. GL_POINTS) a sada cca 120 funkcí tvořících vlastní rozhraní. Existují však i podobné soubory s deklaracemi pro další programovací jazyky, například Fortran, Object Pascal či Javu; tyto soubory jsou většinou automaticky vytvářeny z Cčkovských hlavičkových souborů.

Z programátorského hlediska se OpenGL chová jako stavový automat. To znamená, že během zadávání příkazů pro vykreslování lze průběžně měnit vlastnosti vykreslovaných primitiv (barva, průhlednost) nebo celé scény (volba způsobu vykreslování, transformace) a toto nastavení zůstane zachováno do té doby, než ho explicitně změníme. Výhoda tohoto přístupu spočívá především v tom, že funkce pro vykreslování mají menší počet parametrů a že jedním příkazem lze globálně změnit způsob vykreslení celé scény, například volbu drátového zobrazení modelu (wireframe model) nebo zobrazení pomocí vyplněných polygonů (filled model). Vykreslování scény se provádí procedurálně - voláním funkcí OpenGL se vykreslí výsledný rastrový obrázek. Výsledkem volání těchto funkcí je rastrový obrázek uložený v tzv. framebufferu, kde je každému pixelu přiřazena barva, hloubka, alfa složka popř. i další atributy. Z framebufferu lze získat pouze barevnou informaci a tu je možné následně zobrazit na obrazovce - viz následující obrázek.



Obrázek 5 : Popis vykreslování

OpenGL nezaručuje, že při spuštění identického programu používajícího knihovnu OpenGL na různých platformách nebo různých grafických akcelerátorech dostaneme vždy přesně stejný výsledek. Pokud bychom oba výsledné rastrové obrázky porovnali pixel po pixelu, mohli bychom zjistit mírné rozdíly v barvách. Může to být způsobeno například odlišnou přesností reprezentace čísel na grafické kartě, odlišnými algoritmy pro interpolaci barvy, normály a texturových souřadnic nebo jinou bitovou hloubkou Z-bufferu. Celkové geometrické a barevné podání scény by však mělo být zachováno.

Pomocí funkcí poskytovaných knihovnou OpenGL lze vykreslovat obrazce a tělesa složená ze základních geometrických prvků, které nazýváme grafická primitiva. Mezi tato primitiva patří bod, úsečka, trojúhelník, čtyřúhelník, plošný konvexní polygon, bitmapa (jednobarevný rastrový obraz) a pixmapa (barevný rastrový obraz). Existují i funkce, které podporují proudové vykreslování některých primitiv - lze například vykreslit polyčáru (line loop), pruh trojúhelníků (triangle strip), pruh čtyřúhelníků (quad strip) nebo trs trojúhelníků (triangle fan). Na vrcholy tvořící jednotlivá grafická primitiva lze aplikovat různé transformace (otočení, změna měřítka, posun, perspektivní projekce), pomocí kterých lze poměrně jednoduše vytvořit animace. Vykreslovaná primitiva mohou být osvětlena nebo pokryta texturou. Dále je možné celou vykreslovanou scénu "skrýt" v mlze.

Z hlediska datové reprezentace vykreslované scény poskytují funkce OpenGL, podobně jako Direct 3D Immediate Mode, pouze základní rozhraní pro přístup ke grafickým akcelerátorům. Existují však rozšiřující knihovny, které funkcionalitu dále zvyšují. Jednou ze základních knihoven používaných společně s OpenGL je knihovna GLU (OpenGL Utilities), která umožňuje využívat tesselátory (rozložení nekonvexních polygonů na trojúhelníky),

evaluátory (výpočet souřadnic bodů ležících na parametrických plochách) a vykreslovat kvadriky (koule, válce, kužely a disky). Další nadstavbovou knihovnou je Open Inventor, pomocí kterého lze konstruovat celé scény složené z hierarchicky navázaných objektů. V porovnání s Direct 3D Retained Mode, kde se také pracuje s hierarchií scény, je však Open Inventor knihovna mnohem mocnější a přitom má poměrně jednoduché rozhraní.

3.3 The OpenGL Shading Language

The OpenGL Shading Language (*GLSL, glslang*) [3], [4] je high level programovací jazyk, který je součástí specifikace standardu OpenGL 2.0. GLSL je určen pro přímé programování jádra moderního GPU. Tento programovací jazyk dovoluje obejít nativní funkci akcelerátoru v tzv. pevném režimu (*fixed functionality*) a nahradit ji režimem programovatelným, kdy se grafický procesor ovládá pomocí kódu napsaného v GLSL. Takovému zdrojovému kódu říkáme shader. GLSL navrhla společnost 3Dlabs, Inc. v roce 2004. Základní stavební kameny jazyka stojí na třech inspiracích. První je jazyk C/C++, druhá je grafický standard Pixar (Apple) RenderMan a třetí je OpenGL. Engine pro shadering používá právě tento jazyk.

II. PRAKTICKÁ ČÁST

4 REALIZACE 3D ENGINU V OPENGL

Praktická část této práce se zakládá na implementaci 3D grafického enginu v OpenGL 2.1. Engine je programovaný tak, aby byl lehce použitelný a dal se snadno rozšířit o nové funkce (například fyzikální solver, nebo nové renderovací techniky). Při programování jsem se snažil maximálně využít možností jazyka C++. Jádro enginu vyvíjím již přes rok, poskytuje velké množství jednoduše použitelných featur, pro tvorbu jednoduchých ukázkových dem, je již dnes plně dostačující. Pro tvorbu rozsáhlé 3D aplikace by bylo zapotřebí ještě asi tak rok usilovného vývoje. Momentálně kód enginu tvoří asi 120 souborů a přes 30 000 řádků.

4.1 SDL

Obrázek 6 : Logo knihovny SDL

Simple DirectMedia Layer [9]. SDL je multiplatformní multimediální knihovna vytvořená tak aby poskytovala low level přístup k zvuku, klávesnici, myši, joystiku, 3D hardwaru skrze OpenGL a samozřejmě k vytvoření 2D framebufferu. SDL podporuje operační systémy Linux, Windows, BeOS, MacOS, FreeBSD, Atari, Solaris, Symbian a mnoho dalších. SDL je napsáno v jazyce C, ale samozřejmě se dá používat v kombinaci s C++. SDL má bindingy pro různé jazyky, například C#, Haskel, Java, Lisp, Lua, Pascal, Perl, PHP, Pyhon a další. SDL je distribuováno pod GNU LGPL 2 licencí. To znamená, že ji můžeme použít zdarma i pro komerční účely,ale musíme ji linkovat dynamicky.

4.2 OpenGL

Open graphics library. Podrobně viz kapitola 3 .

4.2.1 The OpenGL Shading Language

Shaderovací jazyk. Podrobně viz kapitola 3.3 .

4.3 Code::Blocks

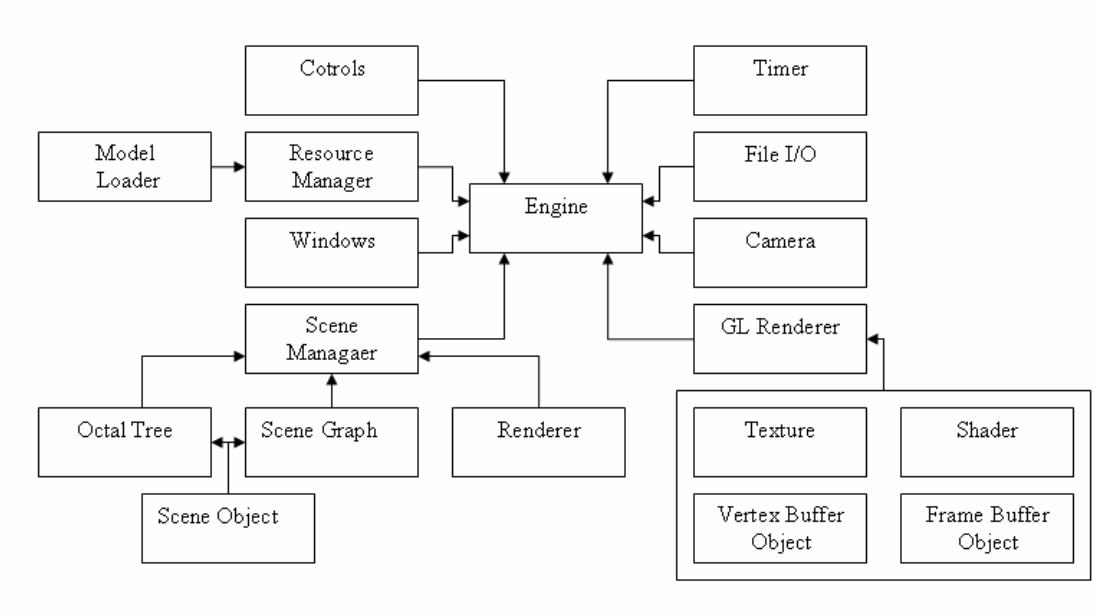
Obrázek 7 : Logo prostředí Code Blocks

Code::Blocks je IDE napsané v C++ s použitím knihovny wxWidgets. Je zcela zdarma. Je snadno rozšiřitelné a konfigurovatelné. Je uvolněno pod GPL 2 licencí. Funguje pod operačními systémy Linux a Windows. Umožňuje import projektu z microsoft visual studia a z dev-cpp. Podporuje také devpacky z dev-cpp. Umí pracovat s kompilerem GCC ale také s kompilerem od Microsoftu MSVC++. Obsahuje mnoho předdefinovaných šablon k tvorbě projektu (nastavení knihoven, include souboru…). Velmi snadno se s ním pracuje pod linuxem i windowsem. Editor podporuje zvýrazňování kódu, automatické doplňování kódu a prohlížení tříd.

Code::Blocks je neustále pod usilovným vývojem, k vývojem jsem používal vždy nejaktuálnější night-build.

5 STRUKTURA ENGINU

5.1 Class Diagram



Obrázek 8 : Diagram hlavních tříd enginu

Toto je hrubý nástin spolupráce jednotlivých tříd enginu. Není to zcela přesná reprezentace, jelikož většina tříd je naprogramovaná jako singleton, tudíž si lze vytvořit ukazatel na instanci třídy v podstatě kdekoliv. Základní třídou je třída CENGINE. Která obaluje všechny třídy. V době psaní této práce není použit žádny prostor jmen (namespace), protože engine nemá oficiální název.

5.2 Popis nejdůležitějších tříd

5.2.1 Vytvoření okna

CWINDOW umožňuje vytvořit okno, s renderovacím kontextem, nebo mu lze říct, že okno bylo vytvořeno, a engine bude využívat tento kontext. Lze vytvořit okno s libovolným rozlišením, například 452 x 755, nebo okno přes celou obrazovku s různým rozlišením. Bohužel SDL neumožňuje nastavit obnovovací frekvenci monitoru.

CCAMERA Umožňuje ovládání myší, pohyb pomocí kláves, spravuje také aktuální pohledový jehlan (frustum). Implementována pomocí funkce gluLookAt(vektor pozice, pohledovy vektor, up vektor).

5.2.2 Matematická část

CVERTEX popisuje jeden bod v prostoru, skládá se z tří prvků typu float. Pro snadnou práci má přetíženo mnoho operátorů. Engine obsahuje CVERTEX2F, CVERTEX3F, CVERTEX4F. Což jsou názvy tříd pro dvou, tří a čtyř - složkový vektor. Jsou implementovány základní potřebné funkce, jako normalizace vektoru, zjištění délky vektoru, zjištění úhlu mezi vektory a podobně.

CMATRIX4F reprezentuje matici. Skládá se z šestnácti prvků typu float. Má přetíženo mnoho operátorů. Používá se pro posunování objektů, otáčení, změnu velikosti, popřípadě typ promítání. Matice má metody na získání invertované matice a matice transponované.

CCOLLIDOR umožňuje zjišťování základních druhů kolizí, například dvou rovin, roviny a přímky, trojúhelníku a přímky, dvou trojúhelníků atd.

5.2.3 Načítání ze souborů, zápis do souborů

Třída CPARSER slouží k načítání z textových souborů, primárně byla napsána k načítání z souborů ASE. Tato třída poskytuje jednoduché rozhraní pro načítání, nicméně její implementace je poněkud svazující. Třída CLOG umožňuje textový zápis do souboru. Slouží k ukládání informací o průběhu načítání a inicializace enginu. Pomocí této třídy se vytváří losovací soubor o průběhu načítání a běhu programu. Engine také umožňuje načítání a zápis do souborů typu XML. Toto umožňuje díky knihovně TiniXml.

5.2.4 Modely

Načítání modelů umožňuje třída CMODEL_LOADER. Engine primárně podporuje načítání pouze z formátu ASE. To je ASCII obdoba formátu 3ds. Načítání je ovšem pomalejší. Při načítání modelů dochází ještě k různým optimalizacím v struktuře modelu, to má za následek rychlejší kreslení, ale také násobně delší dobu načítání, to se znatelně projeví hlavně při načítání modelu laboratoře, tento model má asi 50 MB. Třída popisující model se nazývá CMODEL, každý model se skládá z jednoho nebo více objektů. Vykreslovací

část enginu pracuje s těmito objekty, třída CMODEL slouží pouze k načítání a k určení které objekty spolu tvoří spojitou část. Objekt je popsán třídou COBJECT. Tato třída je potomek třídy CSCENE_OBJECT. COBJECT může obsahovat mesh (třída CMESH) a materiál (CMATERIAL). Mesh je datová struktura která obsahuje „tělo“ modelu, skládá se z vertexů. Některé enginy podporují různé vertex formáty pro meshe. Moje implementace meshe podporuje tuto strukturu : povinně vertexy, volitelně normály, volitelně tangenty, volitelně texturovací koordináty pro 2 mapovací kanály(v bakalářské práci nepoužito). Materiál je datová struktura popisující vzhled meshe.Obsahuje definice pro osvětlovací parametry, jako diffusní, ambientní a odrazovou složku světla. Dále může volitelně obsahovat několik druhů textur (diffusní, light mapu, normal mapu, height mapu).

5.2.5 Obálky

Takzvané boundingy. Engine podporuje Boudning Spere (BS) a Axes Aligned Bounding BOX (AABB). Slouží k obalení objektů ve scéně. Urychlují detekci viditelnosti a detekci kolizí. Každý objekt má položku CBOUNDING_SET, což je množina jednoduchých obálek (bud BS nebo AABB). Tím se dá dosáhnout velmi dobré aproximace tvaru modelu. Obálky mají implementovány metody na zjištění jestli se nachází v pohledovém jehlanu (frustum).

5.2.6 Světla

Třída CLIGHT reprezentuje světlo. Umožňuje nastavit prakticky všechny parametry které podporuje OpenGL. Pozici, směr, barvu, útlum, reflektorový tvar. Engine předává parametry osvětlení do shaderu pomocí opengl parametrů. Proto nelze najednou aktivovat více než 8 světel, v této práci je ale aktivní vždy pouze jedno světlo. Světlo je potomek třídy CSCENE_OBJECT, proto s ním lze zacházet jako z každým jiným objektem ve scéně.

5.2.7 Textury

K načítání textur engine používá knihovnu SDL_IMAGE. Tato knihovna podporuje načítání nejrůznějších formátů (BMP, XPM, PCX, JPG, PNG, TIF a další). Třída CTEXTURE umožňuje buď načíst texturu ze souboru nebo vytvořit texturu s pomocí parametrického konstruktoru. Engine podporuje vytváření mipmapovaných textur, anisotropního filtrování

textur, vytváření textur o rozměrech které nejsou mocninou čísla dvě (pokud to umožňuje HW), dále vytváření float textur a komprimovaných textur.

5.2.8 Shadery

Engine podporuje načítání glsl shaderů. Třída CSHADER. Umožňuje jednoduché aplikování shaderů, a nastavování uniformních proměnných. Shadery jsou načítány z textových souborů, zvlášť pixel a zvlášť vertex shader. Lze také načíst a aplikovat pouze jeden z nich.

5.2.9 Effekty

Třída CEFFECT má za úkol usnadnit vytváření komplexních efektů. Například vytvoření efektu bump mapping docílíme tak že do virtuální funkce Bind(), napíšeme kód který vybere texturu normal mapy, aktivuje příslušný shader, a nastaví příslušné parametry pro shader. Tato třída je jakési materiálové rozšíření. Z této třídy se dědí třída CPOST_EFECT. Tato třída popisuje takzvané post proces efekty, tyto efekty se aplikují jakoby až po vykreslení celé scény (HDR, BLOOM), ve skutečnosti je to ale trochu jinak. Na demonstračním programu je vidět použití efektu bloom.

5.2.10 Text

K vypisování textu používám knihovnu freetype. K ulehčení práce s freetype jsem použil zdrojový kód ze stránky www.nehe.ceskehry.cz. Engine umožňuje načítání ttf fontů, výpis na různých pozicích obrazovky, různé barvy textu a průhledný text.

5.2.11 Representace scény

Objekty ve scéně jsou uspořádány dvěma způsoby. První uspořádání je „logické“. To znamená například že miska položená na stole, změní svoji pozici pokud pohnu stolem. Toto logické uspořádání obstarává třída CSCENE_GRAPH. Scéna se skládá z uzlů (třída CSCENE_GRAPH_NODE). Každý uzel může obsahovat různý objekt poděděný z třídy CSCENE_OBJECT (například COBJECT, CLIGHT…). Každý uzel navíc může obsahovat různý počet uzlů, které jsou hierarchicky „pod ním“. To znamená že metody volané na nadřazené uzly se provedou i pro všechny uzly podřazené. Druhý typ representace, je representace prostorová, tu obstarává třída C_OCTREE. Je to implementace oktávového

stromu. Nejdříve se definuje velikost prostoru, který bude tato třída spravovat. Po té do oktávového stromu vkládají objekty. Třída sama vytváří nové uzly. Oproti jiným implementacím nemusí být všechny kreslené objekty v leaf uzlech. Oktávový strom urychluje frustum culling a testování kolizí. Při práci s objekty ve scéně tedy nevoláme přímo metody objektů ale metody uzlu. Díky tomu se například po transformaci uzlu z jednoho rohu do druhého automaticky aktualizuje pozice uzlu i v oktávovém stromu.

5.2.12 Vykreslovací Pipe - line enginu

K vykreslování slouží 2 třídy. Třída CGL_RENDERER vytváří abstrakci nad funkcemi OpenGL. Ulehčuje nastavování parametrů pro vykreslování, pro shadery, světla, texturování… Nastavování parametrů vykreslování musí probíhat právě přes tuto třídu, jinak nelze zaručit správnou funkcionalitu. Třída si drží hodnoty nastavení různých parametrů (blending, ligting, culling…), díky tomu se dá v určitých situacích předejít zbytečné změně stavového nastavení vykreslovacího serveru (grafické karty). Průběh vykreslení je asi následovný. Třída CSCENE_MANAGER si nechá zjistit od třídy COCC_TREE, které objekty jsou v zorném poli. Poté z těchto objektů získá všechny další vykreslovací cíle (render target, například odraz na vodné hladině). Dojde k zpracování těchto úkolů. Toto by se dalo nazvat aktualizací textur. Pokud je aktivní nějaký post effect, aktivujeme ho, nyní dojde k vykreslení všech objektů které tvoří scénu.

5.3 Popis vybraných použitých technik

5.3.1 Frustum Culling

Je to optimalizační technika která největší měrou zvyšuje výkon rozsáhlejších aplikací. Principem je to, že se z vykreslování vyřadí objekty které nejsou v pohledovém jehlanu. Pokud aspoň jeden bod objektu je v pohledu, objekt se vykreslí. Protože složité objekty mohou mít třeba sto tisíc bodů, a každý by se musel otestovat, obalují se objekty do tzv. obálek. Obálka je jednoduchý geometrický útvar, který aproximuje tvar objektu. Nejpoužívanější obálky jsou bounding spere (BS), axis aligned bounding box (AABB) a object oriented bounding box (OOBB).

BS – nejednodušší, charakterizuje ji střed koule a poloměr koule. Výhodou je, že odpadá problém s řešením rotace objektu, nevýhodou je že ve většině případů je aproximace tvaru objektu velmi špatná. Test na viditelnost je nejrychlejší ze všech typů obálek.

AABB – je popsán středem a velikostí os. Dosahuje relativně lepší aproximace tvaru objektu než BS, ovšem při rotování objektu se musí přepočítat. Testování na viditelnost se provádí jako test osmi rohových bodů proti všem rovinám pohledového jehlanu.

OOBB – v bakalářské práci není použito. Definuje se středem, velikostí os, a vektorem rotace. Při rotaci objektu se otáčí s objektem. Test na viditelnost je stejný jako u AABB.

Frustum culling se dá vypnout a zapnout z třídy CGL_RENDERER.

5.3.2 Scene Graph

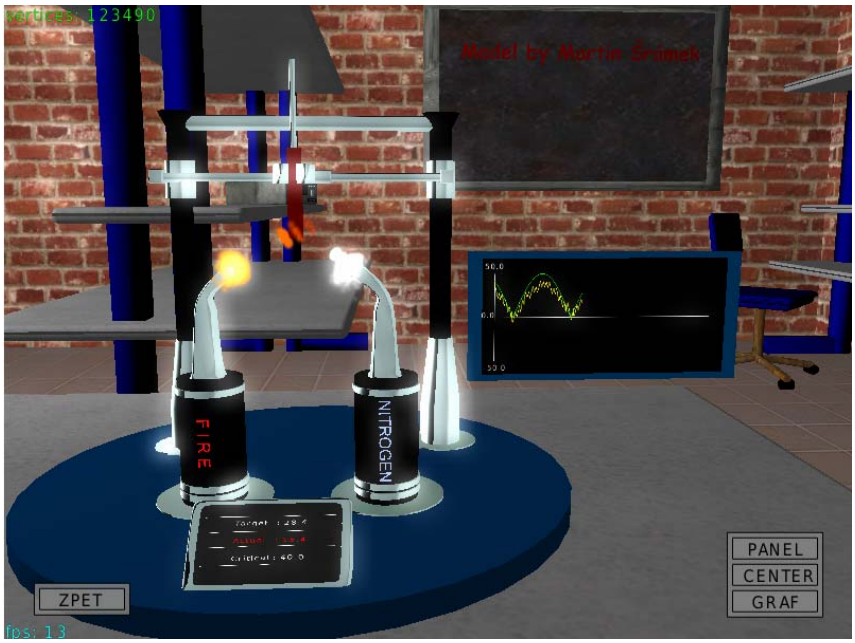
Jedná se o datovou strukturu, která se používá k strukturovanému ukládání objektů. Jednotlivé implementace se mohou lišit. Způsob ukládání objektů se velmi podobá struktuře stromu, tzn. že ve scene graphu jsou objekty uloženy od kořenu k listům. Provedeme-li nějakou operaci s kořenem (například změna pozice), provede se změna na všechny jeho poduzly. Scene graph tedy neslouží k zefektivnění vykreslování ale k docílení logického chování objektů.

5.3.3 Render Targets

OpenGL stejně jako Direct3D umožňuje „přesměrovat“ grafický výstup z back bufferu, do jiné části paměti. Toto se používá k vytvoření nejrůznějších efektů, například k vytvoření vodní hladiny s podporou odrazu objektů nad hladinou a k rozmazání objektů pod hladinou, nebo k vytvoření post process efektů, jako motion blur (rozmazání pohybu), depth of field (rozmazání objektů které neleží v ohniskové vzdálenosti), nebo třeba HDR [3], [4], [7], [8] efektu. Získat obraz lze dvěma způsoby. Buď kreslit do back bufferu a poté zkopírovat obraz do textury, nebo využít možnosti přesměrovat výstup přímo do textury, čímž se ušetří čas strávený při kopírování z back bufferu. Já jsem použil druhý způsob, k tomu je ale nutné aby grafická karta podporovala opengl rozšíření frame bufferu object.

6 POPIS SCHÉMAT

6.1 Regulace teploty



Obrázek 9 : Regulace teploty

Schéma zobrazuje regulaci teploty čidla podle zadaného průběhu. Zapojení se skládá z teplotního čidla, ohřívacího ventilu, chladícího ventilu, displeje, a grafu zaznamenávajícího průběhy teplot.

6.1.1 Implementace regulátoru

Regulátor je implementován jako dvoupolohový regulátor s penalizací. To znamená že systém může nabývat pouze dvou stavů. Ohřev nebo chlazení. Tento regulátor má pro prezentaci naprosto dostačující výsledky a není složitý na implementaci. Původní záměr ovšem byl PID [2] regulátor.

6.1.2 Implementace trysek

Obě trysky jsou implementovány stejně. Jedná se o částicové systémy. Částicové systémy jsou seskupení bodů, kde každý bod má svou pozici, směr, rychlost. Každý bod lze různě ovlivňovat, například „větrem“, gravitací… Částice jsou vykreslovány pomocí point spritu, tzn. že na každou částici připadá pouze jeden vertex. Nevýhody plynoucí z použití tohoto způsobu se zde vůbec neprojevují. Výkon každé trysky je řízen hodnotami z regulátoru.

6.1.3 Implementace displeje a grafu

Oba tyto prvky jsou implementovány velmi podobně. Oba objekty mají materiál, na který je přesměrován grafický výstup. Display je aktualizován v každém snímku, graf je aktualizován po uplynutí zadané doby.

Obrázek 10 : displej s aktuálními hodnotami

6.1.4 Popis funkce

Účelem schématu je zobrazit regulaci teploty čidla. Regulace probíhá pomocí dvou trysek, plamenu a dusíku. Průběh regulace je vidět na grafu. Graf zobrazuje průběh žádané a aktuální teploty. Panel ve spodní části zobrazuje číselné hodnoty žádané, aktuální a kritické teploty. Pokud teplota čidla dosáhne kritické hodnoty, dojde k ukončení regulace. Stisknutím mezerníku vyvoláme menu, v něm lze nastavovat tyto parametry :

- perioda vzorkování
- žádanou hodnotu
- kritickou hodnotu
- penalizaci regulátoru

Lze také restartovat regulační pochod, popřípadě vyexportovat data z regulátoru do textového souboru.

6.2 Regulace výšky vodní hladiny

6.2.1 Implementace regulátoru

Regulace je u tohoto schématu velice jednoduchá. Do nádržky neustále přitéká voda. Po dosažení nastavené hladiny se přítok zastaví, a dojde k postupnému vypuštění. Po vypuštění se opět aktivuje přítok vody a celý cyklus se opakuje.

6.2.2 Implementace přítoku vody

Přítok je implementován podobně jako trysky v předchozím schématu. Je to opět částicové systém, ale s trochu jinými parametry. Největší rozdíl je v tom že se nepoužívá bodový emitor částic, ale emitor plošný, tzn. že částice nejsou generovány z jednoho bodu, ale z určité vymezené plochy.

6.2.3 Implementace displeje

Naprosto shodné s implementací displeje z předchozího schématu.

6.2.4 Implementace vodní hladina

Třída popisující vodní hladinu plně těží z objektového návrhu celého enginu. Třída CWATER je potomek třídy z COBJECT. Díky tomu má třída CWATER atributy CMESH a CMATERIAL, ty jsou nahrazeny adekvátně poděděnými třídami CWATER_MESH a CWATER_MATERIAL. CWATER_MESH má navíc oproti standardní třídě popisující mesh navíc pole, kam se ukládají vektory reprezentující „ vektory “ vodní hladiny. Má také přetíženou metodu Update(float delay), která obstarává vlnění vodní hladiny. Třída CWATER_MATERIAL obstarává aplikování shaderu a textur a barvy vody.

K vytvoření odrazu a lomu na vodní hladině (reflection / refraction) se používají dva render targets (viz. kapitola o Render targets). Jeden zachytává tělesa pod vodou a druhý tělesa nad vodou. Tyto textury jsou pak míchány v shaderu a tak vzniká efekt realistické vodní hladiny.

7 UKÁZKY KÓDU

7.1 Textury

7.1.1 Kód načte texturu a přidá ji do všeobecných zdrojů :

```
CTEXTURE *Tex;
Tex  = new CTEXTURE;
Tex->FileName = "../Data/Pics/bump.jpg";
Tex->Load();
Tex->Name = "NormalMap";
Engine->ResourceManager->AddTexture(Tex);
```

7.1.2 S detailnějším nastavením :

```
Tex = new CTEXTURE;
Tex->FileName = "../Data/Pics/aniso.tga";
Tex->Load(GL_LINEAR, true, true,  // filtrování , mipmaping, komprese
          10.0, GL_UNSIGNED_BYTE, // anisotropní filtrování, pixel
          GL_CLAMP_TO_EDGE, GL_CLAMP_TO_EDGE); // clampování
Tex->Name = "AnisoTex";
Engine->ResourceManager->AddTexture(Tex);
```

7.1.3 Připravení prázdné float textury :

```
Tex = new CTEXTURE(1024, 1024, // rozmery
                   RGB_FLOAT, // interní formát
                   RGB_FLOAT, // formát
                   false,     // mipmaping
                   0.0, NULL, // anisotropní filtrování,ukazatel na data
                   GL_NEAREST, GL_TEXTURE_2D, // filtrování, typ textury
                   GL_FLOAT, // typ pixelu
                   GL_CLAMP_TO_EDGE, GL_CLAMP_TO_EDGE); // clampování
```

7.2 Shadery

7.2.1 Načtení shaderu ze souboru.

```
CSHADER * Shader;
Shader = new CSHADER;
Shader->Name = "Water"; // jméno shaderu
Shader->Load("../Data/Shaders/Water/water.frag",  // pixel shader
"../Data/Shaders/Water/water.vert"); // vertex shader
Engine->ResourceManager->AddShader(Shader);
```

7.3 Modely

7.3.1 Načtení ASE modelu ze souboru

```
    CMODEL * Model;
    CLOAD_PARAM Param;
    float a = this->Engine->GLRenderer->GetMaxAnisotropi();// anisotropní
                                                           // filtrování
    Param.Texture_Anisotropy = a; // maximální možné
    Param.MatrixTransform = true; // požít matice z modelu
    Param.ComputeTangents = false; // nepočítat tangenty
    Param.From3dMax = true; // export z 3D max studia
    Param.CreateBuffer = true; // vytořit VBO na graf. kartě
    Param.NormalsFlag = PER_FACE_NORMALS; // příznak normálových vektorů
    Param.RelativeTexturePath = true; // textury v adresáři s modelem
    Param.FileName = "../Data/Models/Tabule/tabule.ASE"; // cesta modelu
    Model = Engine->ResourceManager->LoadModel(Param); // načtení
    if (Model)
        Model->Name = "Tabule"; // přiřazení jména
```

ZÁVĚR

Cílem práce bylo seznámit se s programy Control Web a InTouch, zhodnotit jejich vlastnosti a vhodnost použití. Dále popsat knihovnu OpenGL a ostatní knihovny vhodné pro použití při tvorbě vizualizačního programu. Na základě těchto znalostí vytvořit vizualizační prostředí a předvést jeho funkci na konkrétním modelu. Pro tyto modely vybrat vhodné řídící algoritmy.

Programy Control Web a InTouch jsou popsány v kapitole 1, respektive 2. Obecně se dá říci, že obě aplikace jsou ve svém oboru na špičce. Jsou to velmi komplexní programy s širokým záběrem použití.

OpenGL je charakterizováno v kapitole 3, další použitá knihovna je SDL, ta je popsána v kapitole 4.1. Knihovnu SDL jsem použil k vytvoření okna a k obsluze klávesnice a myši. OpenGL slouží pouze pro vykreslování. V aplikaci jsem se snažil maximálně využít prostředků, které OpenGL poskytuje. Další použité knihovny jsou SDL_IMAGE, FREETYPE a GLEW.

Samotný program vznikl úpravou zdrojových kódu, na kterých jsem pracoval od začátku studia na UTB. Je psaný v jazyce C++. Snažil jsem se maximálně využít objektového přístupu k programování. Vzhledem k použitým knihovnám je aplikace multiplatformní, přiloženy jsou binární soubory pro operační systémy Windows a Linux. Vykreslovací jádro jsem se snažil optimalizovat tak, aby bylo schopné co nejvíce využít schopnosti moderních grafických karet.

Regulační pochody jsou pouze ukázkové, jsou použity pouze jednoduché typy regulátorů.

Z této aplikace by se ale zcela jistě dalo vycházet při tvorbě komplexního programu. Bylo by vhodné doplnit ho o IDE pro jednoduchou tvorbu scény. Dále pak o kód, který by uměl tyto scény načítat a ukládat do souboru. Nutné by také bylo vytvořit podporu pro složitější typy regulátorů.

RESULT

Ambition of my work was to familiarize with programms Control Web and InTouch, review their properties and usage. Then describe library OpenGL and other libraries suitable for creation render programme. With this knowledge create visualization system and demonstrate his function on specific example.

Programms Control Web and InTouch are described in chapter 1 and 2. Generally we can say that both aplications are on the top in their specialization. Both of them are very complex programs, with miscellaneous usage.

OpenGL is characterized in chapter 3, another used library is SDL. SDL is described in chapter 4.1. I had use SDL to create window and handling keyboard and mouse. Opengl is used only to rendering. I had try to use maximum of capabilities which OpenGL provides. Other used libraries are SDL_IMAGE, FREETYPE and GLEW.

Programme was created from source codes, which I made since the start of studies on UTB. Programme is written in C++. I had try to use maximum of object method of programming. In regard of used libraries is aplication cross-platform, provided are binary files for operating systems Windows and Linux. I had try to optimalize render core so it can use ability of modern graphics cards. Regulation was made by simple regulator.

This aplication could be a base of more complex programme. It would be good add IDE to simplify creation scenes. Another important thing is ability to load and save scene in data file. Necessarily is also support for complex regulator types.

SEZNAM POUŽITÉ LITERATURY

[1] SHREINER, D. a kol.: OpenGL – průvodce programátora. Computer press,2006

[2] BALÁTĚ, J.: Vybrané statě z automatického řízení. Skripta FT, VUT Brno 1996

[3] *OPENGL – The Industry standard for High Performance Graphics* [online]. 2007, [cit. 2007-05-17]. Dostupné z WWW: <http://www.opengl.org>

[4] *GPGPU – General Purpose Computation Using Graphics Hardware* [online]. 2007, [cit. 2007-05-17]. Dostupné z WWW: <http://www.gpgpu.org>>

[5] TIŠŇOVSKÝ, P. *Grafická knihovna OpenGL* [online]. 2005, [cit. 2007-05-17]. Dostupné z WWW: <http://www.root.cz/clanky/opengl-33-souhrn-temat/>

[6] PORTÁL ČESKÉ HRY – komunita českých herních vývojářů [online]. 2007, [cit. 2007-05,17] . Dostuplné z WWW : <http://www.ceske-hry.cz/forum/>

[7] *NVIDIA CORPORATION – Software developer Kit* [online]. 2007, [cit. 2007-05-17]. Dostupné z WWW: <http://developer.nvidia.com>

[8] *ATI TECHNOLOGIES – Developer pages* [online]. 2007, [cit. 2007-05-17]. Dostupné z WWW: < http://www.ati.com/developer>

[9] *SDL – cross-platform library* [online]. 2007, [cit. 2007-05-17]. Dostupné z WWW: < http://www.libsdl.org>

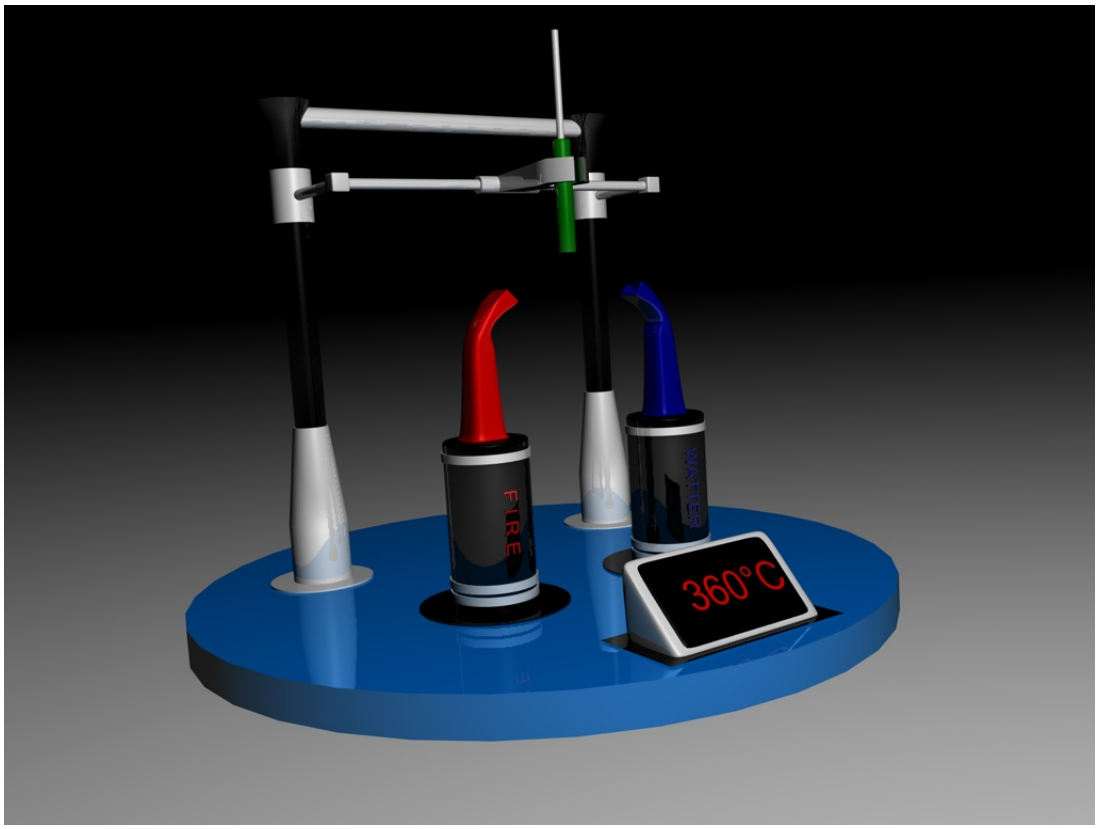
SE**ZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK**

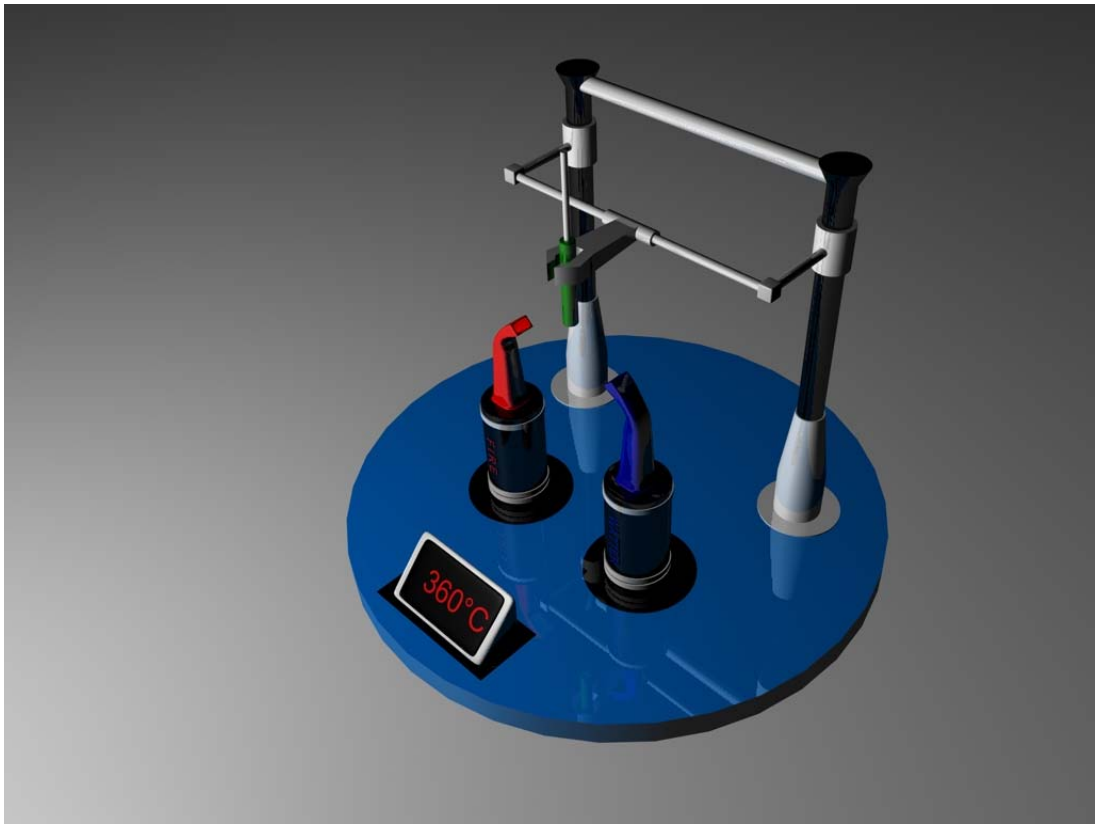
OGL	**O**pen **G**raphics **L**ibrary. Knihovna určená k 2D/3D vykreslování.
SDL	**S**imple **D**irectMedia **L**ayer. Multimediální knihovna.
GLSL	The Open**GL** **S**hading **L**anguage. Jazyk pro programovatelné grafické karty.
VERTEX	Datová struktura popisující jeden bod v prostoru.
SHADER	Kód, který se vykonává na grafické kartě.
MESH	Síť vertexů tvořící model.
ENGINE	Jádro programu.
FRAME-BUFFER	Paměťový prostor, místo do kterého vykreslujeme.
API	**A**pplication **P**rogram **I**nterface. Aplikační programové rozhraní.
Pixel	**P**icture **El**ement. Základní jednotka rastrového obrazu.

SEZNAM OBRÁZKŮ

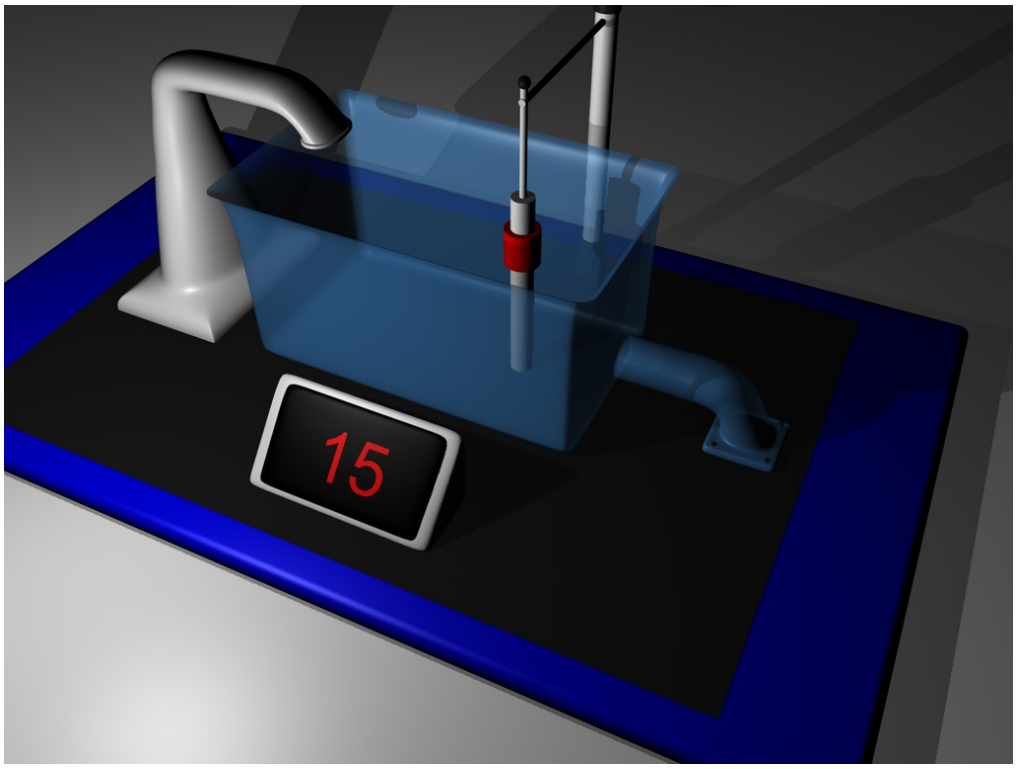
SEZNAM PŘÍLOH

PŘÍLOHA P I: MODEL PRO REGULACI TEPLOTY





PŘÍLOHA P II: MODEL PRO REGULACI HLADINY



PŘÍLOHA P III: UKÁZKY Z PROGRAMU