VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ROČNÍKOVÝ PROJEKT

PAVEL GUNIA

AKADEMICKÝ ROK 2004/2005

Knihovna pro OpenGL rendering založená na OpenGL Performer API

Pavel Gunia 3.5.2005

Prohlášení

Prohlašuji, že jsem tento ročníkový projekt vypracoval samostatně pod vedením Ing. Jana Pečivy.

Dále prohlašuji, že jsem uvedl všechny literární prameny ze kterých jsem čerpal.

Pavel Gunia

Abstrakt

Projekt se zabývá problematikou zobrazení grafické informace jako prostředku komunikace a toku informací od počítače k člověku. Je zaměřen především na popis a zobrazení událostí a scén v trojrozměrném prostoru. Analýza je založena na studiu profesionální grafické knihovny OpenGL Performer, obecných technik a metod, i vlastních inovacích.

Výsledkem je návrh a implementace vlastní grafické knihovny pro snadnou a efektivní tvorbu grafických aplikací.

Klíčová slova

grafická knihovna, zobrazení prostorových dat, OpenGL, Microsoft DirectX, OpenGL Performer, hierarchická databáze scény, objektově orientovaný návrh, optimalizace, aplikační fáze, ořezávací fáze, vykreslovací fáze, multiplatformová implementace, abstraktní třída

Obsah

1. Úvod

Ve vnímání člověka je drtivá většina podnětů zrakových. Je přirozené, že je každý člověk svým způsobem citlivý na vzhled a design věcí okolo něj. S rozvojem počítačů a zvyšováním výpočetní síly nabývá grafika na významnosti. Zároveň jde o vliv společnosti a snahu upoutat na produkt nebo dílo. V nasyceném trhu je efektivním prostředkem upoutání pozornosti právě design.
Rozvoj a aplikace počítačů do stále více aspektů našeho života dalo vzniknout množství technik, postupů a rutin k zobrazení potřebných dat způsobem pro člověka co nejpřijatelnějším a nejpříjemnějším. Jsou zde letecké simulátory pro výcvik pilotů, lékařské přístroje umožňující doktorům pracovat na dálku, rozličné měřící a zaznamenávací přístroje, systémy virtuální reality či hry které musí oku prezentovat své výsledky intuitivním a srozumitelným způsobem. V neposlední řadě jde také o reklamu, která využívá lidské citivosti ke grafice.

Protože jsou rutiny a postupy pro prezentaci dat specifické pro každý typ aplikace, vznikly, a vznikají, knihovny grafických nástrojů a technik zapouzdřující problematiku zobrazení dat dané specifické oblasti. Jelikož je náš svět trojrozměrný, patří právě knihovny pro zobrazení prostorových virtuálních dějů či světů k nejrozšířenějším a zároveň k nejobecnějším.

Za nejjednodušší grafickou knihovnu se dá považovat ovladač grafické karty. Specifikace jeho API poskytuje rozhraní mezi strojovým kódem hardwaru a přijatelnějšími programovacími jazyky vyššího stupně. Mezi nejrozšířenější a výrobci hardwaru podporované specifikace patří Microsoft **DirectX**, poskytující API pro C++, VisualBasic, Jawu, Delphi a další, a **OpenGL** firmy Silicon Graphics pro jazyk C. Na těchto specifikacích staví vyšší grafické knihovny, které poskytují rozhraní k ještě jednoduššímu a efektivnějšímu programování. Takto vrstvená abstrakce dovoluje vývojářům soustředit se co nejvíce na vlastní aplikaci.

Obecně je základem grafické knihovny vyššího stupně databáze obsahující informace o modelovaném prostředí *(angl. scenegraph)*, zobrazovací algoritmus *(angl. rendering pipeline)* a umístění jednoho či více pozorovatelů.
Databáze je typicky objektová ve formě hierarchického stromu vyjadřujícího závislosti mezi jednotlivými objekty. Objekty jsou buď reprezentace vlivů a předmětů jak je známe z reálného světa (jako tvary, světla, částice) nebo abstraktní objekty reprezentující vnitřní stavy a chování modelovaného světa (jako pohyby, animace, reakce na podněty apod.).

Zobrazovací algoritmus provádí podle programátorem specifikovaných požadavků sérii sousledných rutin pro zobrazení výsledného obrazu na výstupní zařízení. Důležitou součástí jsou optimalizace v závislosti na použitém hardware a specifikace programátora. Typicky jde o reorganizaci hierarchického stromu databáze tak, aby se nevykreslovaly části které jsou zcela mimo pohled pozorovatele, úprava detailu zobrazení pro vzdálené objekty a seřazení vykreslovacích příkazů tak, aby bylo co nejefektivněji využito grafické vybavení.

2. OpenGL Performer

2.1 Co je to OpenGL Performer

OpenGL Performer firmy Silicon Graphics (SGI) je multiplatformová robustní knihovna pro tvorbu výkonných a efektivních grafických aplikací. Je postavena na vlastní specifikaci grafického rozhraní OpenGL firmy SGI. V technické praxi jde například o vizualizace přesných fyzikálních, chemických, matematických a jiných simulací, u filmových efektů zase o věrohodné zobrazení s fotografickou kvalitou. V neposlední řadě jsou hry a systémy virtuální reality, pro které je dostatečná rychlost zobrazení nezbytná, kvalita obrazu pak určuje kvalitu vizuálního zážitku. Obecně, aplikace které vyžadují vysoký grafický výkon, kvalitní obrazový výstup a přitom co nejvyšší rychlost zobrazení budou úspěšně využívat velkého potenciálu Performeru.

OpenGL Performer razantně snižuje práci potřebnou k vyladění aplikace na maximální grafický výkon. Optimalizace zahrnují vysoce efektivní rutiny pro vyladění kritických míst a reorganizace grafických dat a operací pro rychlejší zobrazení. OpenGL Performer také obsahuje nástroje pro optimalizace závislé na architektuře počítače a využití více procesorů najednou. Jako součást SGI poskytuje sofistikovaný vizualizační systém ve výkonné, flexibilní a rozšiřitelné formě.

2.2 Application Programming Interface

OpenGL Performance API je jak objektově orientované, obsahující třídy a jejich metody, tak i čistě strukturované pro použití v C.

OpenGL Performance API používá konvence pro pojmenování funkcí, metod a identifikátorů které samy indikují funkci daného příkazu.
Prefix názvu funkce nebo třídy říká ve které knihovně se daný příkaz nebo třída nachází. Všechny příkazy a třídy standardní knihovny OpenGL Performer začínají na **„pf“**.
Knihovny přídavných funkcí používají k standardnímu prefixu písmeno navíc, jako **„pfu“** pro knihovnu utilit libpfutil, **„pfi“** pro knihovnu vstupních operací libpfui a **„pfd“** pro knihovnu pro přístup k databázím.

Každá metoda třídy pro programování v C++ má svou reprezentaci strukturovanou funkcí pro použití v C. Obecně, název strukturované funkce zahrnuje název třídy, ke které je doplňkem.

```
C:    pfGetPipeScreen();
C++:  pipe->getScreen();
```

V některých případech obecných rutin je jméno třídy vynecháno.

```
C:    pfAddChild(node, child);
C++:  node->addChild(child);
```

2.3 Principy OpenGL Performer

OpenGL Performer a jeho pojetí jak virtuálního světa tak i zobrazovacích technik a nástrojů je implementován ryze objektově. Veškeré objekty, rutiny a techniky, které se podílí na tvorbě virtuálního světa a jeho zobrazení jsou obsaženy v tzv. visuální databázi (dále jen jako databáze), která reprezentuje modelovaný virtuální svět i druh a parametry jeho zobrazení.

Standardní knihovna **libpf** obsahuje systém pro práci s databází viruálního světa a zobrazení dané databáze. Kořenem je třída **pfScene** která jako své potomky obsahuje celou hierarchii scény.
Objekty databáze se dělí do 2 kategorií podle toho, zda-li reprezentují vzhled a chování virtuálního světa nebo zda-li určují jeho zobrazení.

Objekty reprezentující vzhled a chování scény

Vycházejí z abstraktní bázové třídy **pfNode**.

pfNode je třída implementující techniky pro:

- seznam otcovských objektů
- ohraničující geometrii (bounding geometry)
- callback funkce a data
- uživatelsky definovaná data

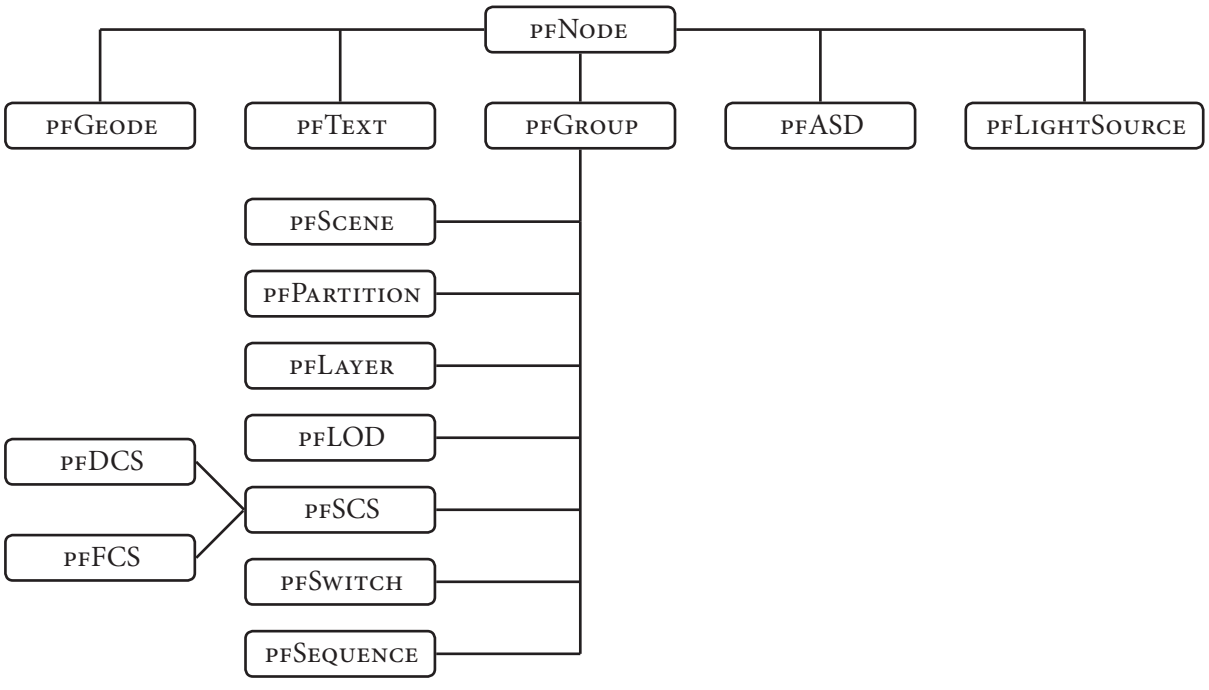
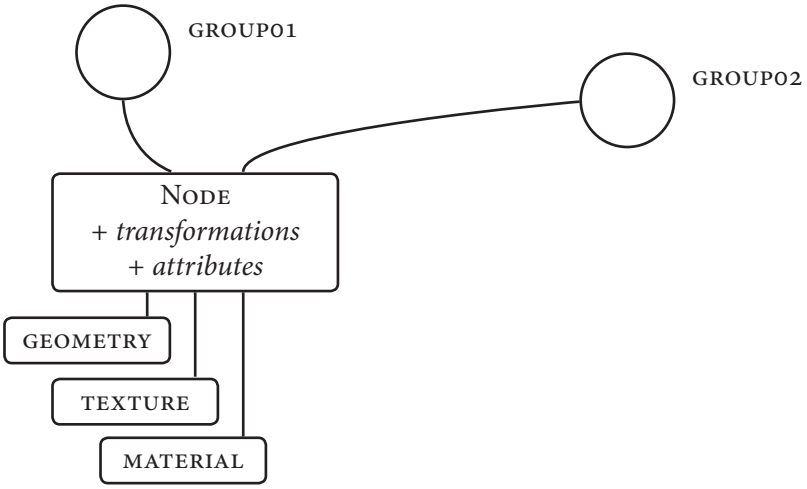


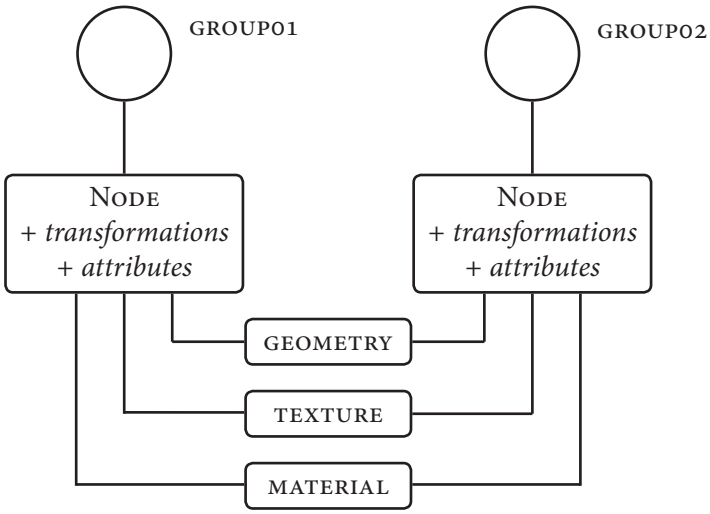
Diagram hierarchie tříd pro reprezentaci vzhledu scény

Z třídy **pfNode** vycházejí:

pfGroup implementuje systém potomků pro vytváření hirearchií. Každý potomek je opět typu pfGroup. Každý objekt scény může být součástí jednoho i více objektů pfGroup. OpenGL Performer umožňuje 2 druhy vytváření instancí.
Sdílené instance jednoduše ukazují na stejný objekt scény.



Proto se veškeré operace nad tímto objektem projeví u obou instancí. Tento efekt je v někerých případech nežádoucí.
Klonované instance vytvářejí kopie objektů které mění své atributy a sdílené instance těch objektů, které jsou statické.



pfGeode je zkratkou pro "geometry node" a je bázovou třídou pro objekty reprezentující geometrii a tvary. Z této třídy vychází pfGeoSet, která navíc implementuje definice materiálů a textur tomuto objektu.

pfLightSource reprezentuje světelný zdroj.

pfASD implementuje dynamické vytváření a prolínání viditelné části geometrie podle rozdílých stupňů detailu. Zajištuje plynulou změnu detailu u rozlehlých a komplexních povrchů. Typickým využitím je rozlehlý terén se sníženým detailem v pozadí.

pfLOD je objekt který mění svůj detail v závislosti na vzdálenosti od pozoroatele. Kromě svých potomků má navíc seznam objektů typu pfGeometry, které definují tvar pro různé stupně detailu.

pfPartition podobně jako pfGroup imlementuje systém potomků. Ti jsou seřazeni do statické struktury efektivnější pro výpočty průsečíků a urychlující vykreslování.

pfSCS reprezentuje statický systém souřadnic. Obsahuje statickou transformační matici která nemůže být po vytvoření změněna. Slouží k umístění objektů v prostoru virtuálního světa.

pfDCS reprezentuje dynamický systém souřadnic. Transformační matice může být za běhu programu libovolně měněna. Typicky se používa k vyjádření pohybu objektů.

pfSwitch je objekt, který vybírá jeden, všechny nebo žádný ze svých potomků.

pfSequence je objekt který postupně vybírá své potomky, zobrazující každého po určitou dobu. Každý jeho potomek si lze představit jako snímek animace.

Objekty reprezentující zobrazení

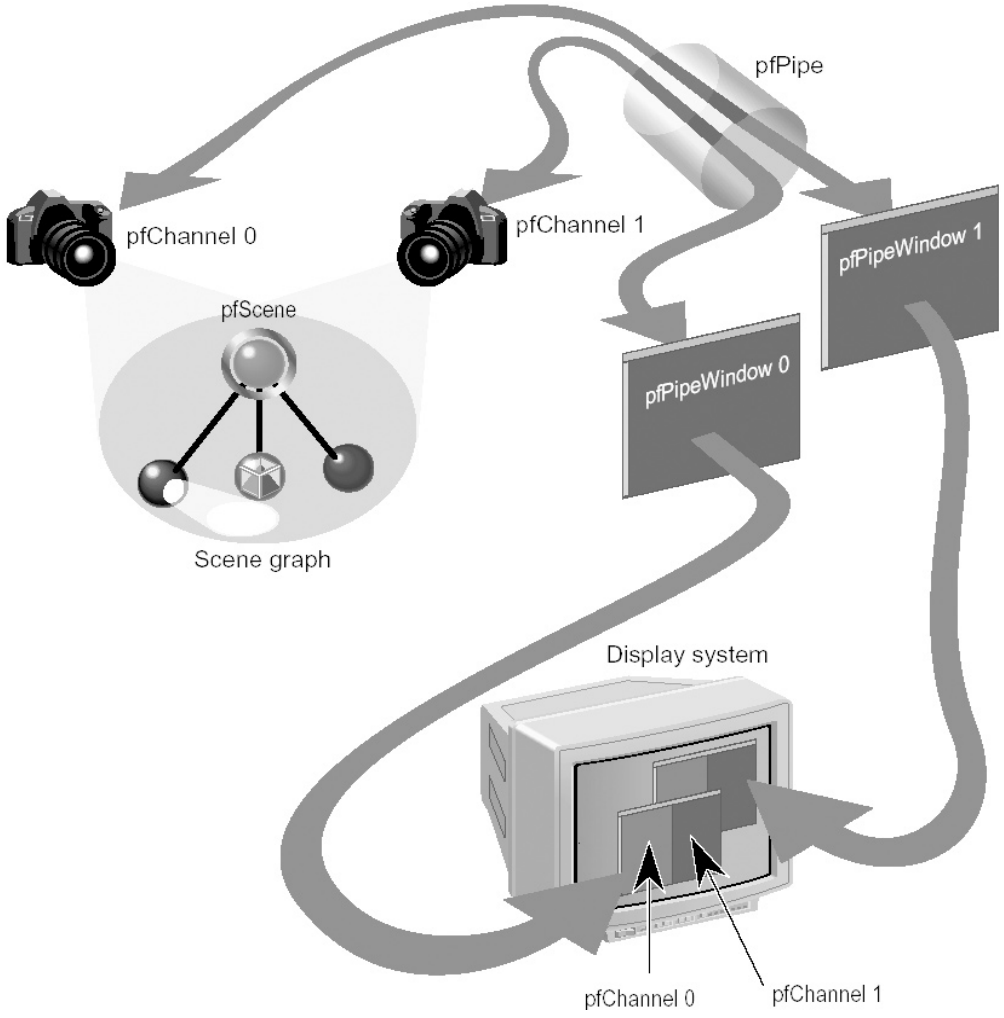
Základním krokem zobrazení virtuálního světa je projekce prostorových souřadnic do dvourozměrnou prostoru, který reprezentuje obrazový výstup. Pro definici projekce je implementována třída **pfChannel**.

pfChannel je objektem databáze a reprezentuje pozorovatele. Obsahuje projekční a transformační matici. Každá scéna musí obsahovat alespoň jeden objekt typu pfChannel.

Projekce scény je pak vykreslena objektem **pfPipe** do framebufferu.

Třída **pfPipeWindow** obsluhuje framebuffer a jeho zobrazení do okna.

Sled událostí při vykreslování scény



Vykreslovací pipeline (pfPipe)

Vykreslovací pipeline je technikou samotného zobrazení scény, tedy obsahu visuální databáze. Jedná se o nástroj zapouzdřující jak hardwarově akcelerované rutiny tak i optimalizace na úrovni softwarového programování. Míra mezi hardwarově akcelerovanou částí a softwarovým výpočtem je závislá na použitém grafickém vybavení.
Každá vykreslovací pipeline kreslí do jednoho nebo více oken. Skládá se ze tří po sobě jdoucích funkčních fází.

APP aplikační fáze

V této fázi provádí aplikace své specifické algoritmy. Dochází k aktualizaci visuální databáze, atributů jednotlivých objektů, upravují se transformační matice podle hierarchických závislostí případně se řeší specifické simulační algoritmy. Animace a pohyb objektů se typicky řeší v této fázi. Obecně jde o všechny změny viditelné ve výsledném obraze.

CULL ořezávací fáze

Provádí se průchod databází a určuje se, které části scény jsou potenciálně viditelné a provádí se výpočet detailu objektů které toto podporují. Významnou částí této fáze je reorganizace dat za účelem optimalizace managementu vnitřních stavů vykreslovacího stroje a generování postupů a seznamu objektů pro konečné vykreslení.
Tato fáze je převážně optimalizační a tudíž kritická. Velice často bývá řešena na úrovni softwarového programování.

Průchod databází se provádí DFS metodou (depth first search) pro každý aktivní objekt pfChannel. Hierarchie objektů uchovává zároveň i informaci o vzájemné poloze a průniku otcovských objektů a potomků. Takto mohou nastat tyto možnosti:

všichni potomci jsou uvnitř ohraničující geometrie objektu

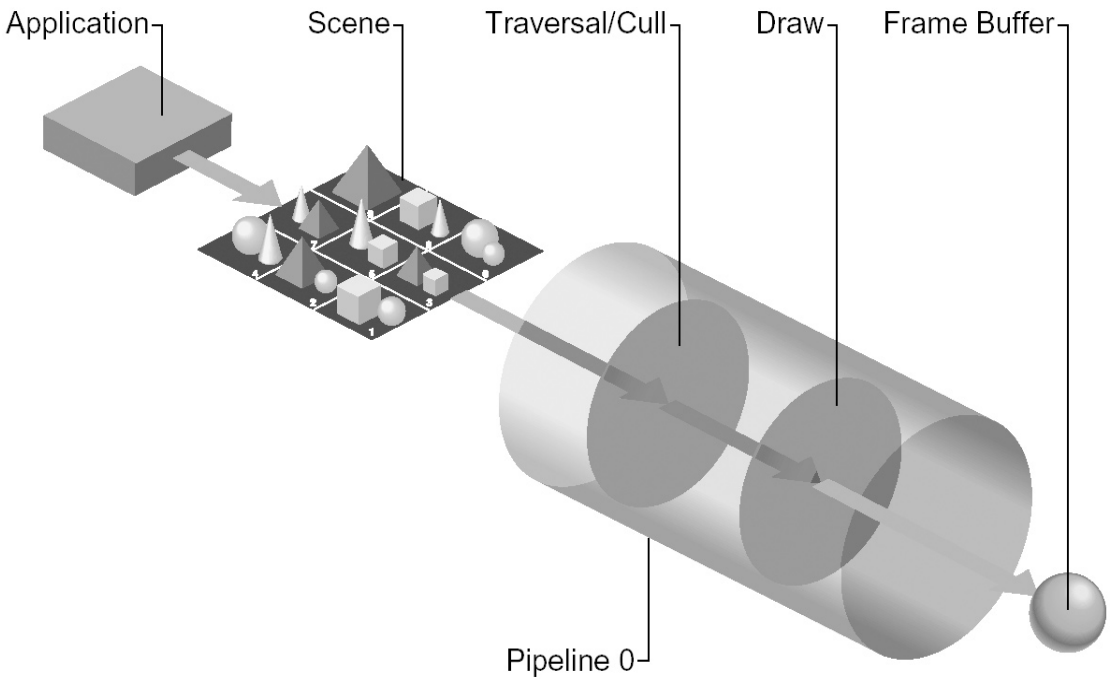
Pro všechny potomky jsou nastavena stejná pravidla pro viditelnost jako otcovskému objektu, větev stromu se již dále neprochází a pokračuje se další.

některý z potomků neleží uvnitř ohraničující geometrie

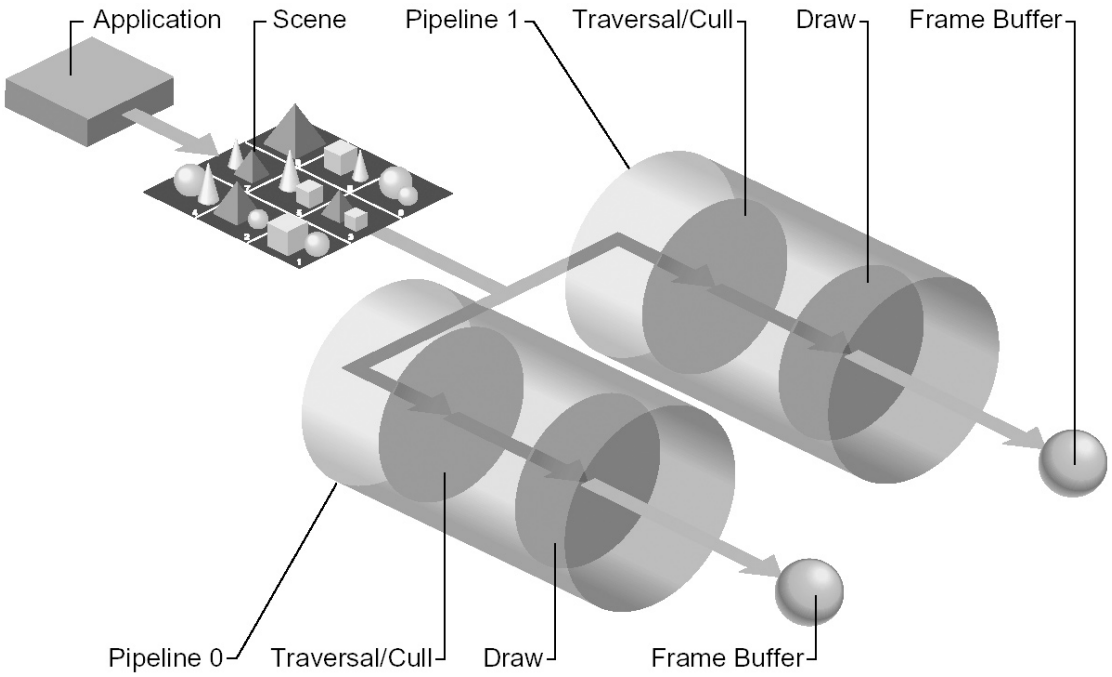
Pro všechny potomky ležící uvnitř ohraničující geometrie objektu jsou nastavena stejná pravidla pro viditelnost jako otcovskému objektu, větev stromu pokračuje pro zbylé potomky.

DRAW VYKRESLOVACÍ FÁZE

Proces vykonání požadavků sestavených předchozí fází. Objekty jsou vykreslováný voláním příkazů ovladače grafického hardwaru. Tato fáze je téměř vždy řešena grafickým hardwarem.



Vykreslovací proces systému s jednou vykreslovací pipeline. V aplikační fázi aplikace vytváří a mění definici scény (visuální databáze) a pozorovatele skrze pfChannel objekt.
V ořezávácí fázi se ze získaných atributů pfChannel objektu určuje, které části scény jsou viditelné. Vytvořený seznam pokynů je vstupem poslední, vykreslovací fáze, která dané příkazy vykoná.



Systém s dvěma vykreslovacími pipeline. Každá pipeline je nezávislá, ořezávací a vykreslovací fáze probíhají odděleně.
Aplikační fáze je probíhá ještě před dělením na více pipeline, protože se jedná o tutéž scénu. Tímto zpracovávají všechny pipeline stejnou visuální databázi.

Každá fáze může být vykonána jedním procesem nebo rozdělena na více procesů pro zlepšení výkonu na víceprocesorových systémech.

2.4 Práce s OpenGL Performer

```
#include <math.h>

// standartní knihovna OpenGL Performer
#include <performer/pf.h>

// knihovna databázových operací
#include <performer/pfdu.h>
```

```
int main( int argc, char *argv[])
{
        // jedna pipeline pro zpracování scény
        pfPipe          *pipe;

        // okno pro výstup
        pfPipeWindow  *pw;

        // kořenový objekt databáze
        pfScene                 *scene;

        // jeden pozorovatel
        pfChannel               *channel;

        // reprezentace vlastních objektů
        pfNode                  *model1;

        // jejich transformace
        pfDCS           *node1;

        // model ze souboru
        char                    *file = "file.obj"


        pfInit();

        // cesty pro filesystem Performeru
        pfFilePathv(
                ".",
                "./data",
                NULL);

        // jednovláknově
        pfMultiprocess(PFMP_DEFAULT);

        pfConfig();

        // model nahrajeme ze souboru
        model1 = pfdLoadFile(file);

        // vytvoříme dynamický souřadnicový systém
        node1 = pfNewDCS();

        // pro náš model
        pfAddChild(node1, model1);
```

```
        // implicitní pipeline
        pipe = pfGetPipe(0);

        // nové okno
        pw = pfNewPWin(pipe);
        pfOpenPWin(pw);

        // alespoň jeden pfChannel (pozorovatel)
        channel = pfNewChan(pipe);

        // zařadit do scény
        pfChanScene(chan, scene);


        // nastavit viewport
        pfSetVec3(view.xyz, 0.0f, 0.0f, 15.0f);
        pfSetVec3(view.hpr, 0.0f, -90.0f, 0.0f);
        pfChanView(channel, view.xyz, view.hpr);


        // spustí vykreslovací pipeline
        pfFrame();

        sleep(3);

        // konec
        pfExit();

}
```

3. VLASTNÍ IMPLEMENTACE

NÁVRH

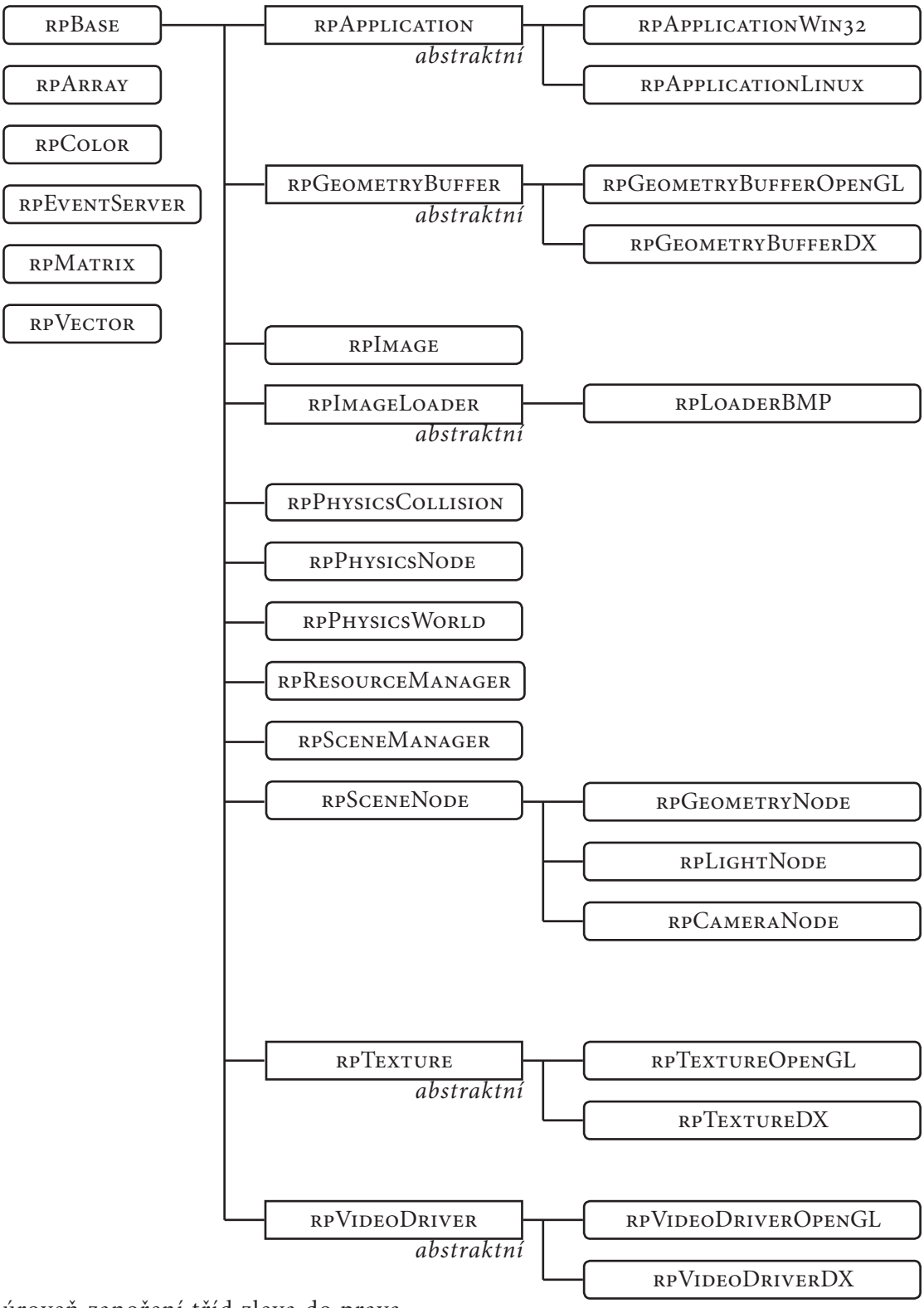
Implementace vlastní knihovny vychází jak z obecných technik tak z principů knihovny OpenGL Performer. Nicméně byly použity odlišné konvence pro pojmenování tříd a metod stejně tak vlastní implementace dílčích algoritmů. OpenGL Performer je licensován pro komerční využití, proto nejsou jeho algoritmy a techniky veřejně přístupné.

Stejně jako OpenGL Performer je návrh pojat multiplatformově s zaměřením na systémy Microsoft Windows a Linux. Rozšířením nad rámec OpenGL Performer je návrh a implementace zobrazovacího stroje využívajícího jak specifikaci OpenGL tak i Microsoft DirectX. V této části je výrazná odlišnost od OpenGL Performer, který využívá čistě specifikace OpenGL.

PRINCIPY

Narozdíl od OpenGL Performer se jedná o čistě objektově orientovanou knihovnu. Elementární třídou je třída **rpBase** sloužící jako základ všem ostatním. Implementuje jak jednoduchý systém pro správu paměti a hlídáni korektnosti ukazatelů tak techniky pro účely ladění. Všechny třídy jsou odvozeny právě z této. Výjimkou jsou jednoduché třídy které správu paměti nepotřebují, jako třídy reprezentující matice, vektory a podobně.

Graf hierarchie tříd



úroveň zanoření tříd zleva do prava

Multiplatformová implementace

Základem implementace podpory pro více operačních systémů či architektur počítače je abstraktní bázová třída **rpApplication**. Obsahuje rozhraní pro obecný model aplikace. Převážně se jedná o práci s okny a jednoduché grafické operace, zprávy operačního systému,obsluhu uživatelských vstupů a zajištění běhu programu pod operačním systémem.

Z této odvozená třída **rpApplicationWin32** resp. **rpApplicationLinux** implementuje funkčnost pod operačním systémem Microsoft Windows resp. Linux.

poznámka: Podpora operačního systému Linux ještě není implementována.

O úroveň výše je specifický vykreslovací stroj pro zpracování trojrozměrné grafiky. Variantou k vykreslovacímu algoritmu pfPipe v OpenGL Performer je třída **rpVideoDriver**. Tato abstraktní bázová třída poskytuje rozhraní pro práci s ovladači grafického vybavení.

Odvozená třída **rpVideoDriverOpenGL** resp. **rpVideoDriverDX** implementuje algoritmy pro práci s ovladači grafického zařízení pod specifikací OpenGL resp. Microsoft DirectX.

poznámka: Podpora specifikace Microsoft DirectX ještě není implementována.

V rámci využití potenciálu grafického hardwaru uchovávat textury resp. geometrii ve vlastní paměti jsou navrženy abstraktní bázové třídy **rpTexture** resp. **rpGeometryBuffer**. Poskytují rozhraní pro práci s daty potenciálně umístěnými v paměti grafického zařízení.

Ze třídy **rpTexture** je odvozena **rpTextureOpenGL** resp. **rpTextureDX** implementující postupy a techniky pro práci s texturami pod specifikací OpenGL resp. Microsoft DirectX. Obecně obsahuje metody pro nahrání textury do paměti a následné vyvolání při samotném zobrazovacím procesu.

Ze třídy **rpGeometryBuffer** vychází třída **rpGeometryBufferOpenGL** resp. **rpGeometryBufferDX** která implementuje techniky pro nahrávání a vykreslování geometrického objektu grafickým ovladačem pod specifikací OpenGL resp.

Tento přístup představuje znatelné zrychlení na moderních grafických kartách.

Technika vykreslování

Třídy **rpTexture**, **rpGeometryBuffer** a **rpVideoDriver** poskytují rozhraní pro veškeré grafické operace a komunikaci s ovladačem grafického zařízení.

O úroveň výše je třída **rpSceneManager** reprezentující celou scénu. Implementuje jak metody pro práci se scénou tak i techniky pro optimalizaci vykreslování. Visuální databáze má opět strukturu hierarchického stromu s vyjádřením závislostí otec-syn. Narozdíl od OpenGL Performer však neobsahuje objekty pro definici zobrazovacích technik. Vykreslování probíhá pouze v jedné pipeline.

Každý objekt v databázi je odvozen ze třídy **rpSceneNode**, která implementuje dynamický systém souřadnic, transformace a atributy objektu. Odvozené třídy **rpGeometryNode**, **rpLightNode** resp. **rpCameraNode** implemetují definice tvarů, světelných zdrojů resp. pozorovatelů ve scéně.

Vykreslování probíhá podobně jako v OpenGL Performer. Aplikační a ořezávací část je sloučena tak, že dochází pouze k jednomu průchodu databází. Postupně se provádějí transformace objektů v závislosti na otci a vlastní transformaci. Následně jsou ořezány větve které nejsou potenciálně viditelné nebo je jejich zobrazení zakázáno uživatelem. Seznam objektů připravených k vykreslení je následně optimalizován.

Grafické zařízení obsahuje systém pro obsluhu vnitřních stavů. Veškeré změny ve způsobu vykreslování se provádějí právě změnou vnitřních stavů. Změny některých stavů jsou náročnější než změny jiných. Obecně je časově nejnáročnější změna textury a mateiálu. Není-li daná textura v paměti grafického zařízení, musí se kopírovat. Proto je důležitou částí optimalizace seřazení pokynů pro grafické zařízení tak, aby docházelo k co nejméně takovým změnám.

poznámka: Implemetována je pouze optimalizace přepínání stavů použitých textur.

Obraz produkovaný během vykreslovací fáze je ukládán do tzv. framebufferu. Konečnou fází zobrazovacího procesu je vykreslení framebufferu na výstupní zařízení. Tím je zaručena plynulost vykreslování zřejmá především u pohyblivých objektů a animací.

Pro účely statistik a měření výkonu je imlementován systém měření časové náročnosti vykreslení snímku. Klasicky se rychlost zobrazení udává v počtu zobrazených snímků za sekundu. Pro plynulou animaci a pohyb je nezbytné dosáhnout alespoň 30 snímků za vteřinu.

Sdílené prostředky

V rámci úspory paměti a urychlení vykreslování je vytvořen systém pro správu a sdílení prostředků, které nemění své vnitřní stavy a vzhled. Jmenovitě jde o textury a geometrii.

Třída **rpResourceManager** imlementuje systém pro práci se soubory a načítání textur a geometrie. Každý načtený prostředek je uveden v seznamu. Při dalším dotazu na tentýž prostředek je vrácen ukazatel na již načtený objekt.

Systém pro zpracování fyziky

Knihovna implementuje rozhraní pro freewarový fyzikální systém **Newton Collisions**. Jde o pokročilý systém s věrohodným fyzikálním modelem.
Rozhraní poskytují třídy **rpPhysicsWorld**, **rpPhysicsCollision** a **rpPhysicsNode**.

rpPhysicsWorld

Představuje izolovanou komplexní oblast pro simulaci fyzikálních dějů. Veškeré objekty se zde mohou navzájem ovlivňovat, nepůsobí však žádné podněty zvenčí. Scéna může obsahovat více takovýchto oblastí.

rpPhysicsCollision

Představuje tvar kolizní geometrie objektu. Nemusí se shodovat s visuálním tvarem objektu, často jsou složité polygonální tvary simulovány jednoduššími tvary typu koule, válec a podobně.
Toto jsou taktéž sdílené prostředky.

rpPhysicsNode

Představuje dynamické vlastnosti modelovaného objektu. Převážně se jedná o hmotnost, těžiště, momenty setrvačnosti, pozici a působící síly.

Dále implementuje spojení mezi fyzikálním a grafickým systémem. Výsledná transformace fyzikálního modelu se přenáší na specifikovaný objekt scény reprezentující vizuální stránku modelu.

Práce se scénou

Aplikace vychází z instanciace třídy rpApplicationWin32 resp. rpApplicationLinux pod operačním systémem Microsoft Windows resp. Linux. Je vytvořeno okno a podle výběru instanciována třída rpVideoDriverOpenGL resp. rpVideoDriverDX pro práci s grafickým zařízením pod specifikací OpenGL resp. Microsotf DirectX. Následně je vytvořen manager scény s prázdnou databází.

Třída **rpSceneManager** byla navržena tak, aby co nejvíce zjednodušila modelování virtuálního světa. Obsahuje metody pro vkládání objektů do databáze zapouzdřující správu sdílených prostředků a nastavení implicitních parametrů. Každá scéna musí obsahovat minimálně jednu kameru, tedy objekt třídy **rpCameraNode**. Ten vychází z bázové třídy **rpSceneNode** a je proto umístitelný do hierarchie databáze. Všechny transformace jeho předchůdců i jeho vlastní se promítnou na výsledném umístění v prostoru. Dále se do databáze umístují objekty představující tvary resp. světelné zdroje. Jsou typu rpGeometryNode resp. rpLightNode vycházející z rpSceneNode.

Všechny objekty v databázi mohou mít libovolný počet potomků a vytvářet tak hirearchii. Ta se promítne na transformaci a obecných atributech jako je příznak viditelnosti a podobně.

4. Závěr

OpenGL Performer je robustní a výkonná grafická knihovna jejíž síla je z veliké míry v efektivním vícevláknovém provozu a využití více procesorů pro zpracování a produkci obrazu. Specifikace uvádí efektivní využití až 64 procesorů. Je zřejmé zaměření na tvorbu grafických nástrojů pro výkonné grafické stanice či vědecké simulace. Jednoprocesorový počítač běžný pro většinu uživatelů využije jen malé procento výkonnosti této knihovny.

Tento projekt byl naopak zaměřen na využití u běžných osobních počítačů. Cílem bylo vytvořit knihovnu pro jednoduchou práci s trojrozměrnou grafikou a s dostatečným výkonem. Vícevláknový provoz nebyl zvažován již z důvodu komplexnosti a složitosti tohoto tématu. I v dnešní době je však většina moderní osobních počítačů vybavena pouze jedním procesorem. S rychlým rozvojem grafického hardware a jeho možností šlo převážně o implementací optimalizačních postupů a technik. Po analýze procesu zpracování obrazu grafickým hardware a prostudování obecných technik byla navržena a poté implementována vlastní grafická knihovna. Její využití je především v oblasti rychlého zobrazování trojrozměrných scén bez nutnosti fotorealistické kvality obrazu jako jsou hry, jednoduché systémy virtuální reality nebo zobrazení vědeckých simulací které nevyžadují vysokou kvalitu obrazu. Bylo dosaženo dobrých výsledků, kdy se rychlost zobrazení průměrně složité scény pohybuje okolo 100 snímků za vteřinu. Přínosná je i jednoduchost použití, kdy lze scénu namodelovat ve velmi krátkém čase.

Do budoucna je na tomto projektu stále hodně práce. Již při návrhu bylo počítáno s dalším vývojem a proto je knihovna snadno rozšiřitelná. Rychlý vývoj grafického hardware přináší stále více jak možností optimalizace zpracování a zobrazení informací tak i prostředků ke zvýšení kvality výsledného obrazu. Použití vertex a pixel shaderů je další nedílnou součástí moderní a kvalitní grafické knihovny. I multiprocesorový provoz nabývá na významu a v budoucnu jistě bude nedílnou součástí každé aplikace. Bude jistě zajímavé se zaměřit i na tuto problematiku.

Nakonec, nebo spíše na začátku dalšího rozšiřování možností, technik a algoritmů této knihovny stojí implementace Linuxové verze aplikačního rozhraní a podpora specifikace Microsoft DirectX.

5. Literatura

OpenGL Performer

OpenGL Performer Getting Started Guide

OpenGL Performer Programming Guide

OpenGL a Microsoft DirectX

www.opengl.org/documentation/blue_book_1.0/

www.opengl.org/resources/tutorials/

nehe.gamedev.net

Obecné

Tomas Akenine-Moller, Eric Haines
Real-Time Rendering

Randima Fernando
GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics

DEMONSTRAČNÍ ZDROJOVÝ KÓD

Hlavičkový soubor knihovny

```
#include "rpLib.h"


int main(int argc, char* agrv[])
{
```

Instanciace třídy pro správu běhu aplikace na úrovni operačního systému. Parametry určují specifikaci užitého grafického rozhraní, velikost okna a volbu režimu celé obrazovky.

```
rpApplicationWin32 app(E_VD_OPENGL, 640, 480, false);
app.setCaption("Rocnikovy projekt");
```

Aplikace sama vytvoří objekt typu rpVideoDriver pro práci s grafickými funkcemi a rpSceneManager pro práci se scénou.
Obecně není ukazatel na objekt typu rpVideoDriver potřebný, veškeré operace se provádějí přes manager scény. Zde je kvůli demonstraci statistických funkcí.

```
rpSceneManager*     smgr    = app.getScenemanager();
rpVideoDriver*      driver  = app.getVideoDriver();
```

Vytvoření oblasti pro zpracování fyzikálních dějů. Nepovinné pokud aplikace nevyužívá zabudovaný fyzikální systém.

```
smrg->newPhysicsWorld();
```

Každá scéna musí obsahovat minimálně jednu kameru. Parametr funkce newCameraNode je ukazatel na otcovský objekt (NULL pro kořen databáze).

```
rpCameraNode*       camera = smgr->newCameraNode(NULL);
```

Posunout kameru od středu.

```
camera->move(E_AXIS_Z, -5.0);
```

Jednoduchý kolizní tvar, kostka o velikosti 1

```
rpPhysicsCollision*      box     = smgr->newPhysicsCollision( E_PC_BOX,
                                                              1.0f,1.0f,1.0f);
```

Geometrie reprezentující tvar naší kostky, načteno ze souboru geometry.oo, použitá textura ze souboru texture.bmp. Otcem objektu je kořen databáze.

```
rpGeometryNode*      box01   = smgr->newGeometryNode(  NULL,
                                                       "geometry.oo",
                                                       "texture.bmp");
```

Spojení mezi fyzikálním modelem a jeho prafickou reprezentací.

```
rpPhysicsNode*       pbox01 = smgr->newPhysicsNode(box, box01);

rpGeometryNode*      box02   = smgr->newGeometryNode(  box01,
                                                       "geometry.oo",
                                                       "texture.bmp");
```

Světelný zdroj, otcem je objekt box02, bude tedy „sledovat" jeho transformace

```
smgr->newLightNode(box02)->move(2.0,2.0,0.0);
```

Nastavení intervalu pro statistické funkce, v ms.

```
driver->setStatsInterval(1000);

while (app.run())
{
```

Působící rotační síla na fyzikální objekt.

```
    pbox01->setTorque(rpVector3D(1.0,0.0,0.5);
```

Tento objekt nemá přidružený fyzikální model proto s ním můžeme manipulovat přímo.

```
    box02->rotateRel(E_AXIS_Y,  0.1);
```

Aktualizace fyzikálního systému.

```
smrg->updatePhysics();
```

Příprava na zahájení vykreslování, prametry specifikují, chceme-li vyprázdnit backbuffer, zbuffer a barvu pozadí.

```
smgr->beginScene(true, true, rpColorf(0.5,0.5,1.0));
```

Vykreslení celé databáze. Vyvolá aplikační, ořezávací a vykreslovací fázi.

```
smgr->drawAll();
```

Ukončení vykreslování, okopírování framebufferu na výstupní zařízení.

```
smgr->endScene();
```

Zobrazení statistiky počtu snímků za vteřinu.

```
app.setCaption(IntToStr(driver->getLastFramesCount()));
```

```
    }

}
```