

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.

2012

Autor: Jan Vacata

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2011/2012

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

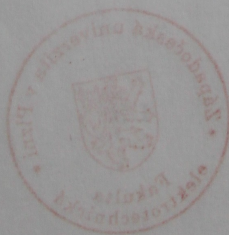
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Jan VACATA**
Osobní číslo: **E09B0243P**
Studijní program: **B2644 Aplikovaná elektrotechnika**
Studijní obor: **Aplikovaná elektrotechnika**
Název tématu: **Platforma aplikací XNA a vzorové řešení**
Zadávací katedra: **Katedra elektromechaniky a výkonové elektroniky**

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

Vypracujte vzorové příklady využívající technologii Microsoft XNA

1. Porovnejte technologii DirectX a XNA především ve směru technických aplikací.
2. Příklady zaměřte na přehrávání médií a vykreslování grafů apod.
3. Porovnejte s možnostmi nativního zobrazování prostředí OS Windows.
4. Diskutujte systémové požadavky. Ověřte možnost nasazení na alternativních platformách (Linux).



Rozsah grafických prací: podle doporučení vedoucího

Rozsah pracovní zprávy: 20 - 30 stran

Forma zpracování bakalářské práce: tištěná/elektronická

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

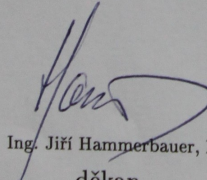
Vedoucí bakalářské práce:

Ing. Petr Weissar, Ph.D.

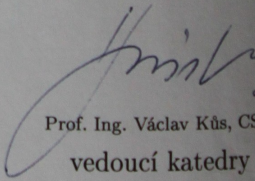
Katedra aplikované elektroniky a telekomunikací

Datum zadání bakalářské práce: 17. října 2011

Termín odevzdání bakalářské práce: 3. června 2012


Doc. Ing. Jifí Hammerbauer, Ph.D.
děkan




Prof. Ing. Václav Kůs, CSc.
vedoucí katedry

V Plzni dne 17. října 2011

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 médií operačního systému Windows. Porovnává systémové požadavky jednotlivých technologií. Ověřuje nasazení na alternativních platformách.

Klíčová slova

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

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 8. 6. 2012

Poděkování

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

Obsah

OBSAH

ÚVOD

1 DirectX

1.1 Co je to DirectX?.....	4
1.1.1 Direct3D.....	5
1.2 Historie, vývoj rozhraní.....	7

2 XNA

2.1 Co je to XNA?.....	9
2.2 Vývoj platformy.....	10
2.3 Prostředí Microsoft Visual Studio 2010, Game Studio 4.0.....	10
2.4 Vývoj	
2.4.1 Proces.....	11
2.4.2 Grafika.....	13
2.4.2.1 2D.....	14
2.4.2.2 3D.....	15
2.4.3 Audio.....	16
2.4.4 Vstupy.....	16
2.4.5 Distribuce.....	17

3 Porovnání DirectX a XNA

3.1 Hardwarová náročnost.....	18
3.2 Náročnost vývoje.....	18
3.3 Rozdíl ve vykreslování grafiky.....	18

4 Popis vypracovaných příkladů.....

5 Porovnání s nativními aplikacemi OS Windows22

6 Alternativní platformy

6.1 Linux.....	24
6.2 Konzole XBOX 360.....	24
6.3 Handheldy.....	25

ZÁVĚR

POUŽITÁ LITERATURA

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

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.

Jakožto nadstavba, je dobré vědět něco o základu této platformy, abychom pochopili její sílu. 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 ukázkou jsem vytvořil pár příkladů ukazující jaké jsou možnosti tohoto prostředí a jak snadno se s nimi pracuje, v porovnání s psáním 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, 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 využít 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

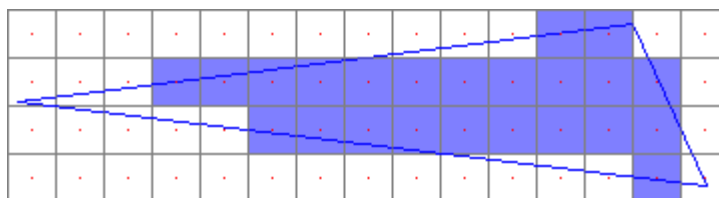
Toto API se využívá v případech, kdy je potřeba vykreslit grafiku ve třech dimenzích (3D). Využívá hardwarovou akceleraci, což je jedna z technik, jak zvýšit výkon při výpočtu. Zjednodušeně řečeno využívá hardware, nejčastěji na grafické kartě (pokud jej podporuje), který je přímo navržen pro vykonávání 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 patří jmenovitě:

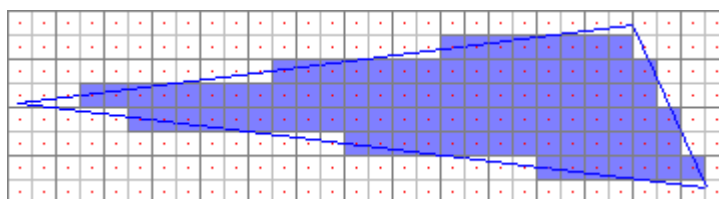
Z-Buffering- Řeší problém viditelnosti objektů. Ke každému vygenerovanému pixelu v prostoru uložíme mimo souřadnic X a Y také hodnotu souřadnice Z – hloubku. Pokud nastane situace, že na stejné X a Y souřadnici musíme vykreslit další pixel, tato metoda porovná hloubku jednotlivých pixelů a zobrazí ten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

Anti-aliasing- Objekty jsou vykreslovány tak, že každý pixel je vybarven barvou, která se nachází v jeho středu. Problém 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.



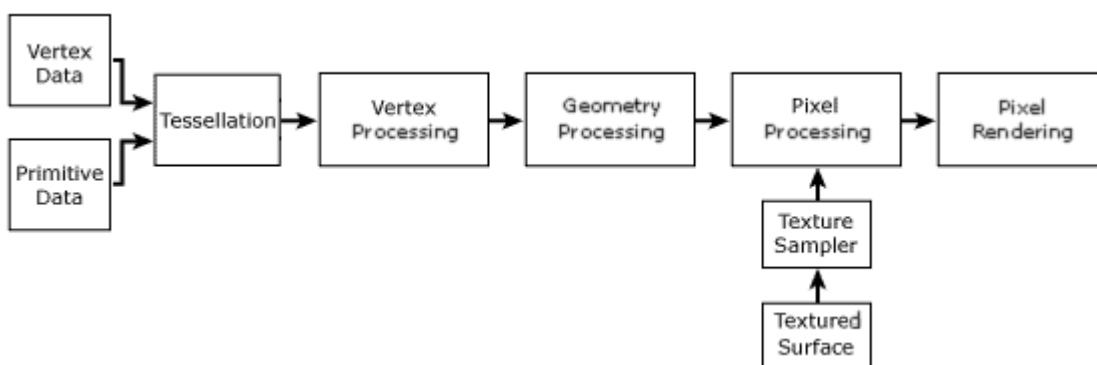
Obr.2 Oversampling

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 uloženy 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, uloženy 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

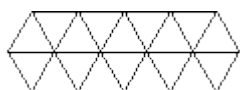
vykreslování objektů pouze v záběru (clipping).

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 laptotech. 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 textur 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 environment 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 Tessellation. Vydán také pro Xbox.

DirectX 9.0- Jedna z nejdůležitějších 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 přístupu 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

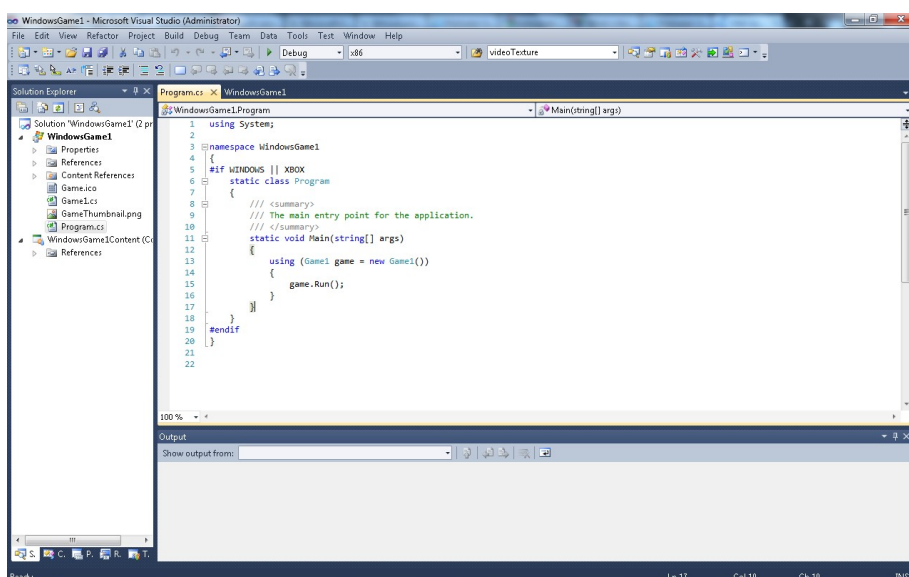
2.1 Co je to XNA?

Vyvíjen firmou Microsoft, XNA je sada nástrojů, která výrazně usnadňuje tvorbu 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 zmíněné dva. Framework XNA je 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, které jsou hardwarově urychlovány. V této verzi je nyní umožněné rozsáhlé nastavení tzv. Effect (Viz Kapitola XX). Nově možná práce s mikrofonom. Vylepšená práce s obrázky. [14]

2.3 Prostředí Visual Studio 2010

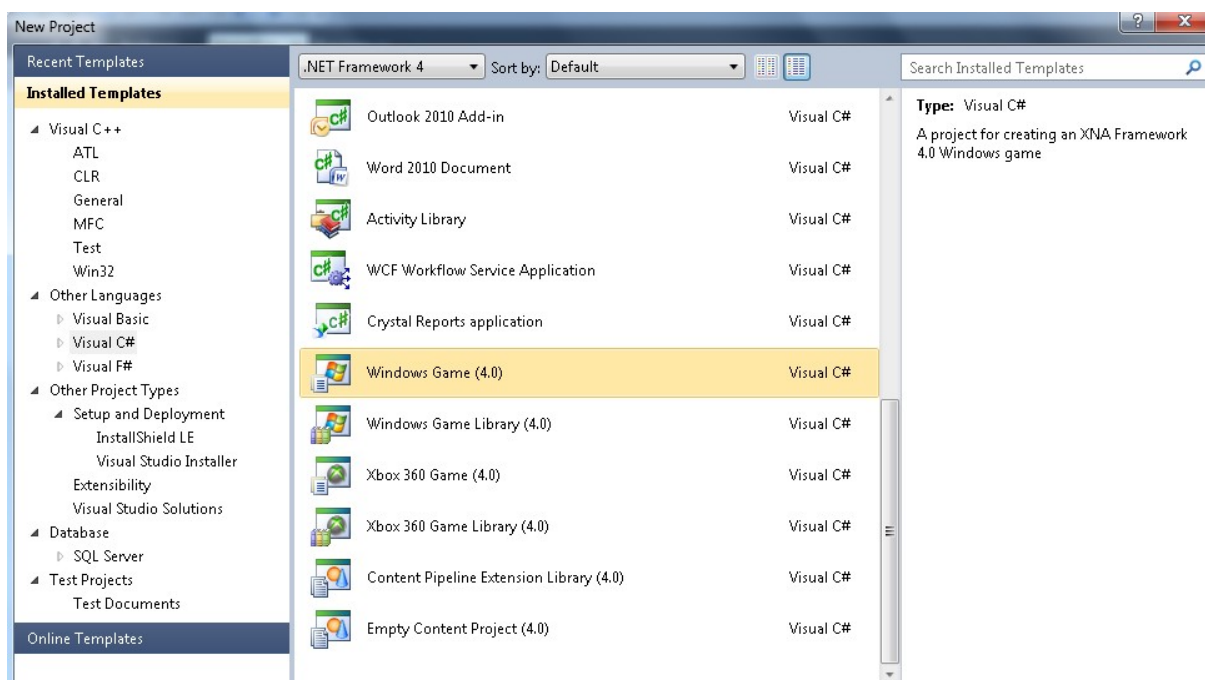
K dispozici je nám pár užitečných nástrojů.



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 nastavení 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.



Obr.7 Přehled typů projektů

2.4 Vývoj

2.4.1 Proces

Zde se podíváme na jednoduchou aplikaci v XNA 4.0., která nám vykreslí prázdné okno. Na té si ukážeme stěžejní součásti programu. Převzato z [15].

a)Direktivy

Pomocí nichž kompilér ví, že používáme komponenty, které byly již dříve vytvořeny.

Př.:

```
1 using System;
```

Obr.8 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.9 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.10 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.11 Inicializace

e)Load a Unload Content

```

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

```

Obr.12 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 |
339 |
340 |
341 |
342 |
343 |
344 |
345 |
protected override void Update(GameTime gameTime)
{
    if (player.State == MediaState.Stopped)
    {
        player.IsLooped = true;
    }
}

```

Obr.13 Update

Tato metoda je zavolána každý snímek 60x. Stará se o průběh aplikace, jestli se něco změnilo. Např.: Jestli bylo zmáčknuto tlačítko, nebo zde napsaný příkaz znamenající, že se zrovna přehravající video bude pouštět ve smyčce.

g)Draw

```

378 |
379 |
380 |
381 |
382 |
383 |
Zde prc
sekund
protected override void Draw(GameTime gameTime)
{
    spriteBatch.Begin(SpriteSortMode.BackToFront, BlendState.AlphaBlend);
    spriteBatch.Draw(obrazek, new Vector2(400, 381), Color.White);
    spriteBatch.End();
}

```

Obr.14 Draw

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:Reach podporuje shader model 2.0, maximální

velkost textur 2048. [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 následovně: *Pozadi = Content.Load<Texture2D> ("Nebe");* Název souboru je *Nebe* a odted' 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 pipeline, např.: výpočet mipmapů, kde se nemusí zaokrouhlovat, jelikože rozměr je vždy dělitelný 2.

Obrázky vykreslujeme pomocí tzv. Dávkovače (spriteBatch).

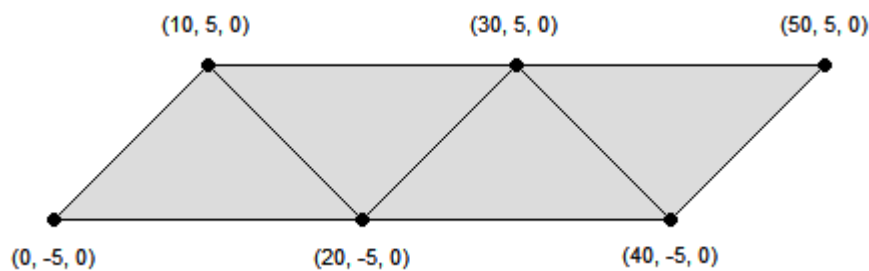
```
463 spriteBatch.Begin(SpriteSortMode.BackToFront, BlendState.AlphaBlend);  
464 spriteBatch.Draw(sipka, new Vector2(410, 340), Color.White);  
465 spriteBatch.End();
```

Obr.15 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

Jak je známo, všechny objekty jsou složeny z primitivních útvarů - trojúhelníků, skládajících se do sebe, formující modely.



Obr.16 Primitivní obrazce

XNA využívá ke kreslení tzv. *effecty*. *Effect* je kód, který určuje, jak má grafická karta tyto trojúhelníky vykreslit. Má podobu souboru s příponou *.fx, který nahrajeme do naší Content složky [24]. Každý effect se skládá ze tří základních částí [25]:

- Definice parametrů – Zde se nastaví proměnné před vykreslením, které jsou později použity ve výpočtu, jako projekční matice, osvětlení, nebo definice textur.
- Definice technik – Určuje, jakým způsobem vykreslíme objekt. Například jednou technikou vykreslíme objekty bez textur, nebo v určité barvě apod..
- 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.17 Matice

viewMatrix – Tato matice v sobě ukládá data, jak se bude kamera dívat na naši 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 [26].

Když máme tyto matice definované, předáme je našemu *Effect* kódu přes *effect.Parameters*.

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 velká úspora výkonu, např.: při vykreslení krychle zobrazíme jen vnější povrch. Kdybychom umístili kameru uvnitř krychle, což může být nežádoucí, a dívali se směrem ven, nevidíme 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 knihovnu XACT. Jelikož můj systém podporuje pouze režim Reach, který ale knihovnu XACT nepodporuje, budeme se zabývat druhou možností a to zjednodušené audio API vyvinuté pro tyto případy. Zde se se zvuky nakládá jako s ostatním obsahem, např. s texturami. Jednolivé zvuky nahrajeme do Content, nastavíme jako typ *SoundEffect* a inicializujeme funkcí *Content.Load*. Pak již jen velmi jednoduše zavoláme funkci pro přehrání:

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

Obr.18 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.19 Pauza videa

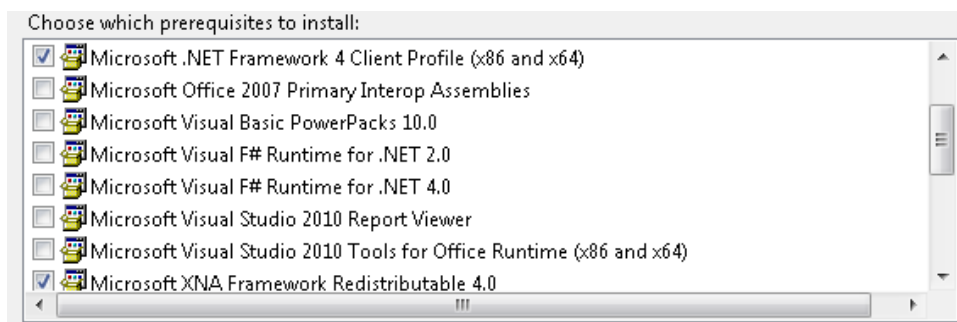
Standartně je na OS Windows podporována klávesnice, myš, gamepad a od verze 4.0 i mikrofon. Na windows phone jsou navíc podporovány i dotykový panel nebo accelerometer.

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 zjednodušuje distribuci programů. Před instalací zkontroluje zda máme nainstalované potřebné komponenty, které jsme zvolili v nastavení před kompilací. [16]



Obr.19 Výsek z přehledu možných instalačních balíků

Případně 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, detekce kolizí, práci se vstupem atd.. Největší zátěží by se dalo říci že je 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.

4 Popis vypracovaných příkladů

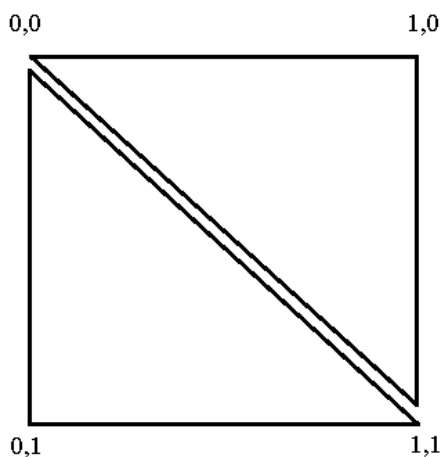
Knihovnu XNA jsem demonstroval v pár jednoduchých příkladech. Program se skládá z menu, ve kterém si vybereme jednotlivé příklady. Nyní popíšu příklad č.1.

Tento příklad demonstruje přehrávání videa na rotující krychli. Tu jsem sestavil tak, že na každou stěnu jsem použil dva stejné trojúhelníky.

```
130 private void SetUpVertices()  
131 {  
132     vertices = new VertexPositionTexture[36];  
133  
134     vertices[0].Position = new Vector3(-5f, 5f, 5f);  
135     vertices[0].TextureCoordinate.X = 0;  
136     vertices[0].TextureCoordinate.Y = 0;  
137
```

Obr.20 Vykreslení trojúhelníku

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



Obr.21 Formát textur

Krychli je možno rotovat pomocí šipek z klávesnice. Video je možno paunout a poté znovu pokračovat. Promítání funguje tak, že z podmínky

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

Obr.22 Textura z videa

získáme v každý okamžik, kdy video přehrává, jeho momentální frame a uložíme ho do proměnné videoTexture, kterou pak promítneme na stěně krychle. 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 získaná z videa 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.23 Přepnutí na Clamp mód



Obr.24 Obrázek z příkladu 1

Jako druhý příklad jsem představil vykreslování 2D grafiky. Na ukázkou jsem vytvořil tvorbu jednoduchých X-Y bodových grafů. Příklad funguje tak, že zadáme hodnoty bodů X a Y a program nám vykreslí křivku. Hodnoty zadáváme postupně, tím plníme pole dvourozměrných vektorů. Křivky, 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.25 Vlastní funkce DrawLine

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

5 Porovnání zobrazování s nativními aplikacemi OS Windows

a) Přehrávání audia a videa

XNA používá k přehrávání audia knihovnu MediaPlayer. Obsahuje funkce jako opakování skladby, náhodné přehrání, ovládání hlasitosti nebo určení momentální pozice v skladbě. 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 přehrávání 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.26 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 porovnání 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 vypočítat 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 porovnání s nativními aplikacemi Windows, omezené funkce.

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, který podporuje Framework verze 4.0. Samozřejmě by to mělo dopad na systémové požadavky. Lepším řešením by bylo zvolit alternativní cestu vývoje, jako např.: OpenGL který je multiplatformní apod..

6.2 Xbox 360

Jelikož jedním z důvodů proč Game Studio vzniklo, byla podpora vývoje her pro konzoli Xbox, především pro podporu nezávislých vývojářů, tak aplikace vyvinuté pomocí XNA jsou v něm 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 [21].

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 přídatná) 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].

Závěr

Závěrem vyplývá, 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. Zatím ale nedosahují potřebných kvalit.

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.

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 encyclopedia* [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

Obr.1	[28]
Obr.2.....	[28]
Obr.3.....	[12]
Obr.26.....	[29]

Seznam příloh

Příloha 1. dodána na CD