Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Rozšíření herního enginu Innovative (XNA)

Plzeň, 2009 Jan Vytlačil

Prohlašuji, že jsem bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 12. května 2009 *Jan Vytlačil*

Extension of Innovative game engine (XNA)

14. května 2009

Abstrakt

XNA is a set of tools provided by Microsoft that allows us to create games for Windows, Xbox a Zune platforms.

First part of this work contains a description of the XNA technology, free XNA engines analysis and comparsion of their main features.

Next part describes creation of some new components for Innovative game engine. This components add dynamic sky with adjustable daytime, dynamic procedurally generated clouds and efficient rain particle effect.

Obsah

1 Úvod

Tato práce se snaží zmapovat možnosti, které v současné době nabízí technologie XNA těm programátorům herních aplikací, kteří nechtějí začínat s vývojem úplně od začátku, ale raději by využili některý z dostupných herních enginů. Pro XNA existuje několik komerčních enginů, které si kladou za úkol zjednodušit vývoj komplexních herních prostředí. Ale jak je na tom v oblasti volně dostupných enginů? Pro OpenGL a DirectX je nabídka v této oblasti poměrně široká a je z čeho vybírat. XNA je poměrně nová technologie, která disponuje aktivní vývojářskou komunitou a o množství různých projektů není nouze. Otázkou je, v jaké fázi vývoje se nacházejí a nakolik jsou použitelné v praxi. A právě to je jedna z hlavních z otázek, kterými se bude zabývat tato práce. Cílem je informovat čtenáře o technologii XNA a některých zajímavých, volně dostupných enginech pro tuto technologii. Prozkoumat jejich funkce, vlastnosti a pokusit se o jejich srovnání.

Další část práce popisuje rozšíření Innovative enginu o nové komponenty z oblasti modelování počasí. Popsána bude tvorba dynamické oblohy umožňující střídání dne a noci včetně pohybu Slunce. Dále tvorba dynamické vrstvy mraků a deště reagujícího na směr větru.

2 XNA

2.1 Úvod

Technologie XNA od společnosti Microsoft vznikla jako nástroj pro usnadnění vývoje počítačových her, nejen pro platformu Microsoft Windows, ale i herní zařízení Microsoft Xbox360 a v poslední verzi i multimediální přehrávač Microsoft Zune.[1] V principu nabízí herním vývojářům to, co Microsoft NET Framework vývojářům "běžných" aplikací. Tedy především pohodlí řízeného (managed) kódu, moderní programovací jazyk C# a vyšší míru abstrakce při programovaní velké částí rutinních úloh.

XNA Game Studio je označením pro sadu vývojových nástrojů primárně zaměřených na studenty, nadšence a nezávislé vývojáře. XNA Game Studio funguje s vývojovým prostředím Visual Studio, nebo Visual C# Express a umožňuje vytvářet hry pro Windows a Xbox 360. XNA Game Studio obsahuje:

- XNA Framework, sadu vysokoúrovňových vývojářských knihoven pro dosažení vyšší produktivity vývoje.

- XNA Framework Content Pipeline, sadu nástrojů pro snazší import obsahu hry v nejrůznějších formátech.
- Dokumentaci, sadu návodů a ukázkových aplikací.[2]

2.2 Historie

Vznik XNA je úzce spojen s rozhraním Managed DirectX. Druhá verze Managed DirectX byla zrušena ve fázi betaverze. První verze XNA Game Studia převzala velkou část její funkcionality. Stejně jako Managed DirectX je založena na technologii DirectX, integruje technologie XACT a XINPUT. Přesto XNA není náhradou Managed DirectX.[2]

První verze, známá pod názvem XNA Game Studio Express, byla vypuštěna 30. září 2006. Vzbudila veliký zájem médií i nezávislých vývojářů. Nabízela široké veřejnosti nástroj pro vývoj her nejen pro Windows, ale i konzoli Xbox 360. Dále podporu .fbx formátu pro 3D data od společnosti Autodesk. Oznámena byla spolupráce s firmou GarageGames, která do dneška nabízí své komerční nástroje jako nadstavbu XNA Game Studia.[3]

XNA Game Studio 2.0 vyšlo 13. prosince 2007. Zásadnější změnou, kterou přineslo, byla podpora všech verzí Visual Studia 2005, včetně edicí Standard, Professional a Express. Velká část všech vylepšení byla provedena především jako reakce na nejčastější připomínky komunity.[4]

Současná poslední verze XNA Game Studio 3.0 byla vydána 30.10.2008.[5] Výraznějšími změnami, které přinesla, byla podpora všech verzí Microsoft Visual Studio 2008 včetně Visual C# Express 2008. Možnost vytvářet hry pro multimediální přehrávač Zune. Podpora nových vlastností jazyka C# 3.0.[1]

2.3 Vlastnosti

XNA Game Studio 3.0 nabízí jednotné rozhraní pro vývoj her na platformy:

- Windows
- Xbox 360
- Zune*[1]*

Následující část podrobněji popisuje možnosti, které nabízejí jeho základní součásti, XNA Content Pipeline a XNA Framework. Služba XNA Creators Club Online není součástí XNA Game Studia, avšak velmi těsně s ním souvisí a nelze se o ní nezmínit.

2.3.1 XNA Content Pipeline

Jde se o nástroj pro automatizované zahrnutí obsahu v různých formátech do hry. Je založený na sadě programů pro převod (Importers) a překlad (Processors) souborů s různými typy obsahu. Tyto programy jsou distribuovány jako součást XNA Game Studia. Importer slouží pro převod obsahu (například 3D model ve formátu .fbx, obrázek .jpg apod.) do některého z formátů, které podporují programy pro překlad(Content Processors). Content Processor přeloží importovaný obsah do podoby objektu řízeného kódu (managed code object), který může být načten a použit ve hře.[6] Následuje přehled standardních součástí Content Pipeline dodávaných s XNA 3.0. Importers:

- FbxImporter - Import modelů ve formátu .fbx.
- EffectImporter - Import efektů ve formátu .fx.
- FontDescriptionImporter - Import souborů .spritefont.
- TextureImporter - Importuje soubory ve formátech .bmp, .dds, .dib, .hdr, .jpg, .pfm, .png, .ppm, a .tga.
- XImporter - Import modelů ve formátu .x.
- XmlImporter - Slouží pro import XML souborů pro editaci parametrů objektů.
- Import zvuku ve formátu XACT s příponou .xap.

Processors:

- EffectProcessor - Překlad efektu.
- ModelProcessor - Překlad modelu.
- PassThroughProcessor - Pro objekty ve formě, ve které je lze použít bez překladu.
- FontDescriptionProcessor - Překlad popisu fontu.
- FontTextureProcessor - Překlad textury s fontem.
- TextureProcessor - Překlad textury.
- Překlad XACT zvuku.[7]

V rámci Content Pipeline je možné použít jakýkoli kompatibilní program pro převod a překlad třetích stran, nebo si napsat vlastní.[6]

2.3.2 XNA Framework

XNA Framework je sada knihoven řízeného kódu pro Windows, Xbox 360 a Zune, sloužící pro vývoj her. Pokrývá oblasti renderování grafiky, načítání obsahu, manipulace s maticemi a vektory, obsluhy vstupních zařízení, přehrávání audio souborů, práce s médii uloženými v zařízení, ukládání a načítání souborů, správy a nastavení LIVE účtů, síťové komunikace prostřednictvím Xbox Live a Games for Windows - LIVE Servers a další.[8]

2.3.3 XNA Creators Club online

Komunitní server XNA Creators Club Online obsahuje důležité informace, rady, diskusní fóra a návody věnované XNA Game Studiu a zároveň zprostředkovává přístup ke zpoplatněným službám. Zpoplatněno je využívaní Xbox Live a Games for Windows Live serverů a distribuce hotových Xbox 360 her prostřednictvím Xbox Live Arcade.[9]

Hotovou hru pro Xbox 360 lze odeslat na server, kde bude zhodnocena ostatními uživateli. Na základě hodnocení může být hře přidělena cena a umožněna distribuce prostřednictvím služby Xbox Live Arcade.[10]

2.4 Požadavky a instalace

XNA Game Studio ve verzi 3.0 je kompatibilní Visual C# 2008 a všemi verzemi Visual Studia 2008. Žádná z verzí Visual Studia 2005 a dřívějších podporována není. Lze jej provozovat na systému Windows XP ve verzích:

- Home Edition
- Professional Edition
- Media Center Edition
- Tablet Edition

a Windows Vista ve verzích:

- Home Basic Edition
- Home Premium Edition
- Business Edition
- Enterprise Edition
- Ultimate Edition

Pro provoz her postavených na technologii XNA je zapotřebí grafická karta podporující Shader Model 1.1 nebo vyšší a DirectX 9.0c. Ostatní hardwarové požadavky jsou identické s těmi pro Visual Studio 2008. Dalším požadavkem je přítomnost Microsoft NET Framework 3.5 (instalován automaticky s jakoukoliv verzí Visual Studia 2008).[11]

Instalátor XNA Game Studio 3.0 můžete stáhnout ze stránek společnosti Microsoft. Při instalaci postupujte následovně:

1. Odinstalujte předchozí verze XNA Game Studio 3.0 (beta, CTP).
2. Nainstalujte Microsoft Visual C# 2008 Express Edition nebo některou z verzí Microsoft Visual Studio 2008.
3. Nainstalujte si poslední updaty pro Visual Studio z Microsoft Update.
4. Stáhněte a spusťte instalátor XNA Game Studio 3.0.
5. Řiďte se instrukcemi instalátoru.
6. Spusťte Visual Studio 2008 ze složky Microsoft Game Studio 3.0 ve Start Menu.[5]

XNA Game Studio 2.0 funguje jen s Visual C# Express 2005, nebo některou z edic Visual Studia 2005. S novějšími verzemi Visual Studio 2008 jej použít nelze. Vyžaduje Microsoft NET Framework 2.0. Ostatní požadavky i postup instalace se shodují s novější verzí.[12]

3 Volně dostupné enginy pro XNA

3.1 Ox Game Engine

3.1.1 Úvod a historie

Ox Game Engine je 3D XNA herní engine. Projekt sídlí na webu www.codeplex.com. Projekt byl založen 16. ledna 2008. Engine využívá možností dalších projektů ze stránek www.codeplex.com:

- JigLibXPhysics Libary - Knihovna pro herní fyziku.
- XNAnimation Libary - Knihovna přinášející podporu animovaných 3D modelů.
- Doom3 MD5 Importer - Knihovna pro import 3D modelù ve formátu MD5, který využívají hry založené na Doom3 enginu od společnosti idSoftware.

Starší verze projektu jsou určeny pro XNA Game Studio 2.0. První verze (1.0.3.0) pro XNA Game Studio 3.0 vyšla 4. prosince 2008. Projekt se podle informací na jeho webových stránkách nachází ve vývojovém stádiu alfa verze. Některé jeho části proto mohou být neúplné nebo nefunkční.[13]

Vlastnosti i funkce enginu se vzhledem k vývojovému stádiu, ve kterém se nachází, mohou rychle změnit. Následující informace se týkají Ox Game Enginu ve verzi 1.0.3.0 určené pro XNA Game Studio 3.0.

3.1.2 Funkce

Následuje stručný přehled funkcí Ox Game Enginu.

- Procedurální geometrická primitiva. Jmenovitě krychle, kvádry a výškové mapy.
- Volitelné Quadtree/Octree ořezávání.
- Tvorba terénu na základě výškové mapy s možností multitexturingu.
- Práce s ambientním, směrovým a bodovým osvětlením.
- Systém pro snadnou tvorbu částicových efektů.
- Statický SkyBox.
- Zobrazení reflexivní vodní hladiny.
- Skinovatelný, událostmi řízený systém uživatelského rozhraní.
- Systém zjednodušující práci s kamerou.
- Zobrazení stínů technikou směrového shadow mappingu.
- Systém pro skriptování.
- Editor scény a uživatelského rozhraní.
- Podpora základní fyziky prostřednictvím JigLibXPhysics Libary.
- Podpora 3D modelů formátu .x a .fbx včetně animací prostřednictvím knihovny XNAnimation.
- Podpora MD5 modelů včetně animací.

3.1.3 Instalace

Archiv se všemi zdrojovými kódy a dalšími potřebnými soubory lze stáhnout ze stránek projektu na serveru www.codeplex.com. Archiv rozbalte. Protože projekty s obsahem a skripty se kompilují zvlášť, je třeba provést jejich kompilaci dříve, než se pokusíte zkompilovat projekt samotného enginu. Otevřete následující soubory a zkompilujte všechny obsažené projekty:

- CommonContent.sln
- CommonScripts.sln
- ExampleContent.sln
- ExampleScripts.sln

v případě, že používáte Visual C# Express 2008, nebo soubory:

- CommonContentPro.sln
- CommonScriptsPro.sln
- ExampleContentPro.sln
- ExampleScriptsPro.sln

v případě, že používáte některou z verzí Visual Studia 2008.

Nyní můžete otevřít soubor Ox.sln (případně OxPro.sln). Pokud nemáte k počítači připojenou konzoli Xbox 360, odstraňte z okna Solution Explorer všechny projekty určené pro tuto platformu, jinak nebude pokus o přeložení úspěšný. Pokud máte v systému nastavený desetinný oddělovač na znak , (čárka) pravděpodobně skončí překlad vyhozením vyjimky. Problém lze vyřešit pomocí nastavení desetinného oddělovače na znak . (tečka) pomocí ovládacího panelu systému Windows - Místní a jazykové nastavení.

3.1.4 Používání a dokumentace

Architektura enginu je založena na komponentech. Snaží se o co nejvyšší otevřenost a rozšiřitelnost. Bohužel, možná že právě to je důvodem značné nepřehlednosti a těžkopádnosti jeho použití, která je patrná už při prvním pohledu do zdrojových kódů některé z ukázkových aplikací. K dispozici není žádná dokumentace, návod, nebo alespoň popis základních tříd. Situaci nijak neulehčuje ani nepřítomnost jakýchkoliv komentářů v již zmíněných ukázkových aplikacích, které jsou bohužel jediným vodítkem pro člověka, který by se pokusil služeb enginu využít. V samotných zdrojových kódech enginu už se komentáře vyskytují, ovšem ve zcela nedostatečném množství.

3.1.5 Licence

Ox Game Engine je šířen pod licencí Microsoft Public License (Ms-PL). Jedná se o open-source licenci, která umožňuje šířit software v kompilované podobě pro komerční i nekomerční účely. Distribuce v podobě zdrojových kódů je možná jen pod stejnou licencí, tedy Ms-PL. Ukládá povinnost na vyžádání zveřejnit případné provedené změny a rozšíření (ve formě zdrojových kódů) provedené na softwaru chráněném touto licení.[14]

3.1.6 Shrnutí

Celkový dojem z enginu je značně rozporuplný. Nabízí velké množství funkcí, ale na druhé straně naprosto selhává v oblasti dokumentace a komentování zdrojových kódů. Navíc je jeho použití těžkopádné a struktura nepřehledná.

Poznámka: 9.1.2009 byla vydána verze 2.0. Autor enginu vyjádřil nespokojenost s dosavadním směřováním projektu. Rozhodl se k radikální změně architektury, vedoucí ke zjednodušení použití enginu. To zásadně ovlivnilo způsob práce s enginem. Informace v tomto dokumentu se týkají verze předchozí, nyní již nemusí být zcela aktuální.

3.2 Quick Start

3.2.1 Úvod a historie

QuickStart Engine je podle statistik jedním z nejoblíbenějších enginù ze serveru www.codeplex.com. Projekt byl založen 3. října 2007. V roce 2008 byl vývoj zastaven a odstartován zcela od začátku. Důvodem byla celková změna architektury, která by umožnila implementaci dalších funkcí.

Původní verze nabízela velké množství zajímavých vlastností. Generování terénu z výškové mapy, pokročilou vodní hladinu včetně interakce s ostatními objekty, fyziku založenou na technologii Ageia PhysX, SkyBox systém, částicové efekty a další. Vývoj byl pravděpodobně zastaven z důvodu problémů s implementací dalších funkcí.

Na nové verzi se pracuje teprve několik měsíců a v současné době ještě nedosáhla úrovně verze předchozí. Přesto se jedná o velice ambiciózní projekt, především díky své (v porovnání se svými konkurenty) dlouhé historii. Projekt se podle informací na jeho webových stránkách nachází ve vývojovém stádiu alfa verze. Některé jeho části mohou být neúplné nebo nefunkční.[15]

Vlastnosti i funkce enginu se vzhledem k vývojovému stádiu, ve kterém se nachází, mohou změnit. Následující informace se týkají QuickStart Enginu ve verzi 0.196 určené pro XNA Game Studio 3.0.

3.2.2 Funkce

Následuje stručný přehled funkcí QuickStart Enginu.

- Podpora 3D modelů .x a .fbx.
- Podpora přehrávání audio souborů.
- Podpora konfiguračních XML souborů.
- Pokročilý systém práce s kamerou. Podporuje ArcBall, pevně umístěnou kameru, volnou a FPS kameru.
- Generování terénu z výškové mapy. Umožňuje vyhlazování terénu. Obsahuje podporu pro zobrazení různých úrovní detailů a multitexturingu. Také ořezávání terénu pomocí QuadTree stromu.
- Základní podpora osvětlení.
- Podpora materiálů. Materiály mohou být načítány pomocí Content Pipeline, nebo přímo za běhu z .qsm souborů.
- Podpora fyziky prostřednictvím knihoven JigLibXPhysics Libary a Ageia PhysX.

3.2.3 Instalace

Zdrojové kódy i všechny další potřebné soubory lze stáhnout ve formě archivu ze stránek projektu na serveru www.codeplex.com. Rozbalte archiv. Otevřete jeden z následujících souborů podle platformy:

- QuickStart_Windows.sln - verze pro Windows
- QuickStart All.sln - verze pro Windows i Xbox 360
- QuickStart Xbox360.sln - verze pro Xbox360

ve Visual C# Express 2008, nebo některé z verzí Visual Studio 2008. Pro vývoj her pro platformu Xbox 360 budete potřebovat NVidia PhysX Driver, který můžete stáhnout na adrese http://www.ageia.com/drivers/drivers.html. Pro práci na vývoji fyzikální části budete potřebovat:

- Visual Studio 2005 pro vývoj v C++
- NVidia PhysX SDK*[15]*

3.2.4 Používání a dokumentace

Základním prvkem enginu je entita. Pod pojmem entita rozumíme takřka cokoliv, co může být zobrazeno ve scéně, dokonce i kameru.

Funkcionalita entity je specifikována jednou nebo více komponentami, které lze entitám přiřazovat. Potřebujeme-li například, aby se entita pohybovala a reagovala na ostatní objekty, přidáme jí takzvaný PhysicsComponent.

Jednotlivé komponenty spolu mohou komunikovat prostřednictvím systému zpráv. Komunikace je možná v rámci jedné entity i mezi komponentami různých entit.

Engine je z hlediska architektury velice zajímavý, především díky výše zmíněnému systému entit, komponent a zpráv. Jedná se o velice flexibilní systém, především díky možnosti "poskládat" si přesně takovou entitu, jaká je v dané situaci potřeba.

Na stránkách projektu lze stáhnout velice praktický QuickStart Tutorial. Jeho prostřednictvím se lze velice snadno a rychle seznámit se základními principy fungování enginu a použitím jeho základních funkcí. Zdrojové kódy jsou navíc velice důsledně komentovány.

3.2.5 Licence

QuickStart Engine je šířen pod licencí Microsoft Public License (Ms-PL). Více o této licenci viz. 3.1.5.

3.2.6 Shrnutí

Engine vyniká především v oblasti návrhu vnitřní architektury, která je velmi flexibilní při zachování jednoduchého použití. V oblasti dokumentace je na tom také velmi dobře. Jeho zdrojové kódy jsou perfektně komentované, k dispozici je i praktický tutoriál.

Hlavní nevýhodou je, že se zatím nachází ve velmi rané fázi vývoje. V porovnání s konkurencí mu chybí mnoho důležitých funkcí. Ovšem z dlouhodobějšího hlediska jde pravděpodobně o projekt s největším potenciálem. V plánu je přidání podpory umělé inteligence, síťové komunikace, animací modelů, SkyBox/SkyDome systému, vodních ploch, systému osvětlení, zobrazení stínů, projekčního texturování a normálového mapování. Uvažuje se i o editoru scény, kterou půjde uložit ve formě XML souboru.

3.3 X-Engine

3.3.1 Úvod a historie

Jedná se o další 3D herní engin sídlící na stránkách www.codeplex.com. Projekt byl založen 22. března 2008. Mezi jeho hlavní priority patří jednoduchost použití a modularita. Engin je stále ve fázi vývoje. Některé jeho části mohou být neúplné, nebo nefunkční. Vlastnosti i funkce enginu se vzhledem k vývojovému stádiu, ve kterém se nachází, mohou změnit. Následující informace se týkají X-Enginu ve verzi 2.1.5 určené pro XNA Game Studio 2.0. Další verze by měla přinést podporu XNA Game Studia 3.0.[16]

3.3.2 Funkce

Následuje stručný přehled funkcí X-Enginu.

- Podpora 3D souborů ve formátu .x .fbx.
- Fyzika prostřednictvím JigLibXPhysics Libary.
- Generování terénu z výškové mapy. Lze využít multitexturingu.
- Scéna s podporou octree ořezávání.
- Statický SkyBox.
- Dynamický SkyDome s podporou denního cyklu.
- Pohyblivá vodní hladina s odrazem i refrakcí.
- Sytém pro snadnou tvorbu částicových efektů.
- Nastavitelné směrové a ambientní osvětlení terénu.
- Jednoduchý systém uživatelského rozhraní.
- Systém kamer. Obsahuje ArcBall, Chase a FreeLook kameru.
- Podpora jednoduchého skriptování prostřednictvím objemových spouštěčů.
- XCar komponenta pro simulaci vozidel.

3.3.3 Instalace

Archiv obsahující všechny potřebné soubory lze stáhnout ze stránek serveru www.codeplex.com. Rozbalte archiv a otevřete soubor X-Engine.sln v programu Visual C# Express 2005 s nainstalovaným XNA Game Studiem 2.0.

Pokud nemáte připojenu konzoli Xbox 360, smažte všechny projekty pro tuto platformu. Při pokusu o spuštění ukázkové aplikace se může projevit problém s desetinným oddělovačem jako v případě Ox Enginu. Situaci lze řešit změnou nastavení v ovládacím panelu systému Windows - Místní a jazyková nastavení.

3.3.4 Používání a dokumentace

Architektura enginu je tradiční. Je založena na komponentech, které lze libovolně přidávat, nebo odebírat ze scény podle aktuální potřeby. Scéna, renderer, systém fyziky, grafické zařízení a další, jsou sdruženy v objektu třídy XMain. Ten provádí pravidelnou aktualizaci objektů ve scéně a vykreslování prostřednictvím objektu rendereru. Použití je velice jednoduché a intuitivní. Při pohledu do zdrojových kódů přiložené hry to není na první pohled patrné, protože se jedná o rozsáhlou a složitě strukturovanou ukázku. Pro vytvoření základní aplikace je zapotřebí jen minimální úsilí. Také knihovna JigLibX je velice nenásilně integrována do architektury. Její použití nevyžaduje žádné složité konstrukce. V balíku s enginem je k dispozici fyzikální hra s kuličkou, která ukazuje možnosti použití. Bohužel uvnitř kódu hry nejsou žádné komentáře. Zdrojové kódy enginu jsou také, až na několik světlých výjimek, prosty jakýchkoliv komentářů. Naštěstí díky jednodušší architektuře není situace tak tragická, jako v případě Ox Enginu. Na stránkách projektu v historii stránek lze dohledat několik tutoriálů a popis základních tříd. Nejsou ovšem aktuální a velká část z nich už nekoresponduje se současným stavem enginu.

3.3.5 Licence

X-Engine je šířen pod licencí Microsoft Public License (Ms-PL). Více o této licenci viz. 3.1.5.

3.3.6 Shrnutí

Celkový dojem z X-Enginu kazí neokomentované zdrojové kódy a neaktuální dokumentace. V porovnání s Ox Enginem postrádá funkční animaci 3D modelů a podporu stínů. Také systém osvětlení není úplně dořešený. Editor scény v aktuální verzi chybí, přestože v předchozí přítomen byl.

Na druhé straně se proti Ox Enginu mnohem snadněji používá, má přehlednější strukturu, propracovanější vodní hladinu i SkyDome s nastavitelnou fází dne.

X-Engine v aktuální verzi má své chyby a nedostatky, ovšem z hlediska praktické použitelnosti je na tom v porovnání s dvěma předchozími konkurenty dobře.

3.4 BetaCell

3.4.1 Úvod a historie

BetaCell je software pro usnadnění programování počítačové grafiky. Projekt sídlí na vlastních internetových stránkách na adrese www.betacell3d.com. Přímo na úvodní stránce se nachází přehled hlavních vlastností, který zaujme na první pohled. Hlavním cílem je nabídnout co nejefektivnější způsob programování bez neustále se opakujících úseků kódu.

Toolkit je určen pro XNA Game Studio 2.0. Byl vydán 20. června 2008 ve verzi nekompatibilní s Xbox 360. Verze s podporou Xbox 360 vyšla 6. července 2008.

Na rozdíl od enginů kterým jsme se zde věnovali, není BetaCell opensource projekt. Nelze tak počítat s častým vydáváním nových verzí a překotnými změnami. Je třeba se spokojit se stávajícími funkcemi, které by ale měli být v dokončeném stavu.[17]

3.4.2 Funkce

Následuje stručný přehled funkcí BetaCell toolkitu.

- Procedurální tvorba grafických primitiv.
- Systém osvětlení s širokými možnostmi nastavení.
- Flexibilní systém dynamické kompozice efektů za běhu programu. Krom jednoduchého obarvení, texturování, nebo osvětlení lze dosáhnout i pokročilejších efektů.
- Podpora modelů ve formátu .bcm a animací ve formátu .bca. Jedná se o proprietární formáty určené pro použití v BetaCellu. Modely v tomto formátu lze tvořit pomocí 3D grafického editoru Blender. Podporu pro tyto formáty lze do Blenderu přidat prostřednictvím exportérů napsaných ve skriptovacím jazyce Python. Je možné je stáhnout přímo na stránkách www.betacell3d.com.

- Generování terénu z výškové mapy. Terén lze rozdělit na části a provádět quadtree ořezávání. Je možné využít multitexturingu.
- Systém pro usnadnění práce s kamerou.
- Funkce výběru objektu pomocí míření, podporuje i animované modely.
- Systém pro jednodušší tvorbu částicových efektů.
- Efekt lesklého povrchu prostřednictvím mapování cube mapy.
- Efekt poloprůhledné vodní hladiny s odrazem.
- Podpora normálového mapování.
- Podpora projekčního texturování.
- Zobrazení stínů technikou shadow mapping.

3.4.3 Instalace

BetaCell se šíří ve formě .dll knihoven a dalšího obsahu, který je součástí startovacího projektu. Ten je možné stáhnout ze stránek www.betacell3d.com ve verzi pro Windows, nebo Xbox 360.

Stáhněte archiv se startovacím projektem ve verzi pro požadovanou platformu. Archiv rozbalte a otevřete soubor starter_windows.sln (v případě verze pro Windows). Projekt by mělo být možné okamžitě přeložit. Po spuštění se zobrazí prázdné okno s ukazatelem počtu snímků za vteřinu.

3.4.4 Používání a dokumentace

Ve způsobu použití se BetaCell od ostatních enginů liší. Svou koncepcí se skutečně jedná hlavně o framework pro snadnější dosažení širokého spektra různých grafických efektů.

Používání celého toolkitu je postaveno na systému dynamické kompozice efektů. Při návrhu bylo využito návrhového vzoru Visitor. Pokud chceme například obarvit model určitou barvou, vytvoříme objekt "návštěvníka". Ten zajistí, že model bude příslušně obarven v případě, že nad modelem zavoláme metodu visit() a předáme objekt návštěvníka jako parametr. Návštěvníky lze různě kombinovat. Jejich prostřednictvím dojde k sestavení požadovaného efektu.

Jde o velice flexibilní systém, který umožňuje vytvářet efekty jednodušeji a bez nutnosti je programovat ručně, například pomocí HLSL.

Jak už bylo zmíněno v úvodu, zdrojové kódy nejsou otevřené. Avšak na stránkách www.betacell3d.com je k dispozici programová dokumentace a velké množství velice propracovaných tutoriálů. Tutoriály se často kromě popisu použití toolkitu věnují i základním principům popisovaných efektů.

3.4.5 Licence

BetaCell lze používat zdarma po odsouhlasení licenční smlouvy, která je k dispozici na stránkách www.betacell3d.com.

3.4.6 Shrnutí

BetaCell zaujme především svým systémem dynamické kompozice efektů. S jeho použitím odpadá nutnost zabývat se programováním efektů v HLSL. Také dostupné tutoriály jsou svoji kvalitou na vysoké úrovni.

Stinnou stránkou jsou uzavřené zdrojové kódy a také chybějící podpora práce s formáty souborů .x a .fbx, které jsou ve světě XNA standardem.

Dále je třeba mít na paměti, že BetaCell je primárně sadou nástrojů k snazšímu dosažení širokého spektra grafických efektů. Postupy pro dosažení některých z nich nejsou zcela přímočaré a vyžadují více úsilí než v případě klasického herního enginu.

3.5 Innovative Engine

3.5.1 Úvod a historie

Innovative Engine staví na základech X-Enginu a oba projekty jsou dílem jednoho autora. Ten se rozhodl zastavit vývoj X-Enginu a převést většinu jeho funkcí do nového Innovative Enginu. Jako důvody autor uvádí nízkou kvalitu kódu a nespokojenost s návrhem celého projektu.[16]

Nový projekt sídlí na stránkách http://www.innovativegames.net. Engin je prezentován formou tutoriálů, z nichž každý se zabývá některou jeho částí. Návštěvník stránek se tak může do nejmenších podrobností seznámit s fungováním celé aplikace.

Nový engin zatím nenabízí všechny funkce, které poskytoval jeho předchůdce. První tutoriál vyšel 9. října 2008 a k 25. březnu 2009 jich vyšlo celkem dvanáct.

3.5.2 Funkce

Následuje stručný přehled funkcí Innovative Enginu.

- Podpora 3D souborů ve formátu .x .fbx.

- Fyzika prostřednictvím JigLibXPhysics Libary.
- Generování terénu z výškové mapy.
- Třída pro snazší práci s kamerou.

3.5.3 Instalace

Na konci většiny tutoriálů se nachází odkaz na archiv obsahující projekt enginu a ukázkový projekt ověřující funkci nového kódu. Stáhněte tento archiv a otevřete soubor InnovationEngine.sln. Kód všech tutoriálů je určen pro XNA Game Studio 3.0 a některou z verzí Visual Studio 2008 nebo Visual C# 2008.

3.5.4 Používání a dokumentace

Návrhem se Innovative Engine podobá X-Enginu, jeho návrh však klade větší důraz na jednoduchost použití a snadnou rozšiřitelnost.

Důležité objekty jsou sdružovány ve třídě Engine obsahující také kontejnery komponentů. Engin automaticky provádí aktualizaci a vykreslování těchto komponentů.

Zajímavým prvkem je systém GameScreen objektů. GameScreen objekty jsou ve své podstatě kontejnery sdružující objekty scény (komponenty). Engine pak s nimi dokáže dále pracovat podle uživatelem definovaných pravidel. Tento systém je možné výhodně využít pro tvorbu různých nabídek, nebo inventářových obrazovek a jejich oddělení od hlavní obrazovky, kde probíhá samotná hra.

Pro zobrazení 3D modelů se používá objekt třídy Actor, který umožní zobrazení modelu a definici jeho fyzikálních vlastností a provázání s fyzikálním enginem.

V oblasti dokumentace je na tom engin velice dobře a to především díky tutoriálům, ve kterých najdete popis všech jeho součástí. Zdrojový kód je ze stejného důvodu velice pečlivě komentován, proto není těžké se v něm zorientovat.

Samotné použití enginu je velice jednoduché. Stačí provést inicializaci a přidat základní služby pro kameru a fyziku. Pak zbývá jen pravidelně volat jeho metody Update() a Draw() v příslušných metodách třídy Game1[1].

[1]Game1 je implicitní pojmenování hlavní třídy každé hry vytvořené pomocí XNA Game Studia 3.0.

3.5.5 Licence

V oblasti licence je situace kolem Innovative Enginu nejasná. Zdrojové kódy jsou doplňkem k veřejně přístupným tutoriálům. Je možné je stahovat bez omezení a jejich licence není blíže specifikována. Autorská práva na zdrojové kódy, stejně jako na veškerý další obsah stránek www.innovativegames.net si vyhrazuje jejich autor v patičce webové prezentace.

3.5.6 Shrnutí

Největší slabinou Innovative Enginu je malé množství funkcí v porovnání s konkurenty. Tento projekt se však vydal odlišnou cestou. Jeho vývoj je dokumentován pomocí tutoriálů. Umožňuje tak zájemcům hlouběji porozumět jeho struktuře a maximálně tak uživateli zjednodušit rozšíření o komponenty, které mu v enginu chybí.

Tvorba vlastního komponentu není zbytečně komplikovaná, většinu potřebného kódu lze získat oddděním od třídy Component a vlastní funkcionalitu dodat jednoduše překrytím metod Draw() a Update().

Všechny potřebné informace pro použití enginu lze nalézt v tutoriálech, jejichž součástí jsou v tomto ohledu velice užitečné ukázkové aplikace pro ověření funkčnosti nového kódu. Samotné použití je jednoduché a intuitivní. Navíc jsou veškeré zdrojové kódy pečlivě komentovány a případné chybějící informace v nich lze snadno dohledat.

Celkový dojem z enginu je spíše pozitivní, ale při jeho použití je nutné počítat s tím, že bude velmi pravděpodobně nutné některé věci doprogramovat. V současné době mu stále chybí některé důležité funkce, které bude mnoho uživatelů postrádat.

3.6 Porovnání enginů

3.6.1 Srovnávací tabulka

Funkce	Ox	QS	X-E	BC	Inn
Terén z výškové mapy	90%	95%	85%	85%	65%
Procedurální grafická primitiva	50%	0%	0%	90%	0%
Fyzika	75%	75%	75%	0%	75%
Vodní hladina	50%	0%	80%	75%	0%
Částicové efekty	60%	0%	85%	80%	0%
Systém osvětlení	85%	55%	30%	85%	0%
Shadow mapping	60%	0%	0%	60%	0%
Normal mapping	0%	0%	0%	90%	0%
SkyBox/SkyDome	60%	0%	95%	60%	0%
Environmental mapping	0%	0%	0%	90%	0%
Quad/OcTree ořezávání	80%	80%	80%	60%	0%
Animace modelů	90%	0%	0%	80%	0%
Dokumentace, komentáře	5%	80%	20%	99%	90%
Průměr	54%	30%	42%	73%	18%

vysvětlivky:

Ox Ox Engine

QS Quick Start

X-E X-Engine

BC BetaCell

Inn Innovative Engine

Tabulka srovnává schopnosti enginů ve vybraných oblastech. To jak si engin vede v určité oblasti je vyjádřeno procentuálně v porovnání s ideálním stavem. Jinými slovy, pokud engin v některé oblasti exceluje a není mu co vytknout, obdrží ohodnocení 100%. Ze všech hodnot je následně vypočten aritmetický průměr.

3.6.2 Vyhodnocení

Srovnání enginů z hlediska nabízených funkcí dopadlo následovně:

1. BetaCell - 73%
2. Ox Engine - 54%

3. X-Engine - 42%
4. Quick Start - 30%
5. Innovative Engine - 18%

Vítězem srovnání se stal BetaCell toolkit. Umožňuje dosáhnout širokého spektra grafických efektů. K dosažení některých z nich je však nutné vyvinout větší úsilí, které je cenou za vysokou variabilitu. Použití usnadňuje množství kvalitních tutoriálů a programová dokumentace. Za nedostatek lze pokládat použití vlastního formátu pro modely a animace. Na rozdíl od ostatních enginů nejsou zdrojové kódy BetaCell toolkitu otevřené. Tento fakt velmi znesnadňuje případné úpravy a rozšíření.

Druhé místo obsadil Ox Engine. Jeho podstatnou nevýhodou je však značně nepřehledné vnitřní uspořádání umocněné nedostatečným komentováním zdrojových kódů. Použití je těžkopádné, navíc není dostupná žádná dokumentace. Pouze několik ukázkových aplikací. Celkový dojem je tak spíše negativní a tento engin nelze ve stávající verzi doporučit.

X-Engine nabízí menší množství funkcí než Ox Engine, je však na rozdíl od něj snadno použitelný, jeho zdrojové kódy jsou přehlednější, bohužel stejně špatně komentované. Ani X-Engine nedisponuje žádnou použitelnou dokumentací. Zaujme především dobrou integrací knihoven pro fyziku, kvalitní implementací vodní hladiny a oblohou umožňující plynule měnit fázi dne. Vývoj tohoto enginu byl však zastaven a nelze očekávat další jeho rozšíření.

Čtvrté místo obsadil Quick Start Engine. Je zajímavý především svojí architekturou založenou na entitách, která umožňuje snadné použití a současně nabízí velikou flexibilitu. Zdrojové kódy jsou velice dobře komentované. Dostupná je i dokumentace, která prezentuje základní principy použití. Nevýhodou je zatím jen velice omezené množství funkcí způsobené odstartováním vývoje znovu zcela od začátku. Jde o projekt s velikým potenciálem, který má šanci stát se v budoucnu jedničkou na poli volně dostupných enginů pro XNA.

Na posledním místě se umístil Innovative Engine, který je nástupcem X-Enginu. Množstvím nabízených funkcí neohromí. Výhodou je ale snadné intuitivní použití. Díky tutoriálům a dobře komentovanému kódu je tento engin velice vhodným kandidátem pro další rozšíření.

4 Rozšíření Innovative enginu

Následující text se věnuje tvorbě komponent z oblasti modelování počasí pro Innovative Engine. V první části je popisován pricip fungování a způsob implementace dynamické oblohy, umožňující plynulou změnu fáze dne.

Druhá je věnována metodě generování dynamické vrstvy mraků s využitím GPU. Poslední třetí část popisuje způsob tvorby částicového efektu deště, reagujícího na směr a sílu větru.

4.1 Dynamická obloha

Dominantní barvou oblohy je světle modrá. Konkrétní barvy a odstíny jsou závislé na mnoha okolnostech. Zásadní roli ovšem hraje poloha Slunce. Barva je výsledkem rozptylu světla v atmosféře a z toho důvodu je závislá na její hustotě, tloušťce i složení. Svoji roli hraje i množství páry a prachových částic.[19] Výsledkem mohou být například barvy zachycené na Obrázku 1. Je vidět, že barvy shora směrem k horizontu vytvářejí barevný přechod, gradient. Na gradientech je založena i metoda, kterou popsal Jesús Alonso Abad ve své práci A fast, simple method to render sky color using gradient maps.[18] Z této metody vychází i zde popisované řešení.

Obrázek 1: fotografie stromu s bezmračnou oblohou v pozadí, zdroj: www.freefoto.com

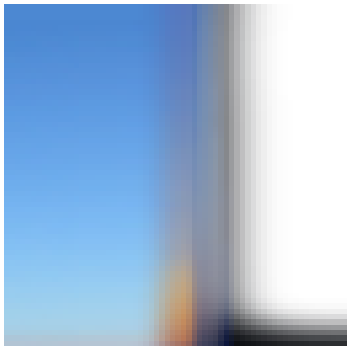
4.1.1 Textura s gradienty

Základní ideou metody je mapování barev textury s gradienty na kupoli tvořenou vrchní polovinou sféry. Mapování je prováděno základě úhlové výšky bodu a denní době, jak popisuje následující funkce:

$$f(a,t) = (r,g,b) \tag{1}$$

V XNA je mapování textury na geometrii definováno pomocí textu-rovacích koordinátů u a v. Tyto koordináty jsou souřadnice udávající polohu uvnitř textury. Barva odpovídající této poloze je použita jako výsledná barva bodu. Koordinát u tvoří horizontální a v vertikální souřadnici. Za předpokladu, že

jako koordinát u použijeme denní dobu a jako koordinát v úhlovou výšku bodu, bude textura vypadat jako na Obrázku 2. Textura na obrázku je součástí projektu Caelum, který rozšířením enginu Ogre3D o zobrazování atmosferických efektů.[20]



Obrázek 2: textura s gradienty použitá v projektu Caelum

Gradient pro jednu denní dobu má konstantní hodnotu v a hodnota u je dána úhlovou výškou. První gradient zleva odpovídá poledni, kdy je Slunce nejvýše.

Tyto gradienty lze vygenerovat pomocí některé z komplexních metod pro simulaci barev oblohy, nebo je vytvořit ručně, případně na základě fotografických podkladů.

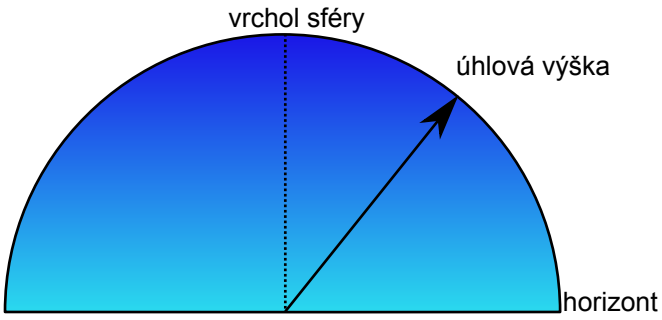
Velikost použité textury je 64x64 bodů, ačkoliv už s velikostí 16x16 lze dosáhnout přijatelných výsledků. Vyšší rozlišení však umožní plynulejší přechody při změně denní doby.[18]

4.1.2 Geometrie oblohy

Jako geometrii oblohy lze použít sféru, nebo její vrchní polovinu. Celá sféra je výhodná především v situacích kdy se lze dostat na okraj terénu. Kamera bude vždy umístěna v jejím středu. Sféra musí být vykreslována vždy jako první s vypnutou pamětí hloubky, aby nedošlo k překrytí jiných objektů.

Požadavkem metody je, že sféra musí mít konstantní hodnotu koordinátu u a hodnoty koordinátu v musí odpovídat úhlové výšce vztažené k jejímu vrcholu. Tím je zajištěno správné mapování barev jako na obr. 3 a jednoduchá změna denní doby modifikací u koordinátu.[18]

Zde popisované řešení používá model sféry vytvořený v 3D modelovacím programu exportovaný do formátu .x (DirectX). Tento formát je podporován v XNA Content Pipeline. Model obsahuje standardní sférické texturovací koordináty a normály vypočítané modelovacím programem. Využití normál je popsáno dále v tomto textu.



Obrázek 3: mapování barev gradientu

Koordináty použitého modelu zcela nesplňují výše uvedené požadavky. Problém lze velmi snadno obejít využitím možností programovatelných grafických akcelerátorů a jazyka HLSL (High Level Shader Language) pro jejich programování. Programovatelným rozhraním je vybavena drtivá většina dnes používaných grafických akcelerátorů a jeho přítomnost je jedním z minimálních hardwarových požadavků pro provoz XNA aplikací (viz. 2.4). Pomocí HLSL lze programovat činnost grafické karty. Kromě transformačních matic a matic zobrazení lze definovat další hodnoty a parametry, které pak mohou být prostřednictvím XNA programu odeslány grafickému akcelerátoru a jím využity k výpočtu výsledné pozice a barvy bodů. Tyto zdrojové kódy mají příponu .fx. Obsahují kód Pixel Shader a Vertex Shader programů. Uvnitř Vertex Shader programu jsou zpracovávány body (vertexy) zobrazované scény. Pixel Shader program zpracovává body výsledného obrazu (pixely).

4.1.3 Mapování gradientů na geometrii oblohy

Jako hodnota koordinátu u je použita proměnná Time reprezentující denní dobu. Ta je spolu s odpovídajícím vektorem směru slunečního záření předána z vnějšku a je pro všechny body v rámci jednoho snímku konstantní. Tím je splněn požadavek metody a zároveň umožněno nastavení denní doby pomocí této proměnné. Dalším krokem je správné nastavení sampleru. Sampler využívá grafický akcelerátor k přístupu k hodnotám uloženým v textuře. Konkrétně nás teď zajimá parametr addressu. Tento parametr definuje, jakou hodnotu textury použít pokud je hodnota koordinátu u mimo rozsah <0,1>. Nastavením parametru na hodnotu mirror zajistíme plynulý průběh denního cyklu. Po opuštění rozsahu dojde k zrcadlovému opakování hodnot.

Barevný přechod bude použit na horní polovinu sféry. Její spodní část bude obarvena poslední barvou přechodu. Roztažení poslední barvy na okraji textury pro hodnoty koordinátu v mimo rozsah <0,1> docílíme nastavením

parametru addressv na hodnotu clamp.

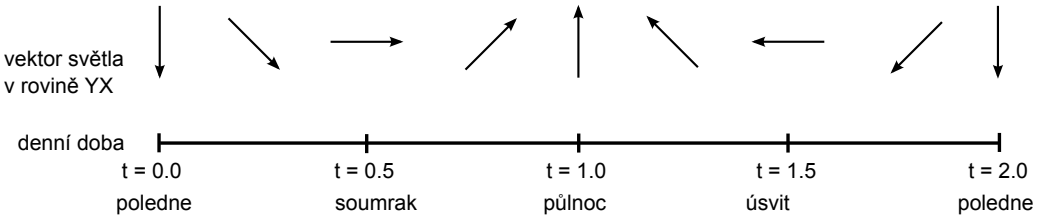
```
// nastavení způsobu práce s texturou a koordináty
texture Texture;
sampler sTexture = sampler_state {
  texture = (Texture);
  addressu = mirror;
  addressv = clamp;
  ...
};
```

Rozsah koordinátu v pro všechny body modelu spadá do rozsahu <0,1>. Pro správné umístění gradientu na horní polovinu sféry je třeba koordináty v vynásobit dvěma. Hodnoty spodní poloviny se posunou mimo rozsah a všechny její body budou obarveny barvou na spodním okraji přechodu.

```
// výpočet koordinátů
float2 TexCoord = float2(Time, input.TexCoord.y * 2.0f);
float4 outputColor = tex2D(sTexture, TexCoord);
```

4.1.4 Slunce a úpravy barevnosti

Zobrazení Slunce je jedním z prvků, které využívají přítomnost normálových vektorů v modelu sféry. Spolu s denní dobou v proměnné Time je HLSL programu předáván odpovídající směrový vektor slunečního světla. Ten je vždy vypočítán rotací vektoru (0,-1,0) v ose Z o úhel daný denní dobou viz. obr. 4. Pohyb Slunce je zjednodušený, nezávislý na ročním období a konkrétní poloze na zemském povrchu. Putuje vždy středem oblohy od východu na západ[2].



Obrázek 4: směrový vektor slunečního světla v závislosti na denní době

Pomocí směrového vektoru světla a normálových vektorů sféry je možné zesvětlit body oblohy na místě, kde se má nacházet Slunce. Skalárním součinem normály a směrového vektoru je získán kosinus úhlu, který svírají[3]. Pro normály se shodným nebo podobným směrem je tato hodnota vyšší. Pokud o tuto hodnotu vynásobenou vhodnou vahou zesvětlíme výslednou barvu

[2]Severní směr je stanoven ve směru záporné osy Z.

[3]Oba vektory jsou před výpočtem skalárního součinu normalizovány.

bodu, na obloze bude patrný světlý kruh se středem v místě polohy slunce. Nejsvětlejší bude ve svém středu, dále od něj se bude světlost plynule snižovat. Ostrost kruhu lze ovlivnit umocněním výsledku skalárního součinu. Tím dojde k potlačení menších hodnot na okrajích a zostření přechodu. V níže uvedeném zdrojovém kódu jsou použity dva kruhy. Jeden s nižší mocninou a tedy s větším průměrem a méně ostrým přechodem pro zesvětlení barev oblohy v okolí Slunce. Druhý s vyšší mocninou pro vytvoření menšího a ostřejšího slunečního kotouče.

Normály bodů na (od Slunce) odvrácené straně sféry produkují hodnoty záporné. Barvy na této polovině jsou tmavší a částečně desaturované, protože paprsky světla osvětlující tuto část oblohy musí urazit delší cestu atmosférou a ztrácí tak část své intenzity.[18] Ztmavení barev se principiálně neliší od zesvětlení popisovaného v předchozím odstavci. Částečné desaturace je dosaženo lineární interpolací původní barvy a barvy po úplné desaturaci, kde poměr je dán mocninou skalárního součinu směrového vektoru světla a normálového vektoru sféry. Barvu po úplné desaturaci stanovíme výpočtem jasu původní barvy pomocí rovnice:

$$I = 0.299R + 0.578G + 0.114B \tag{2}$$

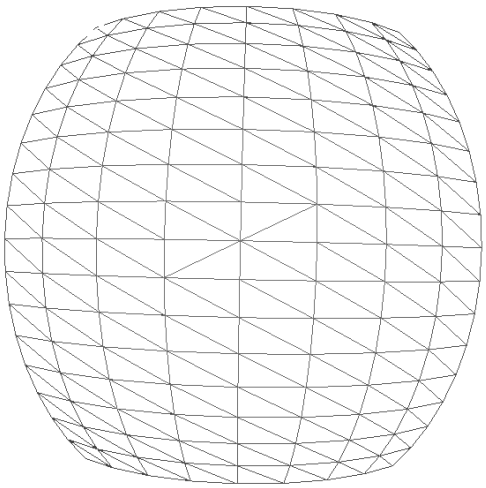
kde R, G a B jsou složky barvy a I hodnota jasu.[21] Pro názornost je uveden fragment zdrojového kódu zachycující výše popisované postupy.

```
// výpočet sklárního součinu
float dotproduct = dot(LightDir, -normalize(input.Normal));
float sunglow = pow(dotproduct, 2) * 0.2f;
if (dotproduct > 0) // přivrácená strana
{
  // zesvětlení kolem Slunce + Slunce
  float sun = pow(dotproduct, 128) * 0.6f;
  outputColor.rgb += sun + sunglow;
}
else // odvrácená strana
{
  // výpočet intenzity
  float intensity = 0.299f * outputColor.r + 0.578f *
      outputColor.g + 0.114f * outputColor.b;
  // ztmavení
  outputColor.rgb -= sunglow;
  // částečná desaturace
  outputColor = lerp(outputColor, float4(intensity, intensity
      , intensity, outputColor.a), sunglow);
}
```

4.1.5 Noční obloha a hvězdy

Při použití textury s gradienty jako na obr. 2, kde barvy obsahují i informaci o průhlednosti, lze dosáhnout efektu zvyšování průhlednosti oblohy s ubýváním denního světla. Průhlednost barev se s příchodem noci zvyšuje. Tato vlastnost umožňuje zobrazení objektů na noční obloze s tím, že jejich viditelnost se bude závislá na denní době.

Na noční obloze je zobrazována textura s hvězdami. Ta je mapována na vlastní geometrii, která je výřezem sféry (viz obr. 5). Vykreslována je dříve než geometrie oblohy, jejíž průhlednost se během dne mění.



Obrázek 5: část sféry pro mapování textury hvězd

XNA lze s průhledností pracovat nejrůznějšími způsoby. V tomto konkrétním případě je barva právě vykreslovaného pixelu vynásobena hodnotou průhlednosti a přičtena k původní barvě vynásobené inverzní hodnotou průhlednosti. Tento způsob práce s průhledností lze v Innovative Enginu nastavit pomocí následujícího kódu.

```
Engine.GraphicsDevice.RenderState.AlphaBlendEnable = true;
Engine.GraphicsDevice.RenderState.SourceBlend = Blend.
    SourceAlpha;
Engine.GraphicsDevice.RenderState.DestinationBlend = Blend.
    InverseSourceAlpha;
```

Pohyb hvězd po obloze je realizován posuvem texturových koordinátů. Jedná se o pohyb zjednodušený, stejně jako je tomu u pohybu Slunce.

4.1.6 Osvětlení v závislosti na poloze Slunce

Slabinou použitého enginu je absence systému osvětlení. Osvětlení je nastaveno pevně uvnitř zdrojového kódu a není umožněno je měnit za běhu. Proto bylo nutné provést úpravy komponentu pro zobrazení terénu tak, aby jeho osvětlení mohlo reagovat na pozici Slunce.

4.2 Dynamické mraky

Základem dynamických mraků je metoda pro generování šumové textury převzatá z tutoriálu 2D Perlin noise na stránkách Riemer's XNA Tutorials.[22] Ta umožňuje generovat texturu pomocí Perlinova šumu v reálném čase s využitím výkonu grafického akcelerátoru. Mraky jsou tvořeny z poloprůhledných vrstev tvořených barevnou texturou, jejichž hodnoty průhlednosti v jednotlivých bodech jsou dány šumovou texturou. Skupina vrstev ve vhodné vzdálenosti vytváří dojem prostorovosti, který je dále umocněn stínováním. Horní vrstvy mraků vrhají stín na vrstvy pod nimi, což se projevuje tmavším odstínem jejich barvy.

4.2.1 Generování textury průhlednosti

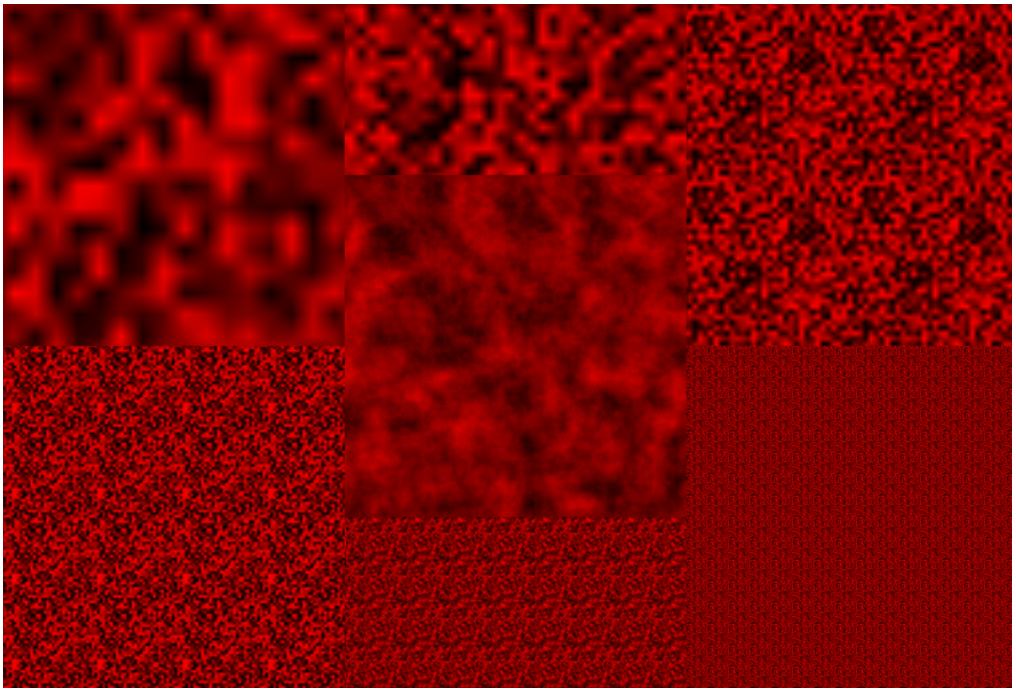
V prvním kroku je vygenerována textura o velikosti 32x32 pixelů náhodného šumu pomocí generátoru náhodných čísel. Z ní je pak za využití GPU skládána šumová textura velikosti 512x512 pixelů. Postup skládání nejlépe zachycuje následující část zdrojového kódu Pixel Shaderu.

```
float4 cloudColor = tex2D(sTextureClouds, input.TexCoord)/2;
cloudColor += tex2D(sTextureClouds, (input.TexCoord)*2)/4;
cloudColor += tex2D(sTextureClouds, (input.TexCoord)*4)/8;
cloudColor += tex2D(sTextureClouds, (input.TexCoord)*8)/16;
cloudColor += tex2D(sTextureClouds, (input.TexCoord)*16)/32;
cloudColor += tex2D(sTextureClouds, (input.TexCoord)*32)/32;
```

Barva výsledného pixelu je součtem šumové textury různých velikostí a vah. Jako první je použita textura v původní velikosti s nejvyšší vahou. Poté je přičtena stejná textura zmenšená pomocí vynásobení texturových koordinátů s poloviční vahou atd. Součet všech vah je roven jedné. Výsledkem je textura Perlinova šumu, která je základem textury mraků. Proces skládání šumové textury je zachycen na obr. 6.

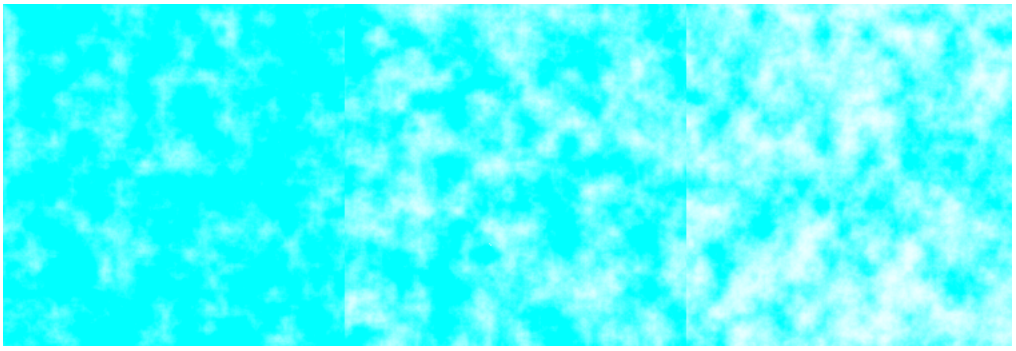
Textura průhlednosti mraků je pak vytvořena úpravou hodnot, která je zachycena následujícím kódem.

```
cloudColor = 1.0f - pow(cloudColor, Overcast) * 2.0f;
```



Obrázek 6: skládání šumové textury, výsledek je uprostřed

Pomocí hodnoty v proměnné Overcast lze regulovat hustotu (neprůhlednost) mraků na textuře. Zároveň je dosaženo vyšší ostrosti a oddělení mraků. Vliv úpravy pro různé hodnoty Overcast ukazuje obr. 7.



Obrázek 7: textura průhlednosti mraku pro hodnoty Overcast 1 (vlevo), 1.5 (uprostřed) a 2 (vpravo)

Pohyb mraků lze realizovat klasicky pomocí posuvu texturových koordinátů. Ovšem reálné mraky kromě pohybu i mění svůj tvar. Podobného efektu lze dosáhnout posuvem texturových koordinátů nikoliv výsledné textury, ale textury náhodného šumu při jejím skládání. Změna tvaru je způsobena tím, že vrstvy jsou posouvány různými rychlostmi jak naznačuje následující kód.

```
cloudColor = tex2D(sTextureClouds, input.TexCoord + Move)/2;
cloudColor += tex2D(sTextureClouds, (input.TexCoord)*2 + Move
   )/4;
...
```

Hodnota obsažená v proměnné Move vyjadřuje směr a vzdálenost pohybu mraků od posledního snímku. Jedná se o 2D vektor.

4.2.2 Reprezentace vrstvy mraků

Nyní známe hodnoty průhlednosti vrstvy mraků, dále ke nutné určit její barvy. Pro jednobarevnou vrstvu je situace jednoduchá, stačí předat programu Pixel Shaderu požadovanou barvu a texturu průhlednosti. Situace se zkomplikuje, pokud budeme požadovat vícebarevnou vrstvu kvůli realizaci stínování. V tomto případě bude nutné barvy uchovávat pomocí další textury. Do ní je vhodné uložit i data o průhlednosti.

4.2.3 Metoda stínování

Úkolem metody je vytvoření textur pro vrstvy mraků. Průhlednost vrstev je známa, proto je hlavním účelem zjištění její barevnosti. Metoda uvažuje rovnoběžné sluneční paprsky jejichž směr je určen vektorem a vrstvy tvořené rovinami rovnoběžnými s rovinou XZ protínající osu Y v ekvidistantních bodech.

Vrchní vrstva[4] pohlcuje část slunečního světla. To se projeví tmavším odstínem barvy nižší vrstvy v místě, kde dochází k zastínění. O kolik bude barva nižší vrstvy tmavší, záleží především na hodnotě průhlednosti vyšší vrstvy a na vzdálenosti, kterou by mezi nimi musel paprsek urazit.

Délka cesty paprsků se mění v závislosti na jejich směru. Nejkratší je, pokud dopadají kolmo na vrstvu. Pak je délka cesty paprsku rovna vzdálenosti mezi vrstvami.

Dále je třeba určit dvojice bodů stínící-stíněný, kde stínící bod je z vyšší vrstvy a je určující pro barvu bodu stíněného, který patří do vrstvy nižší. Určujícím faktorem bude směrový vektor světla, který vzniká rotací vektoru (0,-1,0) kolem osy Z (viz. 4.1.4). Je tedy za všech okolností rovnoběžný s rovinou XY. Pokud bude geometrie vrstvy ve scéně vhodně umístěna tak, že osy texturových souřadnic u a v budou rovnoběžné s osami X a Y, stínící a stínění bod budou mít stejnou hodnotu koordinátu v a budou se lišit[5]

[4] Vrchní vrstvou se v tomto případě rozumí vrstva s nejvyšší hodnotou souřadnice Y.

[5] V případě, že bude paprsek dopadat na vrstvy kolmo, budou i hodnoty koordinátu u obou bodů shodné.

hodnotou koordinátu u o hodnotu závislou na směrovém vektoru světla. Koordinát u se bude u všech dvojic lišit o stejnou hodnotu, stejně tak bude pro všechny dvojice stejná délka cesty paprsku. Proto stačí vypočítat tyto hodnoty pro každý snímek pouze jednou a předat je programu Pixel Shaderu jako parametry.

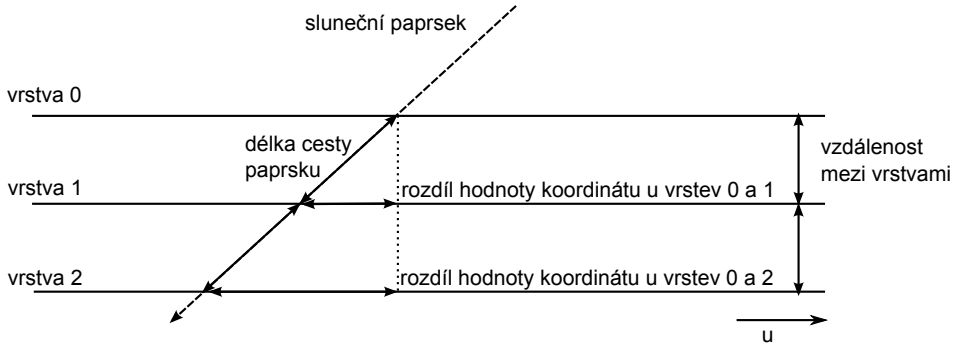
Kód Pixel Shaderu využívá texturu předchozí vrstvy a texturu průhlednosti. V případě první vrstvy, je místo předchozí textury k dispozici jen barva, která je použita pro všechny její body. Texturové koordináty textury předchozí vrstvy se posouvají o hodnotu vypočítanou na základě směrového vektoru světla. S jejich pomocí lze určit parametry stíněného bodu před úpravou. Dále je třeba znát průhlednost stínícího bodu. Ta je určována pomocí nezměněných texturových koordinátů z textury průhlednosti. Z hodnoty průhlednosti stínícího bodu a hodnoty reprezentující útlum světla úměrný délce paprsku je vypočtena hodnota, která je následně odečítána od všech složek barvy stíněného bodu. Tím je určena výsledná barva textury příslušné vrstvy. Textury jsou renderovány postupně, pro správnou funkci musí být vždy k dispozici textura předchozí vrstvy. Všechny textury jsou uchovávány v poli a později namapovány na geometrii ve scéně.

Zdrojový kód Pixel Shaderu, který je uveden níže, je ve výsledku jednoduchý, protože potřebné výpočty nejsou prováděny pro každý bod zvlášť. Provedeny jsou pouze jednou v každém snímku pro všechny body vrstev. Výsledky jsou předávány jako parametry.

```
// výpočet posunu texturových koordinátů
float2 TexCoordShift = float2(LayerIndex * Shift, 0);
// stínící bod
float4 textureColor = tex2D(sTextureClouds, input.TexCoord);
float4 outputColor;
if (LayerIndex == 0) // první vrstva
{
  outputColor =  float4(CloudColor, textureColor.r);
}
else // ostatní vrstvy
{
  // stíněný bod
  float4 prevLayerColor = tex2D(sPrevLayerTexture, input.
     TexCoord - TexCoordShift);
  // o kolik bude stíněný bod tmavší
  float decrement = float(textureColor.r * RayLenght);
  // ztmavení stíněného bodu
  outputColor = float4(prevLayerColor.r - decrement,
     prevLayerColor.g - decrement, prevLayerColor.b -
     decrement, textureColor.r);
}
```

4.2.4 Výpočty parametrů metody stínování

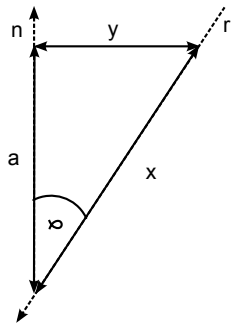
Velikost rozdílu koordinátů v stínících a stíněných bodů a délka cesty paprsku mezi vrstvami (viz. obr. 8) jsou důležitými parametry metody stínování. Jejich výpočet je prováděn při vykreslování každého snímku.



Obrázek 8: rozdíl hodnoty koordinátu u a délka cesty paprsku mezi vrstvami

Prvním krokem zjištění kosinu úhlu α, který paprsek svírá s normálou vrstvy, pomocí skalárního součinu. Z obr. 9 je patrné, že normalizované vektory r a n platí:

$$cos\alpha = -\vec{r}.\vec{n} \tag{3}$$



Obrázek 9: vyjádření matematické závislosti parametrů na směrovém vektoru světla r a vzdálenosti mezi vrstvami a

Pro délku paprsku x platí:

$$x = \frac{a}{cos\alpha} \tag{4}$$

kde a značí vzdálenost mezi vrstvami. Velikost rozdílu koordinátů y je dána vztahem:

$$y = \sqrt{x^2 - a^2} \tag{5}$$

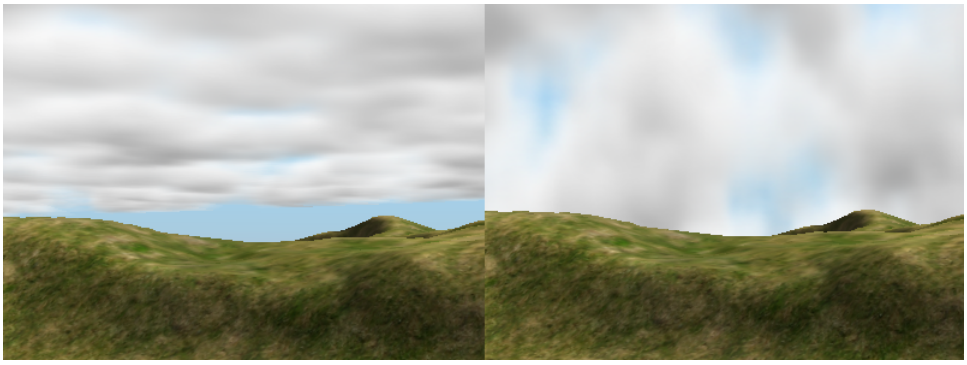
U této hodnoty je nutné určovat směr posunu. K tomu účelu je využito skalárního součinu vektoru paprsku -r s vektorem (1, 0, 0). Pokud je jeho hodnota záporná, jsou koordináty posouvány v kladném směru a naopak.

Dále je třeba určit hodnotu reprezentující útlum světla. Hodnota je počítána jako kvadratická funkce délky paprsku x. Během testování bylo dosaženo dobrých výsledků s funkcí:

$$lightAbsorption = 0.05x^2 + 0.05 \tag{6}$$

4.2.5 Geometrie vrstev

Metoda stínování je prováděna nezávisle na geometrii na kterou je nakonec textura vrstvy namapována. Počítá s vrstvami tvořenými čtverci rovnoběžnými s rovinou XZ. Nabízí se tedy použití vrstev tvořených čtyřmi body ve stejné výšce. Stínování je pak ve výsledku nejpřesnější. Problémem jsou ale okraje vrstev, které jsou viditelné a pod mraky vystupuje obloha. To působí rušivě obzvlášť při vyšší míře oblačnosti. Proto je jako geometrie pro vrstvy využito výřezu sféry viz. obr. 5. To však přináší i nevýhody. Jednou z nich je podstatně vyšší počet bodů, ze kterých je vrstva složena a s tím související vyšší hardwarová náročnost. Další je zvyšující se nepřesnost stínování směrem k okrajům vrstev, která však nepůsobí tak rušivě jako viditelné okraje u vrstev plochých. Okraje jsou zakryty terénem viz. obr. 10.



Obrázek 10: srovnání plochých (vlevo) a sférických vrstev (vpravo)

4.2.6 Změny barevnosti vrstev

V průběhu dne se postupně mění světelné podmínky a tím i barva oblaků. Barevnost vrstev lze snadno ovlivňovat, protože je odvozena od barvy první vrstvy, kterou lze za běhu programu bez omezení měnit. V době kolem poledne použita bílá, s klesajícím Sluncem je zvýrazňována červená složka.

Se západem Slunce barva přechází až k tmavě šedomodré, která dominuje v noci.

4.3 Déšť

Pro modelování jevů jako déšť, oheň, nebo kouř je často používána metoda nazývaná Částicové systémy. Metoda modeluje objekty jako shluky částic. Částice jsou v průběhu času přidávány do systému, mění svoji pozici, rychlost, tvar i další parametry a následně jsou ze systému odebrány, aby je nahradily částice nové. Takový systém dokáže zobrazit jevy, jejichž tvar není možné přesně definovat.[23] Mezi ně patří i déšť. Vytvořením efektu deště se zabývá následující část textu.

4.3.1 Částicové systémy na GPU

Částicový systém použitý k tvorbě efektu, využívá k výpočtu pozic jednotlivých částic GPU. Pohyb a chování částice je definováno kódem vertex shaderu a počátečními parametry, které jsou nastavovány při vložení nové částice do systému. Zpravidla to je počáteční pozice, čas vytvoření částice a případné další parametry. V každém snímku je vertex shaderu předán aktuální čas. Odečtením času vytvoření částice od aktuálního času je získáno stáří částice. To je použito k výpočtu pozice, na které je následně částice vykreslena. Výhodou tohoto řešení je přesunutí většiny výpočtů na grafický akcelerátor, který je pro zpracování geometrie způsobilejší než procesor. Naopak omezením je, že pohyb částice je předem pevně nadefinován kódem vertex shaderu a nastavením parametrů při vytvoření částice. Na vlivy, které vznikají v průběhu jejího života může reagovat jen velmi omezeně.[25]

Částice je zobrazena jako 2D textura. 2D textury bývají často označovány také jako sprity. XNA je zvládne zobrazit v prostoru na pozici udané jedním bodem. Takto v prostoru zobrazená textura se často označuje jako point sprite. Jejich hlavní výhodou jsou nízké nároky na hardware, proto jsou často využívány v částicových systémech.[24]

4.3.2 Definice vertexu

Kromě počáteční polohy musí vertex reprezentující částici obsahovat i čas svého vytvoření a další informace, které se často liší. Proto je často třeba definovat vlastní typ vertexu. Vertex používaný efektem deště obsahuje následující informace:

- Počáteční pozici. Ta je náhodně generována v určitých definovaných mezích.

- Hodnotu velikosti spritu.
- Čas vytvoření částice, který je důležitý pro výpočet jejího stáří.
- Náhodný vektor. Díky němu se budou dráhy jednotlivých kapek do určité míry lišit v rychlosti a směru.
- Vektor větru ovlivňující dráhu částice.

4.3.3 Správa částic v systému

Částicový systém je nutné udržovat v chodu opětovnou inicializací starých částic. Doba života částice je pevně určena. Částice jsou uchovávány v poli, jehož velikost je dána intenzitou deště a maximální dobou života částice. Intenzita je definována jako počet nových částic v průběhu jedné vteřiny. Nové částice jsou inicializovány, určena jejich výchozí pozice a další parametry. Při inicializaci ale není nastaven skutečný čas jejich vytvoření. Časy jsou voleny tak, aby byly v systému rovnoměrně zastoupeny částice všech možných stáří a v poli seřazeny sestupně podle stáří. Rovnoměrné rozložení v čase je zásadní pro dosažení přibližně stejné intenzity po celou dobu činnosti systému. Po inicializaci je nastaven index poslední aktivní částice na nulu.

V každém snímku je kontrolováno stáří poslední aktivní částice. Pokud přesáhne maximální hodnotu, je znovu inicializována a index poslední částice posunut. Nastavena je nová výchozí pozice, čas vzniku a aktuální vektor větru. Ostatní parametry lze ponechat beze změn. Postup se opakuje dokud nenarazí na částici, která zatím nedosáhla maximálního věku tak jako v následujícím kódu.

```
// pokud poslední aktivní částice přesáhla maximální věk
while (currentTime - drops[lastActiveParticle].time >= maxAge
    )
{
  // generuj novou pozici
  drops[lastActiveParticle].position = randomPosition;
  // zaznamenej čas vzniku
  drops[lastActiveParticle].time = (float)currentTime;
  // nastav aktuální vektor větru
  drops[lastActiveParticle].wind = windDirection;
  // posuň index poslední aktivní částice
  lastActiveParticle++;
  lastActiveParticle %= drops.Length;
}
```

4.3.4 Pohyb částic

Pohyb každé částice v tomto konkrétním systému je ovlivňován:

- gravitací
- větrem
- náhodným vektorem

Gravitace působí v záporném směru osy Y. Používána je rovnice:

$$s = \frac{1}{2}at^2 \tag{7}$$

hodnota s je odečítána od souřadnice y výchozí polohy částice. Zrychlení a je nastaveno na hodnotu 9.81 a jako t je použito stáří částice.

Vítr je reprezentován směrovým vektorem, který se přičítá k počáteční pozici částice. Jeho vliv roste společně se stářím částice. Stejně je zacházeno i s náhodným vektorem. Následuje fragment kódu vertex shaderu.

```
// výpočet stáří částice
float DeltaT = CurrentTime - input.Time;
// uplatnění náhodného vektoru
worldPosition.xyz += DeltaT * input.Random;
// uplatnění větru
worldPosition.xyz += DeltaT * input.Wind;
// vliv gravitace
worldPosition.y = worldPosition.y - (9.81f * pow(DeltaT, 2))
   /2.0f;
```

5 Závěr

Analýzou volně dostupných herních enginů pro XNA bylo zjištěno, že kvalitativně zatím nedosahují svých ekvivalentů pro DirectX a OpenGL. Jde o menší projekty často vyvíjené jediným člověkem a většina z nich se nachází ve velmi rozpracovaném stavu. Přesto se mezi nimy najdou některé zajímavé projekty jako BetaCell s možností dynamické kompozice efektů, nebo jednoduchý a snadno rozšiřitelný Innovative engine, který je vyvíjen a uveřejňován po částech formou tutoriálů. Podařilo se shromáždit informace o funkcích, struktuře a používání enginů a bylo provedeno jejich srovnání z hlediska nabízených funkcí. U části z nich byl překážkou nedostatek dokumentace v jakékoliv formě, proto bylo nutné získat většinu informací přímo studiem zdrojových kódů a praktickými pokusy.

Engin Innovative, zvolený pro rozšíření o další komponenty, se ukázal být vhodný pro tento účel. Díky dobrému návrhu je tvorba nového komponentu snadná. Jediným úskalím byla nemožnost práce s osvětlením, kvůli které bylo nutné provést menší úpravy zdrojových kódů enginu.

Vytvořená dynamická obloha působí uspokojivě, barevné přechody jsou ve většině denních dob plynulé. Artefakty v podobě nespojitosti barevného přechodu jsou pozorovatelné jen během přechodu mezi dnem a nocí.

Konečná podoba mraků je taktéž přijatelná. Metoda stínování předpokládá, že jednotlivé vrstvy mraků jsou ploché. Při použití plochých vrstev byl však problém s jejich viditelnými okraji. Proto byly nakonec použity vrstvy tvořené vrchlíkem sféry. Nepřesnost stínování směrem k okrajům vrchlíku se zvyšuje. Okraje jsou však zakryty terénem. Viditelná je pouze část, která se svým tvarem blíží rovině a nepřesnosti jsou zde nepatrné. Bylo by možné je dále redukovat úpravou metody stínování tak, aby uvažovala zakřivení vrstev, ovšem za cenu výrazného zvýšení výpočetní náročnosti.

V ukázkové úrovni jsou předvedeny hlavní funkce Innovative enginu a zároveň i funkce nových komponent. Těch původních nabízí engin v porovnání s konkurenty málo, jde o generování terénu z výškové mapy a fyziku. Z nových je předvedena plynulá změna denní doby a oblačnosti. Dále efekt deště s nastavitelnou intenzitou reagující na směr a sílu větru.

Přehled zkratek

- XACT (Cross-platform Audio Creation Tool) - Jde o rozhraní pro práci se zvukem původně vydanou jako součást DirectX SDK.
- XML (eXtensible Markup Language) - Jedná se o obecný značkovací jazyk určený především pro výměnu dat.
- GPU (Graphic Processing Unit) - Jde o specializovaný procesor určený pro zobrazování 3D grafiky.

Literatura

[1] XNA 3.0: What's new? [online] <http://creators.xna.com/en- US/article/xna3.0whatsnew> [citováno 2009-1-9]

[2] XNA Frequently Asked Questions [online] <http://msdn.microsoft.com/enus/ xna/aa937793.aspx> [citováno 2009-1-9]

[3] Microsoft Invites the World to Create Its Own Xbox 360 Console Games for the First Time [online] <http://www.microsoft.com/presspass/press/2006/aug06/08-13XNAGameStudioPR.mspx> [citováno 2009-1-9]

[4] Announcing XNA Game Studio 2.0 [online] <http://blogs.msdn.com/xna/archive/2007/08/13/announcing-xnagame- studio-2-0.aspx> [citováno 2009-1-9]

[5] Download details: Microsoft XNA Game Studio 3.0 [online] <http://www.microsoft.com/downloads/details.aspx?familyid=7d70d6ed- 1edd-4852-9883-9a33c0ad8fee&displaylang=en> [citováno 2009-1-10]

[6] Overview of the Content Pipeline [online] <http://msdn.microsoft.com/enus/library/bb447756.aspx> [citováno 2009-1-10]

[7] Standard Importers and Processors [online] <http://msdn.microsoft.com/enus/library/bb447762.aspx> [citováno 2009-1-10]

[8] XNA Game Studio 3.0 Programming Guide [online] <http://msdn.microsoft.com/en-us/library/bb198548.aspx> [citováno 2009-1-10]

[9] XNA Creators Club Online - membership [online] <http://creators.xna.com/en- US/membership> [citováno 2009-1-10]

[10] XNA Creators Club Online - FAQ [online] <http://creators.xna.com/en-US/faq> [citováno 2009-1-10]

[11] XNA Game Studio 3.0 Readme [online] <http://creators.xna.com/en-US/XNAGS3readme> [citováno 2009-1-10]

[12] XNA Game Studio 2.0 ReadMe [online] <http://creators.xna.com/enus/ XNAGS2ReadMe> [citováno 2009-1-10]

[13] Ox Game Engine [online] <http://www.codeplex.com/OxGameEngine> [citováno 2009-1-11]

[14] Microsoft Public License (Ms-PL) [online] <http://www.opensource.org/licenses/mspl. html> [citováno 2009-1-10]

[15] QuickStart Engine [online] <http://www.codeplex.com/QuickStartEngine> [citováno 2009-1-11]

[16] X-Engine, http://www.codeplex.com/xengine, nahlíženo 11.1.2009

[17] BetaCell Toolkit [online] <http://www.betacell3d.com/> [citováno 2009-1-11]

[18] J. A. Abad: A fast, simple method to render sky color using gradients maps. 2006. (studentská práce)

[19] Ralf Stokholm Nielsen: Real Time Rendering of Atmospheric Scattering Effects for Flight Simulators. 2003. (studentská práce)

[20] Caelum [online] <http://sourceforge.net/projects/caelum> [citováno 2009-1-11]

[21] ŽÁRA Jiří, BENEŠ Bedřich, SOCHOR Jiří, FELKEL Petr. Moderní počítačová grafika. 2. vydání. Brno: Computer Press, 2004. 609 s. ISBN 80-251-0454-0

[22] 2D Perlin noise [online] <http://www.riemers.net/eng/Tutorials/XNA/Csharp/Series4/Perlin_noise.php> [citováno 2009-1-11]

[23] Wiliam T. Reeves: Particle Systems - A Technique for Modeling a Class of Fuzzy Objects. ACM Transactions on Graphics, Vol. 2, No. 2, 91-108, 1983

[24] Point sprites – Billboarding [online] <http://www.riemers.net/eng/Tutorials/XNA/Csharp/Series2/Point_sprites.php> [citováno 2009-1-11]

[25] Particle 3D [online] <http://creators.xna.com/en-US/sample/particle3d> [citováno 2009-1-11]

Přílohy

A Tvorba vlastního komponentu v Innovative enginu

Nový komponent lze v Innovative enginu vytvořit děděním třídy Component a překrytím metod Update() a Draw(). Po vytvoření objektu třídy engin automaticky volá překryté metody, do kterých je možné umístit vlastní kód.

```
// kostra nového komponentu
public class NewComponent : Component
{
  // konstruktor
  public NewComponet() : base() {}

  // překrytí metody Draw
  public override void Draw()
  {
    // kód pro vykreslování
  }

  // překrytí metody update
  public override void Update()
  {
    // logika
  }
}
```

Volitelně lze implementovat rozhraní I3DComponent pro 3D objekty, nebo rozhraní I2DComponent pro dvourozměrné objekty. Následuje kostra komponentu naznačující možný způsob implementace rozhraní I3DComponent.

```
// kostra nového komponentu s rozhraním I3DComponent
public class NewComponent : Component, I3DComponent
{
  // proměnné potřebné pro implementaci rozhraní
  Vector3 position;
  Matrix rotation;
  Vector3 scale;
  BoundingBox boundingBox;

  // implementace rozhraní
  public Vector3 Position { get { return position; } set {
     position = value; } }
  public Vector3 EulerRotation
  {
    get { return MathUtil.MatrixToVector3(Rotation); }
    set { this.Rotation = MathUtil.Vector3ToMatrix(value); }
```

```
  }
  public Matrix Rotation { get { return rotation; } set {
     rotation = value; } }
  public Vector3 Scale { get { return scale; } set { scale =
     value; } }
  public BoundingBox BoundingBox { get { return boundingBox;
     } }

  // konstruktor
  public NewComponet() : base() {}

  // překrytí metody Draw
  public override void Draw()
  {
    // kód pro vykreslování
  }
  // překrytí metody update
  public override void Update()
  {
    // logika
  }
}
```

Následující kostra komponentu ukazuje implementaci rozhraní I2DComponent.

```
// kostra nového komponentu
public class NewComponent : Component
{
  // proměnné potřebné pro implementaci rozhraní
  Rectangle rectangle;

  // implementace rozhraní
  public Rectangle Rectangle { get { return rectangle; } set
     { rectangle = value; } }

  // konstruktor
  public NewComponet() : base() {}

  // překrytí metody Draw
  public override void Draw()
  {
    // kód pro vykreslování
  }
  // překrytí metody Update
  public override void Update()
  {
    // logika
  }
}
```

B Ukázková úroveň

Tato příloha je věnována tvorbě ukázkové úrovně. Ta ukazuje způsob použití Innovative enginu a jeho funkcí jako je generování terénu z výškové textury, nebo fyzika. Dále prezentuje způsob použití nových komponent.

Vytvoření kostry základní třídy

Hlavní třída každé aplikace vytvořené pomocí XNA je potomkem třídy Game. Třída ukázkové aplikace překrývá její metodu LoadContent() určenou pro načítání obsahu, případně inicializaci. Metodu Update() určenou pro herní logiku, v tomto případě hlavně ovládání. Metodu Draw() pro vykreslování. Kostra základní třídy a vlastně i celé ukázkové aplikace bude vypadat následovně.

```
// kostra ukázkové aplikace
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Innovation;
using JigLibX;
using System;

namespace TestEnvironment
{
  public class Game1 : Game
  {
    // manažer grafického zařízení
    GraphicsDeviceManager graphics;

    // konstruktor
    public Game1()
    {
      // nastavení grafiky
      graphics = new GraphicsDeviceManager(this);
      graphics.PreferredBackBufferWidth = 800;
      graphics.PreferredBackBufferHeight = 600;
    }

    // překrytí metody LoadContent
    protected override void LoadContent()
    {
      // načítání obsahu, inicializace
    }

    // překrytí metody Update
    protected override void Update(GameTime gameTime)
```

```
    {
      // herní logika, ovládání

      base.Update(gameTime);
    }

    // překrytí metody Draw
    protected override void Draw(GameTime gameTime)
    {
      base.Draw(gameTime)

      // vykreslování
    }
  }
}
```

Dále je nutné přidat do projektu reference na knihovny InnovationEngine a JigLibX.

Inicializace enginu a zprovoznění základních služeb

Následující akce je nutné provést v metodě LoadContent(). Prvním krokem je zavolání metody SetupEngine(), které provede inicializaci a nastavení enginu. Dále se musí vytvořit objekty kamery, fyziky, klávesnice a myši a přidat je do služeb enginu.

```
protected override void LoadContent()
{
  // inicializace a nastavení enginu
  Engine.SetupEngine(graphics);

  // vytvoření FPS kamery
  FPSCamera camera = new FPSCamera();
  // nastavení výchozí pozice kamery
  camera.Position = new Vector3(125f, 2f, 125f);
  // přiřazení do služeb enginu
  Engine.Services.AddService(typeof(Camera), camera);

  // přidání fyziky mezi služby enginu
  Engine.Services.AddService(typeof(Physics), new Physics());

  // vytvoření objektů klávesnice a myši
  MouseDevice mouse = new MouseDevice();
  KeyboardDevice keyboard = new KeyboardDevice();

  // přidání klávesnice a myši mezi služby enginu
  Engine.Services.AddService(typeof(MouseDevice), mouse);
```

```
  Engine.Services.AddService(typeof(KeyboardDevice), keyboard
     );
}
```

Vytvoření a zobrazení terénu

Tvar terénu je určen výškovou mapou, druhá textura je namapována na jeho povrch. Obě textury je nutné přidat do složky Content projektu ukázkové aplikace. Objekt terénu je deklarován jako atribut třídy.

```
Terrain terrain;
```

Inicializace je provedena v metodě LoadContent().

```
terrain = new Terrain(
  Engine.Content.Load<Texture2D>("Content/heightmap"),
  Engine.Content.Load<Texture2D>("Content/grass"));
```

Současně s objektem je automaticky vytvořen i kolizní povrch, případné další objekty na něj budou adekvátně reagovat.

Vytvoření jednoduchého ovládání

Ovládání klávesami WASD a myší je vytvořeno pomocí služeb klávesnice, myši a kamery v metodě Update().

```
// objekty klávesnice, myši a kamery
KeyboardDevice keyboard = Engine.Services.GetService<
    KeyboardDevice>();
MouseDevice mouse = Engine.Services.GetService<MouseDevice>()
    ;
FPSCamera cam = (FPSCamera)Engine.Services.GetService<Camera
    >();

// sestavení vektoru pohybu na základě
// stisknutých kláves
Vector3 inputModifier = new Vector3(
  (keyboard.IsKeyDown(Keys.A) ? -1 : 0) + (keyboard.IsKeyDown
     (Keys.D) ? 1 : 0),
  (keyboard.IsKeyDown(Keys.Q) ? -1 : 0) + (keyboard.IsKeyDown
     (Keys.E) ? 1 : 0),
  (keyboard.IsKeyDown(Keys.W) ? -1 : 0) + (keyboard.IsKeyDown
     (Keys.S) ? 1 : 0)
);

// rotace v osách X a Y na základě změny polohy
// myši a následný posun ve směru vektoru pohybu
cam.RotateTranslate(new Vector3(mouse.Delta.Y * -.002f, mouse
    .Delta.X * -.002f, 0),inputModifier * .5f);
```

Zobrazení modelu, přiřazení fyzikální reprezentace

Pro ověření funkce fyziky je v ukázkové aplikaci umožněno odhazovat krychle pomocí levého tlačítka myši. Pro zobrazení modelu s fyzikálními vlastnostmi slouží třída Actor. Kód bude umístěn v metodě Update(). Pokud dojde ke stisku levého tlačítka myši bude vytvořen objekt třídy Actor. Tomu je nutné předat model objektu a jeho fyzikální reprezentaci jako parametry. Aby došlo k odhození, je potřeba nastavit vektor počáteční rychlosti ve směru pohledu kamery.

```
// pokud dojde ke stisku leveho tlačítka myši
if (mouse.WasButtonPressed(MouseButtons.Left))
{
  // vytvoření objektu
  PhysicsActor act = new PhysicsActor(
    Engine.Content.Load<Model>("Content/ig_box"),
  new BoxObject(new Vector3(2.5f), cam.Position, Vector3.Zero
     ));

  // změna měřítka objektu
  act.Scale = new Vector3(2.5f);
  // nastavení hmotnosti
  act.PhysicsObject.Mass = 1000;

  // nastavení vektoru rychlosti ve směru pohledu
  Vector3 dir = cam.Target - cam.Position;
  dir.Normalize();
  act.PhysicsObject.Velocity = dir * 10;
}
```

Zobrazení oblohy

Objekt oblohy deklarujeme jako atribut třídy.

```
Sky sky;
```

Inicializace je provedena v metodě LoadContent().

```
sky = new Sky();
```

Denní dobu lze posouvat kupředu stiskem klávesy T. Obslužný kód je umístěn v metodě Update().

```
if (keyboard.IsKeyDown(Keys.T))
{
  sky.Time += 0.001f;
}
```

Počet vrstev mraků lze přepínat klávesami F1 - F12. Číslo funkční klávesy odpovídá počtu zobrazených vrstev.

```
if (keyboard.IsKeyDown(Keys.F3))
{
  sky.Layers = 3;
}
```

Množství oblačnosti lze zvyšovat pomocí klávesy O a snižovat klávesou P.

```
if (keyboard.IsKeyDown(Keys.O))
{
  // zvýšení oblačnosti
  sky.Overcast += 0.01f;
}
if (keyboard.IsKeyDown(Keys.P))
{
  // snížení oblačnosti
  sky.Overcast -= 0.01f;
}
```

Zobrazení deště

Objekt deště deklarujeme jako atribut třídy.

```
Rain rain;
```

Inicializace je provedena v metodě LoadContent(). První parametr konstruktoru je pozice středu deště. Prostor kde prší je vymezen krychlí s délkou strany udanou druhým parametrem. První parametr určuje pozici středu této krychle. Třetí parametr určuje intenzitu deště a čtvrtý udává maximální stáří částice.

```
rain = new Rain(camera.Position, 100f, 1000, 4);
```

V ukázkové aplikaci je pozice deště vždy umístěna na pozici kamery. Kamera se tak nachází v jeho středu. Posouvání zajišťuje následující kód v metodě Update().

```
rain.Position = cam.Position;
```

Předdefinovaná nastavení

Ukázková aplikace obsahuje také předdefinovaná nastavení oblohy a deště, mezi kterými lze rychle přepínat pomocí kláves 1-4. Jde o kombinace různých nastavení

Ovládání ukázkové aplikace

- WASD + myš - pohyb

- Levé tlačítko myši - odhození krychle
- T - posun denní doby
- F1-F12 - přepínání počtu vrstev
- O - zvýšení oblačnosti
- P - snížení oblačnosti
- 1-4 - přepínání mezi předdefinovanými nastaveními
- ESC - ukončení aplikace

C Příručka k použití komponent

V této příloze naleznete přehled a krátký popis možných nastavení komponentů oblohy a deště. Nastavení obou komponentů lze provádět nad jejich instancí pomocí vlastností (properties). Následuje přehled vlastností komponentu Sky.

- int Layers { get; set; } - počet vrstev, ze kterých je složena vrstva mraků
- float Overcast { get; set; } - hodnota ovlivňující míru oblačnosti
- Vector3 LightDir { get; } - směr světla odpovídající nastavené denní době
- Vector3 LightColor { get; } - barva slunečního světla
- float Time { get; set; } - hodnota určující denní dobu (viz. obr. 4)
- Vector2 WindDir { get; set; } - směr větru, udává směr pohybu mraků
- float WindStrenght { get; set; } - síla větru, udává rychlost pohybu mraků

Přehled vlastností komponentu Rain.

- Vector2 WindDir { get; set; } - směr větru, udává směr vychýlení dráhy kapek
- float WindStrenght { get; set; } - síla větru, udává velikost vychýlení dráhy kapek
- float Side { get; set; } - délka strany krychle ohraničující prostor výskytu dešťových kapek
- Vector3 Position { get; set; } - pozice středu krychle ohraničující prostor výskytu dešťových kapek
- float Intensity { get; set; } - intenzita deště (průměrný počet nových částic za sekundu)