

UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

Vývoj herních aplikací s využitím OpenGL

Antonín Kühtreiber

Bakalářská práce

2012

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Antonín Kühntreiber**
Osobní číslo: **I08098**
Studijní program: **B2646 Informační technologie**
Studijní obor: **Informační technologie**
Název tématu: **Vývoj herních aplikací s využitím OpenGL**
Zadávající katedra: **Katedra informačních technologií**

Z á s a d y p r o v y p r a c o v á n í :

Bakalářská práce se zabývá problematikou tvorby her typu vesmírný simulátor s využitím multiplatformního standardu OpenGL.

V teoretické části budou popsány vybrané technologie, které se v této oblasti používají. Součástí teoretické části bude i základní popis základních principů a použití knihovny OpenGL. Cílem praktické části je návrh a implementace vlastní herní aplikace založené na knihovně OpenGL. K realizaci hry bude vytvořen zjednodušený fyzikální model pro pohyb těles v trojrozměrném beztlakém prostoru a jednoduchá umělá inteligence pro kontrolu říditelných objektů. Implementační část bude realizována pomocí programovacího jazyka C++.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

1. **SHREINER, Dave, et al. OpenGL : Průvodce programátora. Vydání první. Brno : Computer Press, 2006. 679 s. ISBN 80-251-1275-6.**
2. **PRATA, Stephen. Mistrovství v C++. 3., aktualiz. vyd. Brno : Computer Press, a.s., 2007. 1119 s. ISBN 978-80-251-1749-1.**

Vedoucí bakalářské práce:

Ing. Petr Veselý

Katedra softwarových technologií

Datum zadání bakalářské práce: **16. prosince 2011**

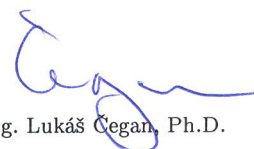
Termín odevzdání bakalářské práce: **11. května 2012**



prof. Ing. Simeon Karamazov, Dr.
děkan



L.S.



Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 30. března 2012

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 14. 08. 2012

Antonín Kuhtreiber

Poděkování

Touto cestou bych rád poděkoval všem, kteří mě při tvorbě této práce podporovali, nebo mi jakkoliv pomohli pak vedoucímu práce Ing. Veselému Petrovi za cenné rady a připomínky v průběhu realizace této práce. Dále bych rád poděkoval mé rodině a přátelům za podporu během studia.

Anotace

Práce se zabývá problematikou vývoje počítačových her, konkrétně typu vesmírný simulátor. Teoretická část se zabývá primárně seznámením s multiplatformní knihovnou OpenGL a knihoven používaných často ve spolupráci s OpenGL, ale i některými dalšími knihovnami a technologiemi užívanými v profesionální sféře při vývoji počítačových her. Praktická část se pak zabývá samotným vývojem herní aplikace v programovacím jazyce C++ s využitím OpenGL a implementací fyziky pohybu těles v beztlžném vzduchoprázdném prostoru.

Klíčová slova

OpenGL, SDL, 3D, vesmírný simulátor, hra.

Title

Development of game applications by using OpenGL

Annotation

This thesis deals with the development of computer games. It specifically deals with space simulator. The theoretical part basically concentrates on the multiplatform library OpenGL, libraries that are usually used in conjunction with OpenGL, other libraries and the technology used in the professional field in the development of computer games. The practical part focuses on the development of game applications in the programming language C++ using OpenGL. It further focuses on the implementation of physics of motion of bodies in the weightless (free fall) vacuum space.

Keywords

OpenGL, SDL, 3D, space simulator, game

Obsah

Seznam zkratek	8
Seznam obrázků	9
Seznam tabulek	9
1. Úvod.....	10
2. OpenGL.....	11
2.1. Alternativy OpenGL.....	11
2.2. Srovnání OpenGL a DirectX.....	11
2.3. Design OpenGL.....	12
2.3.1.Stavový automat	12
2.3.2.OpenGL Pipeline.....	12
2.4. Použití OpenGL.....	13
2.4.1.Vnitřní datové typy	14
2.4.2.Syntaxe příkazů OpenGL.....	14
2.4.3.Display list	14
2.4.4.Buffery	14
- Buffer barev	14
- Buffer hloubky.....	14
- Double buffering.....	15
2.4.5.Grafická primitiva	15
2.4.6.Transformace.....	16
- Pohledová a Modelová transformace.....	16
- Projekční transformace	17
- Zobrazovací transformace.....	18
2.4.7.Barva	18
- Mod RGBA.....	19
- Mód indexové barvy	19
2.4.8.Osvětlení	20
- Složky světla.....	20

- Model osvětlení.....	20
- Textury.....	21
3. Ostatní použité knihovny.....	22
3.1. GLU.....	22
3.2. GLUT	22
3.3. SDL.....	22
3.4. OpenAL.....	22
4. Technologie, knihovny užívané při vývoji her.....	24
4.1. Shader.....	24
4.2. OpenCL	24
4.3. Bullet Physic.....	25
5. Nástroje užitá při realizaci praktické části.....	26
5.1. Xcode.....	26
5.2. Cheetah3D.....	27
6. Realizace praktické části - logická část.....	28
6.1. Návrh fyzikální simulace prostoru	28
6.1.1. Analýza fyzikálních zákonů k implementaci	28
- Klasická mechanika.....	28
- Relativistická mechanika.....	29
- Analýza.....	29
6.1.2. Uchování informací o prostoru	29
6.1.3. Uchování informací o tělesech.....	30
6.1.4. Pohyb těles	31
6.1.5. Detekce kolize.....	32
6.1.6. Reakce na kolizi	32
6.1.7. Třída říditelných těles	33
6.1.8. Třída pozorovatele	33
6.2. Problematika návrhu umělé inteligence	33
6.2.1. Herní umělá inteligence	34
6.2.2. Metody tvorby herní AI.....	34
6.2.3. Realizace herní AI.....	34

7. Realizace praktické části grafická a akustická část.....	36
7.1. Vytvoření a zpráva grafického okna aplikace.....	36
7.2. Zobrazení stavu	37
7.3. Načtení trojrozměrných těles	37
7.3.1.Zobrazení trojrozměrných těles	38
7.3.2.Přehrávání zvuku.....	39
Závěr.....	41
Literatura.....	42
Příloha A - Přiložené CD	44

Seznam zkratek

OpenGL	Open Graphics Library, otevřená grafická knihovna
API	Application Programming Interface, rozhraní pro programování aplikací
GLUT	The OpenGL Utility Toolkit - sada pomocných nástrojů pro OpenGL
OpenAL	Open Audio Library, otevřená audio knihovna
ARB	The OpenGL Architecture Review Board,
OpenGL ES	OpenGL for Embedded Systems, OpenGL pro vestavěné systémy
GLU	The OpenGL Utility Library, dodatečná knihovna nástrojů pro OpenGL
AI	Artificial Intelligence, umělá inteligence
GPU	Graphic Processing Unit, Grafický procesor
GPGPU	General-purpose computing on graphics processing units, víceúčelové počítání na grafických procesorech
SDL	Simple Directmedia Layer, knihovna pro multimediální software
IDE	Integrated Development Environment
GNU	GNU's Not Unix! GNU Není Linux! Projekt zaměřený na svobodný software
GCC	GNU Compiler Collection, kolekce kompilátorů
LLVM	LLVM Low Level Virtual Machine
OpenCL	Open Computing Language, průmyslový standard pro paralelní programování
c	Rychlost světla

Seznam obrázků

Obrazek 1 - OpenGL pipeline diagram,by Bedwyr, CC	13
Obrazek 2 - Ortografická projekce, by Dave Pape, CC	17
Obrazek 3 - Perspektivní projekce, by Dave Pape, CC	18
Obrazek 4 - RGB barvy, by Dave Pape, CC	19
Obrazek 5 - xCode IDE	26
Obrazek 6 - Cheetah3D	27
Obrazek 7 - UML Class diagram třídy prostoru	31
Obrazek 8 - Vzorec rychlost po nárazu těleso 1, by Wikipedia, CC	32
Obrazek 9 - Vzorec rychlost po nárazu těleso 2, by Wikipedia, CC	32
Obrazek 10 - SDL okno	36
Obrazek 11 - UML class diagram, Model a pomocné třídy	37
Obrazek 12 - SDL okno s vykresleným modelem	39

Seznam tabulek

Tabulka 1 - Grafická primitiva	16
--------------------------------------	----

1 Úvod

Téma jsem si vybral z důvodu dlouhodobého plánování, kdy jsem již několik let měl v plánu vytvořit podobný projekt. Hlavním přínosem této bakalářské práce je rozšíření osobních znalostí v oboru počítačové 3D grafiky, modelování a hlubší pochopení knihoven OpenGL používaných v profesionální sféře herních vývojářů, s jejichž pomocí je vytvářena samotná praktická část v podobě hry typu vesmírný simulátor.

Bakalářská práce se dělí na dvě části. První, teoretická, se snaží povrchně seznámit s technologiemi, které by mohly být použity pro vývoj herních aplikací, tedy převážně s různými knihovnami a přístupy k tomu užívanými. A to jak knihovnami pro načtení multimediálních dat, kde se široce využívá knihovna SDL a k ní přidružené knihovny, přes knihovny určené ke zpracování zvuků, až po samotné OpenGL, jenž je nejdůležitější složkou teoretické části této práce.

V praktické části se nachází popis nástrojů, které byly využity pro vývoj herní aplikace typu vesmírný simulátor. Je nastoleno několik dílčích problémů, se kterými jsem se musel při vývoji aplikace vypořádat a pak následný popis vybraných klíčových součástí aplikace.

2 OpenGL

OpenGL API je široce podporovaným průmyslovým standardem v oblasti 2D a 3D počítačové grafiky. Jeho úkolem je poskytnout rozhraní mezi programátorem a grafickým hardware a usnadnit vytvoření trojrozměrných aplikací, jako jsou například hry či CAD programy. (1) (2) (3)

Jeho největší výhodou je multiplatformnost umožňující snadný přesun grafických aplikací mezi různými počítačovými platformami. Některé implementace umožňují i vykreslování na jiném zařízení, než na kterém je prováděn výpočet scény. (1) (2) (3)

OpenGL bylo poprvé uvedeno ve verzi 1.0 roku 1992 jako otevřená alternativa k IrisGL. V tomtéž roce bylo zformováno nezávislé konsorcium ARB odpovídající za řízení budoucnosti a vývoje OpenGL. Roku 2000 řada IT společností včetně Intel, NVIDIA, ATI a SGI založila organizaci jménem Khronos Group zabývající se vytvářením otevřených standardních API. V září roku 2006 bylo ARB včleněno do této organizace jako pracovní skupina pro OpenGL a tím se odpovědnost za řízení vývoje přenesla na Khronos Group. Do dnešní doby bylo uvolněno mnoho nových verzí až do aktuální 4.2. Vedle OpenGL existuje i od OpenGL odvozená varianta pro vestavěné systémy OpenGL|ES. (1) (2) (3)

2.1 Alternativy OpenGL

V knihovnách pro vykreslování grafiky není mnoho alternativ. Fakticky jedinou významně rozšířenou, pokud nemluvíme o OpenGL|ES, jež je odvozeno od OpenGL, je DirectX od společnosti Microsoft. V zájmu objektivitě bude následujících několik odstavců věnováno právě DirectX.

Microsoft DirectX je množina rozhraní mezi hardwarem a softwarem vyvíjené společností Microsoft pro operační systém Windows a herní konzole Xbox a mobilní telefony. První verze byla uvolněna 30. 9. 1995.

DirectX zastřešuje řadu API, pro porovnání s OpenGL jsou podstatné pouze DirectX Graphics, neboť OpenGL je čistě grafické API. DirectX graphics se skládá z dílčích částí pro 2D grafiku - DirectDraw, 3D grafiku - Direct3D a dalších. Funkce, které poskytuje DirectX a které nejsou obsaženy v čistě grafickém OpenGL, jsou dodávány nezávislými API, vyvíjenými třetími stranami. Jako první příklad poslouží OpenAL, dnes udržovaný a vyvíjený společností Creative Technology za podpory společnosti Apple, jež se staví proti DirectSound. Další částí DirectX, na níž budou zmíněny alternativy, je DirectCompute sloužící ke GPGPU výpočtům. Mezi alternativy patří OpenCL spadající pod KHRONOS Group, jež umožňuje využít jakékoliv kompatibilní zařízení pro výpočty, tedy nejen grafické karty. Dalším zástupcem je proprietární CUDA společnosti nVidia.

2.2 Srovnání OpenGL a DirectX

Obě API poskytují v principu stejnou funkcionalitu, nedá se proto říci, které je horší a které lepší, základním důvodem je tedy programátorské hledisko, neboť obě API se v něm diametrálně liší. Zatímco OpenGL je zástupcem strukturovaného programování, čímž je zajišťována jeho kompatibilita, mimo jiné i s jazykem C, DirectX je objektivě

orientovaný. Dalším zajímavým parametrem pro srovnání může být délka kódu při stejném výsledku. (7): „John Carmack, autor her Quake, uvádí, že kód programu plnící stejnou funkci je v Direct3D i čtyřikrát delší než v OpenGL.“ Přes tyto rozdíly však stále zůstávají hlavní rozdíly v podobě Multiplatformnosti na straně OpenGL, jenž je možno používat na mnoha platformách, zatímco DirectX je vázáno pouze na řešení společnosti Microsoft. Z tohoto důvodu bylo rozhodnuto využít OpenGL. (6)(7)

2.3 Design OpenGL

Knihovna OpenGL je navržena určitým způsobem. Obecně se sice nedá říci, jaké postupy byly použity pro implementaci, neboť vždy záleží na konkrétní implementaci. Metody využití ovšem zůstávají stejné. Z toho důvodu nejprve bude popsán obecný přístup k vlastnostem vykreslovaných scény a nejčastěji využívaná pipeline napříč různými implementacemi OpenGL.(1)

2.3.1 Stavový automat

OpenGL je definováno jako stavový automat. Z toho vyplývá i postup práce s ním. Postupně jsou zadávány vlastnosti (jako například transformace, barva, a podobně), které zůstávají platné až do okamžiku jejich změny. Výhodami tohoto principu je menší množství vstupních parametrů pro vykreslovací funkce a fakt, že jsou vždy měněny vlastnosti všech později vykreslených částí scény. Tedy pokud je změněn stav před zahájením vykreslování scény, dojde k ovlivnění celé scény.(1)

2.3.2 OpenGL Pipeline

Různé implementace OpenGL se mohou lišit v pořadí operací vykonávaných nad daty. Většina implementací však využívá jednu posloupnost, která se nazývá OpenGL Pipeline. Obrázek ilustruje pořadí operací v této pipeline. (2)

Display List: Zorazovací seznam umožňuje případné uložení všech dat, jak pro okamžité, tak i pozdější použití. Pokud ovšem pracujeme v módu okamžitého zpracování, je vynechán. V případě užití zobrazovacího seznamu se data v něm uložená zpracují totožně, jako při okamžitém zpracování.(2)

Evaluator: Všechny vykreslované objekty je potřeba mít specifikované pouze jako body pro další zpracování. Nyní jsou, ale popsány různými parametry a polynomy. V této fázi zpracování jsou ze vstupních dat vypočteny souřadnice bodů, normálové vektory, souřadnice textur a barvy. (2)

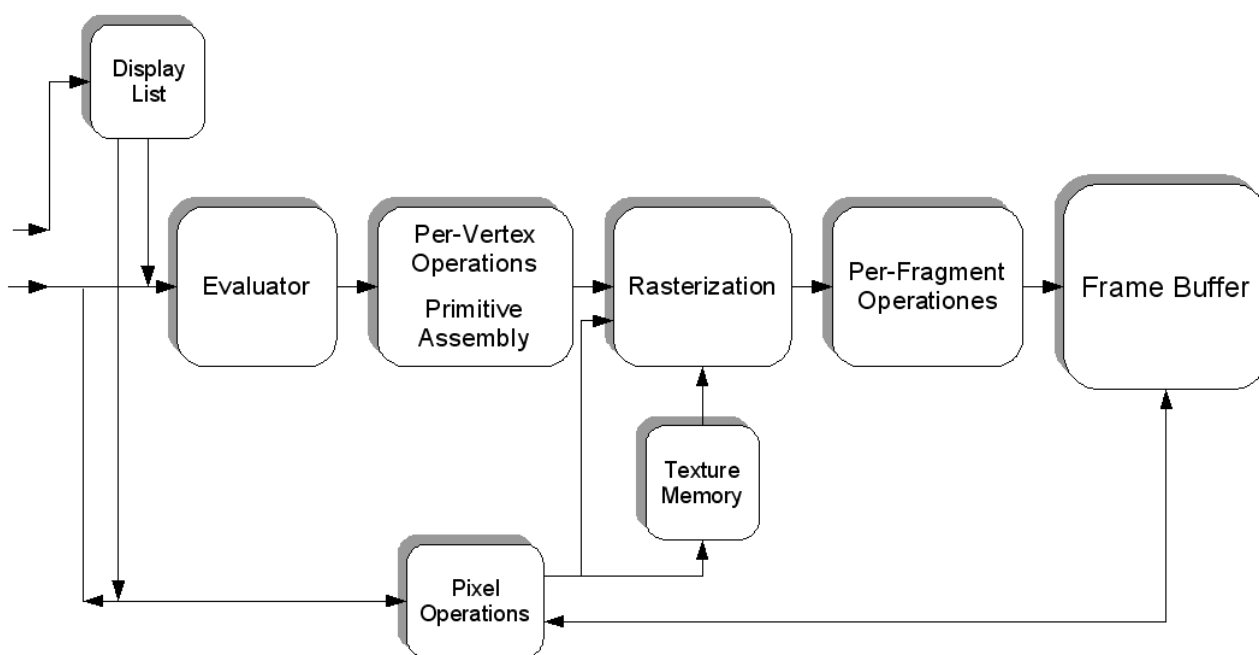
Per-Vertex Operations, primitive assembly: V této části zpracování dochází k různým transformacím, tedy například k rotaci zobrazovaného objektu, či jeho posunu, obojí pomocí 4×4 matic. Je-li povoleno osvětlení, vypočte se barva. Následuje fáze ořezání, kdy se odstraní vše ležící mimo zorné pole kamery. Je-li využívána perspektiva, tedy pokud se objekty blíže kameře jeví větší než vzdálenější, pak jsou v této fázi provedeny operace s hloubkou.(2)

Pixel Operations: Touto fází prochází pouze data obrazová, tedy textury. Zde jsou tato data převedena z různých formátů do formátu, který používá implementace OpenGL. (2)

Texture Memory: Dochází k uložení textur do paměti pro pozdější použití z důvodu odbourání nutnosti načítat texturu při příští potřebě jejího použití.(2)

Rasterization: Obrazová a geometrická data, která se do tohoto okamžiku zpracovávala separátně, jsou nyní zpracována na fragmenty, které reprezentují jednotlivé pixely frame bufferu. Vrcholy definovaných úseček a polygonů jsou propojeny hranami a je-li to povoleno i vyplněno, je uplatněno osvětlení. Každý fragment disponuje na konci dvěma hodnotami a to barvou a hloubkou.(2)

Per-Fragment Operations: Poslední fází před uložením do Frame-Bufferu je řada volitelných operací, jež ovlivňují samotný fragment. Mezi těmito operacemi je například výpočet mlhy či oříznutí dle pozice fragmentu a mnoho dalších.(2)



OpenGL pipeline diagram, by Bedwyr, CC

2.4 Použití OpenGL

V předchozích odstavcích pojednávajících o OpenGL byl uveden krátký soupis historie, byla zmínka o alternativách a teoretickém základu designu samotné knihovny. Nyní se pozornost upoutá k samotnému využívání OpenGL v praktické rovině kterým se bude věnovat v odstavcích následujících.

2.4.1 Vnitřní datové typy

Vzhledem k multiplatformní povaze OpenGL nelze zajistit na všech platformách definování všech proměnných shodně. Z tohoto důvodu OpenGL zavádí vlastní primitivní datové typy, jež využívá ke komunikaci s vnějším světem, ale i k ukládání dat. Vnitřní datový typ OpenGL vždy začíná prefixem GL a následuje jeho název. Tyto datové typy jsou pro hodnoty s plovoucí desetinou čárkou GLfloat (GLclampf) a GLdouble (GLclampd), pro celá kladná čísla pak GLubyte (GLboolean), GLushort, GLuint (GLenum, GLbitfield) a pro celá čísla se znaménkem GLbyte, GLshort, GLint (GLsizei).

2.4.2 Syntaxe příkazů OpenGL

Názvy všech funkcí OpenGL začínají prefixem gl, následuje samotný název. Pokud funkce vrací nějaká data, popřípadě jsou jí předávány parametry, následuje sufix. Ten se může u některých funkcí skládat až ze tří částí. První je číslo, určující počet parametrů, jež jsou funkci předány, následuje znak určující jejich datový typ. Pokud sufix po znaku definující datový typ obsahuje znak 'v', znamená to, že funkce přijímá ukazatel na vektor hodnot, nikoliv 3 separátní proměnné. Tento komplikovaný postup vyplývá z nezávislosti na konkrétním programovacím jazyce, a tedy na nutnosti funkce i bez možnosti přetěžování funkcí.

2.4.3 Display list

Neustále volání stále se opakujících posloupností příkazů je časově náročné. Alternativou k tomuto řešení jsou takzvané zobrazovací seznamy, které jednotlivé příkazy uloží pro opakované použití. Pokud do seznamu uložíme příkaz, kterému předáváme data pomocí proměnné, dojde k uložení hodnoty, která bude použita vždy bez ohledu na to, jakou má proměnná aktuální hodnotu.

2.4.4 Buffery

Buffery slouží k uchovávání informací o jednotlivých pixelech výsledného obrazu. Konkrétní buffer uchovává ke každému pixelu stejné množství dat, ale u různých bufferů to může být různé množství dat. To je způsobeno nutností v Bufferu uchovávat data vždy ve stejném formátu. Pokud máme Buffer barev tak pokud data máme ve formátu RGBA s 8 bity na jeden barevný kanál u jednoho pixelu, pak všechny ostatní pixely musí mít také v RGBA s 8 bity na jeden kanál.

- Buffer barev

Do tohoto bufferu se obvykle kreslí. Uchovává hodnoty o barvách jednotlivých pixelů. Může, ale nemusí obsahovat i hodnoty alfa. Buffer barev nemusí být jeden, některé implementace poskytují pomocné buffery barev. Implementace umožňující stereoskopický obraz umožňují i levý přední a pravý přední buffer barev.

- Buffer hloubky

Slouží k uchování informací o hloubce pixelu. To umožňuje zjistit, zda je pixel překryt nověji vypočítaným pixelem, pokud ano je aktuální pixel v bufferu barev nahrazen novým.

- Double buffering

V případě použití pouze jednoho color bufferu, do kterého se zároveň kreslí i zobrazuje, dochází k situaci, kdy objekty vykreslované, jako poslední mohou být zobrazeny výrazně později, než objekty vykreslené jako první. V další iteraci se situace opakuje. Důsledkem je stav, kdy některé objekty jsou vykresleny výrazně déle, než objekty jiné a zároveň dochází k jevu, kdy krátce zobrazený objekt lidské oko vnímá jako průhledný objekt či v extrémním případě ho nemusí zpozorovat vůbec. Tato situace se řeší pomocí tzv. double-bufferingu. Ten využívá dvou color bufferů tak, že jeden slouží pro uložení scény zobrazované a do druhého se kreslí. V okamžiku, kdy je kreslení dokončeno, dojde k prohození výměně bufferů, tedy zastaralý kompletně vykreslený buffer je nahrazen nově vzniklým, ale již kompletně vykresleným bufferem. Tím fakticky nedochází k postupnému vykreslování před očima diváka a vždy je zobrazen pouze kompletní obraz, který zůstává až do dokončení dalšího vykreslování.

V definici OpenGL není sice double-buffering specifikován, ale různé implementace ho podporují. V případě, kdy chceme double buffering využít, k záměně bufferů použijeme funkci rozšířeného OpenGL dané implementace. V OpenGL pro X Window se jedná o funkci `glXSwapBuffers()`. v OpenGL pro Apple Macintosh `aglSwapBuffers()`. Při využití knihovny GLUT můžeme volat funkci `glutSwapBuffers()`; v SDL pak `SDL_GL_SwapBuffers()`;

2.4.5 Grafická primitiva

Základem každého zobrazovaného grafického primitiva na ploše i v prostoru je vertex, tedy vrchol. OpenGL sice umožňuje zadat souřadnice vertexu jako dvourozměrné, ve skutečnosti je ale vždy počítáno jako s trojrozměrnými souřadnicemi, kdy hodnota souřadnice hloubky je nula. Vrcholy se specifikují voláním vhodné varianty funkce `glVertex` podle datového typu a počtu parametrů.

Před voláním funkce `glVertex` je nutné změnit stav OpenGL tak, aby očekávalo správná data. To se provede funkcí `void glBegin(GLenum grafická_primitiva)`, po skončení je nutné stav změnit ve výchozí funkcí `void glEnd()`. Z výkonnostních důvodů není vhodné volat na jednotlivá primitiva vždy `glBegin` a `glEnd`, místo toho OpenGL očekává mezi těmito dvěma funkcemi sérii primitiv, jejichž konec se ohlásí voláním `glEnd`, výjimkou je mnohoúhelník, u nějž OpenGL nemůže určit, zda následující vrchol je součástí tohoto mnohoúhelníku nebo následujícího. Podporovaná základní grafická primitiva jsou viz Tabulka 1.

Tabulka 1 - Grafická primitiva

Primitivum	OpenGL	Primitivum	OpenGL
Body	GL_POINTS	Pás trojúhelníků	GL_TRIANGLE_STRIP
Úsečky	GL_LINES	Vějíř trojúhelníků	GL_TRIANGLE_FAN
Spojené úsečky	GL_LINE_STRIP	Čtýřúhelníky	GL_QUADS
Úsečky spojené do kruhu	GL_LINE_LOOP	Pás čtyřúhelníků	GL_QUAD_STRIP
Trojúhelníky	GL_TRIANGLES	Mnolehelník	GL_POLYGON

2.4.6 Transformace

Mezi zadáním objektu a jeho vykreslením je řada transformací, které určují jeho umístění vzhledem ke kameře, jeho natočení, zda se nachází v zobrazované oblasti a další. Existují celkem 4 transformace, kterými každý vrchol projde, než je vykreslen.

Transformace se provádí pomocí transformačních matic. Ty mohou být vypočítány aplikací, a nebo zavoláním funkce OpenGL, které to provedou. Následující obrázek zobrazuje pořadí jednotlivých operací, které jsou vysvětleny později.

- Pohledová a Modelová transformace

V této části se vyskytnou dva nové termíny, které je vhodné vysvětlit. Jedním termínem je kamera, kterou je myšlen pozorovatel, tedy něco ve smyslu fotoaparátu, který je na určitém místě, natočený určitým směrem a zaznamenávající dění, které má před sebou. Rozdíl je ale v tom, že místo fotky je výsledkem obraz na displeji. Druhým termínem je scéna, která je vlastně prostor, ve kterém jsou různě umístěny různé objekty. Při návratu k analogii s fotoaparátem je scéna ona připravená scenerie, která je fotografována.

Obě transformace se provádí vynásobením souřadnic vertexu modelovací maticí, neboť nezáleží na tom, zda manipulujeme s kamerou, nebo se scénou v opačném směru.

Nejdříve dochází k pohledové transformaci, která umístí kameru na scénu a její správné natočení. To lze provést pomocí funkce z knihovny glu "gluLookAt", pro kterou jsou parametry pozice kamery - bod, na který se kamera dívá a vektor směru nahoru. Funkce "gluLookAt" může být zastoupena funkcemi transformace posunu a rotace o stejné hodnotě, ale v opačném směru.

```
glLoadIdentity(); // nastavení matice na jednotkovou

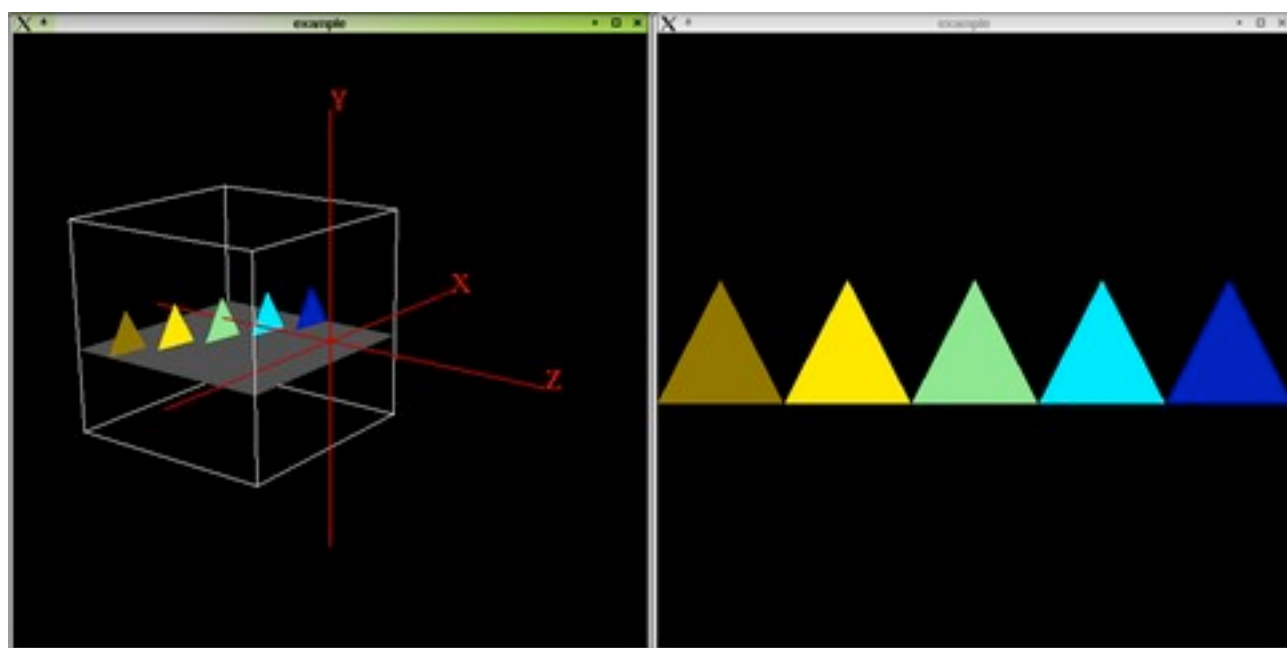
glRotated( uhel, 1, 0, 0); // rotace podle osy x
glRotated( uhel, 0, 1, 0); // rotace podle osy y
glRotated( uhel, 0, 0, 1); // rotace podle osy z

glTranslated(3, 2, 1); // posun o 3 na ose x, o 2 na ose y a 1 na ose z
```

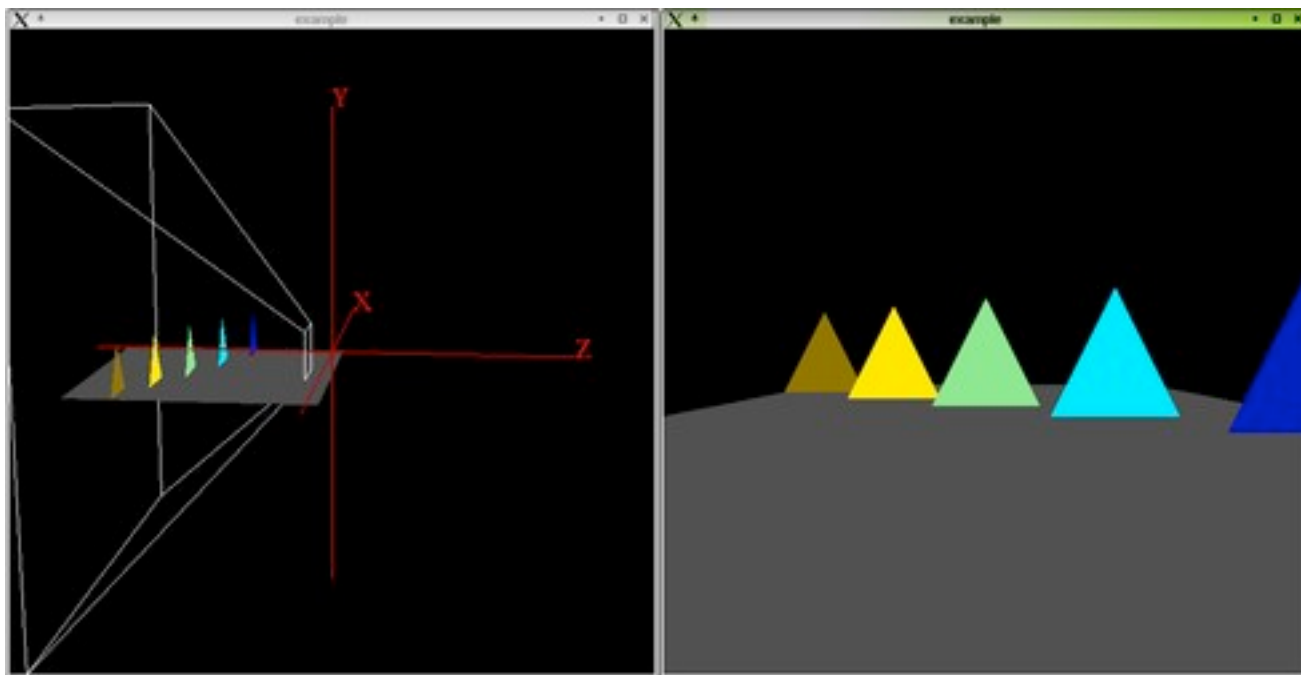
Modelovací transformace slouží k posunu, rotaci, upravení měřítka vzhledem k počátku souřadnic. To je prováděno násobením pohledové matice maticí dané transformace. Funkce, kterými se tyto transformace provádí, jsou znovu ve variantách pro více datových typů, které určuje sufix v jejich názvu. Pro posun slouží funkce “glTranslate”, pro rotaci “glRotate” a změnu měřítka “glScale”

- Projekční transformace

Projekční transformace se ukládají do projekční matice, která definuje zorné pole kamery. Odpovídá i za deformaci objektu vzhledem ke vzdálenosti od kamery. K nastavení projekční matice se může použít jedna ze tří funkcí. Tyto funkce jsou glOrtho, která nastaví zorné pole jako kvádr, nedochází tedy k perspektivní deformaci. Druhou možností je nastavit zorné pole jako komolý jehlan funkcí glFrustum. Poslední možností je symetrický komolý jehlan, který nastavujeme funkcí “gluPerspective”.



Obrazek 2 - Ortografická projekce, by [Dave Pape](#), CC



Obrazek 3 - Perspektivní projekce, by [Dave Pape](#), CC

- Zobrazovací transformace

Zobrazovací transformace slouží definici dvourozměrné oblasti, na kterou je obraz vykreslen. Za samotné okno se zobrazením obsahu sice není zodpovědné OpenGL, ale i přes to může definovat tuto oblast a umožnit tak vykreslení několika různých pohledů na scénu vedle sebe. Provádí se funkcí `glViewport`.

2.4.7 Barva

Barva je vlastnost světla dopadajícího na lidskou sítnici. Lidské oko je schopno zachytit světlo od vlnové délky cca 390 nanometrů, což je vlnová délka fialového světla až po světlo vlnové délky cca 720 nanometrů, kde končí červené světlo jako nejvyšší viditelná vlnová délka. Díky různým druhům čípků v lidském oku je oko schopno zachytit různé barvy zároveň a tyto informace lidský mozek interpretuje jako další barvy. Příkladem budiž bílé světlo, jehož ideální podoba je tvořena všemi frekvencemi viditelného světla zastoupenými stejnou intenzitou.(2)

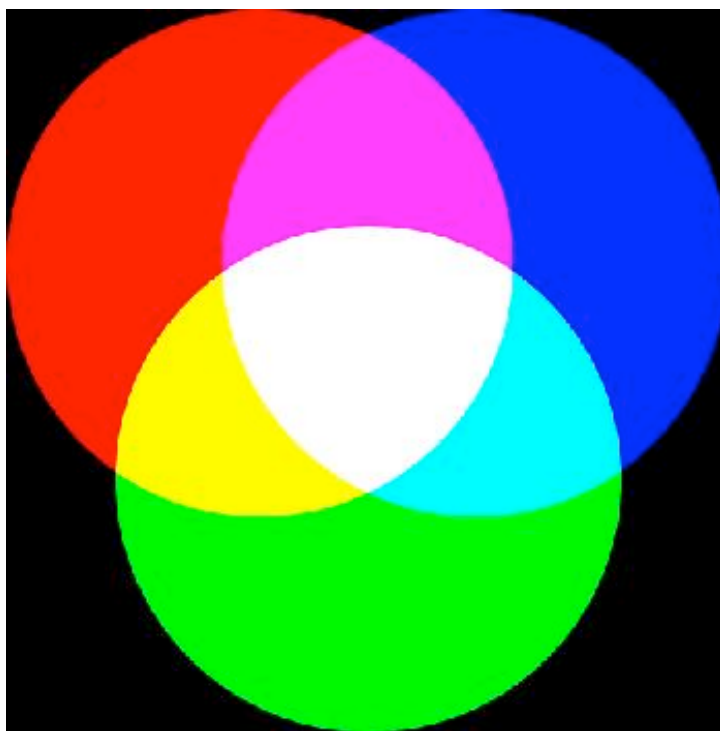
Lidské oko je vybaveno třemi různými variantami čípků, které se neodlišují konstrukcí těla, ale druhem jodopsinu (oční pigment), který obsahují. Každá varianta je specializována na jinou barvu tak, že pokrývají červenou, zelenou a modrou barvu. Pokud je vysláno světlo odpovídající modré a rudé, dojde k nabuzení čípků vnímajících modrou a rudou barvu a mozek toto interpretuje jako barvu fialovou. Tohoto principu je využíváno i v monitorech, kde v jednotlivých pixelech nejsou zastoupeny všechny barvy, ale standardně pouze tři, které odpovídají barvám, na které jsou citlivé čípky, tedy barvy červená, zelená a modrá, jejichž správnými kombinacemi je dosaženo zobrazení námi požadované barvy. Pokud by lidské oko disponovalo více variantami čípků, pravděpodobně by jeden pixel disponoval navíc schopností zobrazit též barvu odpovídající i tomuto čípku. Díky tomu, že se všechny barvy skládají pouze z různých kombinací červené (Red), zelené (Green) a modré (Blue), postačí k popsání zobrazované barvy pouze

údaje o intenzitě těchto tří barev - RGB. Mimo tohoto typu reprezentace barvy existují i další typy. Například CMYK, HLS, ale těch OpenGL nevyužívá.(2)

OpenGL disponuje dvěma způsoby určení barvy, buď jako RGB/RGBA, kde A značí Alfa kanál, nebo pomocí indexových barev. Rozhodnutí, kterou metodu aplikace bude využívat, musí být učiněno na začátku programu a později jej nelze změnit, nebo pouze obtížně. U obou je uloženo na každý pixel určité množství dat, jejichž počet je dán počtem rovin bitů zásobníku, kdy každá rovina poskytuje každému pixelu jeden bit.(2)

- **Mod RGBA**

Bitové roviny bývají často rozděleny na jednotlivé složky RGBA, kdy se nemusí jednat o rozdělení rovnoměrné, ale na většině systémů tomu tak je. Standardně je užíváno 24 bitových rovin - tedy pro každý kanál 8 bitů reprezentujících jeho intenzitu. V zápise do zdrojového kódu se standardně využívá zapisování intenzity jednotlivých kanálů pomocí desetinných čísel, kdy 0 je neaktivní barva a 1.0 je maximální intenzita. Toto systém následně převede do celočíselné reprezentace.



Obrazek 4 - RGB barvy, by [Dave Pape](#), CC

- **Mód indexové barvy**

Při této variantě bitové roviny neudávají přímo barvu pixelu, ale její index, kdy jsou hodnoty intenzity jednotlivých složek dohledány v paletě (vyhledávací tabulce). V tomto módu je limitováno množství barev velikostí palety a množstvím bitových rovin. Velikost palety nejčastěji kolísá mezi 2^8 až 2^{12} . Hlavním důvodem pro tento mód je

možnost jeho užití na platformách s malým množstvím bitových rovin, kde by byly barvy v módu RGB/RGBA příliš hrubé.

2.4.8 Osvětlení

Při pozorování objektu v reálném světě je vjem lidského oka závislý na světle, které k němu doputovalo od tohoto objektu. V drtivé většině objektů se jedná pouze o světlo od něj odražené. Těleso část světla pohltí a odražené světlo poté určuje jeho barvu. Těleso může být lesklé, poté většinu světla odrazí jedním směrem, jiné může světlo rozptýlit světlo kolem sebe rovnoměrně.(2)

OpenGL napodobuje princip světla z reálného světa tak, že využívá popsání barvy světla pomocí modelu RGB a materiál povrchu definuje, kolik z dopadnutého světla bude odraženo a kolik absorbováno. (2): "Rovnice osvětlení v OpenGL jsou pouze aproximacemi, ale poskytují poměrně kvalitní výsledky a hlavně mohou být spočteny hodně rychle. Pokud chcete přesnější výpočty osvětlení (nebo prostě jiné), můžete samozřejmě použít své vlastní výpočty. Takovýto software může být enormně komplexní, k čemuž dozajista dospějete po pár hodinách studia libovolné učebnice optiky".(2)

Osvětlení scény v OpenGL umožňuje osvětlení z jednoho či více zdrojů či světlo obecně rozptýlené po scéně nevystopovatelné ke konkrétnímu zdroji. Takové světlo se nazývá obecné ambientní světlo.(2)

- Složky světla

Světlo v OpenGL je definováno jako složené ze čtyř nezávislých složek. Ambientní, difuzní, zrcadlové a vyzářená složka. I když se s nimi pracuje nezávisle, nakonec jsou složky spojeny v jeden celek.(2)

- ambientní složka je světlo rozptýlené v prostředí natolik, že nelze určit původce. Po dopadu je rozptýleno rovnoměrně po okolí.(2)
- difuzní složka pochází z konkrétního zdroje a po dopadu je rovnoměrně rozptýlena po okolí, je pravděpodobné, že bodové zdroje mají tuto složku.(2)
- zrcadlová složka pochází z konkrétního zdroje, ale odráží se jedním směrem.(2)
- vyzářovací složka je světlo, které nahrazuje světlo, které pochází z objektu, nepřidává však světlo na scénu.(2)

- Model osvětlení

Model osvětlení určuje vlastnosti osvětlení scény. Vlastnosti, které mohou být nastaveny, jsou ambientní barva RGBA pro celou scénu, určení výpočtu úhlu odrazu v závislosti na tom, zda je pozorovatel ve scéně či daleko mimo ni, určení, zda se jedná o jednu složku osvětlení, určení zda se zrcadlová složka počítá odděleně od ostatních. (2)

- Textury

Textury slouží k zjednodušení a zrychlení práce OpenGL. Místo vykreslování jednotlivých částí plochy, například dřevěné desky se šrouby, které desku nepřecházejí, kdy by musela být vykreslena deska a následně šrouby můžeme vykreslit pouze desku a na ni namapovat texturu desky i se šrouby, vedlejším přínosem je přiblížení desky realitě, neboť textury mohou obsahovat detaily dřeva kterých, by jsme museli jinak dosahovat velmi obtížně rozdělením celistvé desky na tisíce polygonů.(2)

Textury mohou být jednorozměrné, dvourozměrné až třírozměrné. Lze je mapovat na zakřivené plochy, množiny polygonů. Definují hodnoty barvy a alfa hodnot, jas atd. Jednotlivé hodnoty v poli jsou nazývány texely. Problémem textur je samo mapování, kdy je třeba texturu tvaru obdélníku namapovat na různé polygony jiného tvaru.(2)

OpenGL uchovává texturu po načtení v podobě interního objektu, nazývaného objekt textury, který usnadňuje jejich použití. Díky těmto objektům můžeme ukládat velké množství textur a později je kdykoliv použít. Jméno textury, se kterým je konkrétní textura svázána, je ve skutečnosti reprezentováno hodnotou GLuint. Použití těchto objektů je nejefektivnější způsob použití textur po jejich načtení do objektů. Jméno textury můžeme použít libovolné, ale z bezpečnostních důvodů se používá funkce `glGenTextures(GLsizei početTextur, GLuint *poleJmenTextur);` která vrátí požadovaný počet doposud nevyužitých názvů textur. Samotné vytvoření textury probíhá funkcí `glBindTextures(GLenum typ1D/2D/3D, GLuint jmenoTextury);` kdy, pokud je jméno textury ještě nepoužito, vytvoří nový objekt. Je-li již použito, stane se tento objekt aktivním a je-li jméno rovno nule, pak se vrací k původní nepojmenované textuře. Objekt textury se maže pomocí funkce `void glDeleteTextures(GLsizei početTextur, GLuint *poleJmenTextur);`.

Zadání textury se provádí příkazem `glTexImage2D(GLenum typTextury, GLint level, GLint vnitřníFormát, GLsizei šíře, GLsizei výška, GLsizei šířeOkraje, GLenum formatTextury, GLenum typTextury, const GLvoid *poleSeVstupnímiHodnotamiTextury);` kde level značí rozlišení textury, pokud používáme texturu s jediným rozlišením, pak je level 0, vnitřníFormát zase značí, která komponenta slouží jako texel obrazu.(2)

Při používání textur je nutné je povolit pomocí `glEnable(GL_TEXTURE_1D/2D/3D);` Následně stačí aktivovat vybranou texturu pomocí fce `glBindTexture` a potom před zadáním každého vertexu zadat, který vertex kterému bodu v textuře odpovídá pomocí `glTexCoord2f(GLfloat x, GLfloat y);`.(2)

3 Ostatní použité knihovny

3.1 GLU

Knihovna utilit OpenGL je nadstavbou čistého OpenGL volající ke své činnosti jeho funkce. Tato knihovna obsahuje řadu funkcí, jež nejsou, nebo nebývaly přímo podporovány grafickými kartami z důvodu složitosti, ale jsou často využívány v praxi. Knihovna GLU obsahuje mimo jiné funkce pro vykreslování kvadrik, vykreslování NURBS ploch. (12)

3.2 GLUT

OpenGL řeší pouze vykreslování, nikoliv správu oken nebo uživatelské vstupy do aplikace, což je pochopitelné vzhledem nezávislosti na systému oken a operačním systému. Většina aplikací však vyžaduje jak otevření a zavření okna, tak vstupy od uživatele, tedy klávesnice, myš a další. Právě na řešení těchto problémů se zaměřuje knihovna GLUT. Její zásadní nevýhodou v dnešní době je zastaralost a zastavení vývoje. Hodí se spíše k využití pro výuku, než pro reálné využití v praxi. Výhodou pak je snadné použití.(2)

Na GLUT navázalo několik dalších projektů, jež sice vzhledem k licenčním podmínkám nemohou převzít zdrojové kódy původního GLUT, ale jejich snahou je vytvářet klon, který kopíruje funkcionalitu, ale navíc je dále vyvíjen. Mezi tyto projekty spadá FreeGLUT.(2)

3.3 SDL

Knihovna SDL vznikla jako obecné multiplatformní nízkoúrovňové API pro hry a multimediální aplikace. Poskytuje relativně malé množství funkcí tvořících základ. SDL poskytuje možnosti vytvoření a správy oken aplikací a podporu pro 2D grafiku, přičemž pro 3D se využívá OpenGL, umožňuje načítat některé základní formáty. Další funkcionalita je doplňována nadstavbami jako je například SDL_image, která k BMP formátu, jenž umí načíst SDL samo o sobě, a přidává řadu dalších formátů. Jmenovitě pár příkladů PNG, TIFF, GIF, TGA, PNM. Dalším příkladem rozšíření je SDL_ttf pro truetype fonty.(18)(20)

Zásadní výhody SDL jsou snadná přenositelnost, jednoduchost a efektivnost, což způsobuje, že je užívána i ve hrách vyvíšenými profesionálními týmy. Jako příklad mohou být uvedeny hry, u nichž je knihovna SDL užita alespoň ve verzi pro jeden operační systém, pro příklad DOOM 3, Quake 4, Sim City 3000 a další.(18)(19)

3.4 OpenAL

Jedná se o multiplatformní API sloužící pro programování přenositelných aplikací využívající zvuku v prostoru. Umožňuje vytvořit skupiny zdrojů zvuku pohybující se relativní rychlostí v různých vzdálenostech vzhledem k posluchači. OpenAL řeší dopplerův jev, směr ze kterého zvuk přichází, intenzitu. Naopak ale neřeší načítání

zvukových souborů, kodeky a podobně. Z těchto důvodů je vhodné k získání zvuku použít externí knihovnu.

4 Technologie, knihovny užívané při vývoji her

Tato část se věnuje ostatním technologiím a knihovnám užívaným při vývoji počítačových her, které sice nebyly využity při vývoji samotné aplikace, ale je vhodné je zmínit, neboť profesionální vývojářská studia je využívají.

4.1 Shader

S rostoucí náročností grafických výpočtů postupně vyvstala potřeba snížit zatížení CPU, který prováděl všechny výpočty, včetně grafických. Řešením se ukázal být specializovaný procesor (GPU) určený pro výpočty spojené s grafikou,. V prvních fázích vývoje tyto procesory měly pouze fixní pipeline, u níž mohl programátor pouze měnit proměnné, ale nemohl řídit samotné provádění operací. Postupem času se v profesionální sféře, převážně u animátorských studií, vyvinula potřeba upravovat si provádění výpočtů v GPU, což byl podnět ke vzniku programovatelné pipeline a s ní spojenými shadery, které jsou funkcemi, jejichž tělo je vykonáváno právě v programovatelné pipeline. K vytváření shaderů vzniklo několik programovacích jazyků, mezi často užívané patří OpenGL Shading Language navržený pro užití s OpenGL, Cg vyvinutý společností nVidia a High Level Shader Language vyvinutý společností Microsoft pro DirectX. (15)

Samy shadery se dělí do několika skupin, podle části grafické pipeline, kterou nahrazují. Konkrétně se jedná o:

- Vertex pipeline, jenž zasahuje do zpracování vrcholů. Často ve vertex shaderu dochází k transformacím, kromě toho je schopen vykonávat další operace nad vrcholy, ale není schopen přidat či odebrat vertex. Používá se na každý vykreslovaný vrchol.(2)(15)
- Geometry shader, zasahuje do sestavení grafických primitiv a na rozdíl od vertex shaderu je schopen přidávat i ubírat vertexy.(15)
- Fragment (pixel) shader, zasahuje do zpracování pixelů. Mezi časté úkoly spadá texturování, výpočet dynamickeho osvětlení a podobně. Výstupem jsou informace o barvě a hloubce daného fragmentu. (2)(15)

4.2 OpenCL

Jedná se o průmyslový standard, jehož hlavním účelem je umožnit využití výpočetního výkonu všech dostupných zdrojů, bez ohledu na jejich hardwarovou platformu v počítači ke zrychlení výpočtů. Tento standard zpřístupňuje k využití pro libovolné výpočty nejen CPU užívaný běžně, ale také GPU, signální procesory.

Softwarová část je multiplatformní, tedy není vázána na konkrétní operační systém, což mu poskytuje výhodu nad Direct Compute, jenž je vázán na produkty společnosti Microsoft.(16)

Standard obsahuje definici abstraktního hardware nutného k provozu, definici OpenCL API a specifikaci programovacího jazyka OpenCL C.(16)

4.3 Bullet Physic

Jedná se o open source fyzikální engine určený k detekci kolizí a dynamiku těles. Je hojně využíván v počítačových hrách a vizuálních efektech ve filmech. Obsahuje podporu pro měkká tělesa (v podobě lan, tkanin a podobně) i pevná tělesa.(17)

5 Nástroje užité při realizaci praktické části

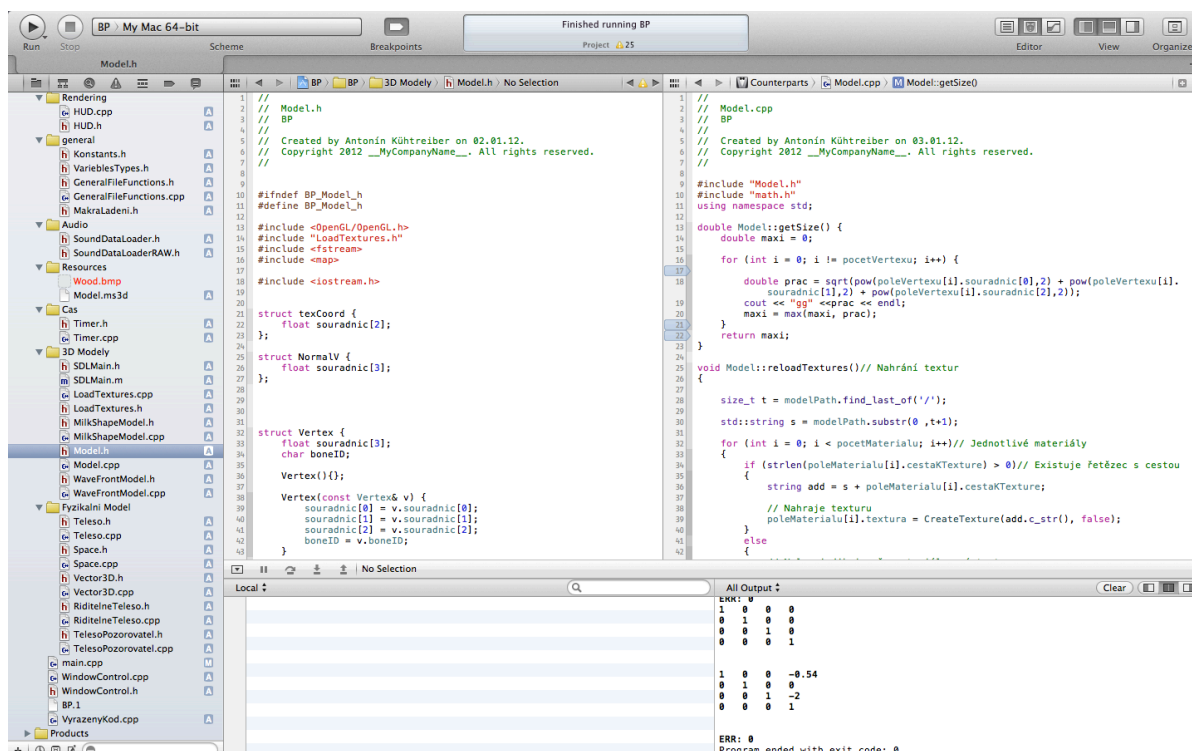
Jako operační systém, ve kterém byl vývoj prováděn, byl zvolen Mac OS X Lion, proto odpovídají této volbě i další nástroje, které byly při vývoji použity. Mac OS X je moderní operační systém postavený na základech OpenBSD.(22)

5.1 Xcode

Jedná se o moderní vývojové prostředí vyvíjené společností Apple inc pro operační systém Mac OS X/OS X. Toto IDE disponuje řadou podpůrných nástrojů, ale část z nich je zaměřena primárně na aplikace psané jazykem Objective-C s využitím frameworku Cocoa. V posledních verzích se objevuje postupný nátlak na opuštění jazyka C++, ale stále toto IDE obsahuje vše potřebné pro vývoj aplikací v C++ včetně debuggeru. Z integrovaných nástrojů lze uvést Interface Builder pro vytváření GUI k aplikacím.(22)

IDE Xcode má přístup k frameworkům uloženým v knihovnách systému a tedy má k dispozici řadu frameworků, mezi nimiž je většina užitych při tvorbě mé bakalářské práce, tedy OpenGL, GLUT, OpenAL a další. Zajímavostí je, že SDL v nich (ve výchozím stavu) obsažen není a musí být tedy stejně jako ve Windows přidán ručně.(22)

Co se kompilace týče ve výchozím stavu je k dispozici hned několik kompilátorů. Výchozím a hojně prosazovaným je Apple LLVM compiler 2.1.(22)

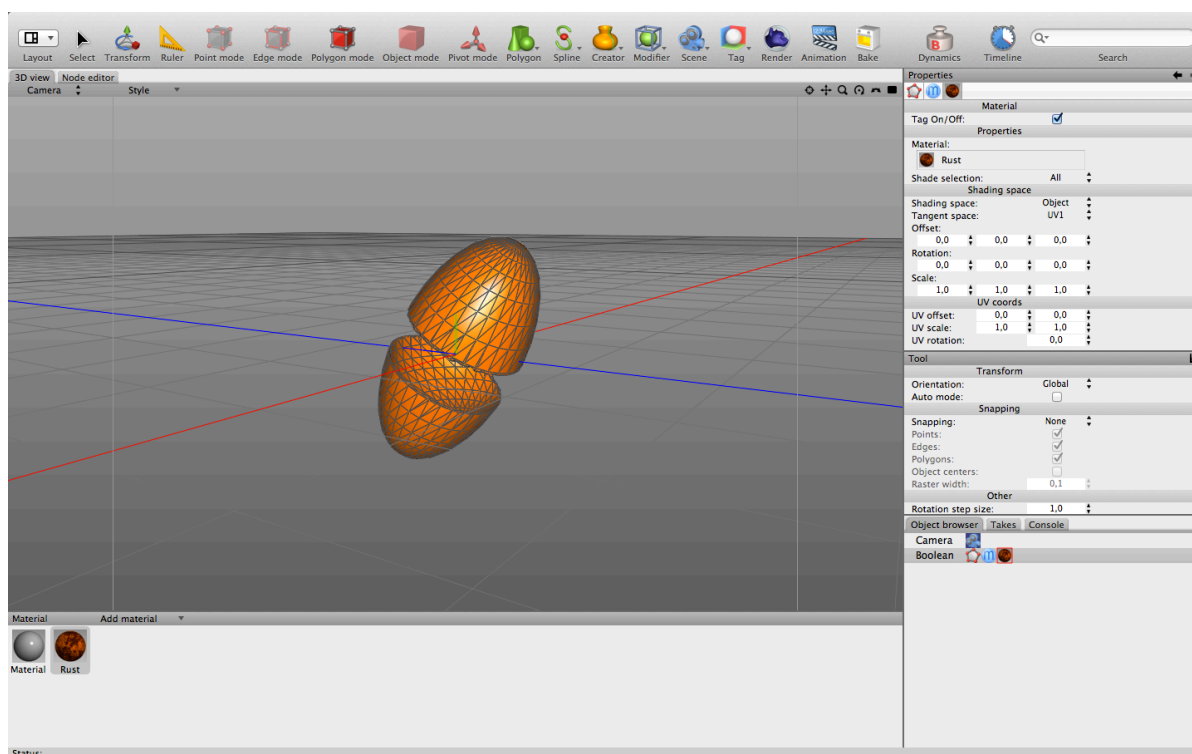


Obrazek 5 - xCode

5.2 Cheetah3D

Jedná se o kvalitní 3D modelovací nástroj německého původu vyvíjený pouze pro operační systém Mac OS X. Jeho cílovou skupinou jsou primárně začínající uživatelé a amatérští 3D umělci, čemuž odpovídá i nízká pořizovací cena ve výši. Krom funkcí modelování obsahuje i podporu pro renderování a animování. Krom V současné době se nachází ve verzi 6.0(23)

Při realizaci praktické části mé bakalářské práce byl využit Cheetah3D pro návrh a tvorbu modelů.(23)



Obrazek 6 - Cheetah3D

6 Realizace praktické části - logická část

6.1 Návrh fyzikální simulace prostoru

Návrh fyzikální simulace prostoru spočívá v metodách, kterými je zajištěno, že objekty v prostoru se chovají dle předpokladu hráče. Stejně jako v mnoha dalších částech i zde je nutné udělat řadu kompromisů vzhledem k výkonům počítačů a potřebě zajistit hladký běh na maximálním počtu strojů. Základní problémy k řešení lze definovat jako způsob uchování informací o prostoru a způsob uchování informací o jednotlivých tělesech v prostoru. Musí být zajištěno řešení vzájemných interakcí těles a způsob přístupu k informacím těles za účelem jejich vykreslení.

6.1.1 Analýza fyzikálních zákonů k implementaci

Pohyb těles ve volném beztlákovém prostoru je popsán buď mechanikou klasickou, která je vhodná pro rychlosti nízké, nebo dynamikou relativistickou, která je přesnější pro chápání těles, jejichž rychlost je blízká rychlosti světla.

- Klasická mechanika

Základ klasické mechaniky těles je založen na Newtonových zákonech, z nichž dva nejsou vytvořeny přímo Isaacem Newtonem, ale astronomem a fyzikem Galileo Galileiem. (13)

- První zákon je zákon setrvačnosti. Zní (13): “Jestliže na těleso nepůsobí žádné vnější síly nebo výslednice sil je nulová, pak těleso setrvává v klidu nebo v rovnoměrném přímočarém pohybu.” Tento zákon popisuje chování těles, které neovlivňuje žádná síla. Tento stav je fakticky běžný u všech těles ve volném prostoru. Primárně pro asteroidy a všechny ostatní tělesa přirozeného původu, pro které se jedná o stav jediný, ale i pro všechny ostatní, která jsou umělá, i když pro ty pouze v době, kdy sama nevyužívají svého pohonu. (13)

- Druhý zákon je zákon Síly (13): “Jestliže na těleso působí síla, pak se těleso pohybuje se zrychlením, které je přímo úměrné působící síle a nepřímo úměrné hmotnosti tělesa.” Tento zákon pro změnu popisuje chování těles, na která působí nějaká síla. Objektivně řečeno, na všechna tělesa působí nějaká síla, byť zanedbatelných hodnot, ale pro potřeby herních aplikací se jich velké množství zanedbává z čistě praktických důvodů. Hodnota gravitační síly je přímo úměrná hmotnostem těles a nepřímo úměrná druhé mocnině jejich vzdálenosti. Díky tomu je jejich vliv na malá tělesa daleko od centrální hvězdy soustavy či planet minimální a proto pro herní aplikace zanedbatelný. Obdobný stav je u většiny sil ovlivňujících tělesa, o nichž je tento typ her. Fakticky jediné síly, které nelze zanedbat, jsou síly působící mezi tělesy a případně pohon některých těles.

- Třetí Newtonův zákon, Zákon akce a reakce (13): “Proti každé akci vždy působí stejná reakce.” Pokud pomineme, že většina, ne-li všechny, motory využívají tohoto principu, pak je nutné si uvědomit, že k vzájemné interakci dochází i u všech srážek těles, popřípadě jiného vzájemného silového působení. (13)

- Dále je vhodné zmínit i tak zvaný Princip superpozice (13): “Jestliže na těleso působí současně více sil, rovnají se silové účinky působení jediné síly, tzv. výslednice sil, která je rovna vektorovému součtu těchto sil.”

- **Relativistická mechanika**

Albert Einstein došel k závěru, že zatímco při nižších rychlostech je klasická mechanika dostatečná k popisu skutečnosti, tak při růstu rychlosti blíže k rychlosti světla (c) dochází ke značným odchylkám od předpokladů. Zatímco u těles v klasické mechanice, pokud působíme na těleso, by těleso vždy zrychlilo stejně, bez ohledu jakou rychlostí se již pohybuje, tak v relativistické mechanice těleso pohybující se rychlostí blízkou rychlosti světla dochází ke zvýšení hmotnosti tělesa a tím i k zvýšeným energetickým nárokům tělesa pro zrychlení. Díky tomuto jevu je reálně nemožné urychlit těleso na rychlost světla, neboť čím je rychlost tělesa blíže rychlosti světla, tím je pro další zrychlení potřeba více energie. Většina těles se však pohybuje rychlostmi výrazně pomalejšími než je rychlost světla. Další komplikací pro rychle se pohybující tělesa je jev zvaný dilatace času. Jedná se o situaci, kdy pro pohybující se těleso plyne čas odlišnou rychlostí, než pro těleso stojící. Pro vysvětlení a pochopení dilatace času lze užít paradoxu dvojčat. Jedno z dvojčat zůstává na Zemi a druhé se vydává na průzkum vesmíru. Když se druhé dvojče vrátí na Zemi, nalezne zde své dvojče výrazně starší, neboť dvojče, které se pohybovalo rychleji stárlo pomaleji, vlastně pro něj čas plynul pomaleji. (14)

- **Analýza**

Už v návrhu hry je důležité učinit rozhodnutí, zda-li se budou vyskytovat tělesa pohybující se rychlostí blízkou rychlosti světla a tedy, zda bude či nebude nutné implementovat relativistickou fyziku chování těles. Samotný pohyb těles není významnější problém, znatelným problémem je ale doprovodný jev, a to dilatace času.(14)

Dilatace času přináší řadu komplikací pro řízení lodí. Hlavním dopadem z pohledu hry je pak problém s reakční dobou, na neočekávanou událost, která je logicky kratší, než by byla bez dilatace času a tedy se snižuje hratelnost. Dalším dopadem je obtížná výpočetní náročnost, neboť to vyžaduje pohyb všech těles zdánlivě rychlejší. A v neposlední řadě dilatace času zcela vylučuje hru více hráčů, pokud tito hráči neovládají různé části jedné lodí, neboť hra musí být pro všechny ve stejný okamžik a není možné nijak simulovat hráči požitky z různých dilatací času různým hráčům tak, aby všichni hráli v jednom herním prostoru. Jediným řešením, které by počítalo s rychlostmi blízkými rychlosti světla v herní simulaci, je i dilatace času pro hráče v reálném světě, což je mimo současné technické možnosti. Jediným uplatnitelným řešením je proto vyhnout se rychlostem blízkým c . (14)

Naštěstí to lze zdůvodnit malými částicemi vyskytujícími se v kosmickém prostoru, i když relativně řídké. Náraz do trupu ve vysoké rychlosti by mohl způsobit jeho poškození až zničení. To je ovšem pouze fyzikální zdůvodnění maximální rychlosti lodí, které nemá s praktickou realizací nic společného.(14)

6.1.2 Uchování informací o prostoru

Tyto informace jsou potřebné ve dvou částech procesu, a to v propočítávání změn mezi dvěma časovými úseky a následně ve vykreslování scény. Pro návrh uchování

informací v prostoru u herních systémů je třeba splnit několik podmínek. Předně je třeba zajistit minimální množství potřebné paměti a zároveň dosáhnout maximální efektivity, ale současně musí podávat kompletní obraz o prostoru. Existuje několik způsobů uchování informací o herním prostoru.

- 1 Základní a nejzřejmější je uchovávání pole, ve kterém je evidován každý bod prostoru. Toto řešení sice obsahuje kompletní popis prostoru, ale je enormně paměťově náročné, neboť každý neobsazený bod prostoru zabírá v paměti prostor pro ukazatel, jehož hodnota je null. Z pohledu výpočetní náročnosti na tom je ještě hůře, neboť jak vykreslování, tak propočítání změn vyžaduje navštívit každou buňku pole, aby u většiny zjistilo, že se jedná o prázdné místo, neboť většina prostoru je prázdná. Z tohoto důvodu je toto řešení nevhodné pro tento typ aplikací. Na druhou stranu snadno nalezne uplatnění v aplikacích typu šachy, kde je hrací plocha dvourozměrná a navíc silně omezena, což se rozsahu týče.
- 2 Další způsob není uchovávání informací o celém prostoru, ale naopak pouze o jednotlivých objektech. Pokud je celý prostor prázdný, s výjimkou popsání objektů, pak je popis prostoru úplný a oproti předchozímu řešení dosahuje výrazně menšího plýtvání paměti i časem procesoru. Vzhledem k předpokladu vzniku a zániku objektů není vhodné užití statického pole, ale struktury umožňující její rozšíření. Vhodným se jeví lineární spojitý seznam.
- 3 Poslední variantou je modifikace předchozího řešení, ovšem s tím rozdílem, že nedochází k zaznamenání všech objektů uvnitř jednoho seznamu, ale jako součásti několika seznamů těles, podle umístění těles v prostoru. Toto řešení je sice nepatrně paměťově náročnější, ale umožňuje distribuci výpočtů mezi více strojů při síťové hře a tím snížit zatížení serveru.

První varianta byla diskvalifikována téměř okamžitě, neboť spojení paměťové náročnosti s výpočetní náročností je zcela nesmyslnou variantou, která nejde obhájit. Třetí varianta má hlavní výhody v možnosti distribuce výpočtů, což není potřebné v této práci, neboť rozsah a předpokládaný počet objektů v této práci zcela jistě nebude tak dramatický, aby bylo potřeba přistoupit k distribuci zátěže. Stejně tak by tato distribuce nebyla ani možná, neboť součástí práce není síťová komunikace, navíc v případě pozdějšího doplnění by případné úpravy nebyly tak rozsáhlé, aby bylo nutné přistoupit k třetí variantě. Z těchto důvodů bylo rozhodnuto využít druhou variantu. K zajištění ukládání objektů těles bylo rozhodnuto použít třídu vector, neboť disponuje požadovanými vlastnostmi.

6.1.3 Uchování informací o tělesech

Objekt tělesa musí obsahovat několik informací. Základní informace jsou statické, které pomáhají definovat samotné těleso. V beztlakovém prostoru stále platí zákon setrvačnosti, a proto mezi tyto údaje spadá hmotnost. Dalším podstatným bodem je informace o rozměrech tělesa. Tvar tělesa je sice specifikován modelem, ale ověřování

Space
<pre> void * winControl; vector<Teleso*> vektorTeles; unsigned int pocetObektu; long *poziceKamery; double natozeniKamery[3]; TelesoPozorovatel *pozorovatel; </pre>
<pre> unsigned short getPocetObektu(); void boundCameraWithObject(TelesoPozorovatel * teleso); void drawHUD(); void tah(); void vykresleni(); vector<Teleso*> getObjectDistantMax(unsigned long maxVzdalenost); void akcelerace(bool stav); void decelerace(bool stav); void rotace(bool stav, char smer); void rotaceSTOP(bool stav); </pre>

Obrazek 7 - UML Class diagram třídy prostoru

kolize dvou těles podle jejich modelů by bylo příliš náročné z pohledu času procesoru, neboť model je složen z velkého množství trojúhelníků a testem by musel projít každý z nich. Pro příklad uvedu, že v případě testu dvou objektů, kdy každý je složen z dvaceti trojúhelníků by muselo být otestováno celých 200 trojúhelníků. Z tohoto důvodu je výhodnější přistupovat k tělesu jako k jednoduššímu prostorovému objektu, tedy jako ke kouli, krychli či kvádru, u kterých se následně rychle lze dopracovat k výsledkům. Dále je vhodné uložit odkaz na model tělesa. Dalšími nutnými informacemi jsou poloha k počátku souřadnic a natočení vzhledem k výchozímu postavení tělesa. Tyto údaje se upravují při každém přepočtu. Poslední kategorií údajů jsou informace o rychlostech. Jedná se o rychlost pohybu tělesa a úhlové rychlosti v jednom časovém kvantu, tyto údaje slouží ke změně údajů z předešlé kategorie. (8)(10)

Objekt Teleso však pouze nezastřešuje informace o tělese ale též poskytuje funkce pro jejich změny v čase i jejich vzájemné interakci.

Změna za časové kvantum:

Prakticky není možné v počítačových hrách počítat s časem jako spojitou veličinou, neboť sám časově závislý výpočet změn byl pomalejší než při diskretní představě času. Bylo by nutné nejen provést přičtení hodnot rychlosti k aktuální poloze, ale předtím i na základě systémového času určit, jakou vzdálenost reálně těleso urazilo. Efektivnější se proto jeví počítat změny vždy pro statický úsek. Vzhledem k potřebě volat překreslování scény vždy až po přepočtu, neboť dříve nemá smysl, je časovač udávající frekvenci přepočtů vně tuto třídu.

6.1.4 Pohyb těles

Přepočet pozic těles nastává vždy, když k tomu dostanou signál z nadřazené vrstvy. Objekt space obdrží signál z časovače a sám následně projde vector, v němž má uloženy všechna tělesa a provede volání jejich funkce void tah(), která způsobí pohyb těles. Jednotlivá tělesa však nemají informace o okolních tělesech a proto nemohou zjistit, zda-li při změně pozic došlo ke kolizi. Z toho důvodu se objekt space dotáže funkcí isCollision(Teleso *t), zdali ke kolizi mezi zmíněnými dvěma tělesy došlo. Pokud ano, dojde k předání této informace jednomu z těles, které provede vyřešení kolize.

6.1.5 Detekce kolize

První částí reakce na kolizi je její samotná detekce. Vzhledem k tomu, že bylo pro potřeby kolizí těleso omezeno na pouhou kouli, je detekce vcelku prostá, a to v podobě testu každých dvou těles, zda vzájemná vzdálenost jejich středů není nižší, než součet jejich poloměrů.

6.1.6 Reakce na kolizi

Pro výpočet reakce na kolizi je nutné si uvědomit, že působení sil mezi dvěma tělesy je pouze v jednom směru pro jedno těleso a ve směru opačném pro těleso druhé. Tyto směry jsou dány vektorem určeným ze vzájemných pozic středů. To umožňuje pro potřeby výpočtu počítat pouze s rychlostí ve směru kolize a tedy využít vzorec pro výpočet rychlostí po srážce. Rychlost, kterou se tělesa pohybují ve směru srážky, pak určíme jako skalární součin vektoru rychlosti s normalizovaným vektorem pohybu tělesa. Následně díky vzorcům uvedeným dále je určena rychlost po srážce pomocí dále zmíněných na obrázcích 8 a 9. Nyní máme k dispozici informaci, jaká je vzájemná rychlost těles ve směru srážky. Vynásobením s normalizovaným vektorem udávajícím směr srážky získáváme vektor změny směru vlivem srážky. Jeho přičtením k vektoru pohybu tělesa získáváme směr po kolizi. Přičemž rychlost je dána velikostí tohoto vektoru.(8)(9)(10)

$$v'_1 = \frac{(m_1 - km_2)v_1 + (1 + k)m_2v_2}{m_1 + m_2}$$

Obrazek 8 - Vzorec rychlost po nárazu těleso 1, Wikipedia, CC

$$v'_2 = \frac{(m_2 - km_1)v_2 + (1 + k)m_1v_1}{m_1 + m_2}$$

Obrazek 9 - Vzorec rychlost po nárazu těleso 2, Wikipedia, CC

6.1.7 Třída říditelných těles

Běžný typ tělesa simuluje pouze těleso neschopné akcelerace, decelerace ani manévrování. Proto bylo rozhodnuto vytvořit nového potomka této třídy, který by těmito schopnostmi již disponoval.

Zatímco rotace tělesa je v principu jednoduchá, samotné těleso rotuje se středem své rotace procházejícím těžištěm a dochází pouze ke zrychlení či zpomalení této rotace, tak při akceleraci či deceleraci je nutné znát směr, kterým je těleso natočeno, neboť hlavní pohon bývá zpravidla nasměrován od přídi k zádi a tedy směr akcelerace musí být určen podle současného natočení letounu. Pokud je známo, jak je letoun natočen, je nejjednodušší využít funkce rotace v OpenGL k výpočtu matice rotace. Tou následně může být vynásoben vektor směru ukazující k přídi tělesa ve výchozí poloze a tím zjistit aktuální vektor ukazující k přídi nyní. Vektor k aktuální přídi je nutno vynásobit hodnotou zrychlení a poté přičíst vektoru rychlosti. Tím ovšem nastává problém, kdy je těleso téměř neovladatelné, neboť platí setrvačnost. Těleso je schopno akcelarovat ve směru k přídi, což pouze rychlost zvyšuje, ale těleso se pohybuje stále stejnou rychlostí v původním směru. Tuto vlastnost lze využít u některých manévřů, ale pro snadnější ovládání tělesa je vhodné umožnit splynutí směru vektoru k přídi s vektorem směru pohybu úpravou směru pohybu. To je jednoduchá funkce, která pouze postupně odstraňuje rozdíl ve směru.

6.1.8 Třída pozorovatele

Tělesem pozorovatele se míní těleso ovládané hráčem a tedy hráč sám je pozorovatelem, neboť vnímá skrze toto těleso a i je reprezentován v aplikaci tímto tělesem. Jako těleso ovládané hráčem krom samotného řízení letu musí poskytovat i vyvolávání odezev na některé situace pro hráče. Primárně je na toto těleso vázána kamera a tedy záběr scény. Zvuk vyvolávaný ostatními tělesy navzájem není podstatný, neboť zvuk se vakuem nešíří, vlivem toho je toto těleso jediné, které může zaslat požadavek na přehrání určitého zvuku vyvolaného kolizí a dalšími jevy. Mimo toto je i stav tohoto objektu zobrazován pro potřeby informování uživatele.

6.2 Problematika návrhu umělé inteligence

Pojem umělé inteligence se objevil krátce po prvních programovatelných počítačích, avšak do dnešní doby není přesně určena jeho definice. Mluví-li se o umělé inteligenci obecně, může se říci, že se jedná o mechanismus samostatného rozhodování při znalosti aktuálního stavu. K určení, co je již umělá inteligence, byl navržen Alanem Turingem test, dnes známý jako Turingův, který sleduje, zda je člověk schopen rozlišit, zda na jeho otázky odpovídá člověk či stroj. Tento test však je již dnes vyřazen argumentem čínského pokoje, kdy stroj může mít předpřipraveny reakce na všechny otázky. Dodnes se však využívá k ověření, zda se jedná o lidi v internetových službách v podobě CAPTCHA.

(5)(15)

6.2.1 Herní umělá inteligence

Od obecného chápání pojmu umělé inteligence se její herní varianta ve většině případů nesnaží vytvořit skutečně inteligentní chování, ale pouze ho natolik napodobit, aby hráče přesvědčilo, že se o inteligentní chování jedná, a to pokud možno s minimálním zatížením procesoru, neboť většina výpočetního výkonu je využívána pro výpočty fyzikálních jevů a vykreslování grafiky. Sama herní AI je podřízena, jako vše ve hře, snaze zabavit hráče, z čehož vyplývají i požadavky na ni. Není vhodné, aby se chovala příliš hloupě, neboť to hráče nezabaví, ale ani příliš dokonale z téhož důvodu. Zatímco naprogramování příliš dobré AI je snadné z důvodu přesnosti výpočtů, které je počítač schopen dosáhnout, nejobtížnější je vytvoření správně vyvážené AI, která ne vždy zasáhne cíl a ne vždy zvolí nejvýhodnější trasu či postup.(5)

6.2.2 Metody tvorby herní AI

Technologie užívaná v herních AI silně zaostává za zbytkem světa. Hlavními důvody jsou výpočetní náročnost, testovatelnost chování a čas tvůrců majících termíny, které se snaží plnit a tedy nemají čas seznamovat se s nejnovějšími poznatky na poli AI, kterými jsou umělé neuronové sítě a genetické algoritmy, které jsou v současnosti široce užívány ve vědecké sféře. Dále je vypsáno několik vybraných metod:(5)

Skript: přímé naprogramování akcí je základním postupem v herních umělých inteligencích. Programovací jazyk obvykle bývá odlišný od jazyku v němž byl vytvořen zbytek hry, a to z důvodů možnosti zvolení odlišné abstrakce. Chování vytvořené touto technikou je přesně definováno, a to se všemi klady i zápory. Pokud nastane situace tvůrcem opomenutá hra nezareaguje správně pokud vůbec.(5)

Rozhodovací pravidla: během celé hry existuje skupina činností, které jsou vykonávány vždy po splnění jejich podmínek. Příkladem budiž nabití zbraně vždy, když nabitá není. (5)

Stavové automaty: je sice možné naprogramovat umělou inteligenci prvním způsobem pro všechny situace, na druhou stranu je ale mnohdy mnohem přehlednější a výhodnější vytvoření několika nezávislých souborů pravidel chování pro určité situace a mezi nimi přepínat podle aktuální situace a tedy stavu, v němž se systém nachází. (5)

Hledání cesty a navigace podle ní: herní umělá inteligence často sleduje předem dané cesty, popřípadě je určuje z grafu algoritmem A*, kde slouží jako vrcholy body vložené tvůrci hry. Následná navigace dále vyžaduje reakce na překážky v cestě, popřípadě i odůvodněné opuštění cesty a následný návrat na ni.(5)

Neuronové sítě: i když se neuronové sítě v akademické sféře k realizaci AI používají, v herní sféře tomu tak není. Důvodů je několik, od výpočetní náročnosti, kdy hra vytěžuje počítač ostatními funkcemi až po nemožnost přesně předpovědět, jak se zachová ve všech situacích a tedy prakticky netestovatelnost.(5)

6.2.3 Realizace herní AI

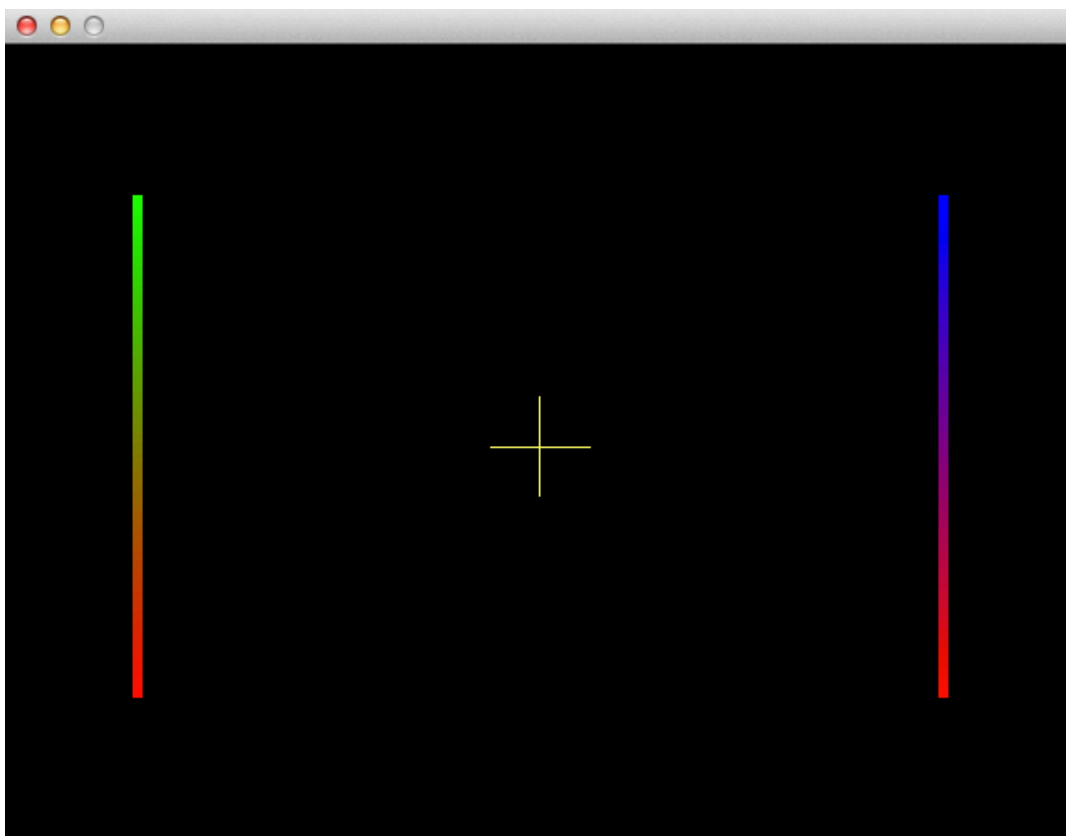
Sama realizace herní umělé inteligence byla provedena na principu stavového automatu a předprogramovaného chování. Princip chování je velmi jednoduchý, neboť není třeba řešit mnoho variant chování. AI se nachází buď ve stavu, kdy nalezla cíl a ten stíhá a nebo ve stavu klidu, kde letí podél trasy zadané formou bodů, kterými musí proletět. Jedinou komplikací jsou překážky, u kterých hrozí kolize. Ty je schopna včas

detekovat a podniknout kroky potřebné k vyhnutí se jim, neboť mu jsou známy pozice okolních těles spolu s jejich rychlostmi.(5)

7 Realizace praktické části grafická a akustická část

7.1 Vytvoření a zpráva grafického okna aplikace

K vytvoření a správě oken je možno použít několik knihoven, z nichž je výběr uveden výše. Původní implementace byla postavena na knihovně GLUT. V pozdějších fázích vývoje bylo ale rozhodnuto nahradit tuto zastaralou a již neaktualizovanou knihovnu knihovnou SDL, která navíc poskytl podporu pro snadné načítání textur, což byl prvotní impuls pro zvážení integrace této knihovny do projektu. Oproti řešení s pomocí knihovny GLUT, kde jsou nastaveny ukazatele na funkce, jež mají reagovat na událost, nastávající stiskem klávesy či žádostí o překreslení scény, bylo třeba vytvořit vlastní cyklus kontrolující, zda byla stlačena či uvolněna klávesa či zda je požadavek na překreslení scény. Místo volání `glutMainLoop()` se tedy volá vlastní funkce obsahující nekonečný cyklus. Stejně tak se místo `glutPostRedisplay()` volá funkce vlastní, jenž uloží požadavek pro překreslení, jemuž bude vyhověno v okamžiku, kdy se k testu připravenosti k překreslení dostane vlastní naprogramovaný cyklus.



Obrazek 10 - SDL okno

7.2 Zobrazení stavu

V reálné hře je nutné informovat hráče o jeho stavu, neboť na rozdíl od reality nemůže vnímat bolest, vibrace a podobně. Proto je vše pokud možno převedeno do přehledných hodnot, které jsou zobrazeny buď číselně, nebo graficky. Z důvodu přehlednosti bylo zvoleno grafické znázornění dvou veličin, a to integrity trupu a množství energie štítů, kdy obojí sice symbolizuje, kolik zásahů ještě je schopen ustát, ale zatímco integrita trupu po snížení se není schopna obnovovat, energie štítů ano. Mimo těchto dvou údajů je zobrazován i zaměřovací kříž.

Na rozdíl od dalších částí se v tomto případě nejedná o 3D grafiku, ale o 2D grafiku. V OpenGL mezi nimi není významný rozdíl, neboť se řeší položením třetí souřadnice rovné vhodně zvolené konstantě.

7.3 Načtení trojrozměrných těles

V OpenGL jsou k dispozici řady grafických primitiv, která mohou popsat objekt v prostoru. Sám objekt tedy musí být popsán pomocí těchto grafických primitiv. V současné době existuje nepřehledná řada souborových formátů vytvořených za účelem nést data o podobě objektů. Některé nesou pouze geometrické informace, jako například Wavefront modely, jejichž soubory mají příponu .obj, k nimž existuje druhý souborový formát s příponou .mtl, který je vlastně knihovnou materiálů pro modely v *.obj souborech. Pak existují souborové formáty, které obsahují nejen geometrické informace, ale i informace o materiálech. Mezi takové spadá i nativní souborový formát programu MilkShape 3D s příponou ms3d, který jsem se rozhodl použít.(10)

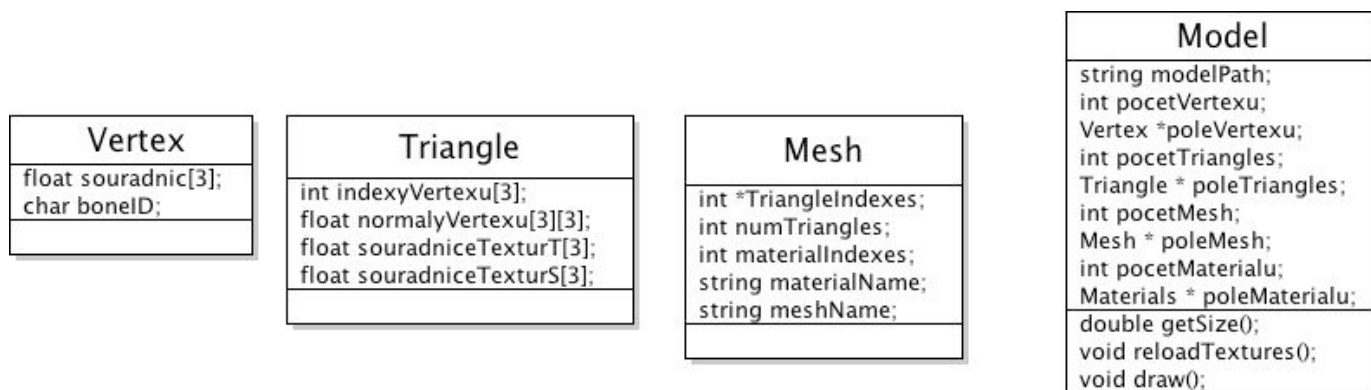
Oproti DirectX má OpenGL nevýhodu v podobě absence model loaderu, tedy funkcí schopných nahrát a zpracovat data ze souborů s modely do podoby použitelné pomocí OpenGL a tedy tuto část musí každý udělat sám. Z důvodu, kdy neexistuje pouze jeden souborový formát, bylo rozhodnuto učinit možnou pozdější implementaci podpory dalších formátů vytvořením modelu základní třídy Model, jenž uchovává geometrická a materiálová data a loader je obsažen v potomku této třídy označovaném jako MilkShapeModel. Tato třída obsahuje pole, v němž jsou uloženy jednotlivé vertexy, pole obsahující trojúhelníky, kde jsou nahrazeny údaje o vertexech. Pouze jejich indexy a pole obsahující mesh sdružující tyto trojúhelníky, které tvoří dílčí část celého modelu, jejichž optické vlastnosti jsou definovány totožným materiálem a ten k nim i váže.(10)(11)

Formát ms3d je člověkem nečitelný a strukturovaný. Soubor začíná hlavičkou obsahující informace o verzi souboru, která je postupně následována daty všech vertexů, trojúhelníků, které místo hodnot vrcholů obsahují pouze indexy vertexů. Následují data mesh, u nichž jsou opět pouze indexy na trojúhelníky. Následují data materiálů a vše je zakončeno daty pro animaci. Z důvodu člověkem nečitelných dat v souboru je nutné k datům přistupovat jako k blokům binárních dat s přesně určeným pořadím a velikostí. (10) (11)

Velikosti jednotlivých bloků jsou dány. Hlavička má vždy délku 14 bytů, kde prvních 10 je 10 znaků, jež vždy nabývají hodnoty "MS3D000000". Následují 4 byty označující číslo formátu souboru. Následuje dvou bytová celočíselná kladná proměnná udávající počet vertexů, po níž následují samotné vertexy. Každému vertexu připadá 15 bytů, kdy první byte obsahuje příznaky, dalších 12 je vyhraněno pro 3 čísla s plovoucí

desetinou čárkou, jež tvoří samotné souřadnice vertexu. Poslední dva byty jsou vyhrazeny pro boneID a referenceCount, které nejsou pro tuto aplikaci důležité. Po vertexech následují trojúhelníky, ale stejně jako u vertexů i jim předchází jejich počet vyjádřený dvěmi byty. Každý trojúhelník je zapsán 70 byty. První dva byty jsou vyhrazeny příznakům, následuje pole dvoubytových proměnných o délce 3, které nese hodnoty indexů vertexů. Následuje dvourozměrné pole float o velikosti 3x3 obsahující normálové vektory k vertexům a další 2 pole o délce 3 obsahují znovu hodnotu float se souřadnicemi textur. Poslední dva byty zabírají smoothingGroup a groupIndex. Znovu se opakují dva byty s počtem Meshů, tedy skupin trojúhelníků. Ty již nemají pevnou velikost, neboť počet trojúhelníků v meshy je proměnlivý. První byte každého meshe obsahuje příznaky, dalších 32 jsou znaky s názvem. Následuje počet trojúhelníků ve dvou bytech následovaný polem s indexy vertexů zabírajících 2 byty a o délce rovné počtu trojúhelníků. Následující byte obsahuje index materiálu. Následují už pouze materiály, jimž znovu předchází dva byty s jejich počtem. Celková velikost jedné položky materiálu je 361 bytů. Prvních 32 obsahuje jméno materiálu. Následují 4 pole o délce 4 s hodnotami float. Ty obsahují optické vlastnosti ambientní, diffusní, specularní a emisiivní. To vše je zakončeno jednou hodnotou float pro lesk a druhou pro transparentnost. Další byte slouží k uložení modu. Po něm následuje 128 znaků pro název souboru s texturou a stejný počet poté znovu pro alpha mapu. Tyto údaje se musí načíst a uložit do připravené struktury.

Po načtení modelu je třeba načíst texturu, pokud existuje. Za tímto účelem je vhodné využít knihovnu SDL_image, která poskytne možnost načíst velké množství obrazových formátů.

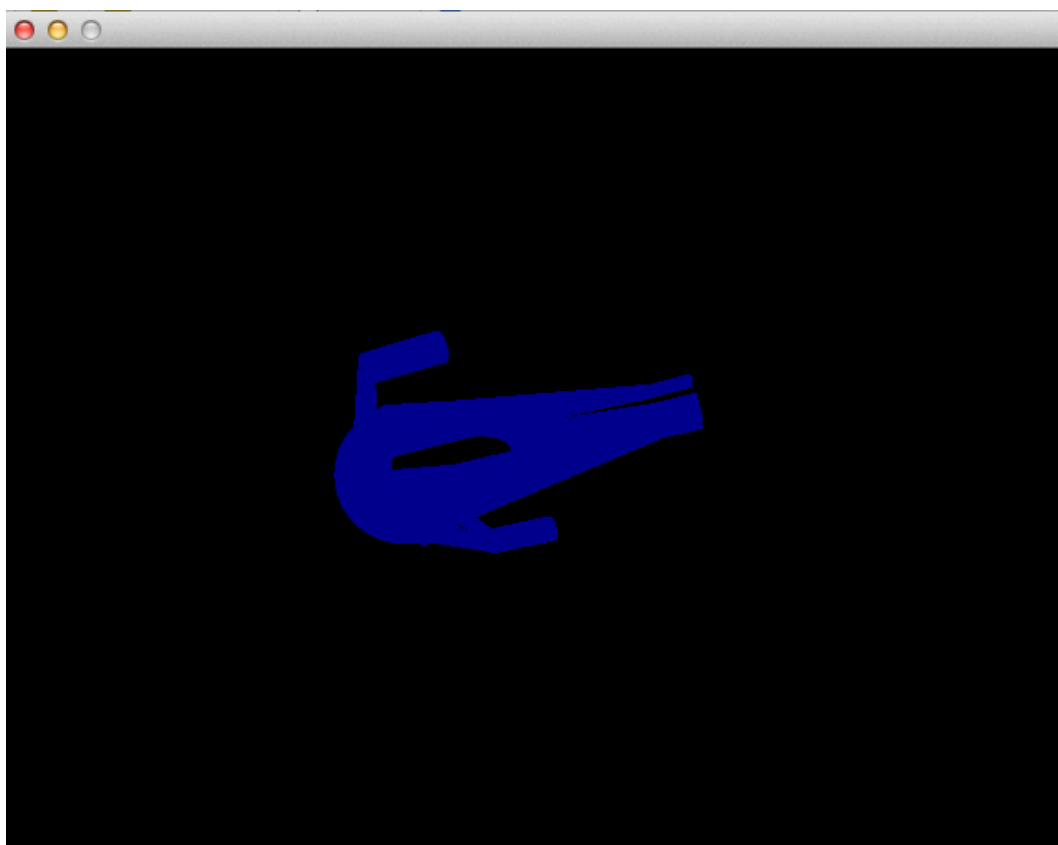


Obrazek 11 - UML class diagram, Model a pomocné třídy

7.3.1 Zobrazení trojrozměrných těles

Po načtení dat je nutné je umět zobrazit. Tato funkce je obsažena v rodičovské třídě třídy MilkShapeModel, tedy v třídě Model. Struktura objektů, kde by ji mohly využívat i

soubory jiných formátů. OpenGL není objektově orientovaná knihovna, ale knihovna využívající principy strukturovaného programování, která pracuje na principu stavového automatu. Z toho důvodu se nejdříve musí nastavit požadované vlastnosti na požadované hodnoty, tedy nastavení optických vlastností zobrazovaných primitiv a tedy i výsledného objektu. Do okamžiku další změny jsou poté všechna grafická primitiva zakreslena s těmito vlastnostmi. Poté již nic nebrání zahájit vykreslování jednotlivých grafických primitiv. V tomto případě pouze trojúhelníků, neboť ty jsou obvykle používány na popis celých modelů. Pro urychlení samotného vykreslování bylo rozhodnuto toto vykreslení provést jako zápis Display listu a následně vždy volat samotný display list.(10)



Obrazek 12 - SDL okno s vykresleným modelem

7.3.2 Přehrávání zvuku

Jak bylo uvedeno již výše, jednotlivé zvuky je důvod přehrát pouze tehdy, pokud je přímým účastníkem těleso pozorovatele. Způsob přehrávání zvuků je založen na principu producent - konzument, kde těleso pozorovatele produkuje žádosti o přehrávání zvuku, které jsou ukládány do pole o fixní délce, zatímco funkce obstarávající spárování konkrétního zvuku se zdrojem je konzument, který z tohoto pole odebírá. V případě přetečení pole se

příští žádost o přehrání zvuku uloží na začátek a tím přepíše nejstarší neuspokojenou žádost.

K vyřešení tohoto problému bylo vytvořeno nové posixové vlákno, které slouží pro konzumenta, zatímco producent pracuje přímo v hlavním vláknu. Tímto řešením vyvstaly nové problémy v podobě synchronizace vícevláknové aplikace. Tyto problémy byly vyřešeny pomocí mutexu hlídajícího zápis a čtení do/z pole. Tím však byla vyřešena pouze část problému, neboť konzument může stále provést pokus o čtení prázdného pole a tím způsobit chybu. Vyřešení tohoto problému bylo vykonáno semaforem, který vpustí konzumenta pouze v případě prázdného pole.

Závěr

Úkolem projektu bylo vytvoření aplikace vesmírného simulátoru. Toho bylo dosaženo, ale projekt v tomto stádiu vývoje stále není plně dokončen a není tedy určen k distribuci mimo potřeby této práce. Aplikace splňuje cíle stanovené na začátku vývoje.

Samotný vývoj aplikace byl obtížný a postavil mne před řadu problémů, při jejichž řešení jsem získal řadu zkušeností. Tyto zkušenosti jsou primárně v oblasti OpenGL, kde jsem začínal pouze se základními znalostmi, které jsem výrazně prohloubil. Díky přechodu z dříve používané knihovny GLUT na modernější SDL, která na rozdíl od GLUT kromě vytvoření okna aplikace, a zprávy vstupů disponuje i funkcemi pro načítání různých multimediálních souborů, což ještě více rozšiřují podpůrné knihovny, jsem získal i nové poznatky o této knihovně. Obdobně jsem získal nové zkušenosti s knihovnou OpenAL a rovněž jsem si prohloubil a procvičil znalosti jazyků C a C++.

Samotná aplikace je funkční, ale stále poskytuje prostor pro rozšíření. Vlivem toho mám v plánu doplnit do aplikace v budoucnu možnost načítání různých formátů 3D modelů. Zdokonalit umělou inteligenci, integrovat zobrazení zjednodušené formy okolního prostředí pro snadnější orientaci do uživatelského rozhraní, umožnit u poškoditelných těles detekci konkrétní části tělesa, jež byla zasažena a dle toho ovlivnit vlastnosti tělesa. Výrazně zdokonalit umělou inteligenci protihráčů, rozdělit skupinu hráčů na týmy schopné spolupráce. Umožnit hru dalším živým hráčům přes počítačovou síť a jejich komunikaci. Upravit jádro aplikace tak, aby bylo schopno distribuovat výpočetní zátěž mezi více strojů v počítačové síti. Dále nevylučuji další možnosti vylepšení, které nejsou uvedeny v tomto výčtu.

Literatura

- 1.KHRONOS GROUP. *The Khronos Group Inc.* [online]. 2012 [cit. 2012-02-20]. Dostupné z: <http://www.khronos.org/>
2. SHREINER, Dave. *OpenGL: průvodce programátora*. Vyd. 1. Brno: Computer Press, 2006, 679 s. ISBN 80-251-1275-6.
- 3.THE KHRONOS GROUP INC. *OpenGL: The Industry's Foundation for High Performance Graphics* [online]. 1997 [cit. 2012-02-21]. Dostupné z: <http://www.opengl.org/>
- 4.CREATIVE TECHNOLOGY LTD. *OpenAL* [online]. 2003 [cit. 2012-02-22]. Dostupné z: <http://connect.creativelabs.com/openal/>
- 5.JIRKOVSKÝ, Jan. *Game industry: vývoj počítačových her a kapitoly z herního průmyslu*. Praha: D.A.M.O., 2011, 135 s. ISBN 978-80-904387-1-2 (BROŽ.).
- 6.TUREK, Michal. OpenGL a Direct3D. *Root.cz* [online]. 2004 [cit. 2012-04-25]. Dostupné z: <http://www.root.cz/clanky/opengl-a-direct3d/>
- 7.TUREK, Michal. OpenGL a Direct3D II. *Root.cz* [online]. 2004 [cit. 2012-04-25]. Dostupné z: <http://www.root.cz/clanky/opengl-a-direct3d-2/>
- 8.Kolize a následná reakce. *České-hry.cz: Komunita herních vývojářů* [online]. 2008, 27.8.2008 [cit. 2012-05-02]. Dostupné z: http://newwiki.ceske-hry.cz/Kolize_a_následná_reakce
- 9.Součinitel restituce. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2012 [cit. 2012-05-02]. Dostupné z: http://cs.wikipedia.org/wiki/Součinitel_restituce
- 10.CZ NeHe *OpenGL: vše o programování 3D grafiky s knihovnou OpenGL* [online]. Michal Turek. 2002 [cit. 2012-05-03]. Dostupné z: <http://nehe.ceske-hry.cz/>
- 11.11.MilkShape 3D 1.8.2: File Format Specification. *Chumbalum soft* [online]. 2007 [cit. 2012-05-03]. Dostupné z: <http://chumbalum.swissquake.ch/ms3d/ms3dspec.txt>
- 12.TIŠNOVSKÝ, Pavel. OpenGL a nadstavbová knihovna GLU. *Root.cz* [online]. 2004 [cit. 2012-05-04]. Dostupné z: <http://www.root.cz/clanky/opengl-a-nadstavbova-knihovna-glu/>

13. Newtonovy pohybové zákony. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-08-08]. Dostupné z: http://cs.wikipedia.org/wiki/Newtonovy_pohybové_zákony
14. REICHL, Jaroslav a Martin VŠETIČKA. *Encyklopedie fyziky* [online]. © 2006 - 2012 [cit. 2012-08-10]. Dostupné z: <http://fyzika.jreichl.com/>
15. 15. Umělá Inteligence. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-08-12]. Dostupné z: http://cs.wikipedia.org/wiki/Umělá_intelligence
16. JURČÍK, Adam. *Techniky programování shaderů* [online]. 2008 [cit. 2012-08-13]. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Petr Tobola. Dostupné z: http://is.muni.cz/th/172993/fi_b/
17. OpenCL. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-08-13]. Dostupné z: <http://cs.wikipedia.org/wiki/OpenCL>
18. Bullet (software). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-08-13]. Dostupné z: [http://en.wikipedia.org/wiki/Bullet_\(software\)](http://en.wikipedia.org/wiki/Bullet_(software))
19. SDL: Hry nejen pro Linux (1). TUREK, Michal. *Root.cz* [online]. 2005 [cit. 2012-08-16]. Dostupné z: <http://www.root.cz/clanky/sdl-hry-nejen-pro-linux-1/>
20. List of games using SDL. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-08-16]. Dostupné z: http://en.wikipedia.org/wiki/List_of_games_using_SDL
21. Simple DirectMedia Layer. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-08-16]. Dostupné z: http://cs.wikipedia.org/wiki/Simple_DirectMedia_Layer
22. APPLE INC. *Apple* [online]. © 2012 [cit. 2012-08-16]. Dostupné z: <http://www.apple.com/>
23. MW3D-SOLUTIONS. *Cheetah3D: Mac 3D Software for modeling, rendering and animation* [online]. © 2001-2012 [cit. 2012-08-16]. Dostupné z: <http://www.cheetah3d.com/>

Příloha A - Přiložené CD

Přiložené CD obsahuje text bakalářské práce, zdrojové kódy, 3D modely a samotnou aplikaci.