Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Alternativní způsoby vývoje počítačových her a grafických aplikací

Martin Mikloš

Bakalářská práce
2010

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2009/2010**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin MIKLOŠ**
Osobní číslo:	**I07928**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Alternativní způsoby vývoje počítačových her a grafických aplikací**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Práce se bude zabývat možnostmi alternativní tvorby grafických aplikací.
V teoretické části se budu zabývat popisem a srovnáním prostředků pro vytváření grafických aplikací a porovnáním práce v jednotlivých softwarech. Na vybraném software detailně popíši práci a principy fungování produktu.
V praktické části ve vybraném software vytvořím grafickou aplikaci a porovnám její funkčnost s dostupnými programy.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

***Moderní počítačová grafika, 80-251-0454-0, Jiří Žára a kol,**
***Jazyky C a C++, 80-247-1494-9, Miroslav Virius**

Vedoucí bakalářské práce: **RNDr. Josef Rak**
Katedra informačních technologií

Datum zadání bakalářské práce: **15. ledna 2010**

Termín odevzdání bakalářské práce: **14. května 2010**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2010

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 10. 5. 2010 Martin Mikloš

Souhrn

Tato práce se zabývá popisem alternativních způsobů vývoje grafických aplikací a her. Detailně popíši tvorbu v nástroji, který k takové tvorbě slouží a porovnám výkon ukázkové aplikace s výkonem obdobné aplikace vytvořené pomocí standartních prostředků. Zabývat se budu také porovnáním ostatních aspektů tvorby standartními a nestandartními prostředky.

Klíčová slova

Grafické aplikace, hry, test výkonu, alternativní způsoby programování

Title

Alternative ways of developing graphic applications and games

Abstract

This work deals with characteristic of alternative ways of developing graphic applications and games. I will describe the way how does an alternative-developing tool work and I will compare performance of sample application with application, which was created in a standard way of developing. I will also comapre other aspects of developing both in standard and non-standard ways.

Keywords

Graphic applications, games, performance test, alternative ways of developing

Poděkování

Rád bych touto formou poděkoval vedoucímu mé bakalářské práce panu RNDr. Josefu Rakovi za odborné vedení a cenné rady, které mi poskytl v průběhu práce.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Cílem bakalářské práce je poukázat na možnosti alternativních způsobů tvorby grafických aplikací pro platformu Windows, detailně popsat, na jakém principu fungují, ukázat praktickou tvorbu pomocí těchto přostředků a popsat hlavní výhody a nevýhody v porovnání se standartními prostředky určeným k vývoji aplikací. Mezi standartní prostředky, které nám slouží k tvorbě grafických aplikací, bych zařadil některý z vyšších programovacích jazyků (například Delphi, C/C++) ve spojení s některou z grafických knihoven (například SDL[1], OpenGL[2], DirectX).

Kombinace takovýchto technologií nám poskytuje velmi dobré možnosti pro tvorbu kvalitního software. Avšak má i svá úskalí. Ovládnutí těchto technologií je poměrně náročné. Vyžaduje týdny studií a nedostatečné zvládnutí zvolené technologie má za následek nekvalitní produkt, který nesplňuje naše očekávání. Především mám na mysli konečný vzhled, funkčnost a efektivitu výsledku. Zéjména v 3D aplikacích, které jsou zvlášť citlivé na počet vykreslených snímků za vteřinu, je potřeba strávit mnoho času optimalizací kódu a zefektivňováním algoritmů pro vykreslování scény – zkrátka o dané problematice něco *vědět.*

Na trhu se nicméně objevila celá řada nástrojů, které se snaží tvorbu grafických aplikací usnadnit. Jedná se o velmi zajímavé vizuální programovací nástroje, které se snaží o maximální zjednodušení tvorby. To ocení nejen začínající programátoři, ale i designéři, kteří se mohou více než na ladění a optimalizaci programů soustředit na samotný vývoj produktu.

V této práci jeden takový nástroj detailně popíši a ukáži tvorbu aplikace v takovém software. Nastíním jeho výhody a také jeho omezení. V práci vytvořím celou řadu ukázkových aplikací demonstrující funkčnost nástroje. Nakonec vytvořím aplikaci, jejíž efektivnost porovnám s efektivitou programu, který vytvořím pomocí standartních prostředků.

1 SDL – Simple DirectMedia Layer
2 OpenGL – Open Graphic Library

2 Vizuální programovací nástroj Petr

2.1 O co se jedná

Jedná se o vizuální programovací nástroj, který je určený pro snadnou a rychlou tvorbu grafických (2D/3D) aplikací pro Windows 95/98/2000/XP/NT. Tento nástroj se odprostil od vyjádření zdrojového kódu pomocí textu, jednotlivé programové struktury jsou zde vyjádřeny grafickými prvky. Tyto prvky se skládají do stromové struktury pomocí myši. Jednotlivé prvky v zásadě zastupují běžné programové prvky typu cykly, podmínky, funkce (podrobně později), které do sebe zapadají trochu jako skládačka. Výhodou zde je, že nelze kombinovat prvky které spolu logicky nesouvisí, a tedy přímo odpadá možnost udělat syntaktickou chybu. Vytvořený program je možné okamžitě spustit. Uložený program je plnohodnotnou aplikací Windows, která je přenositelná a samostatně spustitelná (pro její spuštění není zapotřebí žádné klientské aplikace ani speciální knihovny).

2.2 Co umí

Nástroj poskytuje prostředky pro tvorbu plnohodnotné grafické aplikace. Kromě řídících struktur programu nabízí matematické funkce, funkce pro práci s 2D/3D grafikou, zvuky a hudbou, síťovou podporu DirectPlay, podporu dialogových oken, ale i konzolí. Mimoto obsahuje funkce pro práci s běžnými vstupními zařízeními (klávesnice, myš, pákový ovladač) a funkce pro práci se soubory nejrůznějších typů. Pokročilého uživatele zaujmou funkce pro práci s komunikačními porty nebo prostředky pro volání API funkcí.

2.3 Okno aplikace Petr

Okno aplikace je rozděleno na pět částí.

2.3.1 Programová část

V prostřední částí se sestavuje náš program, jedná se v podstatě o ekvivalent zdrojového kódu, který je však reprezentován grafickými prvky.

2.3.2 Základní prvky

Program se sestavuje pomocí základních prvků umístěných v pravé horní části obrazovky. Těmito prvky se budu zabývat v následující kapitole o tvorbě programu v nástroji Petr.

2.3.3 Knihovna proměnných a funkcí

Petr má omezené množství typů proměnných:

Typ proměnné	Ekvivalent v jazyce C++	Poznámka
Číslo	Double	
Příznak	Bool	Ano / Ne
Text	String	
Obrázek	-	
Zvuk	-	
Plocha	Podobné 2D poli	Skládá se z předmětů
Předmět	-	Ikona 32px * 32px
Zvuk	-	Mp3 / wav
Hudba	-	Midi
Sprajt	-	Zajímavá struktura usnadňující animaci grafického prvku. Řeší také hladiny výskytu a překrývání objektů.
Funkce	Function	Klasicky s jedním výstupním a více vstupními parametry.
Seznam	Statické pole struktur	Umožňuje procházet prvky pomocí indexů.

Tabulka 1: Typy proměnných

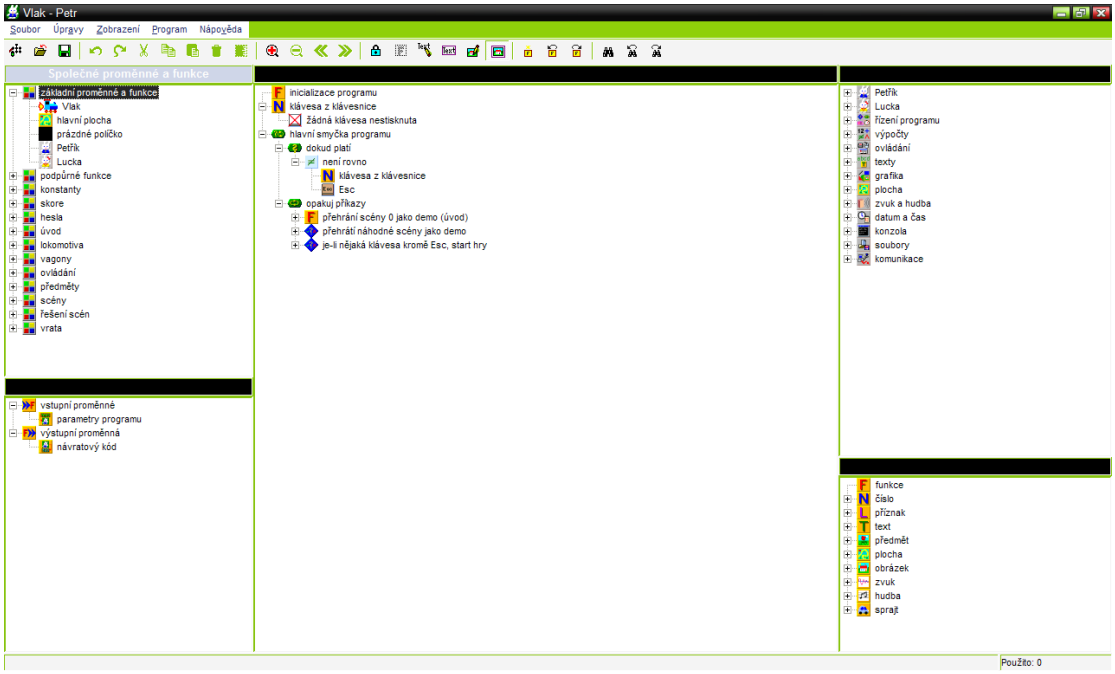
Proměnné se vyskytují v pravé dolní části okna. Odtud si je můžeme přetáhnout do našeho programu. Petr také nabízí rozsáhlou knihovnu zvuků, obrázků, ikon a předrenderovaných 3D sprajtů, které můžeme v naší aplikaci využívat.

2.3.4 Společné proměnné a funkce

Vlevo nahoře si ukládáme proměnné, se kterými náš program pracuje. Ty sem přetahujeme z knihovny proměnných a funkcí. Jedná se o ekvivalent „deklarace globálních proměnných“ tak, jak ho známe z programovacích jazyků.

2.3.5 Místní proměnné a funkce

Petr pracuje také s lokálními proměnnými, které jsou umístěny v levé dolní části obrazovky. Jedná se o vstupní, výstupní a místní proměnné, se kterými pracují námi vytvořené funkce a které nejsou mimo ně vyditelné. V případě hlavního programu se jedná o vstupní parametry programu a návratový kód, který naše aplikace může vracet.

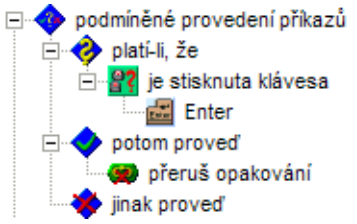
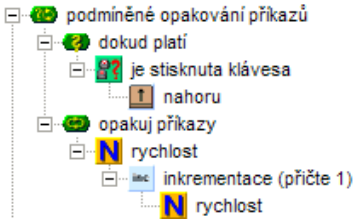
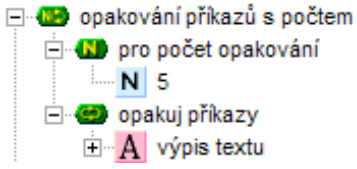
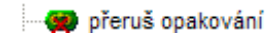
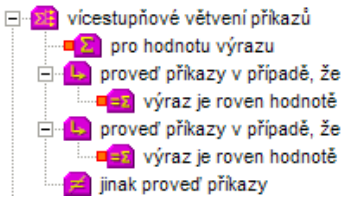
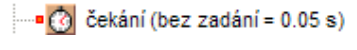
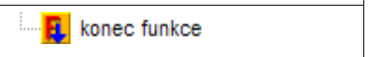


Obr 1: Hlavní okno aplikace

3 Tvorba programu v nástroji Petr

Jak jsem již zmínil, program se sestavuje pomocí prvků, které skládáme dohromady přetahováním myši. Tyto prvky nahrazují základní, ale i složitější prvky, struktury a funkce, které známe z běžných programovacích jazyků. Prvky jsou reprezentovány graficky ve stromové struktuře. Textové popisky jsou informativní a plní funkci komentářů. Libovolné větve stromu (se kterými např. zrovna nepracujeme) můžeme schovat a program tak zpřehlednit.

3.1 Prvky pro řízení programu

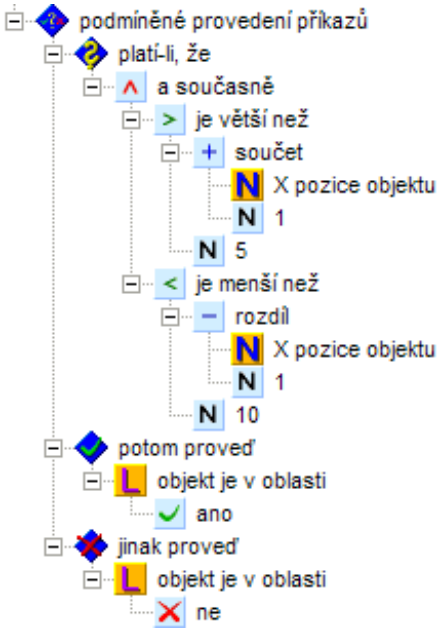
Typ příkazu	Ekvivalent v jazyce C++	Ukázka
Podmíněné provedení	If - else	
Podmíněné opakování	While	
Opakování s počtem	For	
Přeruš opakování	Break	
Vícestupňové větvení	Switch	
Čekání	Wait	
Konec funkce	Return	

Tabulka 2: Řídící struktury

3.2 Výpočtové funkce

Nedílnou součásti editoru jsou matematické funkce sloužící k definici podmínek podmíněných příkazů a cyklů a ostatním výpočtům. K dispozici jsou běžné aritmetické operace, logické operace (konjunkce, disjunkce, negace, logické ano a ne), goniometrické funkce, funkce pro náhodná čísla, bitové operace (bitový součet, součin, inverze, doplněk, rotace vlevo a vpravo, maximální bitové číslo). Pro zjednodušení práce jsou také definovány některé konstanty, jako je Eulerova konstanta, Ludolfovo číslo, nebo úhlové konstanty.

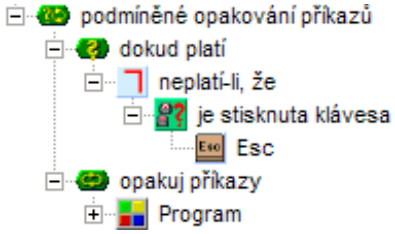
Jelikož jsou i tyto funkce reprezentovány pouze graficky, může se zdát jejich zápis trochu nepřehledný. Jejich zvládnutí je však o zvyku (viz obr.8).



Obr 2: Ukázka struktury kódu

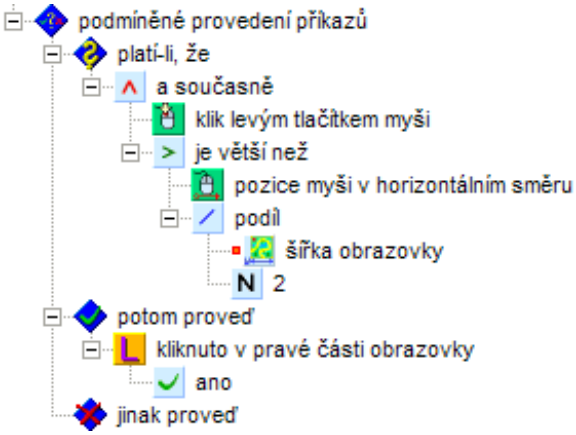
3.3 Ovládací funkce

Slouží k obsluze funkcí myši, klávesnice a pákového ovladače. Jejich obsluha je velmi jednoduchá. V následující ukázce je vytvořen cyklus programu, který je vykonáván, dokud uživatel nestiskne klávesu escape.



Obr 3: Cyklus v aplikaci Petr

V další ukázce je znázorněn podmíněný příkaz. Podmínkou jeho vykonání je, že uživatel kliknul levým tlačítkem myši a současně je pozice myši v horní polovině obrazovky.



Obr 4: Podmíněné vykonání príkazů

Mezi ovládací funkce jsou též zařazeny dialogové prvky pro práci s oknem, tlačítky, textovými poli, posuvníky a dalšími. Popisovat je zde však, stejně jako mnohé další funkce editoru, nebudu.

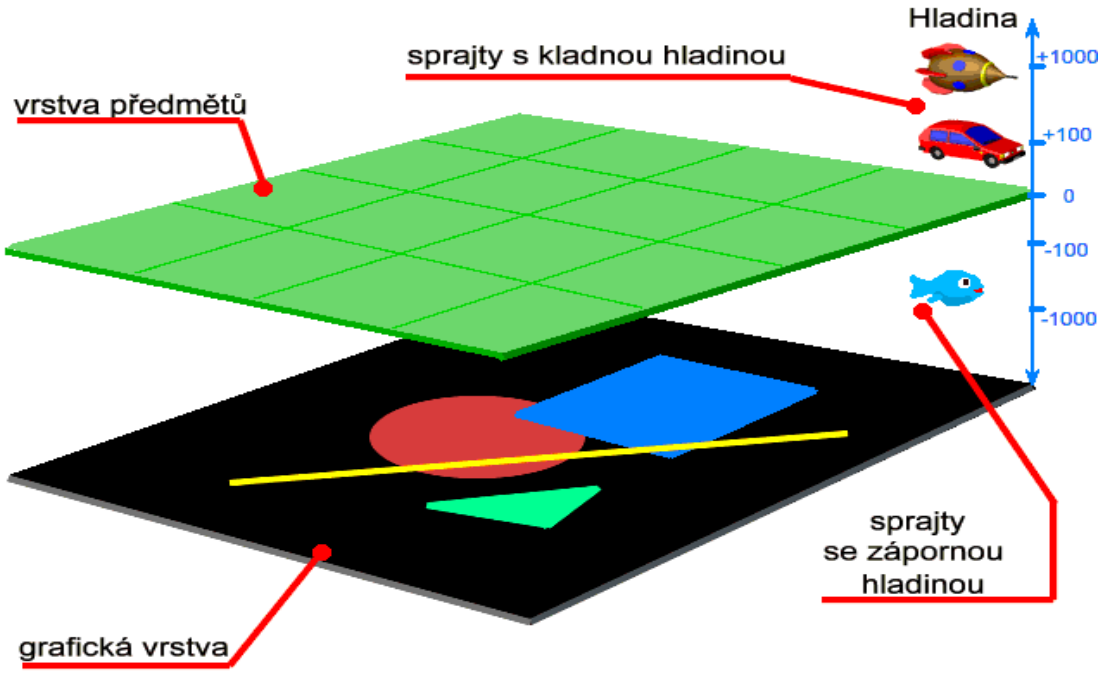
4 Tvorba 2D grafických aplikací

4.1 Obecné informace

Podstata jednoduchosti tvorby v aplikaci Petr je právě v tom, že standartně složité úkony maximálně zjednodušuje. Okno námi tvořené aplikace je vytvořeno automaticky spolu se založením projektu. Spuštěním prázdného projektu se otevře okno s námi definovaným rozměrem, s tlačítky pro minimalizaci, maximalizaci a ukončení a se status-panelem. Kombinací kláves alt a enter se můžeme přepínat do a z celoobrazovkového režimu. Aplikace se samozřejmě okamžitě ukončí, jelikož zpočátku nemá žádné příkazy k vykonání.

4.2 Vrstvy

2D Aplikace vytvořená v Petrovi je standartně rozdělena do několika vrstev (viz obr.). Úplně vespod je grafické plátno, do kterého se provádí vykreslování běžných grafických funkcí. Druhou vrstvou je vrstva předmětů, v Petrovi nazývaná plocha. Plocha je složená z takzvaných předmětů, což jsou ikony o rozměrech 32 x 32 pixelů. Tato vrstva je vhodná zejména pro výukové účely. Má sice některé zajímavé funkce (každé políčko plochy má přepínače a číselné hodnoty definující jeho stav), ale ve většíne aplikací jsem ji příliš nevyužil. Nad a pod vrstvou plochy je prostor pro vykreslování sprajtů.



Obr 5: Grafické vrstvy ve 2D režimu

4.3 Sprajty

Sprajt je struktura umožňující snadno animovat grafické objekty. Skládá se ze sady obrázku, které mohou například definovat objekt otočený do několika směrů a v jednotlivých směrech jeho animaci. Díky tomu můžeme napříklat vytvořit animaci postavy (viz obr.). Každému sprajtu přiřazujeme hladinu výskytu a díky tomu můžeme snadně řídit překrývání (například letadlo se bude vždy vykreslovat nad lidmi). Vlastností sprajtů je, že při změně jejich polohy nemusíme překreslovat ostatní grafické prvky naší aplikace.

Obr 6: Sprajt

Podstatné je, že souřadnicový systém se odvíjí od jednotlivých předmětů. X-ová souřadnice 0 je u levého okraje obrazovky, X-ová souřadnice 1 začíná na 32. pixelu zleva.

4.4 Realizace 2D aplikace vykreslující v reálném čase

Základem grafické aplikace je obvykle smyčka stále se opakujících příkazů. Ve 2D režimu by poslední příkaz ve smyčce měl být příkaz čekání s parametrem 0.01. Ten zajistí optimální vykreslení bez problikávání grafického plátna. Problikávání známe i z grafických aplikací tvořených v C++, kde jej obvykle můžeme řešit pomocí metody zvané double-buffering.

4.5 Podporované funkce pro 2D grafiku

2D Grafické funkce v Petrovi nám umožňují jednak kreslení (bod, úsečka, kruh, kružnice, obdélník, text, trojúhelník) se zadáním parametrů jako je barva a tloušťka čáry, ale také například načíst barvu bodu ze zadané souřadnice, mixovat barvy dle složek (včetně alfa-kanálu, průhlednosti). Samozřejmě se zde vyskytují funkce pro otočení, výřez, změnu jasu, transformace, maskování, náhradu barev, vyplňování a podobně.

5 Tvorba 3D grafických aplikací

Jelikož opravdová síla nástroje Petr je podle mě právě ve tvorbe 3D aplikací, rozhodl jsem se jejich tvorbě věnovat celou kapitolu, ve které podrobně popíši, jak 3D grafika v Petrovi funguje.

5.1 Inicializace 3D

Okno aplikace Petr slouží implicitně pro kreslení 2D grafiky. Pokud chceme používat 3D grafiku, musíme nejprve inicalizovat 3D okno, což je jediný nezbytný úkon před využitím 3D funkcí Petra. Volitelně můžeme nastavit videomód, rozhraní a ovladač.

5.1.1 Inicializace okna 3D grafiky

Inicializaci provedeme jediným příkazem „Okno 3D grafiky“. Zadáním příkazu se v hlavním okně programu zobrazí okno pro 3D grafiku s požadovanými souřadnicemi a rozměry. Souřadnice a rozměry okna jsou udávány v souřadnicích Petra. K vytvoření 3D okna je použito zadané rozhraní a ovladač. Novým zadáním příkazu okno 3D grafiky je možné pozici a rozměry 3D okna dodatečně měnit. Není-li žádný z parametrů uveden, vytvoří se 3D okno přes celou plochu programového okna. Okno 3D grafiky může mít i větší rozměry než je plocha programu. Např. v celoobrazovkovém režimu 800x600 bodů využívá program plochu 25x18 políček, tj. 800x576 bodů. Okno 3D grafiky přitom může mít rozměry přes celou plochu 800x600. Uvedením nuly namísto šířky nebo výšky 3D okna je 3D režim ukončen.

Všechny prvky 3D grafiky je možné využívat nezávisle na aktivitě 3D režimu. Proto je možné 3D okno zapínat a vypínat podle potřeby, stejně tak měnit rozhraní nebo ovladač. Všechny nastavené parametry přitom zůstávají zachovány, není potřeba po změnách provádět přenastavení.

Nastavení videomódu

Prvek videomód slouží ke zjištění a nastavení videomódu pro celoobrazovkový režim programu. Prvek při čtení navrátí víceřádkový textový seznam videomódů. Každý řádek seznamu obsahuje jeden videomód ve tvaru: šířka*výška/bitů. Číselné parametry "šířka" a "výška" představují horizontální a vertikální rozlišení displeje. Parametr "bitů" je počet bitů na jeden grafický bod. Parametr může mít hodnotu 8, 16, 24 nebo 32.

Nastavení rozhraní

Nastavením prvku „Rozhraní“ určujeme, jaké rozhraní má být použito k obsluze

3D grafiky. Nastavením na hodnotu 1 až 8 bude použito rozhraní podle tabulky níže. Nastavením na 0 bude použito rozhraní podle automatické detekce. Zvolené rozhraní je použito k inicializaci 3D režimu příkazem okno 3D grafiky. Není-li 3D okno aktivní nebo nebylo-li nalezeno požadované rozhraní, navrací prvek 0. Jinak navrací číslo aktivního rozhraní. Rozhraní je možné měnit i po aktivaci 3D okna. Ke změně rozhraní může dojít též automaticky při přepnutí celoobrazovkového režimu nebo při změně aktivního videomódu Windows.

5.1.2 Nastavení ovladače

Prvek ovladač určuje, který ovladač aktivního rozhraní má být použit k obsluze 3D grafiky. Nastavením na hodnotu 1 až 6 bude použit ovladač podle tabulky níže. Nastavením na 0 bude použit ovladač podle automatické detekce.

Číslo	Ovladač
1	HAL (hardwarový)
2	TnLHal (hardwarový s transformačním a osvětlovacím modulem)
3	REF (referenční)
4	RGB (softwarový)
5	MMX (softwarový s instrukcemi MMX)
6	Ramp (softwarový s MONO osvětlením, bez textur)

Tabulka 3: Nastavení ovladače

5.2 Práce s 3D objekty v aplikaci Petr

5.2.1 Principy

Petr každému 3D objektu vytvořenému v naší aplikaci automaticky přiděluje unikátní číslo, které si uschováme v číselné proměnné. Pokud chceme nad nějakým konkrétním objektem provádět např. transformace (otočení, posunutí) nebo ruzná nastavení (stíny, osvětlení...), musíme tento objekt nejprve nastavit jako „aktivní“. To provedeme prvkem „Aktivní objekt“, jemuž jako číselný parametr předáme číslo požadovaného objektu. Je to poměrně zajímavé řešení, mající výhody i nevýhody. Výhodou je, že po aktivování objektu už je manipulace s ním velmi snadná. Nevýhodou je, že přiřazení hodnot (například pozice) jednoho objektu druhému, je poměrně složité (aktivace objektu *a*, úschova hodnot do proměnných, aktivace

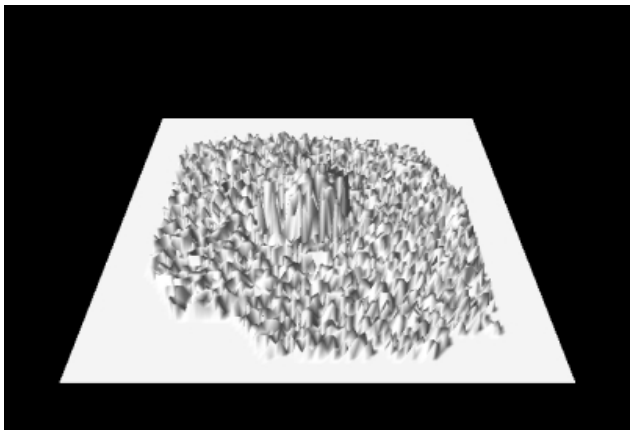
objektu *b*, přiřazení hodnot).

5.2.2 Vytváření

Petr nabízí automatické vytváření celé řady objektů. Každý objekt se vytváří pomocí vlastního prvku vracející jeho identifikační číslo.

Kromě možnosti vytvoření základních 3D objektů, jako je stěna, koule, kvádr, kužel / komolý kužel (s parametrem průměru horní stěny), umožňuje Petr vytváření celé řady zajímavých struktur.

Velmi zajímavá je možnost vytvoření 3D map (prvek „**Terén z obrázku**“) pomocí výškové mapy. Výšková mapa je obrázek který zadáme na vstupu prvku. Jas jednotlivých bodů obrázku definuje výšku ve výsledné 3D mapě. Jinou funkcí je vytvoření takzvaného „**statického 2D objektu**“. Na vstupu je obrázek, který se namapuje na 2 vzájemně proložené kolmé plochy. Takto se snadno vytvoří například strom.



Obr 7: Objekt vytvořený pomocí výškové mapy

I když si často v našich aplikacích vystačíme se základními tvary, nezbytnou součástí je možnost definice tvarů vlastních. K jejich načtení slouží prvky „**Objekt z textu**“ a „**Objekt ze souboru**“, s parametrem textové proměnné / textového souboru definujícího načítaný objekt. Petr pracuje pouze s jediným formátem a to formátem DirectX. Formát je to vhodný a hojně podporovaný (většina nástrojů pro 3D modelování podporuje export do formátu DirectX).

Speciálním 3D objektem je prvek „Skupina“. Ke skupině můžeme připojit pomocí prvku „Připojení“ libovolný počet jiných objektů a skupin a s touto skupinou pak manipulovat jako s celkem (např. vypnout viditelnost). Připojovat objekty nemusíme jen ke skupině, ale také k jinému (rodičovskému) objektu. Transformace

(otočení, velikost, pozice) připojovaného objektu je pak relativní vzhledem k rodičovskému objektu.

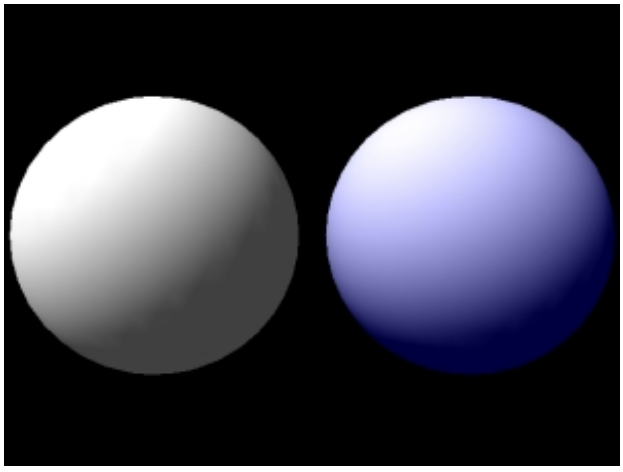
5.2.3 Transformace

Transformace se uplatní pro aktivní objekt a vztahují se k rodiči objektu. Není-li objekt připojen k jinému objektu, vztahují se transformace ke scéně. Transformace vychází z nulové výchozí pozice objektu. Nejedná se proto o změnu oproti předchozí poloze, ale spíše o aktuální souřadnice objektu. Transformace se provádí v pořadí: změna měřítka - rotace - posun. U rotací je možné určovat pořadí. Implicitně se rotace provádí v pořadí X - Y - Z.

K provádění transformací slouží prvky: „Posun ve směru X“, „Posun ve směru Y“, „Posun ve směru Z“. Obdobně máme prvky pro otočení po jednotlivých osách / změnu měřítka (relativní velikosti) v jednotlivých směrech.

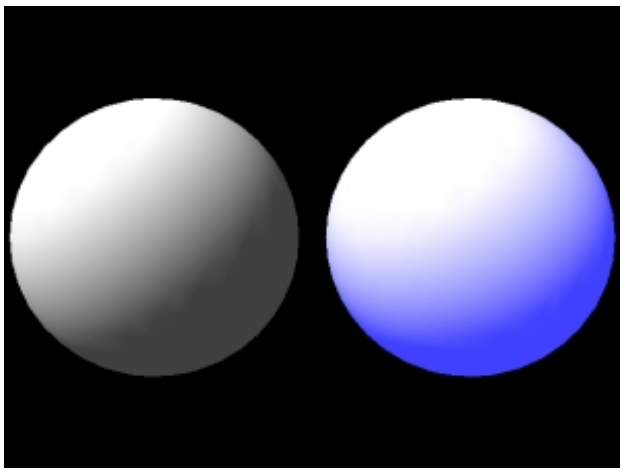
5.2.4 Povrch

U objektů můžeme nastavovat barvu (pomocí prvku „**Barva**“ s číselným parametrem udávajícím barvu). Prvek „**Odraz rozptýleného světla**“ slouží k nastavení barvy určující, jak je aktivním objektem odráženo rozptýlené světlo. Na obrázku ukázka s implicitním nastavením (vlevo) a s nastavením odrazu na modrou barvu (vpravo).



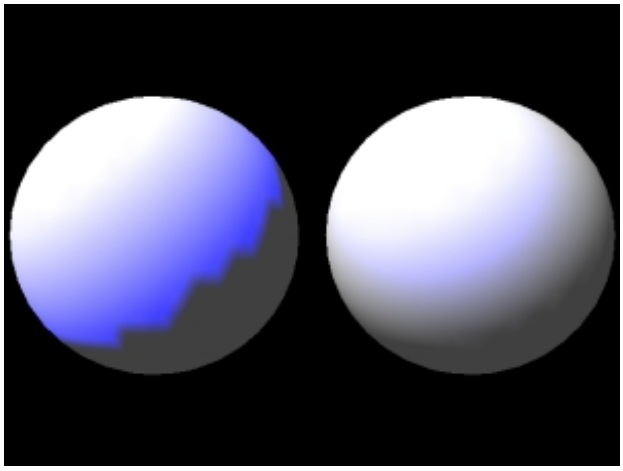
Obr 8: Objekt s a bez nastavení odrazu světla

Prvek „**Barva svítivosti**“ slouží k nastavení vlastní svítivosti aktivního objektu. Svítivost je přídavná složka barvy objektu, která se přičítá k výsledné celkové barvě objektu. Není závislá na okolním osvětlení. Na obrázku ukázka s implicitním nastavením (vlevo) a s nastavením svítivosti na modrou barvu (vpravo).



Obr 9: Objekt s a bez nastavené barvy svítivosti

Pro zvýšení realističnosti vzhledu objektů je možné vytvářet na povrchu objektů odlesky od okolního osvětlení. Prvek „**Matnost**“ slouží k nastavení velikosti odlesků aktivního objektu. Matnost může mít hodnoty od 0 do 128. Nastavením matnosti na 0 a nastavením barvy odlesku na černou barvu se odlesk vypne. Matnost s hodnotou 5 se používá pro kovový povrch. S rostoucím stupněm matnosti se zmenšuje velikost odlesku. Matnost s hodnotou kolem 50 se používá pro plastový povrch. Prvek „**Barva odlesku**“ slouží k nastavení barvy a intenzity odlesků. Odlesky závisí na okolních světlech a na úhlu pozorovatele. Na obrázku ukázka s dvou koulí. Obě mají nastavenou barvu odlesku na modrou barvu. Levá koule má matnost nastavenou na implicitní hodnotu, druhá na hodnotu 5.



Obr 10: Objekt s a bez nastavené barvy odlesku

Prvkem „**Obrysy**“ lze řídit u aktivního objektu zobrazení povrchu objektu v závislosti na alfa složce (průhlednosti). Jako parametr prvku se zadává číselný parametr v rozsahu 0 až 1. Jsou zobrazeny jen ty části povrchu objektu, jejichž alfa složka má vyšší hodnotu, než je zadaný parametr. Obrysy mají význam především ve spojení s texturami. Nejčastěji se využívají u 2D objektů a u statických 2D objektů. Při kreslení textury se okolí objektu vyplní průhlednou barvou. Průhledná barva je černá barva s alfa složkou 0. Ostatní barvy mají alfa složku 1. Tak je zajištěno, že část textury s průhlednou barvou bude potlačena (má nižší afla složku než prahová úroveň zadaná prvkem obrysů), bude zobrazena jen část obrázku s běžnými barvami. Na obrázku je 2D objekt ve 3D scéně s vypnutými obrysy (vlevo) a se zapnutými (vpravo).

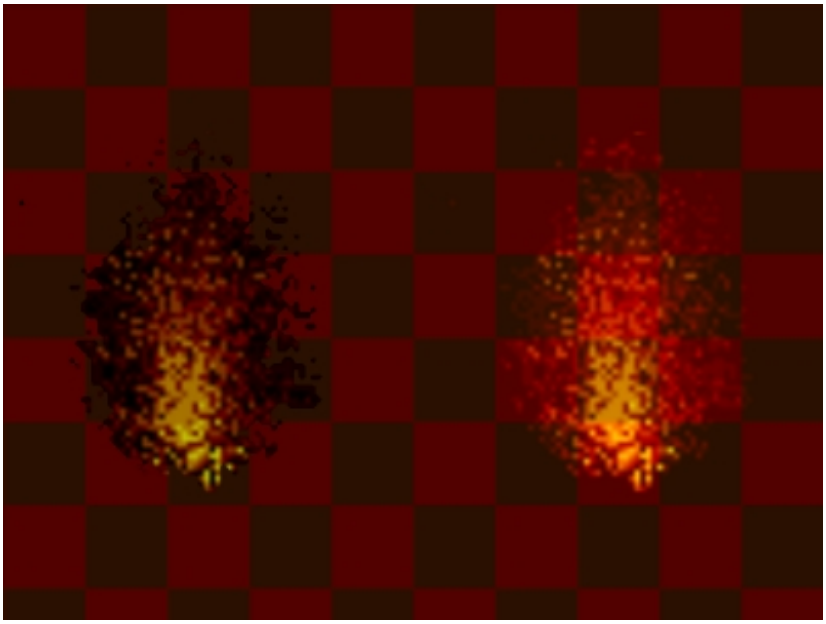
Obr 11: Objekt s vypnutými a zapnutými obrysy

Prvek „**Průhlednost**“ řídí u aktivního objektu způsob, jakým je kombinována barva povrchu objektu s barvou pozadí. Při vykreslování objektu se vypočítávají složky výsledné barvy (včetně alfa) každého vykreslovaného bodu objektu podle vztahu: výsledná_složka = zdrojová_složka*zdrojový_koeficient + cílová_složka*cílový_koeficient. Průhlednost je 2-místný číselný kód. Číslice na pozici jednotek určuje kód operace pro zdrojovou barvu, číslice na pozici desítek určuje kód operace pro cílovou barvu. Nejčastěji použité kombinace jsou:

Kód	Výsledek
1	běžný (implicitní) mód, objekt plně překrývá pozadí
11	mód pro oheň a střely, barva objektu se sečítá s barvou pozadí
20	mód pro sklo, barva objektu se vynásobí barvou pozadí (modulace)
54	mód pro průhledný překryv, alfa složkou objektu lze plynule řídit průhlednost

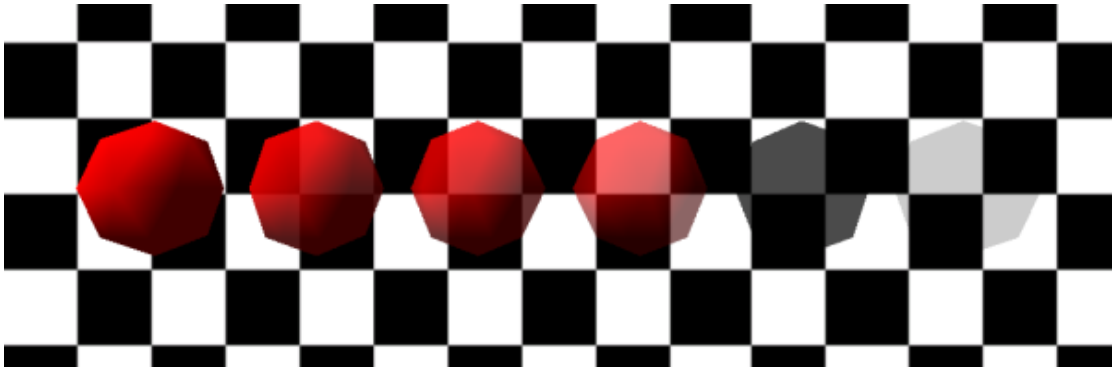
Tabulka 4: Nastavení průhlednosti

Na obrázku níže je levý objekt bez využití průhlednosti (se zapnutými obrysy). Pravý objekt má průhlednost nastavenou na hodnotu 11 (efekt ohně).



Obr 12: Objekt bez a s nastavením barvy průhlednosti

Na dalším obrázku se nachází objekty s nastavením průhlednosti na hodnotu 54 (překryv) a postupným zvyšováním alfa složky.

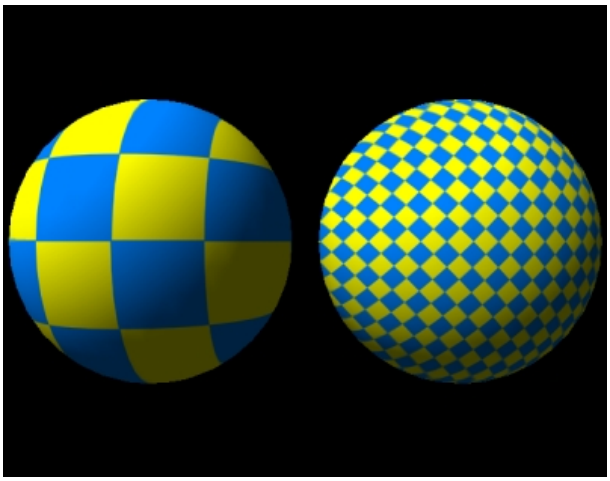


Obr 13: Různé nastavení parametru průhlednost u objektu

5.2.5 Textury

Povrch objektu může být pokryt barvou nebo texturou. Textura je v podstatě běžný obrázek, který lze vytvořit v grafickém editoru aplikace Petr, nebo v jakémkoliv jiném editoru. V Petrovi se nachází celá řada funkcí pro práci s texturami. Popisovat všechny není v rámci této práce možné, takže popíši pouze nejdůležitější a nejpoužívanější z nich.

Asi nejužitečnější je jednoduše prvek „**Textura**“. Parametrem je obrázek, který se namapuje na aktivní objekt. Obvykle potřebujeme přesněji řídit pokrytí objektu texturou. K tomu slouží prvek „**Mapování textury**“. Tento prvek má poměrně velké množství vstupních parametrů, pomocí kterých můžeme definovat, odkud a jakým směrem se textura na povrch objektu nanáší. Umožňuje také určit tzv. „Stupeň zmenšení / zvětšení textury“, což se projeví tak, že se textura zmenší a na objekt nanáší opakovaně (využijeme například pokud texturujeme zeď apod.). Na obrázku je objekt s jednodduchým pokrytím texturou pomocí prvku „**Textura**“ (vlevo) a objekt s mapováním zmenšené textury otočené o 45° (vpravo).



Obr 14: Ukázka mapování textur

Zajímavé jsou logické přepínače „**Filtrace zmenšených textur**“ a „**Filtrace zmenšených textur**“. Jsou-li zapnuty, jsou texely (tj. body textury) při zmenšení / zvětšení textury lineárně interpolovány - textury jsou při zmenšení / zvětšení vyhlazeny. Jsou-li vypnuty, použije se k vykreslování zmenšené / zvětšené textury nejbližší bod - u vykreslované textury jsou patrné ostré přechody mezi texely.

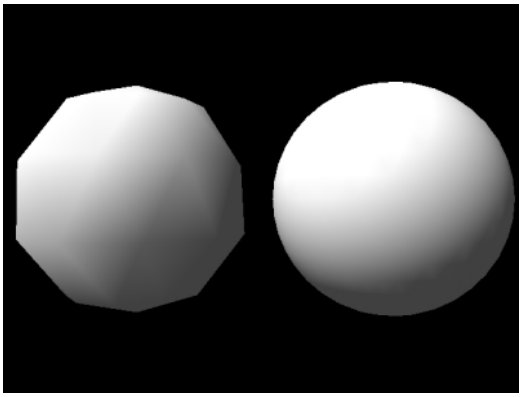
Prvek „**Filtrace vzdálených textur**“ je logický přepínač. Je-li zapnut, jsou u vzdálených objektů použity zmenšené textury (nazývané mipmap). Tím je možné u vzdálených objektů dosáhnout přirozenějšího vzhledu (rozostření). Je-li přepínač vypnut, je použita základní nezmenšená textura (povrch vzdálených objektů "zrní"). Způsob filtrace vzdálených textur je možné ještě dále upřesnit prvky „**Počet úrovní vzdálených textur**“ a „**Zjemnění vzdálených textur**“.

5.2.6 Funkce pro práci s objekty

S objekty lze provádět mnoho věcí, které velmi usnadňují tvorbu 3D aplikací, případně jsou pro tvorbu přímo nezbytné. Tato skupina funkcí je také poměrně rozsáhlá a opět uvádím jen nejdůležitější z nich.

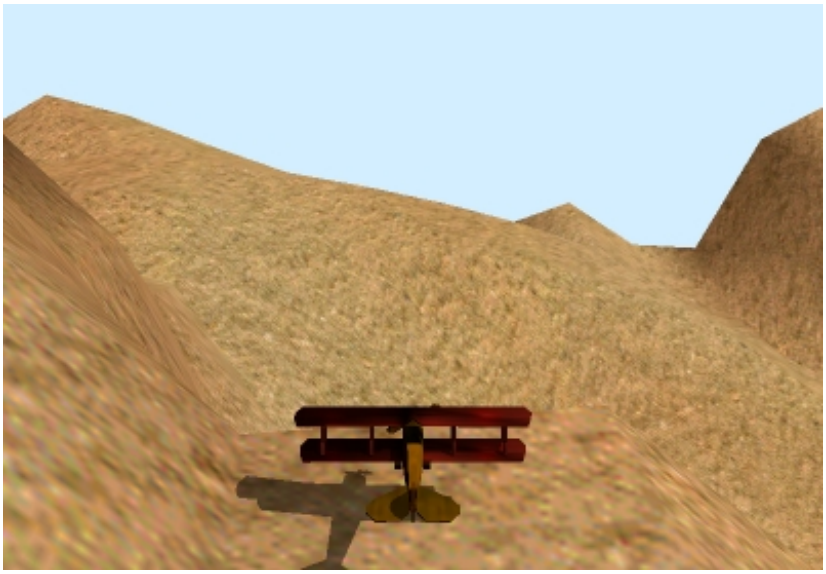
Pomocí prvku „**Viditelný**“ nastavujeme zda se objekt vykresluje nebo ne. Prvkem „**Zrušení objektu**“ provedeme odstranění objektu jak z aplikace, tak z paměti (obdoba dealokace). Prvek má velký význam při optimalizaci rychlosti programu. Pomocí prvku „**Klonování**“ můžeme duplikovat libovolný prvek, nebo skupinu prvků. V případě klonování skupiny však ztrácíme možnost zpřístupnit jednotlivé pod-objekty klonu.

Velmi důležitým prvkem je prvek „**Složitost objektu**“, který hraje roli při vytváření objektů. Určuje složitost, s jakou budou vytvářeny rotační objekty (kruh, koule, polokoule, válec atd.). Složitost objektu je celé číslo v rozsahu 2 až 100 a udává počet úseků, na které je rozdělena polovina obvodu rotačního objektu.



Obr 15: Různé nastavení parametru složitosti objektu

Prvek „**Dosah stínu**“ slouží k aktivaci dynamického stínu aktivního objektu. Dynamické stíny jsou stíny vrhané pohybujícími se objekty. Dynamický stín vytváří pouze hlavní osvětlení scény. Dynamický stín objektu je vytvářen metodou objemového stínu. Nejdříve je vytvořena obrysová křivka objektu (obrys z pohledu světla) a potom je z obrysové křivky vrhnut kuželový stín mající vrchol na opačné straně než se nachází světlo. V místech, kde se kuželový stín setká s jiným objektem, vzniká stín. Stín vzniká i na objektu samotném. Ikdyž to zní trochu komplikovaně, jediné o co se uživatel Petra stará je, že nastaví, do jaké vzdálenosti se stín vykresluje.



Obr 16: Ukázka dynamického stínování

Příkaz „**Normalizace objektu**“ zajistí změnu velikosti a pozice aktivního objektu tak, aby co nejlépe vyplnil jednotkovou krychli. Funkce příkazu spočívá v tom, že pro všechny vrcholy objektu zjistí maximální rozteč vrcholů ve směrech X, Y a Z. Poté objekt vystředí ve všech třech směrech a vydělí souřadnice vrcholů největší z roztečí. Největší z rozměrů objektu tedy bude mít jednotkovou hodnotu.

Prvek „**renderovací skupina“** je číselný parametr aktivního objektu umožňující určovat pořadí, v jakém jsou jednotlivé objekty renderovány (vykreslovány). Renderovací skupina může mít hodnotu 0 až 15. Renderování probíhá od objektů patřících k renderovací skupině 0 až po objekty renderovací skupiny 15. Použití renderovacích skupin může být libovolné. Implicitně jsou skupiny rozděleny následovně:

Skupina	Objekty
0 až 7	běžné objekty (implicitně 4)
8 až 11	průhledné objekty (implicitně 10, hloubkové třídění)
12 až 15	2D obrázky (implicitně 14)

Tabulka 5: Nastavení renderovacích skupin

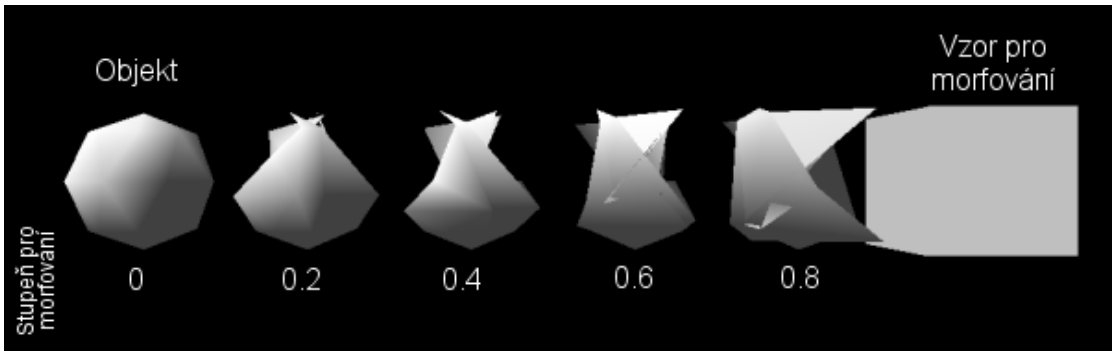
Ke každému 3D objektu lze přiřadit jeden další objekt se sníženou složitostí, který bude použit namísto původního objektu při určité vzdálenosti od pozorovatele. Tím lze dosáhnout značného zvýšení rychlosti vykreslování 3D grafiky bez viditelné ztráty složitosti objektů. Pomocí prvku „**Objekt se sníženou úrovní detailů“** lze k

aktivnímu objektu přiřadit jeden objekt se sníženou úrovní detailů. Prvek „**Vzdálenost pro snížení úrovně detailů“** určuje vzdálenost od pozorovatele, od které se namísto původního objektu použije objekt se sníženou úrovní detailů. Objekty lze řetězit – objekt s již sníženou úrovní detailů může mít ještě méně detailní atd.

Operace morfování je plynulá změna podoby 3D objektu mezi dvěmi nebo více vzory objektu. Morfování se provádí změnou souřadnic jednotlivých vrcholů objektu. Pomocí prvku „**Přidání vzoru pro morfování**“ je k aktivnímu objektu přidán další vzor pro morfování. Parametrem prvku je číslo objektu použitého jako vzor. Ze vzorového objektu je uchován současný vzhled objektu, samotný objekt již není dále využíván a může být v případě potřeby zrušen. Vzorovým objektem může být i jiný morfovaný objekt nebo dokonce sám aktivní objekt. Vzorů může být libovolný počet. Nastavením hodnoty -1 jako parametr prvku se všechny vzory pro morfování zruší a objekt se stává běžným objektem, jeho vzhled zůstane zachován podle posledního stavu morfování.

Prvek „**Stupeň morfování**“ slouží k řízení operace morfování. Parametrem prvku je libovolná hodnota v intervalu od 0 po počet vzorů morfování+1. Hodnota 0 odpovídá původnímu vzhledu objektu, hodnota 1 vzoru objektu 1 až hodnota N odpovídá vzoru objektu N. Změnou hodnoty od N po N+1 je objekt morfován zpět k původnímu vzhledu objektu, tedy k hodnotě 0.

Na obrázku se nachází ukázka morfování. Jako základní objekt byla vytvořena koule, ke které byl přidán vzor pro morfování (objekt krychle). Stupeň morfování byl zvyšován v intervalech po 0.2.



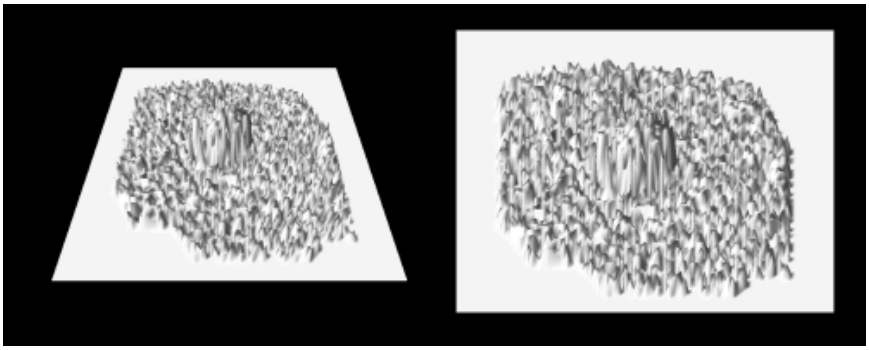
Obr 17: Morfování objektů - koule v krychli

5.2.7 Funkce pro nastavení scény

Umožňuje nastavit barvu pozadí scény (prvek „**Barva pozadí scény**“), nastavit pozadí scény jako obrázek (prvek „**Textura pozadí scény**“), nebo měnit vzdálenost promítací roviny (prvek „**Vzdálenost dohledu**“), což je maximální vzdálenost, ve které jsou objekty ještě viditelné.

Zajímavou funkcí je funkce „**Vzdálenost promítací roviny**“, pomocí které snadno realizujeme efekt „zoom“.

Dále jsou k dispozici funkce pro zobrazení mlhy ve scéně. Nastavit můžeme barvu, typ (lineární, exponenciální, kvadratickou), počátek a konec mlhy. U exponenciální a kvadratické mlhy můžeme nastavit hustotu. Výsledek je na následujícím obrázku.



Obr 19: Perspektivní a ortografická projekce

Obr 18: Nastavení mlhy v 3D scéně

Prvek „**Typ projekce**“ je číselná proměnná určující způsob promítání scény do 3D okna:

Číslo	Projekce
0	perspektivní projekce (implicitně)
1	ortografická projekce
2	perspektivní projekce, pravoruký souřadný systém
3	ortografická projekce, pravoruký souřadný systém

Tabulka 6: Nastavení typu projekce

Na následujícím obrázku je objekt vykreslený v perspektivní projekci (vlevo) a v ortografické projekci (vpravo).

5.3 Realizace 3D aplikace vykreslující v reálném čase

5.3.1 Základ aplikace

Projekt založený v aplikaci Petr je implicitně určen ke kreslení ve 2D. Režim 3D inicializujeme funkcí „**Okno 3D grafiky**“, kterému předáme parametry rozměrů a umístění 3D okna.

Základem 3D aplikace je obvykle smyčka stále se opakujících příkazů. Ve 3D režimu by poslední příkaz ve smyčce měl být příkaz čekání s parametrem 0. Ten zajistí optimální vykreslení bez problikávání grafického plátna. Problikávání známe i z grafických aplikací tvořených v C++, kde jej obvykle můžeme řešit pomocí metody zvané double-buffering.

5.3.2 Zajištění plynulosti animace

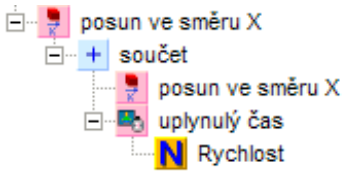
Jedním z problémů při realizaci 3D aplikace vykreslující v reálném čase je, že rychlost vykreslování je velmi závislá na výkonu počítače, videokarty a na použitém ovladači. Pokud by například auto v naší aplikaci měnilo pozici v každém cyklu o hodnotu 1, na některých počítačích by se pohnulo za jednu vteřinu o 50 a na jiných třeba jen o 20 jednotek. To je samozřejmě nedostatečné řešení, animace musí být nezávislá na počtu vykreslených snímků za vteřinu. Proto zpravidla 3D programy pracují maximální rychlostí a program je časován podle skutečně uběhlého času.

Prvek „**Uplynulý čas**“ měří čas uplynulý od posledního vykreslení 3D snímku. Měření času se provádí s rozlišením 1 milisekunda nebo přesněji (je-li k dispozici multimediální časovač). Není-li zadán žádný parametr, navrátí prvek uběhlý čas v sekundách. Jako parametr prvku je možné uvést požadovanou časovou změnu za 1 sekundu (tedy rychlost). Prvek navrátí uběhlý čas vynásobený zadaným číselným parametrem. S pomocí tohoto prvku můžeme vykreslovat v závislosti na reálném čase. Například pohyb objektu tak můžeme realizovat pomocí následujícího vzorce:

Posun ve směru X = Posun ve směru X + Uplynulý čas (Rychlost)

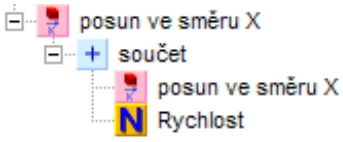
Na následující stránce je grafické znázornění správné a špatné realizace animace v aplikaci Petr.

Správné počítání pozice objektu v závislosti na čase:



Obr 20: Správné animování

Chybné počítání pozice objektu:



Obr 21: Chybné animování

V tomto případě by animace nebyla plynulá a pohyb trhaný. Celkový dojem z hraní by tak byl velmi špatný.

6 Testování výkonu aplikací vytvořených v Petrovi

V následující kapitole budu porovnávat výkon programu vytvořeného v aplikaci Petr s výkonem aplikace vytvořené v C++ s pomocí knihovny SDL. K tomuto účelu jsem vytvořil dvě grafické zcela totožné aplikace, se kterými následně provedu sérii testovacích experimentů s různými parametry. Kritériem rychlosti běhu aplikace bude počet vykreslených snímků za jednu vteřinu.

6.1 Testovací aplikace

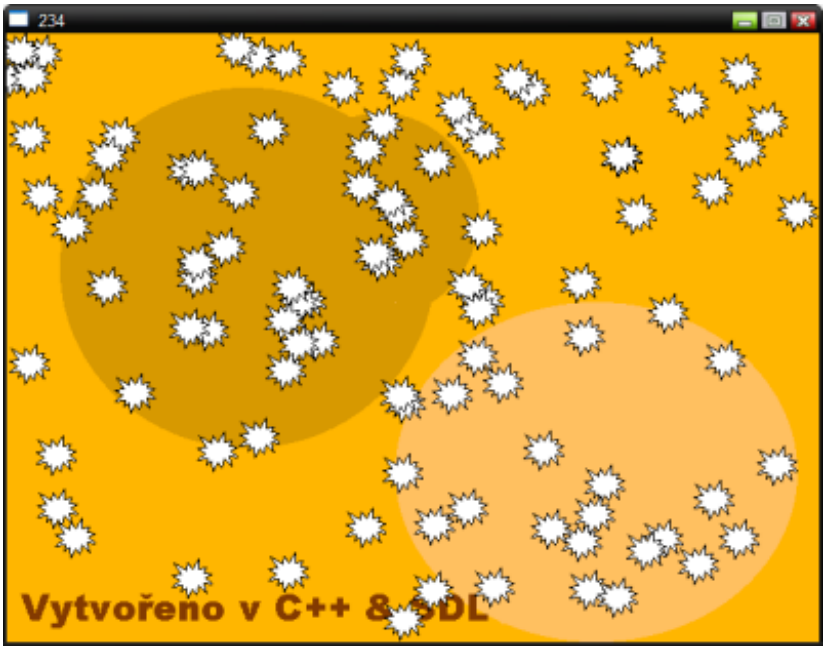
V rámci testovací aplikace bude nejprve inicializované 2D okno s šířkou 640px a výškou 480px. Aplikaci musí být možno ukončit tlačítkem na liště a také klávesou Escape. Aplikace obsahuje *n* struktur které se pohybují v prostoru okna. Každý objekt je při spuštění inicializován na náhodnou pozici v prostoru okna s náhodnou rychlostí pohybu. V každém cyklu aplikace dojde neprve k překreslení grafického plátna grafickým pozadím, aktualizaci pozic jednotlivých objektů a k jejich vykreslení. V případě, že dojde ke kolizi objektu s okrajem okna, dojde k „odražení“ objektu náhodným směrem s náhodnou rychlostí. Objekt je obrázek s šířkou 32px a výškou 32px a má nastavenou jednu barvu jako průhlednou.

V průběhu spuštění aplikace je průběžně propočítávána aktuální hodnota počtu vykreslených snímků za vteřinu. Ta se zobrazí jako titulek okna. Důležité je, že vykreslování ani výpočty jsem žádným způsobem neoptimalizoval a jedná se o využití prostředků tak, jak nám je jednotlivé nástroje a komponenty implicitně nabízejí.

Vzhled aplikace vytvořené v Petrovi je na následujícím obrázku.

Obr 22: Ukázková aplikace vytvořená v nástorji Petr

Vzhled porovnejme se vzhledem aplikace vytvořené v C++ & SDL. Je prakticky shodný.



Obr 23: Ukázková aplikace vytvořená v C++ s pomocí knihovny SDL

6.2 Hypotéza

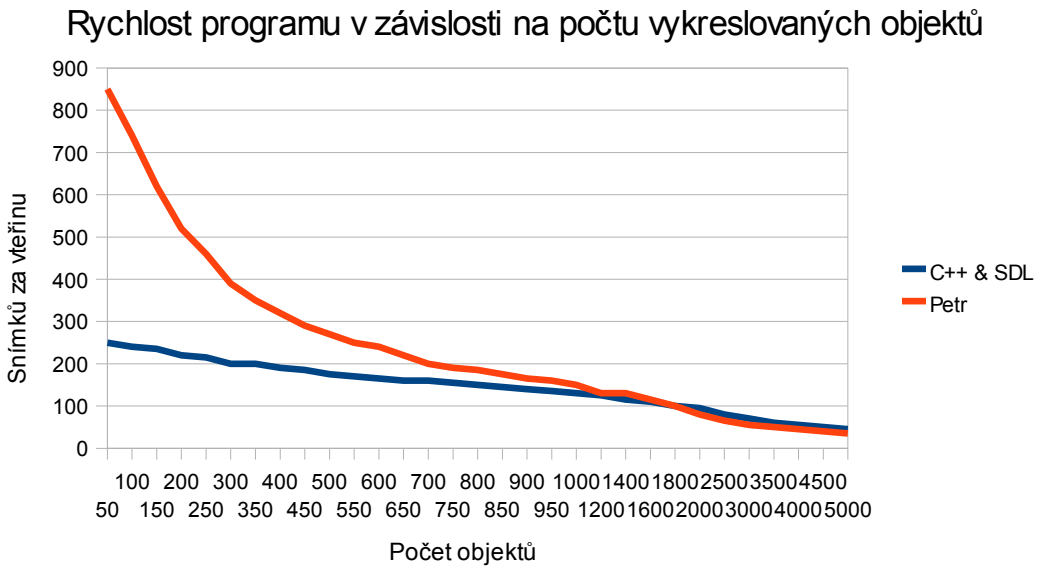
Očekávám, že program vytvořený v aplikaci Petr bude pomalejší. Je to z toho důvodu, že se přeci jen jedná o nástroj který usnadňuje tvorbu aplikací a bude zjednodušovat na úkor výkonu. Dalo by se očekávát, že čím nižší úroveň vývoje aplikací zvolíme, tím výkonnější budou. Volání různých dodatečných funkcí a služeb by mělo celkový výkon zpomalovat.

6.3 Experimentování

Počet objektů	C++ & SDL	Petr
50	250	850
100	240	740
150	235	620
200	220	520
250	215	460
300	200	390
350	200	350
400	190	320
450	185	290
500	175	270
550	170	250
600	165	240
650	160	220
700	160	200
750	155	190
800	150	185
850	145	175
900	140	165
950	135	160
1000	130	150
1200	125	130
1400	115	130
1600	110	115
1800	100	100
2000	95	80
2500	80	65
3000	70	55
3500	60	50
4000	55	45
4500	50	40
5000	45	35

Tabulka 7: Výsledky experimentu

Parametrem experimentu je počet vykreslovaných objektů. Testovat budu postupně od počtu 50 do 1000 po 50ti, od 1000 do 2000 po 200 a od 2000 do 5000 po 1000. Výsledky experimentu jsou zaznamenány v následující tabulce.



Výsledky jsou přehledněji zaznamenány také v následujícím grafu.

6.4 Výsledky experimentu

Výsledek experimentu prakticky vyvrátil hypotézu, že program vytvořený v aplikaci Petr je automaticky pomalejší než aplikace vytvořené pomocí standartních prostředků. Výsledek je překvapivý. Když se podrobněji podíváme na graf, program vytvořený v C++ a SDL běží pro malý počet objektů skoro čtyřikrát pomaleji než obdobná aplikace vytvořená v Petrovi.

Rozdíl je však v tom, jak se mění rychlost programu s postupným přibýváním objektů. Zatímco rychlost aplikace vytvořené v C++ je nepřímo úměrná počtu objektů, výkon aplikace vytvořené v Petrovi nejprve prudce klesá a s přibývajícím počtem objektů klesání výkonu zpomaluje. V určitém bodě (konkrétně 1800 objektů) se výkon aplikace vytvořené v C++ stává větším, než výkon aplikace vytvořené v Petrovi.

6.5 Zhodnocení experimentu

Experiment ukázal, že programy vytvořené v aplikaci Petr mohou, co se výkonu týče, konkurovat programům vytvořeným pomocí standartních prostředků a dokonce ho i převyšují.

6.6 Další možnosti experimentování

Určitě by bylo zajímavé experimentovat i se složitějšími aplikacemi. Obrázky by mohly například rotovat nebo být poloprůhledné s nastaveným alfa kanálem. Experimenty by mohly prokázat, že například jedna aplikace vykresluje efektivně

jednoduché obrázky, ale v případě složitějších efektů se výrazně zpomaluje. Stejně tak by bylo možné obdobně provést bezpočet experimentů s 3D grafikou.

Porovnávat bychom také mohli více technologií, jako je třeba Delphi s využitím komponent PaintBox, DelphiX, nebo také SDL. V případě 3D grafiky by bylo možné porovnat C++ v kombinaci s OpenGL, DirectX a stejně tak aplikaci vytvořenou v Petrovi, která taktéž volitelně pracuje s OpenGL nebo DirectX. Případně by bylo možné provést komplexní srovnání jak standartních prostředků, tak různých nástrojů jako je právě Petr.

7 Závěr

Závěrem bych rád zhodnotil práci s aplikací Petr. Jedná se o velmi zajímavé řešení poskytující celou řadu možností, které neocení jen začínající programátoři, ale i designeři s vizí tvořit, ne se učit kvanta technologií nutných k efektivní tvorbě. Napříč českou komunitou nezávislých vývojářů se objevilo mnoho programátorů, kteří si zvolili Petra jako prostředí pro vývoj her a jejich úspěchy objektivně dokládají výsledky nejrůznějších soutěží ve tvorbě počítačových her, které tyto aplikace často vyhrávají. Pokud je zadání vytvořit během dvou týdnů 3D hru na určité téma, nástroj Petr prakticky nemá konkurenci. Že má však svá omezení a nedostatky naopak jasně dokazuje fakt, že se dosud masivně nerozšířil a co se technologie týče, hernímu průmyslu stále dominuje vývoj v C++ ve spojení s Open GL nebo Direct X.

Mezi hlavní výhody bych tedy zařadil rychlou a efektivní tvorbu. Naučit se s touto aplikací pracovat trvá tak týden a poté již není problém efektivně vyvíjet jakékoliv aplikace. Aplikace jsou stabilní – nikdy mi ještě žádná aplikace nespadla a nevyhodila byť jedinou chybovou zprávu. Spustitelné jsou na jakémkoliv počítači s operačním systémem Windows bez jakékoliv další podpory. Vytvořené programy jsou kompatibilní jak s Windows 95, tak s nejnovějšími Windows Vista a Windows 7.

Mezi nevýhody bych zařadil problematickou optimalizaci programu v okamžiku, kdy výkon přestane stačit. Petr veškeré textury a obrázky ukládá pouze ve 256 barvách, což považuji za velmi omezující. V případě 2D režimu toto omezení považuji za tristní. Využití v komerční sféře například pro různé efektní prezentace tak nepřipadá v úvahu. Pokud se odprostíme od grafiky – Petr sice obsahuje množství dialogových prvků, ale na jejich využití nahlížím poněkud rozporuplně vzhledem k tomu, že Petr stejně nepodporuje žádné napojení na databáze. Že Petr přistupuje k tvorbě ryze strukturovaně, tedy ne objektově, bych také hodnotil záporně. Ale zásadní nedostatek to není a současný přístup je plně dostačující. Zásadnější problém je, že prostředí žádným způsobem nepodporuje spolupráci více lidí na vývoji jednoho projektu a pokročilé organizování vývoje projektu. Výhodná by také byla možnost tvorby modulů a znovu-použitelných nebo přenosných částí kódů (které by se daly sdílet).

Pro oddychovou tvorbu nebo menší projekty však mohu tento nástroj vřele doporučit, protože má mnoho co nabídnout a volnost tvorby je opravdu velká.

8 Seznam použité literatury a ostatních zdrojů

[1] LANTINGA, Sam. Simple DirectMedia Layer [online]. 2004-10-03 [cit. 2010-04-24]. Simple DirectMedia Layer. Dostupné z WWW: <http://www.libsdl.org/intro.cs/toc.html>.

[2] KOMPPA, Jari. Sol's Tutorials [online]. 2005-03-04, 2005-08-08 [cit. 2010-04-24]. Sol's Graphics for Beginners. Dostupné z WWW: <http://sol.gfxile.net/gp/>.

[3] ANDRA, Marius. Cone3D Programming - SDL, OpenGL and C++ Tutorials [online]. 2001-05-23 [cit. 2010-04-24]. Lesson 6: Space Shooter. Dostupné z WWW: <http://cone3d.gamedev.net/cgi-bin/index.pl?page=tutorials/gfxsdl/tut6>.