ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA ELEKTROTECHNICKÁ

Katedra elektromechaniky a výkonové elektroniky

Bakalářská práce

Platforma aplikací XNA a vzorové řešení

Vedoucí práce: **Ing. Petr Weissar, Ph.D.**

Autor: **Jan Vacata**

2012

Anotace

Tato bakalářská práce se zabývá technologiemi DirectX a XNA od firmy Microsoft. Práce popisuje postupný vývoj těchto technologií a jejich porovnání.Také porovnává technologii XNA s nativními zobrazovacími prostředky operačního systému Windows a porovnává systémové požadavky jednotlivých technologií. Ověřuje nasazení na alternativních platformách. V příloze je uveden ukázkový kód ve třech různých verzích.

Klíčová slova

XNA, DirectX, GameStudio, Grafika, Hry, Hardware, C#, 3D, GDI

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této bakalářské práce.
Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

V Plzni dne 24. 8. 2012 .....

Poděkování

Chtěl bych poděkovat panu Ing. Petru Weissarovi, Ph.D., za cenné rady potřebné k vypracování této bakalářské práce.

Obsah

Úvod

Technologie DirectX od firmy Microsoft je dnes nezbytnou součástí dnešních softwarových aplikací běžících na OS Windows. Její nadstavba, XNA, jakožto nadstavba Managed DirectX, byla vytvořena zejména pro účel jednoduššího vyvíjení počítačových her, což zapříčinilo rozmach aplikací vytvářených pro konzoli Xbox nezávislými vývojáři, studenty a tzv. „hobby“ programátory. Využívá objektově orientovaný jazyk C#, rovněž od stejné firmy.

DirectX byl spočátku velmi složitý a nepřehledný, což se postupem časem stále zlepšilo až na takovou úroveň, že relativně složité věci dnes zvládne i „laik“ za dobu několikanásobně menší než dříve. V dnešní době internetu je také k dispozici mnoho tutoriálů obsahující nejčastější problematiky při vývoji.

Její největší soupeř na tomto poli je API OpenGL, jehož největší síla tkví v multiplatformní podpoře. Stará se o něj nezisková organizace Khronos Group. Tyto dvě API jsou dnes nejhlavnějšími prostředky pro vývoj aplikací.

V praxi při velkorozpočtových a náročných projektech se spíše užívá nízkoúrovňového přístupu k vývoji aplikací, kde se na vývoji podílí organizované skupiny vývojářů.

Na ukázku jsem vytvořil pár příkladů ukazující jaké jsou možnosti tohoto prostředí a jak snadno se s nimi pracuje, v porovnání s psaním v nativním kódu.

1 DirectX

1.1 Co je to DirectX

Dnes se již každý, kdo někdy obsluhoval počítač s operačním systémem Windows, či si zahrál nějakou počítačovou hru na konzoli, setkal s touto technologií. DirectX je soubor pomocných softwarových nástrojů, které tvoří aplikační rozhraní (tzv. API). API, z anglického „application programming interface“, je specifikace, jakýsi „most“ mezi softwarovými komponenty, který umožňuje jednotlivým částem spolupracovat. Příčina velmi hojného využití této technologie je dána jednak vysokým procentem uživatelů OS Windows a jednak tím, že nástroje DirectX jsou velmi kvalitní, jelikož umožňují práci jak s 2D, tak s 3D grafikou, zvuky, vstupními zařízeními jako myš a klávesnice, tak i na úrovni síťové, jako komunikace mezi počítači [1].

Mezi nejznámější API DirectX patří **Direct3D**, o které si podrobněji popíšeme v následující kapitole.

Direct Input je rozhraní umožňující číst výstup z vnějších zařízení, jako je myš, klávesnice, joystick a další. U zařízení, která jsou podporována, je možno využit tzv. „force feedback“, což je odesílání dat zpět do zařízení, čili fungující ne pouze jako vstup z pohledu hlavního zařízení (počítač), ale jako vstup/výstup. Příkladem může být vibrační motorek umístěný v ovladačích na herní konzole. Díky technologii action mapping lze číst data, aniž by software musel vědět, o jaký typ zařízení se jedná [2].

Direct Sound zvládá přehrát, zaznamenat či vytvořit určitý zvukový efekt nebo hudbu. Realizuje komunikaci se zvukovou kartou. Používá se v případech, kdy je potřeba např.: přehrát více zvukových stop za sebou, jelikož má blíže k hardwarové části než DirectMusic [3].

Direct Music pracuje s audiem (převážně s MIDI) jako Direct Sound, ale na vyšší úrovni, čímž může být pomalejší, zato má více funkcí [4].

Direct Play zajišťuje síťovou podporu [5].

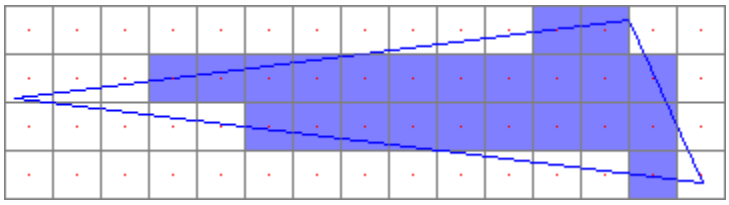
DirectShow je navržený pro provádění různých operací s multimediálními soubory, jako přehrávání [6].

1.1.1 Direct 3D

Toto API se využívá v případech, kdy je potřeba vykreslit grafiku ve třech dimenzích (3D). Využívá hardwarovou akceleraci, což je jedna z technik, jak zvýšit výkon při výpočtu. Zjednodušeně řečeno využívá hardware, nejčastěji na grafické kartě(pokud jej podporuje), který je přimo navržen pro vykonávání instrukcí, zaměřených na tuto oblast.
Pro účely této práce budu referovat DirectX jako práci s nativním kódem [7].
Podporuje pokročilé operace s grafikou, mezi nezákladnější patřící jmenovitě:

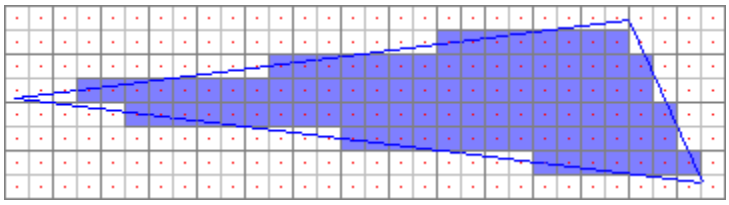
Z-Buffering- Řeší problém viditelnosti objektů. Ke každému vygenerovanému pixelu v prostoru uložíme mimo souřadnic X a Y také hodnotu souřadnice Z – hloubku. Pokud nastane situace, že na stejné X a Y souřadnici musíme vykreslit další pixel, tato funkce porovná hloubku jednotlivých pixelů a zobrazí ten blíže k pozorovateli [8].

Anti-aliasing-Obrazce jsou vykreslovány tak, že každý pixel je vybarven barvou, která se nachází v jeho středu za podmínky, že se tento střed nachází uvnitř zmíněného obrazce. Problé nastává, když rozlišení pixelů není dostatečně veliké, aby dokázalo vykreslit objekt (geometrický útvar) v dostatečné kvalitě. Tento problém se nazývá Jagged Edges, čili zubaté hrany [9].



Obr.1 Jagged Edges

Řešením je zvícenásobit počet pixelů, tzv. Oversampling. Tím se ale samozřejmě zvýší nároky na výpočet.



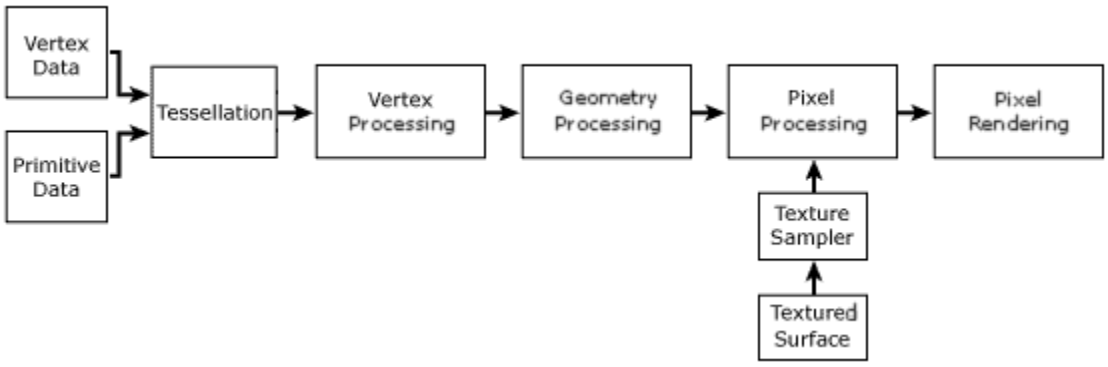
Obr.2 Oversampling

Mipmapping- Každá textura má sadu předvypočtených obrázků nižšího rozlišení, než je textura sama. Příklad: Máme texturu o velikosti 32x32 pixelů, tudíž obsahuje dalších 5 o velikosti 16x16, 8x8, 4x4, 2x2 a 1x1. Využívá se toho tehdy, kdy se na texturu díváme z větší dálky a stačí použít obrázek o nižším rozlišení, nejbližší tomu, které potřebujeme [11].

Alpha blending – Kombinování barev objektu s barvou pozadí k vytvoření efektu průhlednosti [10].

Jelikož vypracované příklady stojí na verzi DirectX 9.0, zde si ukážeme jak vypadá proces vykreslování v této verzi. Nutno podotknout, že ve vyšších verzích se schéma pozměnilo a neustále se vyvíjí.

Direct3D 9 pipeline [12]:



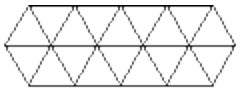
Obr.3 Pipeline [12]

Vertex Data - Vektory modelů, které jsou uložené ve vertex bufferech.
Primitive Data – Data jednoduchých geometrických útvarů, jako jsou trojúhelníky
Tessellation – Neboli rasterizace.Vezme geometrické útvary a vytvoří z nich síť, jejíž spojnice uloží ve vertex bufferu.
Vertex Processing – Na vektory, uložené v bufferu, je aplikována transformace - převedení modelů do prostoru.
Geometry processing – Provede geometrické úpravy, jako zobrazování útvarů jen z určitých stran z důvodu zlepšení výkonu (culling), nebo vykreslování objektů pouze v záběru (clipping).
Textured surface – Zde se zjistí a dodají potřebné souřadnice pro textury pro objekty.

Texture sampler – Filtrování textur podle potřebné úrovně detailu.

Pixel processing – Spojení dat textury a jednotlivých bodů geometrických útvarů. Výsledkem je pixel o dané barvě.

Pixel Rendering – Na jednotlivé pixely se aplikují efekty, např.: alpha(úroveň kombinace barvy pixelu s pozadím), nebo efekt mlhy.



Obr.4 Rasterizace

1.2 Historie, vývoj rozhraní

DirectX 1.0 – Vydán v roce 1995 pod jménem Windows Games SDK jako 32-bitová náhrada za platformu WinG, která měla na starosti grafické procesy na operačním systému Windows 3.1.

DirectX 2.0 – Vydán v červnu 1996. Byl expedován již jako součást Windows 95 OSR2 a Windows NT 4.0. Spolu s ním byl poprvé vydána první verze Direct3D a DirectPlay.

DirectX 3.0 – Vydán v září 1996. V očích veřejnosti začíná být brán vážně, vyvíjí se hry, které k chodu potřebují nainstalovanou verzi DirectX 3.0.

DirectX 4.0 – Čtvrtá verze měla být původně vydána společně s čipy od firmy Cirrus Logic, které měly být umístěny v laptopech. Jelikož se dodávka čipů opozdila, Microsoft rozhodl, že vydá rovnou pátou verzi.

DirectX 5.0- Vydán v červenci 1997 pro windows NT 5.0. Nově se objevila třeba podpora Force Feedback nebo hardwarově podporovaný prostorový 3D zvuk. Verze 5.2 byla vydána i pro Windows 98.

DirectX 6.0- V srpnu 1998 byla vydána další, která podporovala hardwarové prvky, jako stencil buffery, multitexture nebo bump mapping. Pro podporu textor o vyšším rozlišení byla zakoupena licence na technologii kompresi textur od firmy S3 Graphics Ltd. a následně implementována jako DXTC. Opravná verze 6.1a byla vydána pro Windows 98 SE.

DirectX 7.0- V září 1999 vydána verze pro Windows 9x a v únoru tatáž verze pro Windows 2000. V této době se rodili první GPU s podporou hardware Transform & Lighting (TnL), se kterým již tato verze počítala. Další novinky byly Vertex Blending, Cubic enviroment mapping, podpora Visual Basic nebo podpora formátu DirectDraw Surface.

DirectX 8.0- Vydaná v listopadu 2000. Nově podpora programovatelných grafických processorů, tím pádem i podpora Pixel a Vertex Shader processingu. DirectDraw, používaný pro 2D grafiku byl převzat společně s 3D do jednoho API, Direct3D.

DirectX 8.1- Cílem bylo zjednodušit vývoj v Direct3D, který byl v dřívějších verzích obtížný. Nově podporován Pixel Shader 1.4 a Tesselation. Vydán také pro Xbox.

DirectX 9.0- Jedna z nejhlavnějších verzí. Podpora Pixel a Shader verze 2.0 (Dohromady Shader Model 2.0). Další novinkou byl High Level Shader Language, pro usnadnění vývoje vizuálních efektů. Verze 9.0c přinesla shader model 3.0.

Directx 10.0- Vydán společně s Windows Vista. Zpětně nepodporoval OS WindowsXP. Podpora shader model 4.0. Od této verze se začal využívat programovatelný pipeline místo fixního, jako tomu bylo u předešlých verzí, čímž se na určitých úrovních změnilo přistupování k tomuto API.

Direct 11.0 – Společně s Windows 7 byla vydána zatím poslední verze této platformy. Shader model 4.1. Vylepšené ovládání GPU z vícejádrového procesoru.

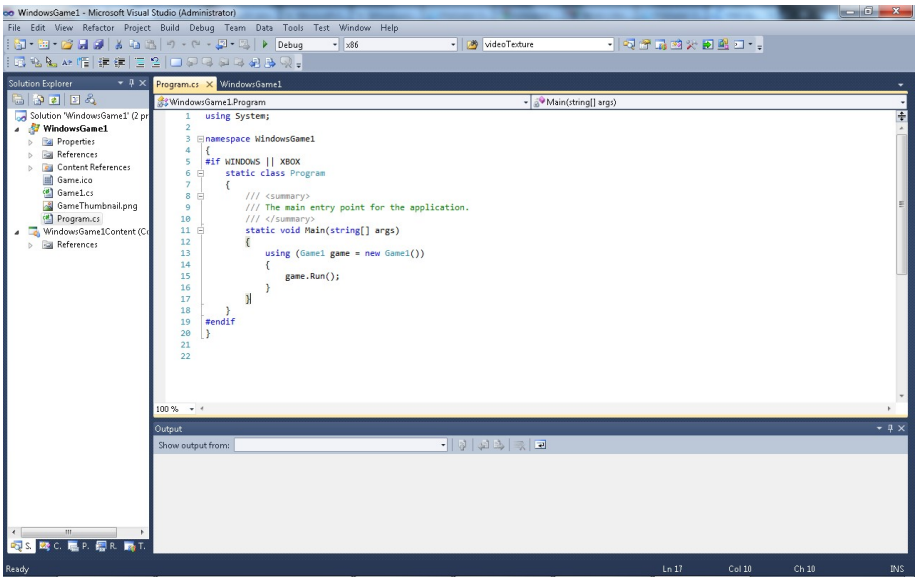
Převzato z [13]

2 XNA

2.1 Co je to XNA?

Vyvíjena firmou Microsoft, XNA je sada nástrojů, která výrazně usnadňuje tvorbu aplikací, převážně videoher. Je umístěna v běhovém prostředí (runtime environment), které zbavuje programátora psaní „zbytečných“ příkazů, které se neustále opakují, jako při vykreslování obrázků apod.. Podporovány jsou jazyky C# a Visual Basic .NET. Technicky může být program napsán v jakémkoliv jazyce, ale do dnešní doby jsou podporovány pouze výše zmíněné dva. První framework XNA byl postaven na frameworku verze 2.0 pro Xbox a pro OS Windows. Byl kladen důraz na to, aby byla snadná konverze z jedné platformy na druhou. XNA plně podporuje vývoj 2D a 3D grafiky. Dále obsahuje XACT, což je zvuková knihovna vyšší úrovně umožňující práci se zvukem. Na OS Windows tato knihovna využívá API DirectSound. Samozřejmostí je podpora ovladače pro konzoli Xbox. Hry vytvořené tímto frameworkem na konzoli Xbox mohou být vydávány pouze členy Microsoft XNA Creator´s Club, jejíž členský příspěvek činí $99 na rok.

Prostředí, ve kterém probíhá vývoj, se nazývá XNA Game Studio. Toto prostředí je provozováno pomocí programu Visual Studio. [14]



Obr.5 Prostředí Visual Studio 2012

2.2 Vývoj platformy

Game Studio 1.0

Neboli Express. První vydání tohoto prostředí bylo určeno pro nezávislé vývojáře (INDIE), studenty a hobby programátory. Na OS windows je vývoj a distribuce her zadarmo, avšak jak bylo napsáno výše, vývojáři pro Xbox musí platit členský poplatek. Obsahuje jakési „startovací balíčky“, dle toho, jaký druh hry se chystáme vyvíjet [14].

Game Studio 2.0

Bylo možné jej použít se všemi verzemi Visual Studia 2005. Bylo vylepšené propojení s konzolí Xbox 360. Nová funkce jednoduché konverze projektu mezi formáty Windows a Xbox 360. Od této verze má každý projekt integrovaný „podprojekt“, ve kterém je veškerý obsah, který projekt obsahuje [14].

Game Studio 3.0

Podpora Xbox LIVE Community Games – Umožňuje distribuovat hry pomocí Xbox LIVE Marketplace. Je kompatibilní se všemi verzemi Visual Studia 2008. Podpora verze 3.0 jazyka C#. Vylepšená podpora více platformových aplikací [14].

Game Studio 3.1

Xbox LIVE Party – Umožněna komunikace mezi hráči. Nově je umožněna manipulace s videi [14].

Game Studio 4.0

Momentálně nejnovější verze, operující ve všech verzích Visual Studia 2010. Podpora pro vývoj aplikací na Windows Phone 7. Nové framework profily Reach a HiDef. V této verzi je nyní umožněné rozsáhlé nastavení tzv. Effect (Viz Kapitola XX). Nově možná práce s mikrofonem. Vylepšená práce s obrázky. [14]

2.3 Prostředí Visual Studio 2010

K dispozici je nám pár užitečných nástrojů.

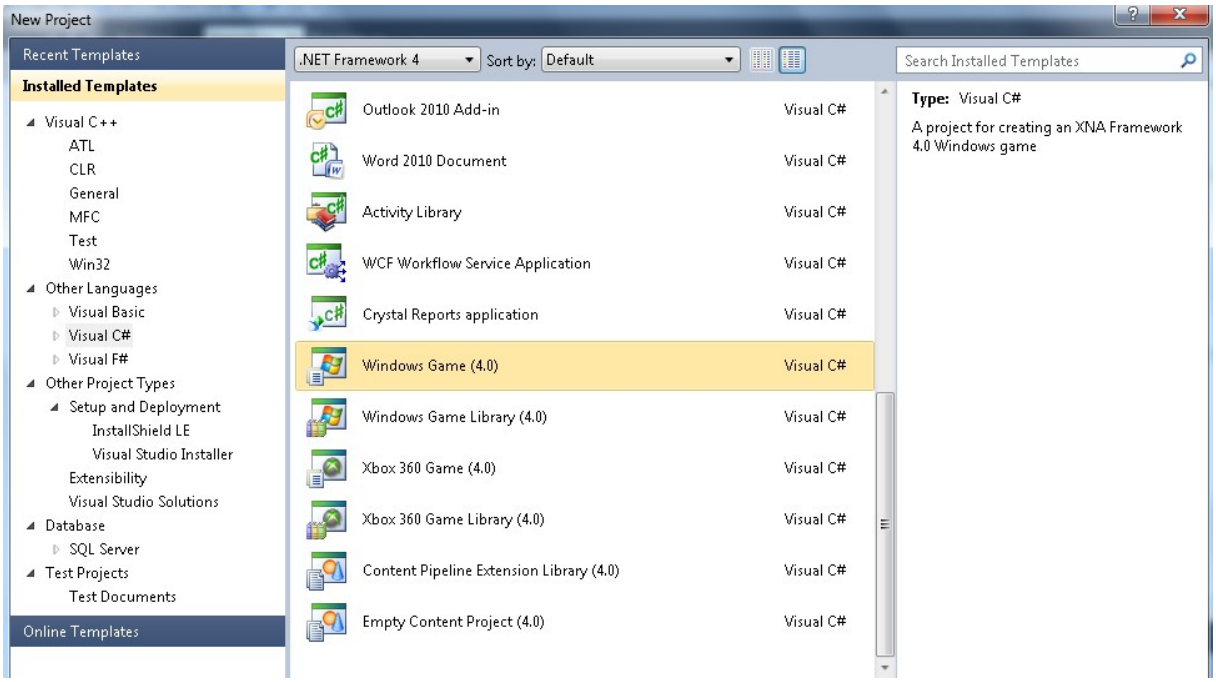
Obr.6 Nástroje Visual Studia

Solution explorer, ve kterém vidíme všechny prvky našeho projektu. Celý strom má dvě hlavní části- v jedné se nacházejí veškeré zdrojové soubory včetně nastevení projektu. V druhé je veškerý obsah, jako textury, zvuky atd.. Pomocí Class View je možno zobrazit symboly, které jsme použili v našem projektu. Property Manager umí uchovat nastavení a sdílet jej mezi více projekty. Resource View je správa resource souborů s příponou . rc .
Team Explorer je klientský nástroj pro TFS.

2.4 Vývoj

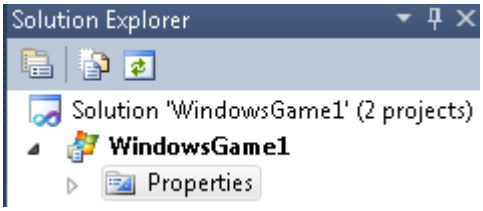
2.4.1 Proces založení a nastavení projektu až po jednoduchý program

Zde se podíváme na jednoduchou aplikaci v XNA 4.0., která nám vypíše na obrazovku náš libovolný text. Na té si ukážeme stěžejní součásti programu. Začneme založením projektu *File->New->Project.* Zde si vybereme, v jaké verzi frameworku chceme pracovat. Jediné důvody, které vidím při zvolení staršího frameworku jsou buď, že verze je nová a nekompatibilní se staršími systémy, jako v případě nejnovějšího frameworku verze 4.5, kde na rozdíl od 4.0 odpadla podpora OS Windows XP se SP3, nebo při chybějící dokumentaci k nové verzi, která je ale v dnešní době již k dispozici na internetu.



Obr.7 Možnosti projektu

Po založení je projekt nutno patřičně nastavit. K tomu nám slouží záložka Properties v Solution Exploreru.



Obr. 8 Properties

Zde je možno nastavit věci jako jméno projektu (Pozor - při změně jména ale nedochází ke změně jména složky v dokumentech), nebo ikony. Dále nastavíme, zda se jedná o aplikaci, která může být konzolová nebo běžící v okně a nebo třeba o knihovnu. Důležitou věcí je zde zvolit správný herní profil. Dále můžeme zvolit chování při debuggingu, jako spouštění externích programů, nastavení složek a argumentů při spouštění. Při publikaci si lze vytvořit k programu vlastní certifikát a dále nastavit, jak bude probíhat instalace a následné updaty programu v případě, že k distribuci využijeme vestavěnou technologii ClickOnce.
Po nastavení projektu se podíváme na jednotlivé součásti samotného programu.
Následující body převzaty z [15].

a)Direktivy

Pomocí nichž kompilátor ví, že používáme komponenty, které byly již dříve vytvořeny.
Př.:

```
1  using System;
```

Obr.9 Direktiva

b)Hlavní třída

Zde definujeme hlavní třídu naší aplikace:

```
20     public class Game1 : Microsoft.Xna.Framework.Game
21     {
22         SpriteBatch spriteBatch;
23         GraphicsDeviceManager graphics;
24
```

Obr.10 Hlavní třída

Hned jsme definovali dvě proměnné typu SpriteBatch a GraphicsDeviceManager. První zmíněná má na starost vykreslování 2D, ta druhá manipuluje s nastavením a managementem grafického zařízení.

c)Konstruktor

Zde probíhá hlavní nastavení aplikace.

```
97          public Aplikace()
98          {
99              graphics = new GraphicsDeviceManager(this);
100             Content.RootDirectory = "Content";
101         }
```

Obr.11 Konstruktor

Zde jsme připravili novou třídu grafického zařízení. Také jsme nastavili, kde se bude nacházet náš obsah.

d)Initialize

Zde můžeme inicializovat určitý obsah přímo ze začátku startu aplikace. Většinou se sem dá obsah který je potřeba nahrát jen jednou, jako velikost nějakých bufferů atd..

```
106         protected override void Initialize()
107         {
108             base.Initialize();
109         }
```

Obr.12 Inicializace

e)Load a Unload Content

```
278         protected override void LoadContent()
279         {
280             font = Content.Load<SpriteFont>("MujFont");
281             video = Content.Load<Video>("video");
282         }
```

Obr.13 Content

Podobné metodě Initialize, zde se nahrává a odebírá obsah v průběhu aplikace. Sem spíše patří multimediální obsah, jako obrázky, videa, fonty apod..

f)Update

```
338         protected override void Update(GameTime gameTime)
339         {
340             if (player.State == MediaState.Stopped)
341             {
342                 player.IsLooped = true;
343
344             }
345         }
```

Obr.14 Update

Implicitně je tato metoda volána 60x za každý snímek. Tato hodnota se dá změnit. Stará se o průběh aplikace, jestli se něco změnilo. Např.: Jestli bylo zmáčknuto tlačítko, nebo zde napsaný příkaz, který znamená, že se zrovna přehravající video bude pouštět ve smyčce.

g)Draw

```
385         protected override void Draw(GameTime gameTime)
386         {
387             spriteBatch.Begin(SpriteSortMode.Immediate, BlendState.AlphaBlend);
388             spriteBatch.DrawString(MujFont, "Zde piseme text", new Vector2(200,200), Color.Black);
389             spriteBatch.End();
390             base.Draw(gameTime);
391         }
392     }
```

Obr.15 Draw

Zde probíhá veškeré vykreslování, jako pozadí, videa atd.. Počet zavolání téhle metody za jednu sekundu závisí na výkonu našeho počítače, zejména grafické karty.

2.4.2 Grafika

Od verze 4.0 jsou nabízeny dva druhy tzv.“herních profilů“. Reach a HiDef. První je napsán tak, aby byl kompatibilní hlavně s Windows, Xbox 360 a Windows Phone, na rozdíl od HiDef, kde není limitovaná API, ale je kompatibilní pouze s Xbox 360 a Windows, kde je podmínka grafické karty s podporou minimálně DirectX 10. Pro srovnání příklad:Reach podporuje shader model 2.0 a maximální velkost textur 2048, zatímco HiDef má podporu shaderu 3.0 a velikosti textur až 4096. [23]

2.4.2.1 2D

Do obsahu (content) si nejdříve nahrajeme požadované obrázky. Podporující formáty jsou *.bmp; *.jpg; *.png; *.tga a *.dds. Poté je nastavíme jako typ proměnné Texture2D: *Texture 2D Pozadi* a v metodě Load Content je inicializujeme funkcí Content.Load následujícím příkazem: *Pozadi = Content.Load<Texture2D> ("Nebe");* Název souboru je *Nebe* a odteď pracujeme s obrázkem jako s *Pozadi*.

Pomocí Game Studia se dají u obrázků provádět různá nastavení. Mohou být transparentní, kde se v nastavení zvolí referenční barva která nebude viditelná. Je zde umožněno generovat mipmapy (viz 1.1.1) a s tím trochu související možnost přepočítat velikost obrázku tak, aby její rozměry měly velikosti násobků 2, jako 16x16, nebo 256x256. Dříve existovalo omezení,

že každý rozměr textury musí mít tyto velikosti. Dnešní grafické karty a grafické API už nejsou v tomto ohledu tak striktní, avšak tím, že jsou rozměry v násobku dvou se urychlí výpočet v pipelinu, např.:výpočet mipmapů, kde se nemusí zaokrouhlovat, jelikože rozměr je vždy dělitelný 2.
Obrázky vykreslujeme pomocí tzv. Dávkovače (spriteBatch).

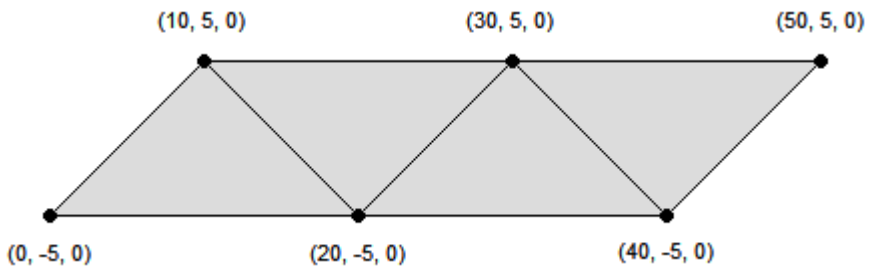
```
463             spriteBatch.Begin(SpriteSortMode.BackToFront, BlendState.AlphaBlend);
464             spriteBatch.Draw(sipka, new Vector2(410, 340), Color.White);
465             spriteBatch.End();
```

Obr.16 Dávkovač

Dávkovač musíme aktivovat a po vykreslení všeho potřebného zase deaktivovat. Jak vidíme, můžeme při začátku nastavit hloubkové pořadí textur, nebo třeba způsob převádění vektorů do pixelů při kreslení primitivních geometrických útvarů. Proč ale používáme dávkovač a nevykreslujeme obrázky bez něj? Kdyby jsme tak učinili, tak při vykreslování mnoha různých obrázků posíláme informace do GPU jednotlivě pro každý obrázek zvlášť, což by bylo pomalé a zbytečné. Mezitím funkcí *Begin* začneme informace o každém obrázku ukládat v bufferu, který pošle veškerá data do GPU naráz až po zavolání funkce *End*. Animace vykreslujeme podobně. Vytvoříme si obrázek, ve kterém si rozkreslíme jednotlivé snímky. Ve funkci *Draw* by jsme aktivovali další parametr a to vykreslování pouze v daném výseku obrázku.Poté si napíšeme jednoduchý algoritmus, který bude s časem přepínat tento výsek mezi jednotlivými snímky a vykreslovali pouze tento výsek pomocí jedné funkce *Draw*.

2.4.2.2 3D

Všechny 3D objekty jsou složeny z primitivních útvarů, skládajících se do sebe a formující komplexnější modely.



Obr.17 Primitivní obrazce

XNA využívá ke kreslení tzv. *effecty*. *Effect* je kód, který určuje, jak má grafická karta tyto trojúhelníky vykreslit.V programu si můžeme vytvořit přednastavený *effect* BasicEffect, který obsahuje základní nastavení, nebo si napsat vlastní, který má podobu souboru s příponou *.fx a který nahrajeme do naší Content složky [24]. *Effecty* jsou psán v HLSL. Každý effect se skládá ze tří základních částí [25]:

a) Definice parametrů – Zde se nastaví proměnné před vykreslením, které jsou později použity ve výpočtu, jako projekční matice, osvětlení, nebo definice textur.
b) Definice technik – Určuje, jakým způsobem vykreslíme objekt. Například jednou technikou vykreslíme objekty bez textur, nebo v určité barvě apod..
c) Funkce – Pro vykonání nejrůznějších početních operací. Mohou pracovat s shader modelem. Jsou napsány v High Level Shader Language.

Dále je zapotřebí definovat si matice.

```
52          private void SetUpCamera()
53          {
54              viewMatrix = Matrix.CreateLookAt(new Vector3(0, 0, 50), new Vector3(0, 0, 0), new Vector3(0, 1, 0));
55              projectionMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
56                                                                     device.Viewport.AspectRatio,
57                                                                     1.0f, 300.0f);
58          }
```

Obr.18 Matice

ViewMatrix – Tato matice v sobě ukládá data, jak se bude kamera dívat na naší scénu. První parametr znamená vzdálenost od bodu na který se díváme. Tento bod definujeme parametrem druhým. Třetím natáčíme kameru podél své osy. *ProjectionMatrix*- Tato matice v sobě ukládá hodnoty jako úhel pohledu na náš bod a poměr stran (např.: 4:3, 16:9). Poslední dva argumenty značí hranice, před a za kterými se již objekty nevykreslují. *WorldMatrix* – Tato matice reprezentuje stupeň velikosti (scale), rotaci a pozici objektu v naší scéně. Z toho vyplývá, že přes tuto matici provádíme nejrůznější transformace, jako např.: translaci a rotaci. [26].

Když máme tyto matice definované, předáme je našemu *Effect* kódu .Další věcí, která je ale již nastavena implicitně je tzv. *Culling*. Jedná se o vykreslování primitivních geometrických útvarů jen z jedné strany a to té, na které jsou seřazeny vektory vrcholů ve směru hodinových ručiček. Je to veliká úspora výkonu, např.: při vykreslení krychle zobrazíme jen vnější povrch. Kdybychom umístili kameru uvnitř krychle, což může být nežádoucí, a dívali se směrem ven, neuvidíme nic. Ke zlepšení výkonu se také používají tzv. indexy. Jejich princip je ten, že u spojených trojúhelníků definujeme společné vektory jako jeden a ne pro každý trojúhelník zvlášť. Tímto se neplýtvá datovým pásmem k GPU.

2.4.3 Audio

Použití audia ve verzi 4.0 závisí na profilu, který využíváme. Ti, co používají HiDef, mohou využít knihovny XACT. Jelikož můj systém podporuje pouze režim Reach, který ale knihovnu XACT nepodporuje, budeme se zabývat druhou možností a to zjednodušenné audio API vyvinuté pro tyto případy. Zde se se zvuky nakládá jako s ostatním obsahem, např. s texturami. Jednotlivé zvuky nahrajeme do Content, nastavíme jako typ *SoundEffect* a inicializujeme funkcí *Content.Load*. Pak již jen velmi jednoduše zavoláme funkci pro přehrátí:

```
292             Zvuk.Play();
```

Obr.19 Přehrání zvuku

2.4.4 Vstupy

Aplikace je nutno nějak řídit a k tomu slouží vstupy. Čtení vstupů v XNA není nijak zvláště obtížné. K zaznamenávání využijeme metodu Update.

```
586             KeyboardState keyState;
587
588             if (keyState.IsKeyDown(Keys.A))
589             {
590                 player.Pause();
591             }
```

Obr.20 Pauza videa

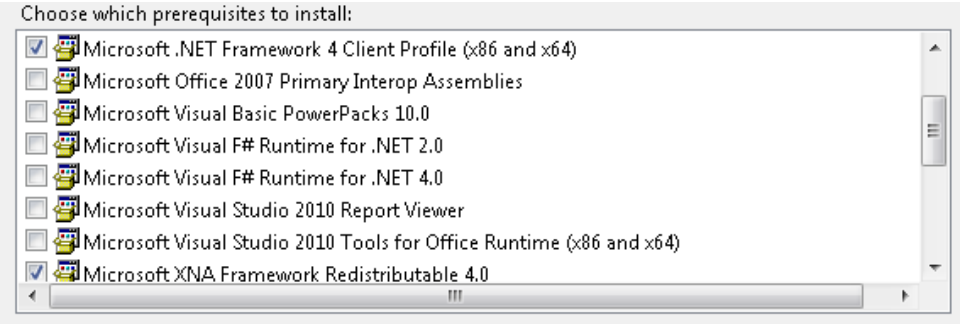
Standartně je na OS Windows podporována klávesnice, myš, gamepad a od verze 4.0 i mikrofon. Na windows phone jsou navíc podporovány i dotykový panel nebo accelerometr.

2.4.5 Distribuce

Distribuce vytvořených aplikací provází menší komplikace. V závislosti na verzi frameworku ve kterém pracujeme potřebujeme, aby na cílovém počítači byly již nainstalované součásti. Jelikož jsem vytvářel ve frameworku verze 4.0, tyto podmínky jsou [17]:

a) Nainstalovaný .NET Framework 4 Client Profile – v případě, kdybych použil vymoženosti jako síťové propojení, musela by se nainstalovat plná verze.
b) XNA Framework Redistributable 4.0

Visual studio 2010 nabízí řešení v podobě ClickOnce. Tato technologie zesnadňuje distribuci programů. Před instalací zkontroluje zda máme nainstalované potřebné komponenty, které jsme zvolili v nastavení před kompilací. [16]



Obr.21 Výsek z přehledu možných instalačních balíků

Případné chybějící komponenty doinstaluje buď stažením z internetu, nebo můžeme instalační soubory distribuovat společně s programem.

Samozřejmě toto není jedinný způsob distribuce, existují také alternativní neoficiální metody.

3 Porovnání DirectX a XNA

3.1 Hardwarová náročnost

a) **CPU -** Má na starosti výpočet logiky, umělé inteligence, detekci kolizí, práci se vstupem atd.. Největší zátěží je asi fyzika objektů. XNA používá Common Language Runtime (CLR), kde je zdrojový kód kompilován do tzv. mezikódu zvaného Common Intermediate Language (CIL) [27]. Poté je spuštěna Just-In-Time kompilace, která vytvoří nativní kód na daném systému. Tato kompilace je kombinací statické, kde se kód takto přeloží jen jednou a interpretované kompilace, kde se kód překládá při každé instrukci [30]. Celý tento proces má na starosti právě CPU. Jelikož při programování přímo s DirectX pracujeme s nativním kódem a není potřeba nic překládat, lze usoudit, že v tomto směru má DirectX na rozdíl od XNA navrch.

b) **GPU –** Má na starosti veškeré vykreslování a práci s grafikou. Tudíž do něj přicházejí již „hotové“ instrukce a je jedno, jestli byly poslány z XNA nebo z DirectX programu. Takže z hlediska náročnosti je rozdíl nepatrný.

Pro malé a středně velké aplikace postačí knihovny XNA. Pokud by jsme prováděli velmi náročné grafické aplikace, tak psaní přímo v nativním kódu bude vždy lepší volbou.

3.2 Náročnost vývoje

XNA bylo přímo vytvořeno za cílem snažšího vývoje. DirectX pracuje na nižší úrovni, tudíž je v něm nutno provádět více práce, které nás XNA zbavuje. Vezměme si např.: Vytváření okna. V XNA se tímto pomalu vůbec nemusíme zabývat zatímco v DirectX musíme projít registrací okna, poté jeho vytvořením, kde v jedné funkci nastavíme všechny jeho parametry a poté jeho zobrazením. Z toho vyplývá, že k psaní v DirectX je potřeba základních znalostí ve Win32 API.

3.3 Rozdíl ve vykreslování grafiky

Jak již bylo zmíněno, XNA je oproti psaní nativního kódu v DirectX velmi zjednodušeno.
V XNA k vykreslení všeho 3D musíme použít effecty. Technicky vzato se vykresluje stejnými principy, jelikož XNA je jen „obalený“ DirectX.

3.4 Ukázkový kód

Příloha č.2 obsahuje ukázkový kód pro vytvoření trojúhelníku v DirectX 9.0 pomocí jazyka C++. Lze srovnat s přílohou č.1 a č.3, kde je naprogramován stejný příklad v XNA a pomocí GDI.

Když se na podíváme příklad, vidíme že ve funkci Main je uzavřená, nekonečná smyčka. Tato smyčka obsahuje funkce pro zpracování tzv. Zpráv. Ať už klikneme myší, zavřeme okno, nebo ho roztáhneme, Windows předá našemu programu zprávu, ten ve smyčce while zavolá funkci pro zpracování a rozpoznání Translate() a poté Dispatch(), která předá zprávu funkci WindowProc, kde můžeme dále psát, co se stane, pokud něco takového nastane.

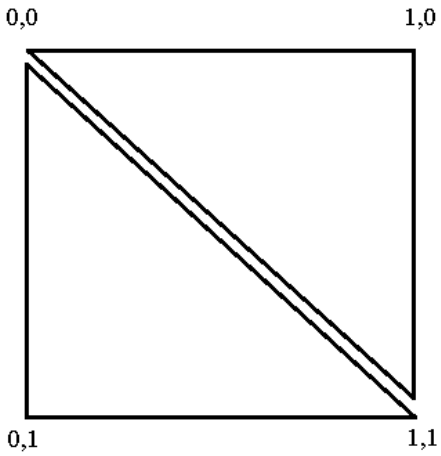
4 Popis vypracovaných příkladů

Platformu XNA jsem demonstroval na pár jednoduchých příkladech. Program se skládá z menu, ve kterém si vybereme jednotlivé příklady. Volba je realizována výběrem jedné ze tří stěn kostky. Během výběru se mění i nasvícení. Každá stěna kostky je sestavena ze dvou trojúhelníků.

```
1505        private void KostkaMenu()
1506        {
1507            KrychlePredek = new VertexPositionNormalTexture[6];
1508
1509            KrychlePredek[0].Position = new Vector3(-5f, 5f, 5f);
1510            KrychlePredek[0].TextureCoordinate = new Vector2(0, 0);
1511            KrychlePredek[0].Normal = new Vector3(0, 0, 1);
```

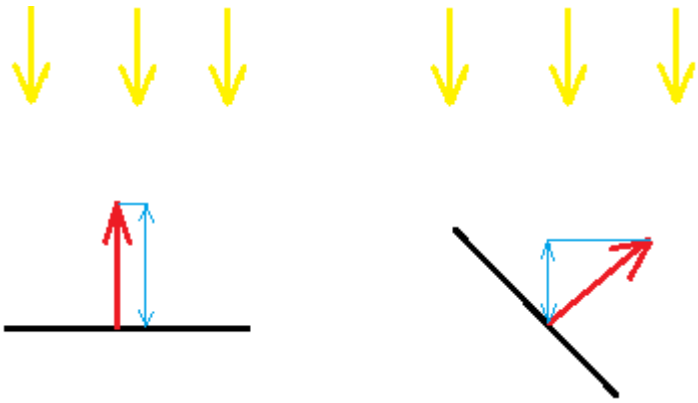
Obr.22 Vykreslení trojúhelníku

Všechny trojúhelníky se uloží do vytvořeného pole vektorů. Každý vektor typu VertexPositionNormalTexture má tři parametry. Pozice jednotlivých bodů, X a Y souřadnice textury a normála. Souřadnice textury znamenají, kde bude textura začínat,končit a jak bude orientovaná.



Obr.23 Souřadnice textur

Normála plochy je důležitá vlastnost při nasvícení.Na dalším obrázku máme dva případy.



Obr.24 Nasvícení

V prvním je normála ve směru (0, 1). Vektor záření je zvolen (0, -1). Jelikož svírají nulový úhel, je zde maximální intenzita. V druhém případě je normála ve směru (1, 1). Zde je již vytvořený úhel mezi normálou a vektorovými čarami záření, tudíž je intenzita snížena. Aby záření na plochu dopadalo, tak úhel musí být menší než 90°.

Při vykreslování vektorů se musí dávat pozor na již dříve zmíněný culling, tedy, pokud ho z nějakých důvodů nevypneme. Při vytváření tohoto příkladu nastal menší problém. Jelikož používám profil Reach, je zde menší kolize při vzorkování textur, které nemají rozměr násobku dvou. Jelikož výsledná textura nemusí mít vždy tento rozměr, bylo nutno nastavit vzorkování textury na mód Clamp. Vzorkování textur určuje způsob, jak se mapují textury mezi pozice 0 a 1.

```
332         SamplerState _clampTextureAddressMode = new SamplerState
333         {
334             AddressU = TextureAddressMode.Clamp,
335             AddressV = TextureAddressMode.Clamp
336         };
```

Obr.25 Přepnutí na Clamp mód

Jako první a druhý příklad jsem vytvořil tvorbu grafů. Na výběr je možnost tvorba grafů pomocí SpriteBatch a nebo pomocí primitivních obrazců. První příklad funguje tak, že zadáme hodnoty bodů X a Y a program nám vykreslí křivku. Hodnoty zadáváme postupně, tím plníme pole dvourozměrných vektorů. Křivek, které na sebe navazují a tvoří graf, můžeme vykreslit libovolné množství. Pokud se dostaneme přes maximální hodnotu na některé z os, graf se přepočítá na větší velikost. Maximální hodnoty jsem zvolil 9999. Hodnoty jsou zadávány tak, že kurzorem myši najedeme na pole do kterého chceme psát a zadáme číselnou hodnotu. Poté klikneme na tlačítko Vykreslit. Pokud budeme chtít začít

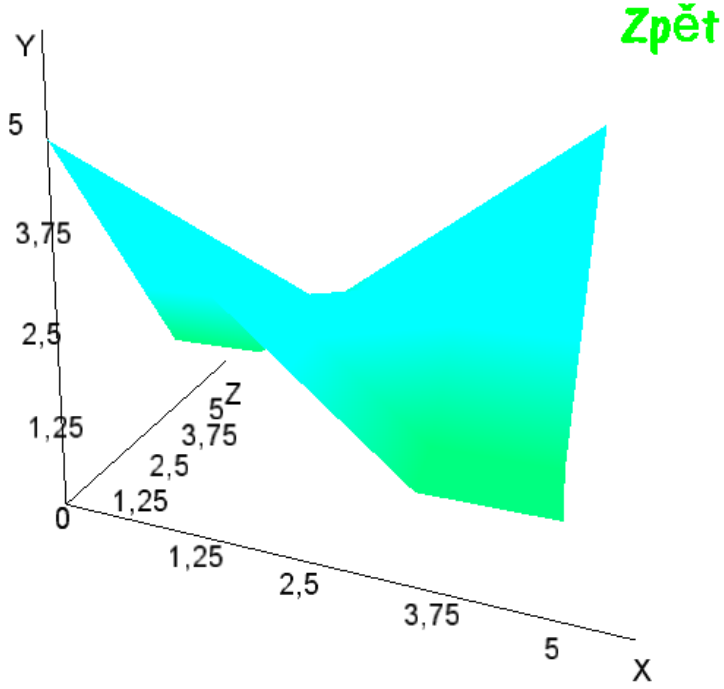
odznovu, je zde tlačítko pro reset. XNA nemá funkci pro kresbu křivek mezi vektory, musel jsem si tedy vytvořit vlastní funkci:

```
410         void DrawLine(SpriteBatch sprBatch, Texture2D spr, Vector2 a, Vector2 b, Color col)
411         {
412             Vector2 pocatek = new Vector2(0.5f, 0.0f);
413             Vector2 rozdil = b - a;
414             float uhel;
415             Vector2 stupnovani = new Vector2(1.0f, rozdil.Length() / spr.Height);
416             uhel = (float)(Math.Atan2(rozdil.Y, rozdil.X)) - MathHelper.PiOver2;
417             sprBatch.Draw(spr, a, null, col, uhel, pocatek, stupnovani, SpriteEffects.None, 1.0f);
418         }
```

Obr.26 Vlastní funkce DrawLine

Tato funkce má pak 5 parametrů. Umístíme ji mezi funkce Begin a End stejně jako Draw třídy SpriteBatch.

Druhý příklad nabízí širší možnosti díky kreslení pomocí primitivních obrazců. Je to především možnost kreslení jak ve 2D, tak ve 3D. Zde je na výběr ze 4 možných druhů grafů. Ve všech případech lze hodnoty načíst z textového souboru. První případ nabízí rozdělení osy x lineárně mezi zvolenými hranicemi. Druhý je X-Y bodový graf, jako v prvním přikladu. Třetí je X-Y-Z bodový graf, kde je možné nastavit kameru do os X-Y, X-Z a neb Y-Z. Čtvrtý je vytvoření plochy z načtené symetrické matice. Hodnoty určují výšku plochy v daném bodě. Mezi jednotlivými body jsou vykreslovány plošky, kde je každá sestavena ze dvou trojúhelníků, podobně jako u kostky v menu. S výškou se mění světlost barvy.



Obr.27 Plošný graf

Třetí příklad je přehravač audia a videa. Je zde možné vytvořit playlist, či přehrát video v různých rozlišeních. Video je možno pauznout a poté znovu pokračovat. Promítání funguje tak, že z podmínky

```
395             if (player.State != MediaState.Stopped)
396             {
397                 videoTexture = player.GetTexture();
398 
399             }
```

Obr.28 Textura z videa

získáme v každý okamžik, kdy video přehrává, jeho momentální frame a uložíme ho do proměnné videoTexture, kterou pak promítneme na vytvořeném obdélníku.

Čtvrtý příklad je kulička, kterou můžeme manipulovat v ose X a Y. Zde jsem demonstroval výběr předmětů v prostoru pomocí myši, detekci kolize a nebo nahrávání a vykreslování modelů. Model jsem si sám vytvořil ve freeware programu Blender. Vytvořil jsem zde dva totožné modely, lišící se pouze barvou, které se prohazují v závislosti na tom, zda na ně bylo ukázáno myší.

5 Porovnání zobrazování s nativními aplikacemi OS Windows

5.1 Nativní aplikace OS Windows

a) Přehrávání audia a videa

XNA používá k přehrávání audia knihovnu MediaPlayer. Obsahuje funkce jako opakování skladby, náhodné přehrání nebo ovládání hlasitosti. Také je možné vyhledat skladby v Knihovně médií na daném systému pomocí knihovny MediaLibrary. Zde je ale podmínka, že hledané skladby byly již dříve nalezeny pomocí aplikace Windows Media Player, což jest nativní aplikace OS Windows. Také je možné přehrát skladbu umístěnou na internetu. K přehrání videa slouží knihovna VideoPlayer. Podporován je pouze formát .wmv a navíc má ještě několik omezení. Musí obsahovat zvukovou stopu ve formátu .wma a konstatní přenosovou rychlost, která má také omezení:

Úroveň	Maximální bit rate	Odpovídající rozlišení
Nízká	2 Mbps	320 x 240, 24Hz
Střední	10 Mbps	720 x 480, 30Hz
		720 x 576, 25Hz
Vysoká	20 Mbps	1280 x 720, 30Hz

Obr.29 Možné šířky pásem

Veškeré aplikace využívající tyto knihovny musí mít v systému nainstalovaný Windows Media Player, jelikož jsou z něho odvozeny. [18] Windows Media Player podporuje širokou škálu audio a video formátů, jako wmv, avi, mpeg atd.. Pokud daný formát není podporován, přehrávač se ho stejně pokusí přehrát. V XNA je přehrávání videa určeno spíše k pouštění video záběrů jako scén do počítačových her.

b) Vytváření a zobrazování obrázků

Nativní Windows aplikací ve verzi 7 je Windows Photo Viewer. Podporuje formáty bmp, jpeg, jpeg XR, png, ico a tiff [19]. V porovnáním s XNA, které podporuje bmp, jpg, tga a png, je lépe vybaven. Visual Studio nám nabízí také tvorbu bitmapů, kde nabízí funkce stejné jako v aplikaci Malování.

c) Práce se zvukem

Od verze 4.0 je možno zaznamenávat zvuk pomocí třídy Microphone. Obsahuje funkce jako Start, Stop, nastavení vzorkovací frekvence, velikost nahrávajícího bufferu, nebo umí vypočíst potřebnou velikost v bytech pro určitou nahrávající délku. Windows obsahují program Sound Recorder, který má od verze Vista méně funkcí než v dřívějších verzích, např.: již v něm nelze přehrát nahraný zvuk a musíme použít přehrávač [20].

Jak vidíme, multimediální funkce v XNA splňují nutné požadavky pro vývoj her a proto stačí jejich, v porovnáním s nativními aplikacemi Windows, omezené funkce.

5.2 GDI

XNA samozřejmně není jediný způsob pro práci s grafikou. V OS Windows se používá API GDI, jehož účel je starat se o veškeré grafické objekty a práci s nimi. Dá se použít v jednoduchých aplikacích, které nejsou příliš náročné na grafiku. Používá se skoro ve všech Windows programech, jako Malování, Obrázky atd..

Když budeme porovnávat GDI s DirectX z hlediska výkonu, bude na tom DirectX lépe. Je to z důvodů jiných přístupů k vykreslování. DirectX běží většinu jeho času v Kernelu, což je most mezi aplikacemi a hardwarem. Také se snaží tento hardware co nejvíce využít, na rozdíl od GDI, které se snaží být hardwarově nezávislé a využívá pro vykreslování převážně CPU. DirectX převádí grafické požadavky do primitivních obrazců a výsledek je počítán v GPU.

Z tohoto důvodu není třeba vhodné používat GDI na přehrávání videí apod.

Nyní se podíváme, jaký je rozdíl při programování v XNA a v GDI v jazyce C#. Ukážeme si to na příkladu, kdy budeme chtít vykreslit jednoduchou 2D čáru. Knihovny XNA neobsahují žádnou funkci, která nám jednoduše vykreslí přímku z bodu A do bodu B. Tudíž máme na výběr dva způsoby. Buď si funkci napíšeme, nebo přímku vykreslíme pomocí effectu v 3D prostoru. V GDI existuje funkce Pen

```
32          Pen Pero = new Pen(Color.Black, 4);
```

,která obsahuje barvu a šířku čáry. Následně jí vykreslíme:

```
47              e.Graphics.DrawLine(Pero, X1, Y1, X2, Y2);
```

, kde X a Y jsou dříve specifikované body typu int. Z tohoto hlediska je kreslení jednodušší, avšak výpočtově náročnější, než kdybychom si napsali vlastní funkci v XNA. Dále si musíme uvědomit princip tohoto vykreslování. V XNA je kreslení psáno v metodě Draw, která se volá

tak často, kolikrát to zvládne grafická karta. Z toho vyplývá, že je funkce pro vykreslování přímky neustále volána dokola. Tudíž kdybychom chtěli přímku smazat, stačí napsat podobný kód do metody Update:

```
66          bool priznak = false;
67          MouseState stavMysi;
68          protected override void Update(GameTime gameTime)
69          {
70              MouseState stavMysi;
71              stavMysi = Mouse.GetState();
72              if (stavMysi.LeftButton == ButtonState.Pressed)
73                  priznak = true;
74              base.Update(gameTime);
75          }
```

Obr.30 Nastavení příznaku

A pak v metodě Draw kreslit krivku pod podmínkou *If(priznak == false)*. Pokud by jsme pak klepli levým tlačítkem myši, křivka se již přestane vykreslovat. V GDI je to složitější. Tam zavoláme funkci pro vykreslení pouze jednou a přímka již zůstane vykreslena napořád. Tam pak musíme např.: uložit bitmap stavu před vykreslením a až budeme chtít křivku smazat, tento bitmap vykreslit.

V ukázkovém kódu není obsažen kód hlavního programu, který jen volá aplikaci Form1 pomocí příkazu `Application.Run(new Form1());`.

6 Alternativní platformy

6.1 Linux

Jelikož XNA závisí na Direct3D, a veškerých knihovnách, které jsou podporovány Microsoftem. Nejreálnějším způsobem, jak spustit aplikaci na tomto systému je použití nějakého Windows emulátoru, např.: WINE. Samozřejmě by to mělo dopad na systémové požadavky. Jiným řešením by bylo zvolit alternativní cestu vývoje, jako např.: OpenGL, který je multiplatformní apod..

Aplikaci jsem bohužel nedokázal spustit pod Wine-Mono, které nabízí portování na Linux.

6.2 Xbox 360

Jelikož jedním z důvodů proč Game Studio vzniklo byla podpora vývoje her pro konzoli Xbox, především pro podporu nezávislých vývojářů, jsou aplikace vyvinuté pomocí XNA plně podporovány. Xbox 360 podporuje verzi DirectX 9.0c. Lze pro něj vyvíjet i v profilu HiDef, jelikož grafický procesor v konzoli podporuje všechny nutné požadavky pro běh tohoto profilu [21].

Tuto konzoli nevlastním a neměl jsem tedy možnost zde otestovat.

6.3 Windows Phone

Mobilní operační systém, následovník systému Windows Mobile. Tyto telefony mají následující minimální požadavky:

-Kapacitní, 4 dotyky podporující display, s minimálním rozlišením 480 x 800

-Procesor ARM v7

-GPU podporující DirectX 9

-RAM o velikosti 256 MB a alespoň 4GB Flash paměť

-Accelerometr, senzory na dálku a okolní světlo, GPS

-FM tuner

-6 Hardwarových tlačítek

Jsou zde 4 druhy možných vstupů- Hardwarová tlačítka, dotykový displej, accelerometr a ostatní (i přídavná) zařízení, jako fotoaparát, senzory atd.. Dotykový displej podporuje tzv. gesta, která můžeme aktivovat ve třídě TouchPanel. Jde o různé pohyby, jako dvojdotek, posun nebo roztáhnutí pomocí dvou prstů. Accelerometr je snímač akcelerace daného zařízení

ve třech osách, generující hodnoty v rozmezí -1 až 1. Aplikace je nutno programovat v profilu Reach. Není možno zde psát vlastní shadery v HLSL, jsou zde vestavěné efekty [22].
K otestování jsem využil Windows Phone Emulator, který je součástí Windows Phone SDK. Bylo potřeba založit nový projekt typu Windows Phone Game 4.0. Program byl emulován ve verzi Windows Phone 7.1. Zde jsem se přesvědčil, že konverze je opravdu relativně snadná. Bylo potřeba program trochu přizpůsobit, aby zde fungoval. Za prvé byl problém s vlastním effect souborem, jelikož Windows Phone nepodporuje vlastní shadery. Bylo tedy nutné vytvořit effect soubor klasicky:

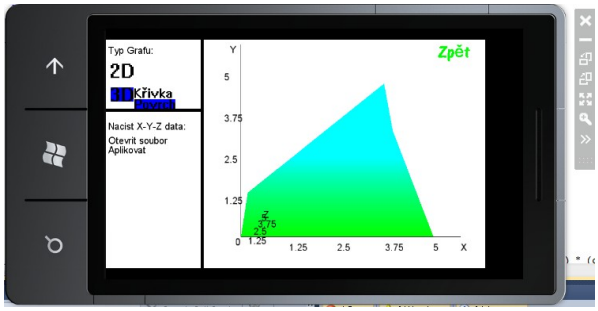
```
631             BasicEffect effect = new BasicEffect(device);
```

a přiřadit matice:

```
634             effect.View = viewMatrix;
635             effect.Projection = projectionMatrix;
636             effect.World = worldMatrix;
```

Obr.31 Přiřazení matic

Další problém byl ten, že program byl napsán pro myš, která má dvě tlačítka - Windows Phone reagoval na „ťuknutí displeje“ jako na levé tlačítko. Tudíž se musí v programu nahradit funkce pravého tlačítka něčím jiným. Také se musí počítat s tím, že zde nelze dynamicky získávat polohu myši, jako to využívám v menu, kdy se podle pozice myši natáčí kostka, nebo při označení tlačítek. Toto by ale šlo s určitou přesností nahradit vbudovaným akcelerometrem. Pokud chceme číst ze souboru, nelze to udělat jednoduše přes třídu StreamReader jako na PC. Zde je pouze podporován tzv. IsolatedStorage, což je izolovaná schránka, kde jsou data schována bezpečněji, než klasicky na disku. Také jsem zde nemohl spustit jeden ze svých příkladů-přehravač, jelikož přehrávání videa není v XNA na Windows Phone doposud podporováno.
Jinak s konverzí nebyl větší problém, až na kosmetické detaily způsobené jiným rozlišením.



Obr.32 Emulátor Windows Phone

Závěr

Závěrem výplývá, že v XNA není problém s programováním obtížnějších technik grafického zobrazování, ale pokud budeme chtít dosáhnout maximálního výkonu, bude lepší sáhnout po nativním kódu. Zde je na místě posoudit naše vývojové možnosti a cíle, jako je počet vývojářů, rozpočet, velikost a účel aplikace, zda pro komerční účely, nebo jen jako freeware a hlavně nejdůležitější faktor – čas vývoje. Problematická může být nutná distribuce instalačních balíčků .NET Framework verze 4.0. Dnes vznikají, nebo už jsou v částečném chodu, projekty, které se snaží rozšířit XNA na více platforem, jako např.: Mono XNA. Pouštění aplikace ve Windows Phone ukázalo, jak snadno lze distribuovat aplikace na ostatní podporované platformy. Ačkoli nevlastním konzoli Xbox 360, věřím, že by se testování aplikace taktéž obešlo bez větších problémů. Programování v C# bylo pro mě novinkou a zajímavým přínosem. Netušil jsem, že by vývoj mohl být tak programově přívětivý a přitom si držet místo v konkurenci vývojových platforem. Ocenil jsem přehlednou práci, jelikož jsem dříve programoval v nativním kódu DirectX s jazykem C++. Určitě jsem se tímto nezabýval naposledy a budu uvažovat o nějaké další činnosti s touto platformou.

Přílohy

Příloha č.1 - Ukázka kódu XNA

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace WindowsGame1
{
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        GraphicsDevice device;
        VertexPositionColor[] vektory;
        Matrix viewMatrix;
        Matrix projectionMatrix;
        BasicEffect effect;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        protected override void Initialize()
        {
            graphics.ApplyChanges();
            base.Initialize();
        }

        protected override void LoadContent()
        {
            device = graphics.GraphicsDevice;
            Kamera();
            NastaveniVektoru();
        }

        private void NastaveniVektoru()
        {

            vektory = new VertexPositionColor[3];

            vektory[0].Position = new Vector3(0f, 0f, 0f);
            vektory[0].Color = Color.Red;
            vektory[1].Position = new Vector3(5f, 5f, 0f);
            vektory[1].Color = Color.Green;
            vektory[2].Position = new Vector3(5f, 0f, 0f);
            vektory[2].Color = Color.Yellow;
        }

        private void Kamera()                                   //Nastaveni Kamery
        {
            viewMatrix = Matrix.CreateLookAt(new Vector3(0, 0, 20), new Vector3(0, 0,
0), new Vector3(0, 1, 0));
            projectionMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
device.Viewport.AspectRatio, 1.0f, 300.0f);
        }
```

```
        protected override void Update(GameTime gameTime)
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
                this.Exit();
            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.Black);
            effect = new BasicEffect(device);
            effect.View = viewMatrix;
            effect.Projection = projectionMatrix;
            effect.VertexColorEnabled = true;
            foreach (EffectPass pass in effect.CurrentTechnique.Passes)
            {
                pass.Apply();
                device.DrawUserPrimitives(PrimitiveType.TriangleList, vektory, 0, 1,
VertexPositionColor.VertexDeclaration);
            }
            base.Draw(gameTime);
        }
    }
}
```

Příloha č.2 - Ukázka kódu Direct3D v C++

```
#include "StdAfx.h"
#include <windows.h>
#include <windowsx.h>
#include <d3d9.h>
#pragma comment (lib, "d3d9.lib")

LPDIRECT3D9 d3d;                    //Ukazatel na d3d rozhrani
LPDIRECT3DDEVICE9 d3ddev;           // Ukazatel na tridu zarizeni
LPDIRECT3DVERTEXBUFFER9 v_buffer = NULL;    //Ukazatel na buffer vertexu

void initD3D(HWND hWnd);    // Nastavuje a inicializuje Direct 3D
void render_frame(void);    // Tato funkce vykresluje jedotlive snimky
void cleanD3D(void);        // Ukoncuje Direct3D a uvolnuje pamet
void init_graphics(void);

struct CUSTOMVERTEX {FLOAT X, Y, Z, RHW; DWORD COLOR;};  // Nas vlastni vertexovy typ
#define CUSTOMFVF (D3DFVF_XYZRHW | D3DFVF_DIFFUSE)

LRESULT CALLBACK WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
//Obdoba funkce Main() v konzolovych aplikacich
int WINAPI WinMain(HINSTANCE hInstance,

                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow)
{
    HWND hWnd;
    WNDCLASSEX wc;
    ZeroMemory(&wc, sizeof(WNDCLASSEX));
    wc.cbSize = sizeof(WNDCLASSEX);
    wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc = WindowProc;
    wc.hInstance = hInstance;
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.lpszClassName = L"WindowClass";
```

```
    RegisterClassEx(&wc);
       hWnd = CreateWindowEx(NULL,                    //Zde vytvarime nase okno
                             L"WindowClass",
                                          L"Direct3D priklad",
                             WS_OVERLAPPEDWINDOW,
                             0, 0,
                             500, 400,
                             NULL,
                             NULL,
                             hInstance,
                             NULL);

    ShowWindow(hWnd, nCmdShow);
    initD3D(hWnd);                 //Inicializace Direct3D
    MSG msg;

    while(TRUE)
    {
        while(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }

        if(msg.message == WM_QUIT)
            break;

        render_frame();
    }
    cleanD3D();
    return msg.wParam;
}

//Tato funkce zpracovava message
LRESULT CALLBACK WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {
        case WM_DESTROY:
            {
                PostQuitMessage(0);
                return 0;
            } break;
    }
    return DefWindowProc (hWnd, message, wParam, lParam);
}

void initD3D(HWND hWnd)
{
    d3d = Direct3DCreate9(D3D_SDK_VERSION);
    D3DPRESENT_PARAMETERS d3dpp;
    ZeroMemory(&d3dpp, sizeof(d3dpp));
    d3dpp.Windowed = TRUE;
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.hDeviceWindow = hWnd;
    d3dpp.BackBufferFormat = D3DFMT_X8R8G8B8;
    d3dpp.BackBufferWidth = 500;
    d3dpp.BackBufferHeight = 400;
    d3d->CreateDevice(D3DADAPTER_DEFAULT,
                      D3DDEVTYPE_HAL,
                      hWnd,
                      D3DCREATE_SOFTWARE_VERTEXPROCESSING,
                      &d3dpp,
                      &d3ddev);
```

```
    init_graphics();
}

void render_frame(void)
{
    d3ddev->Clear(0, NULL, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0, 0, 0), 1.0f, 0);
    d3ddev->BeginScene();
    d3ddev->SetFVF(CUSTOMFVF);
    d3ddev->SetStreamSource(0, v_buffer, 0, sizeof(CUSTOMVERTEX));
    d3ddev->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 1);
    d3ddev->EndScene();
    d3ddev->Present(NULL, NULL, NULL, NULL);
}

void cleanD3D(void)
{                               //Zde uvolnime pamet
    v_buffer->Release();
    d3ddev->Release();
    d3d->Release();
}

void init_graphics(void)      //Zde nastavujeme vektory a vytvarime vertex buffer
{
    CUSTOMVERTEX vertices[] =
    {
        { 200.0f, 5.0f, 0.0f, 1.0f, D3DCOLOR_XRGB(0, 0, 255), },
        { 420.0f, 300.0f, 0.0f, 1.0f, D3DCOLOR_XRGB(0, 0, 255), },
        { 20.0f, 300.0f, 0.0f, 1.0f, D3DCOLOR_XRGB(0, 0, 255), },
    };
    d3ddev->CreateVertexBuffer(3*sizeof(CUSTOMVERTEX),
                               0,
                               CUSTOMFVF,
                               D3DPOOL_MANAGED,
                               &v_buffer,
                               NULL);
    VOID* pVoid;
    v_buffer->Lock(0, 0, (void**)&pVoid, 0);
    memcpy(pVoid, vertices, sizeof(vertices));
    v_buffer->Unlock();
}
```

Příloha č.3 - Ukázka kódu v GDI v C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;


namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
```

```
        }

        private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
        {
            //Vytvoreni bodů
            Point[] body = { new Point(100, 50), new Point(50, 200), new Point(200,
            200) };
            //Vykresleni trojuhleniku s vyplni
            e.Graphics.FillPolygon(new SolidBrush(Color.Red), body);
        }
    }

}
```

Použitá Literatura

[1] DirectX. *Wikipedia, the free encyclopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Directx

[2] DirectInput. *Wikipedia, the free encyclopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Xinput

[3] DirectSound. *Wikipedia, the free encyclopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Directsound

[4] DirectMusic. *Wikipedia, the free encyclopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Directmusic

[5] DirectPlay. *Wikipedia, the free encyclopedia* [online]. 2011 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Directplay

[6] DirectShow. *Wikipedia, the free encyclopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Directshow

[7] Microsoft Direct3D. *Wikipedia, the free encyclopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Direct3d

[8] Z-buffering. *Wikipedia, the free encyclopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Z-buffering

[9] Spatial anti-aliasing. *Wikipedia, the free encyclopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Anti-aliasing

[10] Alpha compositing. *Wikipedia, the free encyclopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Alpha_compositing

[11] Mipmap. *Wikipedia, the free encyclopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Mipmap

[12] Direct3D Architecture(Direct3D 9). *Microsoft MSDN* [online]. 2012 [cit. 2012-06-04]. Dostupné z:http://msdn.microsoft.com/enus/library/windows/desktop/bb219679%28v=vs.85%29.aspx

[13] *Evolution of DirectX* [online]. 2009 [cit. 2012-06-08]. Dostupné z: http://archive.techtree.com/techtree/jsp/article.jsp?article_id=104330&cat_id=584

[14] Microsoft XNA. *Wikipedia, the free encyclopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Microsoft_XNA

[15] Creating an XNA Game from the Template. *RB Whitakers Wiki* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://rbwhitaker.wikidot.com/xna-project-template

[16] Distributing your finished Windows Game. *Microsoft MSDN* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://msdn.microsoft.com/en-us/library/bb464156.aspx

[17] What do I need to make XNA Framework games run on other computers?. *App hub forums* [online]. 2012 [cit. 2012-06-04]. Dostupné z:http://forums.create.msdn.com/forums/p/1988/9924.aspx#9924

[18] MediaPlayer Class. *Microsoft MSDN* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://msdn.microsoft.com/enus/library/microsoft.xna.framework.media.mediaplayer.aspx

[19] Windows Photo Viewer. *Wikipedia, the free encyklopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Windows_Photo_Viewer

[20] Microphone Class. *Microsoft MSDN* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.audio.microphone.aspx

[21] Xbox 360. *Wikipedia, the free encyclopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Xbox_360

[22] Windows Phone. *Wikipedia, the free encyclopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Windows_Phone

[23] Reach vs. HiDef. *Shawn Hargreaves blog* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://blogs.msdn.com/b/shawnhar/archive/2010/03/12/reach-vs-hidef.aspx

[24] The effect file. *Riemers XNA Tutorial* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://www.riemers.net/eng/Tutorials/XNA/Csharp/Series1/The_effect_file.php

[25] Introduction to Effect Files. *MVPS* [online]. 2005 [cit. 2012-06-04]. Dostupné z: http://www.mvps.org/directx/articles/effect/effect1_1.htm

[26] World Space. *Riemers XNA tutorial* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://www.riemers.net/eng/Tutorials/XNA/Csharp/Series1/World_space.php

[27] Common Language Runtime. *Riemers XNA tutorial* [online]. 2012 [cit. 2012-06-04] Dostupné z: http://en.wikipedia.org/wiki/Common_Language_Runtime

[28] Antialiasing - vyhlazování teoreticky i prakticky. *PC Tuning* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://pctuning.tyden.cz/hardware/graficke-karty/14177-antialiasing-vyhlazovani-teoreticky-i-prakticky?start=2

[29] Media Overview. *Microsoft MSDN* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://msdn.microsoft.com/en-us/library/dd254869.aspx

[30] Just-In-Time compilation. *Wikipedia, the free encyclopedia* [online]. 2012 [cit. 2012-06-04]. Dostupné z: http://en.wikipedia.org/wiki/Just-in-time_compilation

Seznam obrázků

Seznam příloh

Příloha č.1 – Ukázkový kód v XNA

Příloha č.2 – Ukázkový kód v DirectX

Příloha č.3 – Ukázkový kód v GDI

Příloha č.4 – Aplikace se zdrojovými kódy na CD

Seznam zkratek

API – Application Programming Interface

OS – Operační Systém

CPU – Central Processing Unit

GPU – Graphics Processing Unit

SDK – Software Development Kit