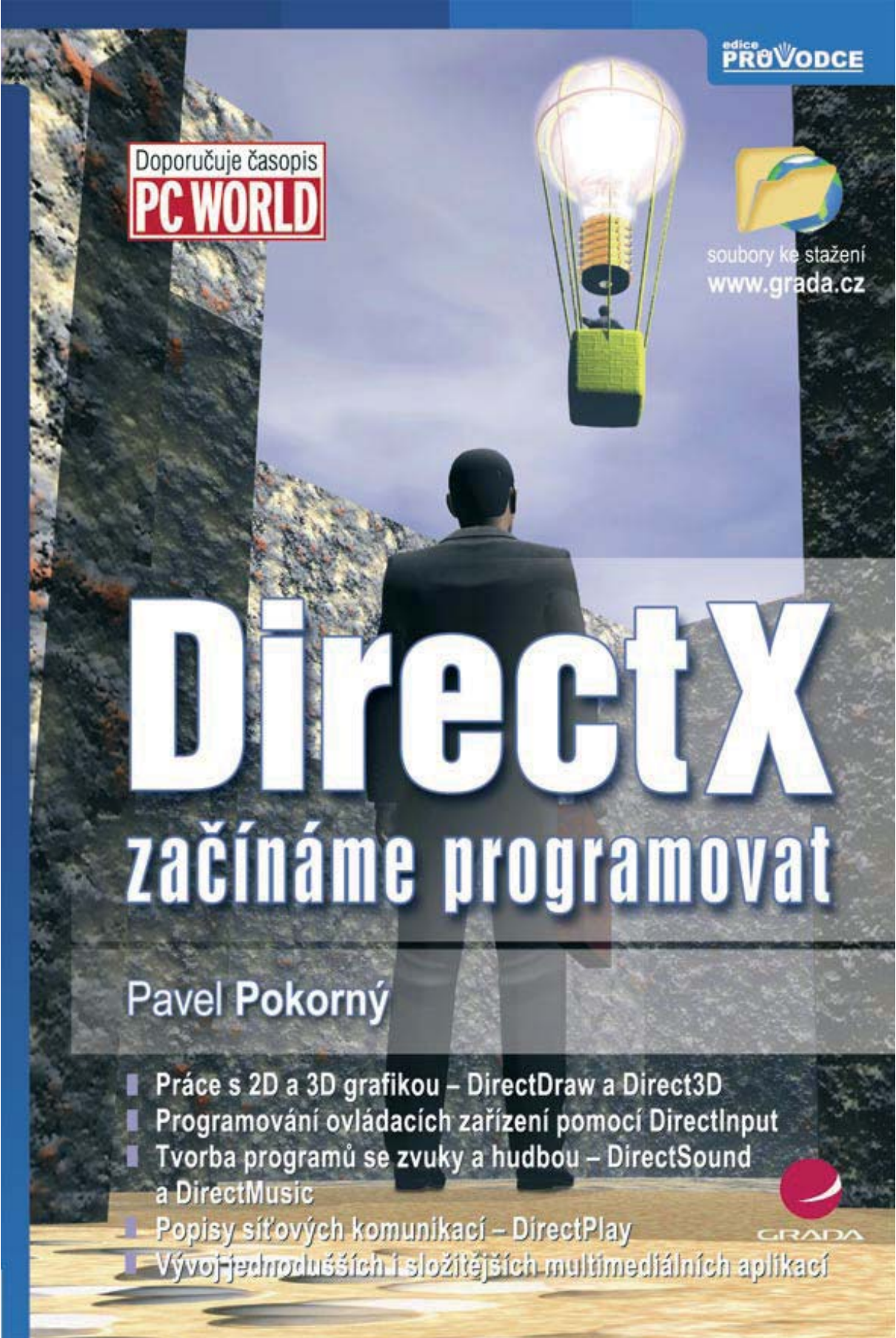

edice PROVODCE
Doporučuje časopis PC WORLD
soubory ke stažení www.grada.cz
DirectX
začínáme programovat
Pavel Pokorný
Práce s 2D a 3D grafikou – DirectDraw a Direct3D
Programování ovládacích zařízení pomocí DirectInput
Tvorba programů se zvuky a hudbou – DirectSound a DirectMusic
Popisy síťových komunikací – DirectPlay
Vývoj jednodušších i složitějších multimediálních aplikací
GRADA

Upozornění pro čtenáře a uživatele této knihy

Obsah

Úvod

Pokud jste otevřeli tuto knihu, s největší pravděpodobností alespoň tušíte, co to DirectX vlastně je. Jde o kolekci pomocných softwarových nástrojů tvořících programátorské rozhraní, tzv. API (*Application Programming Interface*) pro tvorbu multimediálních a herních aplikací. Vyvíjí ho firma Microsoft, proto se s ním můžete setkat jednak v operačním systému Windows a dále na platformách této firmy, jimiž jsou herní konzoly XBox (těmi se v této knize zabývat nebudeme).

V této knize se zaměříme na tvorbu aplikací ve Windows. Snad úplně každý počítačový uživatel si zkusil zahrát nějakou počítačovou hru, a právě u počítačových her se s DirectX můžeme setkat velice často. Důvody jsou jednoduché. DirectX poskytuje velice kvalitní nástroje pro práci s 2D i 3D grafikou, práci se vstupními zařízeními, se zvuky a hudbou, ale i pro komunikaci mezi počítači. A právě tyto věci jsou u multimediálních a herních aplikací stěžejní. Pokud tedy chcete nahlédnout pod pokličku tvorby takových programů, je tato kniha určena právě pro vás. Provede vás základy tvorby DirectX aplikací pro verzi DirectX 9.0c. Po jejím pečlivém prostudování byste měli být schopni ve svých programech používat jednotlivé komponenty DirectX. Nicméně vás zcela jistě nenaučí všemu. Celá problematika má totiž mnohem větší rozsah, který tato kniha nestačí pokrýt. Proto chce-

te-li se této oblasti věnovat opravdu důkladně, doporučuji prostudovat publikace, jejichž seznam se nachází na konci této knihy, a aktuálně sledovat internetové servery, věnující se této problematice.

Předpoklady ke studiu této knihy

Pro úspěšné studium této knihy budu předpokládat, že znáte programovací jazyk C++. Veškeré příklady a programové ukázky jsou právě v tomto jazyce. Nemůžeme zde vysvětlovat obecné programovací principy a základní pojmy jako proměnná, pole, ukazatel, objekt, zapouzdření, přístupová práva ve třídách atd. Pokud s programováním začínáte nebo částečně ovládáte jazyk C, doporučuji nejprve prostudovat nějakou odbornou publikaci, například [2]. Výhodou, ale ne nutností je také alespoň základní znalost Win32 API. Při programování jednodušších DirectX aplikací toho o Win32 API mnoho vědět nepotřebujete. To, co bude nutné z Win32 API vědět, se dočtete v této knize. Nicméně Win32API je velice rozsáhlé a nabízí spoustu užitečných funkcí, které můžete využít i později. Proto je vhodné mít o jeho možnostech alespoň přehled. Něco z Win32API využijeme i v ukázkových programech této knihy. Kompletní popis Win32API můžete získat studiem knih [4] nebo [7]. Kromě tohoto jsou také vhodné znalosti určitých oblastí matematiky a počítačové grafiky (geometrie, trigonometrie, lineární algebra, rasterizace, grafické formáty, základy promítání apod.), které vám usnadní práci s Direct3D.

Samozřejmě, že mnoho informací a ukázkových programů jazyka C++ a Win32 API naleznete i na internetu. Některé internetové odkazy naleznete i na konci této knihy, nebo můžete použít svůj oblíbený vyhledávač. Existuje mnoho webových serverů, které se tímto zabývají. Také tyto zdroje bývají doprovázeny četnými ukázkami programových kódů, jejichž prostudováním a praktickým vyzkoušením se toho naučíte nejvíce.

Koncepce knihy

Celková koncepce knihy se může na první pohled zdát ne zrovna vhodná. Ve většině kapitol jsou jednotlivá rozhraní, jejich objekty a metody vysvětleny přímo na příkladech. Důvodem chybějícího teoretického popisu ještě před příkladem je skutečnost, že nejde o příliš obsáhlá témata a přímo z programového kódu by měl být takový popis mnohem názornější. Naopak v místech, kdy by teorie bylo příliš mnoho, což by narušovalo plynulost popisu programu, je tento teoretický popis na začátku příslušné kapitoly uveden.

V knize je zejména kladen důraz na popis rozhraní, objektů, funkcí a datových typů, které patří do DirectX. Až na poslední dvě kapitoly (věnované DirectSound, DirectMusic a DirectPlay) je vše demonstrováno na funkčních programových ukázkách. Nebojte s nimi experimentovat a libovolně je upravovat. A nenechte se odradit chybami, které vám při tom možná bude hlásit kompilátor. I o tom je totiž práce programátora. Zkušený programátor ví, že nejde jen o napsání nějakého programového kódu, ale i o jeho efektivitu, rychlost, paměťové nároky atd. A nedílnou součástí programů je i vyhledávání a odstraňování chyb v programech, tzv. ladění. Ostatně jedno přísloví říká, že chybami se člověk učí.

Pro přehlednost textu v knize jsou příkazové nabídky či názvy souborů psány tučně a anglické výrazy jsou psány kurzívou. Zdrojové programy a programové příkazy jsou pak odlišným fontem. Občas je možné v textu objevit slova, která jsou v určitých případech odlišným fontem psána a jindy ne. Příkladem je DirectDraw. Zde nejde o chybu, ale opět o odlišení. DirectDraw je myšleno jako komponenta DirectX, zatímco `DirectDraw` představuje objekt, tedy část programu.

V této knize nejsou některé anglické výrazy překládány do češtiny. V některých českých publikacích tohoto typu naleznete pro výrazy „surface“, „front buffer“ a „back buffer“ překlady jako „povrch“, „přední povrch“ a „zadní povrch“. Osobně se však raději držím anglické terminologie, aniž by však tyto výrazy byly psány kurzívou. Je to výhoda i pro čtenáře, kteří budou studovat anglicky psané publikace tohoto typu.

Zdrojové kódy knihy

Programové ukázky, které naleznete v této knize, nemusíte opisovat. Stačí navštívit domovskou stránku nakladatelství Grada (*www.grada.cz*) a tam v sekci *Soubory a příklady ke stažení* tuto knihu vyhledat. Pod přískušným odkazem jsou tyto zdrojové kódy k dispozici. Po dekompresi staženého souboru se vytvoří složky, odpovídající číslům kapitol v této knize. V těchto složkách se nachází programy ve spustitelné formě a dále ve formě zdrojových kódů. Veškeré zdrojové kódy jsou doplněny četnými komentáři, které by měly usnadnit jejich pochopení.

Co budeme potřebovat

Než začneme programovat v DirectX, musíme mít dvě věci. První z nich je vývojové prostředí s kompilátorem. Jak jistě víte, kompilátorů existuje celá řada. U profesionálních vývojářů patří mezi nejčastěji používané Microsoft Visual Studio (nás konkrétně zajímá Microsoft Visual C++). V něm byly vytvořeny všechny programy, které v této knize naleznete. Doporučuji ho i vám, protože u jiných vývojových nástrojů mohou nastat problémy s nastavením a kompilací.

Jak jistě víte, Microsoft Visual C++ je komerční produkt. Nevlastníte-li toto vývojové prostředí, nemusíte zoufat. Existují i zkušební verze určené pro začátečníky a amatéry, kteří se chtějí naučit programovat. Zpočátku tedy nemusíte tímto směrem vynakládat žádné investice a teprve později, pokud budete chtít využívat všech předností, jimiž toto vývojové prostředí disponuje, si můžete některou z komerčních verzí Visual C++ zakoupit. Nejnovější verze, která je k dispozici zdarma, se jmenuje Microsoft Visual C++ 2005 Express Edition a stáhnout si ji můžete z internetových stránek Microsoftu [21]. Po instalaci a následném spuštění ve výběru nových projektů zjistíte, že je možné vytvořit pouze Win32 konzolové aplikace. Pro aplikace využívající DirectX potřebujeme čisté Win32 aplikace. Tuto komplikaci dokáže vyřešit programový balík Platform SDK, který si opět můžete bezplatně stáhnout z internetu. Po jeho instalaci musíme provést ještě určitá nastavení v našem vývojovém prostředí. Nastavení je poněkud více, ale nejsou nijak složitá. Pokud tedy chcete používat bezplatnou zkušební verzi Visual C++ a vytvářet v něm Win32 aplikace, stačí se držet postupu, který je popsán na internetových stránkách Microsoftu [23].

Kromě bezplatné verze Express je tu samozřejmě možnost zakoupení plné verze Microsoft Visual Studia (C++). Pokud s programováním začínáte a ještě nevíte, zda se programování stane vaším koníčkem nebo se jím budete živit, bude vám produkt nejspíše připadat velice drahý. V opačném případě se vám taková investice jistě vyplatí. Pokud však ještě nejste rozhodnuti nebo patříte spíše do první skupiny uživatelů a nechcete zpočátku příliš investovat, nabízí se ještě možnost pořídit si verzi starší. Předchozí verze nese název Microsoft Visual C++ 2003 .NET a té ještě předcházela verze Microsoft Visual C++ 6.0. Tyto starší verze se příležitostně dají zakoupit levněji.

Příklady, které jsou uvedeny v této knize, jsou vytvořeny v Microsoft Visual C++ 6.0. I samotný popis tvorby a nastavení projektů se týká této verze. Nicméně prostředí obou

novějších verzí je do značné míry koncipováno podobně jako ve verzi 6.0. Proto to, co je popsáno v následujících kapitolách, platí i pro novější verze. Samozřejmostí je, že v těchto novějších verzích můžete velice jednoduše zkompilovat všechny programy, které jsou uvedeny v této knize.

Pokud nechcete používat Microsoft Visual C++, nabízejí se i alternativy. Ať už se jedná o libovolný komerční nebo nekomerční vývojový nástroj, vyplatí se pečlivě prostudovat příslušnou dokumentaci a případě zjistit osobní zkušenosti jiných počítačových uživatelů a vyhnout se tak zbytečné investici. Protože je DirectX produkt firmy Microsoft, je nejlépe podporován vývojovými nástroji této firmy. U jiných vývojových nástrojů může mít použití DirectX jistá omezení, například nemusí správně fungovat všechny jeho součásti nebo může existovat podpora pouze zastaralých verzí.

Druhou věcí, kterou budeme pro naše programy potřebovat, je DirectX SDK (*Software Development Kit*). Jde o softwarový balík, který si můžete zdarma stáhnout ze stránek Microsoftu. [14] To nejdůležitější, co z něj potřebujeme, jsou hlavičkové soubory a knihovny, které podle implementovaných částí DirectX vkládáme a přilinkováváme do programových projektů (o způsobu implementace se můžete dočíst v druhé kapitole). Dále zde také můžeme nalézt i podrobnou dokumentaci a spoustu ukázkových programů i se zdrojovými kódy, které nám při studiu DirectX také velice pomohou. V současné době je k dispozici SDK pro DirectX verzi 9.0c. Tato verze již existuje několik let, ovšem každé dva měsíce pravidelně vycházejí aktualizace, které přinášejí jisté změny a vylepšení. To, co přibylo nebo bylo upraveno, je vždy popsáno v souboru **directx9_c.chm**, který naleznete ve složce **Documentation** v instalaci DirectX SDK.

Samozřejmě je nejvýhodnější používat vždy nejnovější SDK. Pokud však při kompilaci chcete používat starší vývojové prostředí Microsoft Visual C++ 6.0, narazíte u nových DirectX SDK na problém, neboť většina programů nejspíše nepůjde zkompilovat. Důvod je ten, že Visual C++ 6.0 je starší vývojové prostředí a není již více než dva roky oficiálně podporováno. Proto používáte-li tento kompilátor, je potřeba si z internetu stáhnout starší verzi DirectX SDK (o podpoře kompilátorů se můžete dočíst v dokumentaci ke každé verzi SDK). Pokud ale s DirectX začínáte a budete programy vytvářet podle této knihy, nemusíte se ničeho obávat. Zde uvedené programy jsou koncipovány tak, aby šly kompilovat ve Visual C++ 6.0 i starších DirectX 9.0c SDK. Na internetových stránkách Microsoftu [14] si můžete stáhnout jak nové, tak i starší SDK.

Poděkování a podpora

Na tomto místě bych chtěl poděkovat všem lidem, kteří mi umožnili tuto knihu napsat. Jde především o množství nejmenovaných programátorů, kteří se prostřednictvím internetu nebáli zveřejnit svoje osobní zkušenosti s DirectX na nejrůznějších příkladech a programech. Výčet často navštěvovaných internetových stránek, které se programováním v DirectX zabývají, najdete na konci této knihy. Ale poděkování patří i všem lidem v mém blízkém okolí, jejichž větší či menší pomoc, trpělivost a tolerance mi umožnily tuto knihu napsat.

Autor

Tato publikace vznikla za podpory výzkumného záměru MSM 7088352102.

1.

Bližší seznámení s DirectX

V úvodu jsme se krátce zmínili o tom, že DirectX je tzv. API rozhraní, které bylo navrženo pro snazší tvorbu multimediálních a herních aplikací. Toto API rozhraní obsahuje několik součástí, které jsou u podobných aplikací často používány. Jde především o nástroje pro práci s 2D/3D grafikou, zvuky (hudbou), síťovou komunikaci mezi počítači, vstupními zařízeními a přehrávání.

Setkat se s DirectX nemusíme jen u počítačových her nebo multimedií, ale stále častěji se také uplatňuje v aplikacích, které jsou vytvářeny pro praktické využití v průmyslových odvětvích (například strojírenství). Důvodem je zejména Direct3D, které je hardwarově podporováno výrobci grafických karet a dokáže tak rychle poskytnout vysoce kvalitní 3D grafiku [13].

Z programátorského pohledu je DirectX založeno na tzv. COM (*Component Object Model*). COM představuje softwarovou architekturu definující vzájemné vztahy mezi jednotlivými komponentami. S rostoucími možnostmi počítačů roste i složitost programů. Proto se programy mohou skládat z komponent (komponenta je část programu, která má jisté specifikace). Jednotlivé komponenty společně komunikují prostřednictvím společného protokolu. COM tak poskytuje sjednocený, rozšiřitelný, objektově orientovaný, komuni-

kační protokol. Jednoznačnou výhodou COM je tak přesná specifikace rozhraní, která zjednodušuje vývojářům tvorbu nových aplikací [1].

Pokud bychom měli uvést nějakou nevýhodu DirectX, je jí platformní závislost. DirectX je spojen pouze s Microsoft Windows, proto vyvíjené aplikace, které ho využívají, nelze zprovoznit pod jiným operačním systémem. Uživatelé, kteří chtějí používat DirectX programy navíc musí mít navíc nainstalovaný tzv. DirectX *end-user runtime*, který bývá součástí médií s takovými aplikacemi, nebo je možné ho zdarma stáhnout z internetových stránek Microsoftu [14]. Samozřejmě je přitom žádoucí, aby byla nainstalovaná nejnovější verze, kterou je ve Windows XP v současné době DirectX 9.0c.

1.1 Součásti DirectX

V této podkapitole si blíže představíme jednotlivá rozhraní, která jsou součástí DirectX 9.0c. Více se o nich samozřejmě dozvíme v dalších kapitolách této knihy, kde bude popsána tvorba programových aplikací využívajících tato rozhraní. Rozhraní je několik a vývojáři ocení jednu velice příznivou skutečnost – jejich programová implementace je velice podobná. Proto se nemusíte obávat, že se musíte složitě učit každou součást DirectX zvlášť. Pokud se tedy naučíte používat jedno z těchto rozhraní, nebude pro vás složité zvládnout rozhraní jiné.

- ✓ **DirectX Graphics** – zahrnuje v sobě DirectDraw a Direct3D.
  - ✓ **DirectDraw** – používá se při práci s rastrovou 2D grafikou, přičemž důraz je kladen na maximální výkonnost. Podporuje hardwarovou akceleraci, tedy čím více funkcí je podporováno grafickou kartou, tím rychlejší jsou i programy, které tyto funkce využívají. Pokud taková podpora chybí, tyto funkce se emulují softwarově, ovšem právě za cenu ztráty výkonu. DirectDraw umožňuje vytvářet aplikace, jež běží v okně i v celoobrazovkovém režimu. V současné době se již DirectDraw nevyvíjí, poslední aktualizace tohoto API byly provedeny v DirectX verzi 7 a od verze DirectX 8 je DirectDraw spojeno s Direct3D. Nicméně ho můžete samozřejmě používat i v DirectX 9.
  - ✓ **Direct3D** – jak z názvu vyplývá, toto rozhraní je určeno pro práci s 3D grafikou (obsahuje i několik funkcí pro práci s 2D grafikou). Podobně jako u DirectDraw je i zde kladen důraz na maximální rychlost a výkonnost. Proto také staví na hardwarové akceleraci a nejvyšších výkonů dosáhneme, pokud máme grafickou kartu, která z tohoto rozhraní hardwarově podporuje co nejvíce funkcí. Mezi možnosti Direct3D patří například kreslení a prohlížení objektů, definování formátu pixelů, používání světelných zdrojů ve scéně, vylepšování obrazu (např. antialiasing, tedy vyhlazení ostrých hran), mapování textur nebo animování. To vše je samozřejmě možné provádět v aplikacích běžících v okně nebo v celoobrazovkovém režimu.
- ✓ **DirectX Input** – jeho součástí je DirectInput, což je rozhraní, které poskytuje služby nejrůznějších vstupních zařízení, jako je klávesnice, myš, joystick nebo gamepad. Součástí je i podpora tzv. *action mapping*. To znamená, že u vstupních zařízeních s programovatelnými tlačítky můžeme těmto tlačítkům přiřadit námi určené funkce.
- ✓ **DirectX Audio** – součástí tohoto API je DirectSound. Jak již z názvu vyplývá, jde o rozhraní umožňující komunikaci se zvukovými kartami, umožňuje tedy vytvářet

aplikace se zvuky a hudbou. Toto rozhraní je sdílené. Ke zvukové kartě tak může přistupovat několika aplikací zároveň. Samozřejmostí jsou základní i rozšiřující funkce, jimiž jsou například míchání (mixování) těchto zvuků, softwarové nastavování hlasitostí či urychlování/zpomalování přehrávání skladeb. Součástí je i práce s 3D zvuky (DirectSound3D).

- ✓ **DirectMusic** – seskupení objektů, které umožňuje podobně jako DirectSound přehrávání zvuků a hudby, ovšem na vyšší úrovni než DirectSound. DirectMusic má více možností, například přehrávání více zvukových formátů (především MIDI), kompletní systém pro implementaci dynamických zvukových stop, které se mohou měnit na základě určitých událostí, dále *downloadable sounds* (DLS – standard pro syntetizaci zvuků z digitálních vzorků uložených v programech, který zajistí stejný zvukový výstup na všech počítačích) a 3D polohové efekty hudebních a zvukových zdrojů.
- ✓ **DirectPlay** – jde o knihovnu objektů, určenou pro komunikaci s jinými počítači, které jsou mezi sebou vzájemně síťově propojeny. Využití tedy nalezne především v multiplayerových počítačových hrách nebo chatovacích aplikacích. Jednotlivé počítače mohou mezi sebou komunikovat přímo (peer-to-peer), nebo pomocí centrálního serveru (klient/server komunikace).

1.2 Pohled do historie DirectX

Vznik DirectX je spojen s operačním systémem Microsoft Windows 95. Oficiálně byl tento operační systém vydán v roce 1994 a pochopitelnou snahou Microsoftu bylo ho co nejvíce rozšířit mezi uživatele. To znamená, že muselo být možné dobře a efektivně tvořit co nejširší spektrum kvalitních aplikací. Oblast tvorby počítačových her pod Windows 95 byla problematická, protože funkce, které Windows poskytovala, byly pomalé a jejich možnosti omezené. Důvodem byl především chráněný paměťový režim, který blokoval přímý přístup k zařízením typu grafická či zvuková karta. Naproti tomu ve starším operačním systému (DOS) tento přímý přístup možný byl, proto i po uvedení Windows 95 byly počítačové hry stále vyvíjeny pro DOS [13].

Microsoft pochopitelně hledal cestu, jak zmíněná omezení Windows odstranit. Z těchto důvodů vzniklo DirectX. První verze byla uvolněna v září 1995 a nesla ještě původní název „Windows Games SDK“. Vývojáři navrhli základní grafickou technologii, která byla již od počátku tvořena tak, aby poskytovala co největší výkonnost. S postupem času, kdy narůstaly možnosti grafických karet a rychlost počítačů, bylo nutné provádět i aktualizace a rozšiřovat schopnosti DirectX. Jen tak mohlo držet krok s dobou a stále více se prosazovat v počítačových hrách, a posléze i mimo ně (v oblasti počítačové grafiky bylo a je nejvážnějším „konkurentem“ OpenGL). Proto vznikaly stále častěji nové verze. Přehled nejdůležitějších verzí s datem vydání a poznámkami obsahuje tabulka 1.1.

Tabulka 1.1: Přehled nejdůležitějších verzí DirectX. Zdroj: [13]

Verze	Datum vydání	Poznámky
DirectX 1.0	30. 9. 1995	
DirectX 2.0/2.0a	5. 6. 1996	Určeno pro Windows 95 OSR2 a Windows NT 4.0.
DirectX 3.0/3.0a	15. 9. 1996	Poslední verze, která podporovala Windows NT 4.0.

Verze	Datum vydání	Poznámky
DirectX 4.0		Nikdy nebylo vydáno.
DirectX 5.0	16. 7. 1997	Jako beta je dostupné i pro Windows NT 5.0.
DirectX 5.1	1. 12. 1997	
DirectX 5.2	5. 5. 1998	Dvě verze – pro Windows 95 a Windows 98.
DirectX 6.0	7. 8. 1998	
DirectX 6.1	3. 2. 1999	Určeno i pro Windows 98 SE.
DirectX 7.0	22. 9. 1999	Verze pro Windows 2000.
DirectX 7.1	26. 9. 1999	Určeno i pro Windows 98 ME.
DirectX 8.0	30. 9. 2000	
DirectX 8.0a	7. 11. 2000	Poslední verze, která šla nainstalovat do Windows 95.
DirectX 8.1	12. 11. 2001	Určeno i pro verzi Windows XP.
DirectX 9.0	19. 12. 2002	
DirectX 9.0a	26. 3. 2003	
DirectX 9.0b	13. 8. 2003	
DirectX 9.0c	13. 12. 2004	
DirectX 10	30. 11. 2006	Určeno pouze pro Windows Vista.

Z této tabulky je patrné, že předposlední oficiální DirectX je verze 9.0c. Nicméně již výše jsme si uvedli, že každé dva měsíce vznikají nové aktualizace DirectX SDK, které samozřejmě neovlivňují uživatele aplikací, ale především vývojáře. Postupně se kromě podpory 32bitových programů objevila i podpora pro 64bitové aplikace a v několika posledních verzích SDK se objevily i hlavičkové soubory s knihovnami a příklady pro nadcházející DirectX 10.

Tvorbou aplikací pro DirectX 10 se v této knize zabývat nebudeme. Přesto se alespoň krátce zmiňme o této (prozatím nejnovější) verzi. V době psaní této knihy (léto 2007) je přímou součástí Windows Vista – pro předchozí verze Windows tedy není k dispozici. V každém případě ale DirectX 10 přinášejí řadu podstatných změn, mezi nimiž nechybí kompletně přepracované API, jeho rozdělení do dvou částí (smyslem je zvýšit stabilitu systému), celkové urychlení nebo unifikovaná podpora shaderů. Změn a rozšíření je pochopitelně více, a jak dokazují první aplikace, které DirectX 10 využívají, zejména v oblasti počítačové grafiky dochází k velkému posunu vpřed. V budoucnosti se tak tato verze jistě dočká masivnějšího rozšíření.

Pohled do historie DirectX zakončíme odstavcem věnujícím se vzájemné kompatibilitě jednotlivých verzí. U všech verzí kromě nejnovějších DirectX 10 platí, že jsou zpětně kompatibilní. V praxi to znamená, že pokud máte například nainstalované DirectX 8.0, spustíte i aplikaci, která byla vyvíjena pro všechna starší DirectX (např. DirectX 6.1). Toto je výhoda modelu COM, na němž je DirectX postaven. Obdobně to platí i při vývoji aplikací. V novějších verzích DirectX SDK naleznete i hlavičkové soubory a knihovny starších verzí. Toto se mění u DirectX 10, jež by mělo být kompatibilní pouze s předposledními verzemi – DirectX 7, 8 a 9. Protože bylo předělané celé API, tato kompatibilita je dána tak, že se starší rozhraní pouze emulují. To znamená, že drtivá většina aplikací, jež využívají starší rozhraní, je sice funkční, ale některé testy ukazují, že takové aplikace mohou být pod DirectX 10 v řádu několika procent pomalejší.

2.

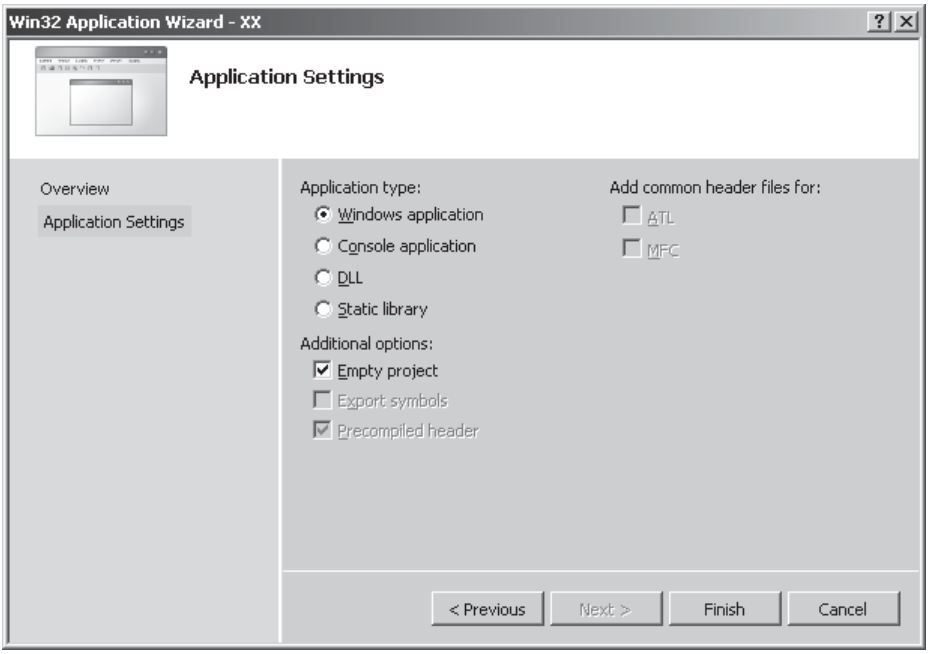
Základní aplikace

Po úvodním seznámení s DirectX se dostáváme k programování. V této kapitole ještě nebudeme vytvářet DirectX aplikace. Její náplní je jednak popis způsobu nastavení prostředí a projektu v Microsoft Visual C++ a vytvoření základní Win32 aplikace. Z takové aplikace budeme vytvářet v dalších částech této knihy programy, které již DirectX budou využívat. Poslední část této kapitoly je věnována způsobu, jak zjistit verzi DirectX, která je v počítači instalována.

2.1 Nastavení DirectX pro naše aplikace

Než si popíšeme tvorbu projektu pro DirectX aplikace, je třeba mít nainstalovaný DirectX *end-user runtime* ve verzi, pro kterou budete aplikace vytvářet (v současné době je to DirectX 9.0c) [14]. V opačném případě by nešly takové aplikace spustit. Tento programový balík vyžadují všechny aplikace, které jsou založeny na DirectX. Dále – pokud ještě nemáte – stáhněte si a nainstalujte DirectX SDK (o tomto balíčku jsme se zmínili již v úvodu).

Nyní však již k vytvoření projektu. Následující popis se vztahuje k Microsoft Visual C++ 6.0.



Obrázek 2.1: Okno Visual C++ Express Edition pro nastavení nové aplikace

Po spuštění Visual C++ zadáme příkaz **File → New** a poté **Project**. V okně projektů si vybereme **Win32 Application**, doplníme jméno projektu a případně si nastavíme cestu do adresáře na disku, kde ho chceme vytvořit. Po stisknutí tlačítka **OK** se dostaneme do další nabídky, kde si můžeme zvolit, jaký typ aplikace Windows chceme vytvořit. Zaškrtneme volbu **An Empty Project** a klepneme na tlačítko **Finish**, čímž návrh projektu dokončíme. Objeví se ještě jedno okno, kde vidíme souhrn námi provedených nastavení. Po dalším stisknutí tlačítka **OK** je projekt vytvořen.

V prostředí Visual C++ Express Edition postupujeme obdobně. Tedy po příkazu **File → New → Project** se opět dostaneme do okna vytvoření projektu. Zde nemáme k dispozici volbu **Win32 Application**, proto si musíme vybrat **Win32 Console Application**. Opět doplníme název projektu a vybereme jeho umístění na disku v počítači. V dalším okně máme dvě nabídky – **Overview** a **Application Settings**. V nabídce **Overview** máme přehled o nastavení projektu. Nás ale zajímá druhá nabídka. Pokud se tedy přepneme do **Application Settings**, můžeme si vybrat typ aplikace (**Application type** – zde zaškrtněte volbu **Windows application**) a v **Additional options** zaškrtněte volbu **Empty project** (prázdný projekt). Toto okno vidíte na obrázku 2.1. Poté již stačí pouze klepnout na tlačítko **Finish** a projekt je vytvořen.

Možná se v tuto chvíli ptáte, proč jsme vytvářeli prázdný projekt. Kdybychom si nevybrali volbu **Empty project**, průvodce vytvoření projektu by počáteční aplikaci Windows vytvořil za nás. To je sice pravda, ale takto vytvořená aplikace nebude objektová, a navíc pokud máme vytvořený prázdný projekt, nic v něm není. V tom projektu bude tedy vždy jen to, co do něj sami vložíme. A konečně, tuto základní aplikaci si ve zbytku této kapitoly popíšeme, takže při jejím vytváření lépe pochopíte, jak vlastně funguje.

Projekt tedy máme vytvořený. Než začneme programovat, musíme ještě provést jistá nastavení. Těmito nastaveními je myšleno nastavení cesty do adresářů, odkud se budou načítat hlavičkové a knihovní soubory DirectX (soubory s příponou ***.h** a ***.lib**). V Microsoft

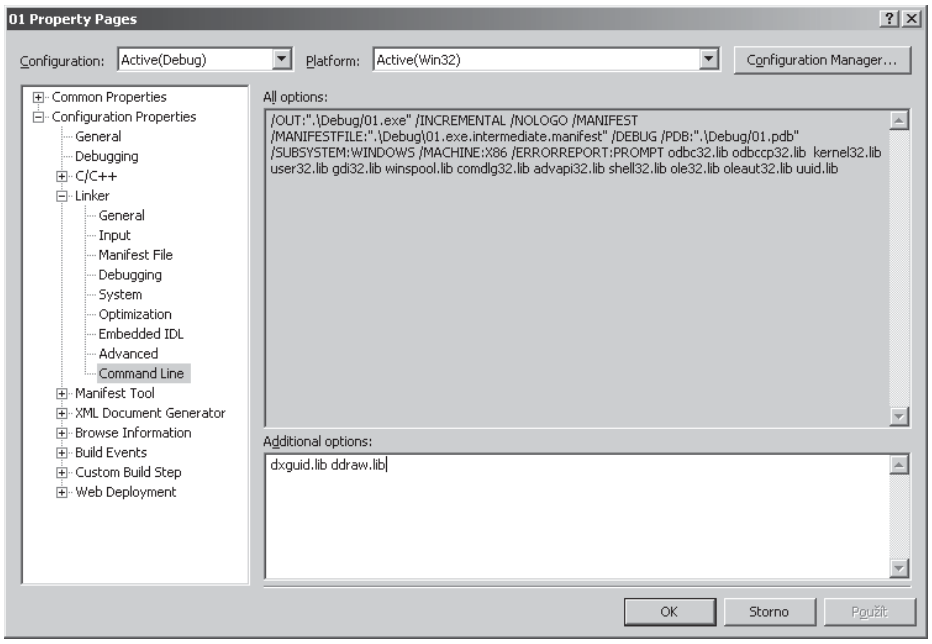
Visual C++ 6.0 toto naleznete v nabídce **Tools → Options** na kartě **Directories**. Zde pro hlavičkové soubory (**Include files**) a knihovní soubory (**Library files**) nastavte cesty do adresářů **Include** a **Lib** na disku, kam jste nainstalovali DirectX SDK. Ve Visual C++ Express Edition naleznete stejné nastavení v položkách **Tools → Options** ve složkách **Projects and Solutions** a **VC++ Directories**.

Úplně poslední věcí, kterou při nastavení projektu můžeme (ale nemusíme) provést, je nastavení knihoven, které se budou přilinkovávat k našemu programu. Toto nastavení provádět nemusíme, protože do zdrojového kódu našeho programu můžeme někam vložit direktivu například v tomto znění:

```
#pragma comment (lib,"d:\\Dx\\Lib\\dxguid.lib")
```

Potom to bude mít stejný účinek, jako bychom soubor **dxguid.lib** přidali do seznamu přilinkovávaných knihoven našeho projektu. Stejným způsobem bychom přidávali i jiné knihovny (o tom, které knihovny jsou kdy potřeba, si povíme na začátku jednotlivých kapitol o částech DirectX). Samozřejmě pokud máme nastavenou cestu ke knihovnám tak, jak jsme si uvedli výše, stačí pouze uvádět jméno souboru a nemusíme psát celou cestu k němu.

Pokud chceme provést nastavení přilinkovávaných knihoven přímo v projektu, ve Visual C++ 6.0 se to dělá příkazem **Project → Settings** pro typ projektu (standardně **Debug** nebo **Release**) na kartě **Link**. Ve formulářovém poli **Object/library modules** je již několik automaticky přidaných knihoven. Stačí k nim tedy připojit jméno souboru, který chceme přidat.



Obrázek 2.2: Okno Visual C++ Express Edition pro přidání přilinkovávaných knihoven

Ve Visual C++ Express Edition se knihovny do projektu přidávají v nabídce **Project → Properties**. Objeví se okno, v jehož levé části se nachází rozbalovací stromová struktura s několika rozbalovacími složkami. Nás konkrétně zajímá složka **Configuration Pro-**

perties – Linker – Command Line (viz obrázek 2.2). Pro zvolenou konfiguraci v horní části tohoto okna (**Debug**, **Release**, nebo nějaká vlastní) se do pole **Additional options** zapisují přilinkovávané knihovny. Jednotlivé zapsané soubory se oddělují mezerou.

2.2 Vytvoření základní Win32 aplikace

Dříve, než si vytvoříme základní Win32 aplikaci, si uvedeme několik informací o koncepci takových programů. Doporučuji tedy prostudovat a zamyslet se nad následujícími řádky především těm z vás, kteří dosud žádný Win32 program netvořili. Usnadní vám to porozumění principům tvorby Win32 aplikací. Tato koncepce je od klasických konzolových programů poněkud odlišná.

Pochopení tvorby takových aplikací nám usnadní zamyšlení nad chováním platformy Windows. Jak jistě víte, jde o grafický operační systém, přičemž v jednom okamžiku můžeme mít spuštěno více aplikací. To znamená, že potřebujeme mít určitým způsobem rozdělenou paměť. Jednotlivé programy také potřebují sdílet nejrůznější zařízení, neboť stejné zařízení může využívat i jiná aplikace. Z těchto důvodů je programování Windows založeno na tzv. událostech. Událostí je například pohyb myši, klepnutí některého tlačítka myši, stisknutí některé klávesy nebo změna velikosti okna aplikace. V programu se dají jednotlivé události rozlišit, proto můžeme zajistit reakci na každou událost zvlášť. Samozřejmě, že také můžeme nechat bez povšimnutí události, která nás nezajímají.

Události můžeme v našem programu zpravidla zachytit podle zpráv, které Windows posílají vždy konkrétnímu oknu aplikace, pro niž je určena. Příslušné okno potom může na konkrétní zprávu zareagovat. Pro nás jako programátory je nejdůležitější vědět, že musíme napsat funkci, které se říká procedura okna. Uvnitř ní definujeme zprávy, které nás zajímají, a dále pak to, jakým způsobem na ně budeme chtít reagovat. O posílání zpráv této funkci se nemusíme starat, to udělají Windows za nás.

Shrnutí těchto poznatků o tvorbě základní Win32 aplikace je tedy následující: po spuštění programu se nejprve registruje třída okna, jemuž definujeme požadované vlastnosti. Na základě této registrace se vytvoří a vykreslí okno Windows aplikace. Potom již probíhá smyčka zpráv, vybírající z fronty ty zprávy, které se týkají naší aplikace, jež se následně předávají proceduře okna, kde můžeme definovat reakce. Zmíněná smyčka zpráv probíhá cyklicky stále dokola, dokud není vyzvednutá zpráva `WM_QUIT` (viz dále) udávající, že má aplikace končit. To je vše.

Jistě uznáte, že na základě předchozího popisu nevypadá princip činnosti programů pro Windows složitě. V tuto chvíli si tedy projděme celý programový kód, který nám tuto aplikaci vytvoří. Jak bylo zmíněno v úvodu knihy, všechny programy, včetně tohoto, budou vytvářeny objektově. V nové Win32 aplikaci tedy vytvoříme šest souborů. První z nich bude obsahovat globální prvky naší aplikace, můžeme ho proto nazvat například **Global.h**. Další soubor **WinMain.cpp** bude obsahovat hlavní smyčku programu, včetně zpracování smyčky zpráv. V tomto souboru se vytvoří instance třídy `CApplication` a následně se provede zavolání metody jejího vytvoření.

Tato třída se bude nacházet v souboru **CApplication.h** a definice jejích metod v souboru **CApplication.cpp**. Funkce třídy `CApplication` je chápána ve smyslu správy celé aplikace. Pro práci s oknem aplikace nám poslouží třída `CWindow`, kterou umístíme do souboru **CWindow.h** a jejíž metody budou definovány v souboru **CWindow.cpp**.

Nyní k jednotlivým souborům podrobněji. Obsah souboru **Global.h** vypadá následovně:

```
#pragma once

#include <windows.h>

#define APP_NAME TEXT("Moje aplikace")
```

Uvnitř tohoto souboru se nacházejí pouze tři příkazy. Direktiva `#pragma once` zajistí, že se tento soubor bude k projektu přidávat pouze jednou, přestože je vložen do jiných souborů vícekrát. Poté vkládáme hlavičkový soubor **windows.h**, čímž si zajistíme možnost využívání proměnných, struktur a funkcí Win32 API. Poslední příkaz přiřazuje symbolické konstantě `APP_NAME` text `"Moje aplikace"`. Jde o text obsahující název aplikace, který se zobrazí v záhlaví našeho okna. Makro `TEXT` nám říká, že jde o textový řetězec v podobě mezinárodního kódování UNICODE.

Nyní si popíšeme strukturu souboru **Winmain.cpp**. Jak bylo zmíněno výše, v tomto souboru začíná „život" celé aplikace, a vlastně zde i končí.


```
#include "Global.h"
#include "CApplication.h"

int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
      CApplication App;
      MSG msg;

      if(!App.Initialize())
      {
            MessageBox (NULL, TEXT ("Chyba při inicializaci!"),
              APP_NAME, MB_ICONERROR) ;

            return (0);
      }

      while(true)
      {
            if(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
            {
                  if(msg.message == WM_QUIT)
                        break;
                  TranslateMessage(&msg);
                  DispatchMessage(&msg);
            }
            // Zde mohou být volány funkce našeho programu
      }

      return (msg.wParam);
}
```

Na počátku vkládáme dvojici hlavičkových souborů – **Global.h** a **CApplication.h**. První, jak víme, obsahuje globální příkazy pro aplikaci, druhý obsahuje třídu `CApplication`, jejíž instanci v hlavní smyčce programu vytvoříme. Zatímco u konzolových aplikací se hlavní funkce programu jmenuje `main`, zde se jmenuje `WinMain`. Vrací hodnotu typu `int` a pokud se podíváme na konec programu, vrací se `msg.wParam`, což je primární parametr zprávy (viz dále). Funkce `WinMain` se spouští se čtveřicí parametrů. Můžete si všimnout,

že v našem výpisu kódu jsou uvedeny pouze jejich typy a nikoliv identifikátory. To je proto, že je nebudeme potřebovat. Nicméně alespoň ve stručnosti si povězme, že první dva parametry jsou typu `HINSTANCE`, což ve znamená handle instance. Handle představuje číslo a první parametr je tedy číslo, které jednoznačně identifikuje program (některé funkce tento parametr mohou vyžadovat). Druhý parametr je stejného typu a je to jakýsi pozůstatek ze starších verzí Windows, kdy se při vícenásobném spuštění programu sdílela paměť pro čtení. Je to tedy handle předchozí instance programu. Třetím parametrem je příkazový řádek, který se používá pro spuštění programu (to samé, co může mít i funkce `main`). A poslední (čtvrtý) parametr udává, jak má být program poprvé zobrazen – okno programu může být zobrazeno normálně, maximalizované nebo minimalizované.

V hlavní smyčce programu vytváříme instanci třídy `CApplication` s názvem `App` a dále vytváříme identifikátor `msg` na základě struktury zprávy `MSG`. Ten budeme potřebovat pro zpracování zpráv Windows. Potom je volána metoda `Initialize()` instance `App`, která provádí inicializaci včetně vykreslení okna. Podrobnostmi této metody se budeme zabývat v dalším textu. V tuto chvíli je pro nás ale podstatné to, že tato metoda vrací binární hodnotu (`bool`) v závislosti na tom, zda inicializace proběhla v pořádku. Pokud ne, zobrazí se dialogové okno, že došlo k chybě (funkce `MessageBox`) a program skončí. Prvním parametrem této funkce je obvykle handle okna (v našem případě žádné není – `NULL`), druhým je textový řetězec, který se objeví uvnitř dialogového okna, třetím parametrem pak je text v záhlaví tohoto okna (makro `APP_NAME`). Čtvrtý argument je kombinace konstant, které nám říkají, jaká tlačítka a ikony se mají v tomto dialogu objevit. V našem programu zobrazujeme chybovou ikonu danou symbolickou konstantou `MB_ICONERROR`.

Zbylou část programu tvoří nekonečný cyklus obsahující dvě části. V první části se pracuje se zprávami Windows a druhou část máme v našem programu prázdnou. Jak napovídá komentář, zde se vkládají funkce a algoritmy, které chceme, aby náš program vykonával. Popišme si podrobněji první část tohoto cyklu. Funkce `Peekmessage` zjišťuje, zda se ve frontě nachází zpráva určená oknu programu. Pokud ano, umístí ji do naší struktury `msg` a vrátí hodnotu `true` (jinak vrací `false`). Protože máme volání této funkce v podmínce, v případě jejího splnění se provede tělo této podmínky. Druhým parametrem funkce `Peekmessage` je handle okna, které zprávy přijímá (v našem případě `NULL`) a třetí a čtvrtý parametr udávají meze zpráv, jež se mají načítat (minimální a maximální). My zde máme uvedeny hodnoty 0, protože chceme načítat všechny zprávy. U posledního parametru zde máme hodnotu `PM_REMOVE`, která udává, že mají být tyto zprávy po načtení z fronty odstraněny.

Pokud naše okno aplikace obdrží nějakou zprávu, proběhne kód uvnitř zmíněné podmínky. Jako první zde máme další podmínku. Jejím smyslem je zjistit, zda zpráva, která byla přijata, není náhodou `WM_QUIT`. Jak z jejího názvu vyplývá, jde o zprávu, která oznamuje, že má program skončit. Pokud je tato zpráva zaslána, celý (nekonečný) cyklus `while` se přeruší a program skončí. V opačném případě proběhne ještě volání dvou funkcí a cyklus bude pokračovat dál. První z těchto funkcí je `TranslateMessage`. Tato funkce předává strukturu `msg` (její parametr) zpět Windows kvůli klávesnicovým převodům. Druhá funkce (`DispatchMessage`) předává strukturu `msg` opět Windows, tentokrát ovšem Windows tuto zprávu zašlou proceduře okna, kde na ní můžeme reagovat (viz dále).

Soubor **CApplication.h** obsahuje následující kód:

```
#pragma once

#include "CWindow.h"
```

```
class CApplication
{
      private:
            CWindow m_Window;

      public:
            CApplication(void);
            ~CApplication(void);
            bool Initialize(void);
            void Terminate(void);
};
```

Obsahem souboru **CApplication.h** je tedy třída, která má stejný název jako soubor. Úkolem této třídy je správa aplikace a jejích algoritmů. V našem případě tu máme pouze konstruktor a destruktor, metody `Initialize` a `Terminate`. Smyslem metody `Initialize` je provést inicializaci naší aplikace (jak víme, tato metoda se volá z hlavní smyčky programu **Winmain.cpp**). Naopak `Terminate` bude provádět ukončovací operace. Kromě těchto metod třída pouze agreguje instanci třídy `CWindow` s názvem `m_Window` pro práci s oknem našeho programu (proto na začátku musíme vkládat hlavičkový soubor **CWindow.h**). Tato třída je jednoduchá, protože máme jednoduchou aplikaci. S rozsáhlejší aplikaci by samozřejmě úměrně rostla složitost této třídy (práce se zvuky a hudbou, umělou inteligencí programu apod.).

Soubor **CApplication.cpp** obsahuje definované metody třídy `CApplication`. Výpis je následující:

```
#include "Global.h"
#include "CApplication.h"

CApplication::CApplication(void)
{
}

CApplication::~CApplication(void)
{
      Terminate();
}

bool CApplication::Initialize(void)
{
      if(!m_Window.Initialize())
            return (false);
      return (true);
}

void CApplication::Terminate(void)
{
}
```

Po vložení potřebných hlavičkových souborů zde máme konstruktor a destruktor. Zatímco konstruktor je prázdný, z destruktoru se volá metoda `Terminate`. Ani jedna z těchto tří metod by tu nemusela být, ale u každé třídy je lepší je (byť prázdné) doplnit s ohledem na přehlednost a snadnější implementace programových algoritmů při případném budoucím rozšíření programu. Poslední metodou je `Initialize`. Protože naše aplikace vlastně jen