VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

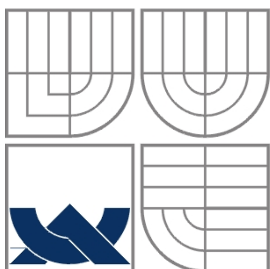
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GRAFICKÉ INTRO 64KB S POUŽITÍM OPENGL

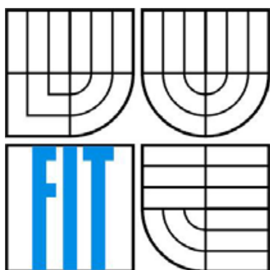
DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE Bc. JAN GERŠL
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GRAFICKÉ INTRO 64KB S POUŽITÍM OPENGL

GRAPHIC INTRO 64KB USING OPENGL

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE — Bc. JAN GERŠL
AUTHOR

VEDOUCÍ PRÁCE — Ing. ADAM HEROUT, Ph.D.
SUPERVISOR

BRNO 2007

Abstrakt

Práce se zabývá fenoménem grafického intra s omezenou velikostí. Vysvětluje motivaci k jeho tvorbě, stručně se zmiňuje také o historii, popisuje obecně principy a techniky běžně používané při vývoji krátkých dem a také detailně vysvětluje vybrané techniky, které byly využity přímo k implementaci praktické části.

Klíčová slova

OpenGL , demo, intro, demoscéna, demoparty, 64kB, digitální umění, optimalizace, procedurální textury, Perlinův šum, generování terénu, čásicový systém, L-systém, kruhové rozmazání, plasma, MIDI, hudební modul, syntéza zvuku, minimalizace exe, komprese spustitelných souborů, záře, odraz na vodní hladině, animace, město, svítící okna.

Abstract

The field of this project is size restricted graphic intro. The paper deals with motivation creating such an intro and talks briefly about history. Main focus is put on general description of various principles common in demo development, techiques used achieving the practical assignment are provided with more detailed description.

Keywords

OpenGL, demo, intro, demoscene, demoparty, 64kB, digital art, optimalization, procedural textures, Perlin noise, terrain generation, particle systém, L-systém, radial blur, plasma, MIDI, sound module, sound synthesis, exe minimalisation, executable compression, glow, bloom, water plane reflection, animation, city, window lights.

Citace

GERŠL, Jan. *Grafické intro 64kB s použitím OpenGL*. Brno, 2008. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií.

Grafické intro 64kB s použitím OpenGL

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Adama Herouta, Ph.D.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……………………
Jan Geršl
19.5.2008

Poděkování

Rád bych na tomto místě poděkoval Ing. A. Heroutovi, Ph.D., který mi jako vedoucí projektu byl ochoten poradit ve věcech odborných, estetických i formálních a věnovat čas konzultacím, kdykoliv bylo potřeba.
Dále bych rád poděkoval Bc. Janu Pinterovi za pomoc při problémech s kompilátorem.

© Jan Geršl, 2008.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Obsah 1
1 Úvod 2
2 Intro s omezenou velikostí 3
2.1 Motivace 3
2.2 Historie 4
2.3 Hlavní směry vývoje 6
3 Grafické aspekty intra 8
3.1 3D scény a jejich reprezentace 8
3.2 Procedurální textury 12
3.3 Vizuální efekty 15
4 Techniky ozvučení intra 17
5 Nástroje při vývoji intra 19
5.1 Komprese spustitelných souborů 19
5.2 Demo Tools 20
6 Návrh a realizace aplikace 21
6.1 Nastavení překladače a linkeru 21
6.2 Model scény 24
6.3 Generované textury 28
6.4 Animace 30
6.5 Efekty 31
6.6 Hudba 37
6.7 Komprese výsledného programu 37
Závěr 39
Literatura 40
Příloha 1. Užitečné odkazy 41
Seznam příloh 42

1 Úvod

Digitální umění má mnoho podob a bezesporu se jedná o nezanedbatelný fenomén dnešní doby. Ze všech stran k nám proudí digitálně vytvořená či vylepšená hudba, filmy s animovanými postavami nebo alespoň speciálními efekty, přičemž většina dat se k nám dostává skrz tradiční média – rozhlas, televize, kino. Ve světě počítačů jde tato móda svou cestou v podobě multimediálních prezentací či krátkých filmů.

Cílem této práce je přiblížit podskupinu počítačového umění – tvorbu krátkých animací s omezením na 64kB, analyzovat používané techniky a vytvořit vlastní ukázkovou aplikaci s použitím OpenGL.

Práce nenavazuje na ročníkový projekt. Navazuje však na stejnojmenný semestrální projekt, který si kladl za cíl zpracovat teoretické základy dané problematiky. Kapitoly 2-4 této práce z něj byly převzaty a dále rozšířeny. Spolu s kapitolou pátou pojednávají o běžných praktikách používaných při tvorbě velikostně omezených inter. Objem prací vyřešených v rámci semestrálního projektu odpovídá zhruba 30 % celkového rozsahu. Z teoretických základů čerpá především praktická část řešené problematiky.

2 Intro s omezenou velikostí

Jak již bylo řečeno v úvodu, *grafické intro* je forma umění tvořená počítačovými nadšenci a programátory. Nejedná se však pouze o umění, ale důležitým aspektem jak při tvorbě tak při hodnocení na soutěžích je technická stránka výsledného produktu.

Celá komunita tvůrců a příznivců *grafického intra* s omezenou velikostí je nazývána *demoscéna*. *Intro* a *demo* jsou v podstatě ekvivalentní názvy pouze s mírným historickým rozlišením. Podle některých pramenů se jedná o *intro*, pokud existuje nějaké velikostní omezení, a o *demo*, pokud je kategorie bez omezení. Toto však není podstatné a oba termíny se běžně zaměňují. Podobně existují z historických důvodů různé hybridní názvy podle účelu, kterému animace sloužila, např. *invtro* – invitation intro, sloužící jako pozvánka na nějakou společenskou událost či akci. Tyto názvy jsou však již v podstatě archaické a používaly se jen v době vzniku prvních inter.

Intra soutěží mezi sebou v rozličných kategoriích, nejznámější kategorie jsou 128kB, 64kB, 4kB, 256B. Samozřejmě, že existují i soutěžní kategorie bez omezení velikosti, ty však nejsou z hlediska technologií vůbec zajímavé a spíše se již jedná o přehlídku animovaných filmů.

Jednotliví tvůrci resp. skupiny (jednotlivci jsou mezi autory spíše výjimkou) se snaží v rámci velikostního omezení přinést co nejlepší a nejpoutavější výsledky, což se v silné konkurenci bez technických inovací neobejde. Proto si každá úspěšná skupina vyvíjí vlastní nástroje, algoritmy a triky, vše s cílem ukázat světu, že dokážou být lepší než ostatní.

2.1 Motivace

Vytvoření grafického intra není pouze prostou realizací nějakého uměleckého nápadu, především se jedná o velkou výzvu. Jak říká vedoucí této práce: „*Naprogramovat dnes pětimegabajtovou aplikaci je malina.*“, ale zkuste si to samé vměstnat do několika kilobajtů.

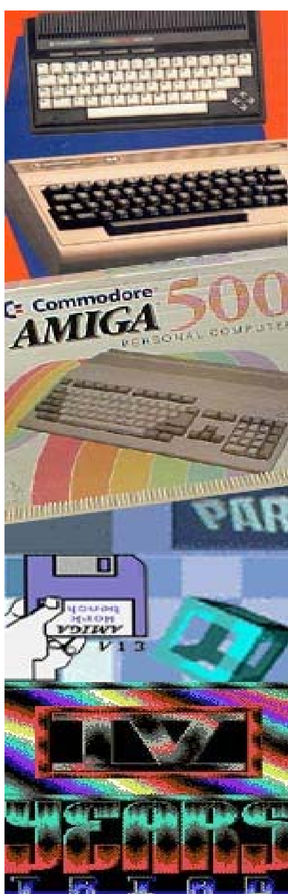
K vývoji multimediálních programů dnes není třeba nic než dobrý nástroj a trocha času. V nějakém vizuálním vývojovém prostředí stačí několik kliknutí a je na světě základní aplikace. Do té přetáhneme myší nějakou hudební skladbu a pomocí některé z mnoha univerzálních grafických knihoven načteme celou 3D scénu včetně informací o materiálech, světlech, kamerách a celé animaci. Za pár okamžiků už Direct3D vykresluje ozvučený film.

Tento přístup programování na vysoké úrovni abstrakce má samozřejmě velkou výhodu v rychlosti, s jakou lze implementovat i velmi složité projekty. Bohužel tak jako vše, co je univerzální a snadno použitelné, si i tento přístup vybírá určitou daň. Snížená rychlost, zvýšené nároky na paměť, kterou výsledný program zabírá, to jsou aspekty, které díky velkému nárůstu výkonnosti počítačů a kapacity úložných médií u většiny aplikací ani nepostřehneme. Na druhou stranu hry nebo náročná grafika stále nutí k zamyšlení nad tímto plýtváním i dnes.

Grafické intro se snaží s co nejmenšími prostředky předvést co nejvíce a využít tak dostupnou technologii na maximum. Zaměřme chvíli pozornost na odvětví, které nutí jít počítačový hardware stále kupředu – hry.

Před několika lety se programy a hry distribuovaly na několika málo disketách, dnes je to již na několika málo DVD médiích. Naproti tomu grafická intra, která jsou stará jako první počítačové hry, stále soutěží ve stejných velikostních kategoriích a grafický výstup je v závislosti na schopnostech nových grafických karet srovnatelný s jejich herními protějšky. Dokonce se objevila hra na 96kB – Kkrieger, která (až na herní rozsah) může směle konkurovat nejnovějším titulům na trhu.

2.2 Historie



Demoscéna je stará téměř jako počítače samy. Již na počátku osmdesátých let 20. století přicházely na trh první osmibitové počítače a s nimi programy a první hry. Objevily se první krádeže softwaru, první ochrany proti nelegálnímu kopírování a první *crackeři*, tedy lidé snažící se o prolomení takových ochran. Pokud byli úspěšní, samozřejmě se tím netajili a své dílo náležitě podepsali – líbivým efektem či krátkou animací. Odtud podle mnoha pramenů původně pochází demoscéna.

Samozřejmě, že toto jsou jen špičky kořenů dnes uznávané oblasti počítačového umění. Kvůli tehdejším (dnes nepředstavitelně miniaturním) záznamovým médiím musel takový zásah do programu být paměťově velmi úsporný a na vysoké technické úrovni. Ani v současné době se nejedná o nic triviálního. Proto se našli lidé, kteří tomuto rodícímu se fenoménu věnovali svou pozornost, začali přemýšlet, jakým způsobem jsou efekty vytvořeny a jak je vylepšit. Začali je nejen napodobovat, ale také vymýšlet nové, které prezentovali již v samostatných aplikacích a přirozeně také začali mezi sebou soutěžit. To byl zlomový moment vzniku demoscény jak ji známe dnes a lidská soutěživost ji udržela na živu a stále ji žene dál již téměř 30 let.

2.2.1 Světová demoscéna

První věc, která čtenáře jistě v souvislosti s demoscénou napadla, byla „Jak asi probíhá ono soutěžení?“ „Jak jinak než celosvětově a jak jinak než online,“ zněla by asi odpověď člověka znalého internetu. Není to tak docela pravda.

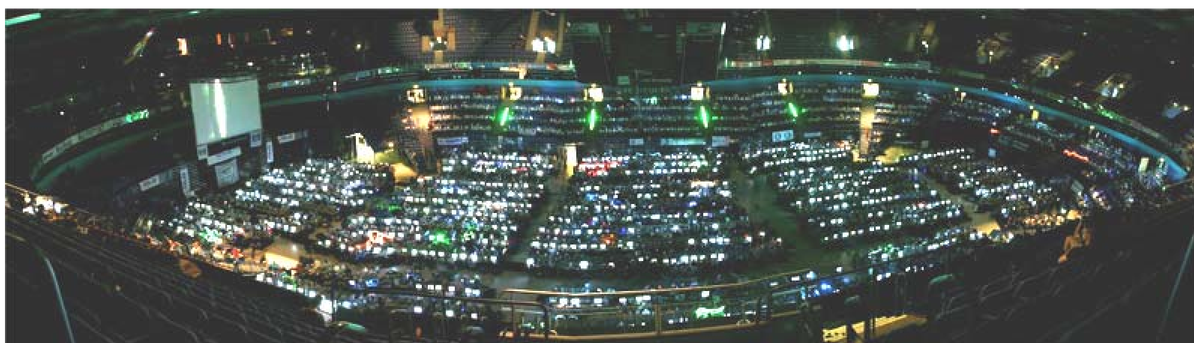
Samozřejmě existují na internetu místa, kde se soustředí velká většina tvorby. Především jsou to portály (rovněž uvedené níže v pramenech) www.scene.org a www.pouet.net. Zde probíhá život

komunity kolem demoscény – stránky poskytují diskuzní skupiny a fóra, novinky „z oboru“ a rozsáhlé archivy obsahující tvorbu všech kategorií, platforem, autorů i časů vzniku.

Na www.scenemusic.eu najdeme portál (známý také jako Nectarine Demoscene Radio) věnující se samostatně hudbě doprovázející dema. Ozvučení dem je většinou i samostatná soutěžní kategorie, protože téma tvorba a přehrávání hudby v miniaturních intrech by svou obsáhlostí vystačilo na samostatnou diplomovou práci. Proto se jí také tento projekt věnuje pouze okrajově.

Samotné soutěže se však (snad naštěstí) konají v reálném světě na setkáních, nazývaných *demoparty*. Celá akce probíhá v několika dnech v prostorách nějakého klubu, tělocvičny či velké sportovní haly (podle velikosti události), kde se účastníci se svými počítači propojí do lokální sítě, na velkém plátně se promítají soutěžní příspěvky, lidé si vyměňují názory, zkušenosti, probíhají odborné přednášky, ale také zábava.

Takovýchto *demoparties* se v Evropě koná do roka hned několik. Z největších můžeme jmenovat např. finskou *Assembly*, německé *Breakpoint* a *Evoke* či norský *The Gathering*. V roce 2007 např. proběhlo zhruba 80 akcí, viz [8].



Obrázek 1: Fotka haly největší demoparty – Assembly

Význam jednotlivých demoparties je těžko měřitelnou veličinou, vodítkem úspěšnosti může však být počet soutěžních příspěvků, který se u největších pohybuje kolem 100, u většiny však nepřesáhne ani 20. Návštěvnost je určitě přímo úměrná tomuto faktu, avšak hodnoty mohou být zavádějící vzhledem k různému počtu soutěžních kategorií.

2.2.2 Demoscéna u nás

V České republice se koná maximálně jedna soutěž ročně a bohužel se nepodařilo najít dostatečný zájem k pořádání pravidelné akce, jakou je v zahraničí např. již zmiňovaná největší *Assembly*.

V letech 1998 – 2001 se konala party s názvem Fiasko, byla to první česká soutěž a také vydržela nejvíce ročníků. Roky 2004 a 2005 patřily také poměrně populární akci Marast, mezi tím proběhlo několik menších, např. Digital Zooo (2000) a Subway p2k (1999).

Na Slovensku stojí za zmínku především úspěšné akce pro osmibitové počítače – Forever, jehož 9. ročník se uskuteční letos (2008) a Demobit, který se tradičně konal v Bratislavě už od roku 1995 a byl pomyslnou startovní čárou české i slovenské demoscény.

2.3 Hlavní směry vývoje

Existuje několik základních proudů vývoje, které se diametrálně liší přístupem k problému. Jedna skupina využívá k vykreslování OpenGL (či obecně i jiné grafické knihovny), kdežto druhá implementuje vlastní vykreslovací mechanismy založené na metodách sledování paprsku. V této kapitole oba přístupy krátce přiblížím. V závěru kapitoly je ještě uvedena polemika na téma použitého operačního systému.

2.3.1 OpenGL

OpenGL je programové rozhraní grafického hardwaru. Toto rozhraní se skládá ze zhruba dvou set příkazů, které se používají pro definici objektů a operací potřebných k vytvoření trojrozměrných aplikací.

Díky koncepci OpenGL jako abstraktního rozhraní je použití jeho funkcí nezávislé na hardware a operačním systému cílového počítače. Jednoduše, co není hardwarově podporováno se emuluje. OpenGL také neobsahuje žádné funkce spojené s konkrétním operačním systémem, takové úkoly jsou ponechány na programátorovi aplikace, např. načítání textur ze souboru, import externích modelů apod. Tím je zajištěna vysoká přenositelnost, která je hojně využívána v průmyslových aplikacích, výzkumných projektech, ale i v mnoha počítačových hrách.

Velkou nevýhodou v oblasti zábavního průmyslu je oproti konkurenčnímu DirectX právě absence funkcí zajišťujících pokročilé úkony, což zpomaluje vývoj nových aplikací. Na druhou stranu tím programátor získává plnou kontrolu nad každým detailem. Samozřejmě také existují knihovny postavené nad OpenGL, které dodatečně dohánějí „sníženou“ funkčnost. Ty však v této práci využívány být nemohou, protože se jedná o univerzální balíky s příliš mohutnou velikostí (v porovnání s uvažovanou velikostí výsledného programu) a protože by stejně většina jejich obsahu zůstala nevyužita.

2.3.2 Sledování paprsku

Pro vytvoření grafického intra samozřejmě není OpenGL jedinou cestou. Existují rovněž intra, která pro vykreslování využívají metody sledování paprsku ve vlastní implementaci. Mohou se potom chlubit takovými grafickými detaily, jako např. měkké stíny, které jsou v OpenGL nedostupné.

Výkon takového řešení však neumožňuje zpracovat příliš rozsáhlé scény, a tak intro realizované tímto způsobem nezaujme množstvím zpracovaných trojúhelníků a rozsáhlými modely, ale spíše grafickými lahůdkami spojenými s optikou.

2.3.3 Windows vs. Linux

Protože mají grafická intra omezenou velikost, snaží se používat veškeré dostupné prostředky ve svůj prospěch s cílem ušetřit cenné bajty pro modely a efekty. To mimo jiné znamená, že jsou intra vždy primárně určena pro určitý operační systém a nejsou bez úprav přenositelná.

Je výhodné využít API konkrétního operačního systému pro vytvoření okna, přehrání zvuku či k jakýmkoliv jiným užitečným účelům. Např. v poslední době mnoho inter využívá hlasový syntetizér zabudovaný v systémech Windows 2000 a pozdějších. Proto je také jedna ze základních otázek, které je potřeba před započetím konkrétní implementace zodpovědět, pro který operační systém bude intro určeno.

Pro operační systém Windows hovoří především jeho velká rozšířenost a s tím spojené publikum. Mnoho programátorů má také obavy z kvality ovladačů, případně nepodporovaného hardwaru pod systémem Linux, či jim chybí pohodlí Microsoft Visual Studia.

Zastánci Linuxu budou zase oprávněně tvrdit, že ovladače jsou stejně kvalitní, komfortní vývojové nástroje existují, největší demoparties přijímají i soutěžní příspěvky pro Linux a v neposlední řadě, že přenést finální produkt z Linuxu na Windows bývá ve většině případů jednodušší, než opačný postup.

Suma sumárum, faktický rozdíl mezi oběma typy přístupu není, je to pouze otázka preferencí a protože je k napsání miniaturního programu potřebná poměrně detailní znalost cílového systému, každý si volí pracovní prostředí podle svých dovedností.

3 Grafické aspekty intra

Zatímco zvuk bez grafické stránky nelze zařazovat mezi grafická intra, grafiku bez zvuku stále můžeme považovat za plnohodnotnou součást demoscény. To je jeden z důvodů, proč je potřeba věnovat dostatek času studiu efektů, textur a výrobě scén.

Kapitola pojednává o běžně používaných technikách, které stručně popisuje. I když nejsou v projektu použity všechny uvedené postupy, byly přinejmenším zvažovány jako alternativa a myslím, že je vhodné se o nich zmínit.

3.1 3D scény a jejich reprezentace

Trojrozměrné modely jsou ve většině případů v počítačové grafice reprezentovány množinou bodů a z ní vytvořených trojúhelníků. K vykreslení celé scény je také potřeba znát pozice světla či světel, pozici kamery, vlastnosti povrchu – tedy barva a vlastnosti jeho materiálu či textur, pro jejichž aplikaci je potřeba jednotlivým bodům povrchu přiřadit normálové vektory.

Vše je modelováno v určitém prostoru nezřídka vyžadujícím větší datové typy pro reprezentaci absolutních hodnot pozic jednotlivých bodů. Každý bod v prostoru navíc zabírá 3 souřadnice, každý vektor má tři složky. Objem dat tak narůstá velmi rychle.

Vzhledem k povaze 3D modelů si nemůžeme dovolit prostě vypustit některá data, jak je např. běžné u ztrátové komprese 2D obrázků, a spoléhat se na nedokonalosti lidského oka. To je totiž citlivé především na jasové přechody, tedy hrany, kterých s ubývajícím počtem polygonů modelu přibývá. Pokud redukujeme počet stěn koule na 6, nikdo nám neuvěří, že se nejedná o krychli. V intru však potřebujeme kvalitní modely – pokud chceme vyprávět nějaký příběh nebo zobrazit nějakou konkrétní věc, je potřeba, aby divák bezpodmínečně rozpoznal, o co se jedná.

Na první pohled slepá ulička však skrývá mnoho možností úsporné reprezentace i rozsáhlých modelů.

3.1.1 Generování modelů

Asi nejúspornější variantou je potřebné modely neukládat do statických dat vůbec, ale za běhu je generovat.

Především přírodní útvary vykazují vysokou míru náhodnosti, což umožňuje připravit takové objekty pro zobrazení až přímo při běhu programu. Buďto se jedná o věci s neurčitým tvarem (na jejichž tvaru příliš nezáleží, např. kámen, asteroid apod., které jsou pouhou kulisou prostředí děje, např. krajina, nebo věci popsatelné gramatikou – tato technika je běžná např. u modelování rostlin.

Na druhou stranu jen stěží si lze představit náhodně generované předměty vytvořené člověkem, mající konkrétní formu. Avšak i zde můžeme využít náhody ve svůj prospěch. Stačí uložit jeden

model a parametrizovat jeho jednotlivé části. Náhodnou volbou parametrů pak můžeme získat velké množství různě vypadajících, ale podobných objektů.

Modely terénu

Pokud potřebujeme ve scéně jakkoliv rozsáhlý terén, nemusíme si pamatovat všechny jeho body, stačí mít hrubou představu reprezentovanou několika parametry a zvolit jeden z mnoha algoritmů pro generování, které většinou vychází z rovinné mřížky a jejichž výsledkem je zvrásněný povrch.

Mezi známé metody patří metoda středního bodu, metoda náhodných poruch, využití perlinova šumu, nanášení částic, frekvenční syntéza nebo multifraktály. Výčet algoritmů není jistě kompletní a ani v tomto případě se fantazii meze nekladou.

Na výsledný terén se poté dají navíc aplikovat erozní a vyhlazovací algoritmy pro dosažení větší detailnosti či naopak odstranění nepřirozeně ostrých výběžků.

Přesouvání středního bodu je nejběžnější metoda, což je rekurzivní algoritmus vycházející ze čtverce. Při inicializaci se nastaví výška rohů čtverce do základního sklonu a pak při každé iteraci určíme střed čtverce, ten náhodně posuneme (podle parametru hrubosti terénu) a zapíšeme novou výšku v tomto bodě výškové mapy. Pootočíme čtverec o 45°, rozdělíme na 4 díly a algoritmus opakujeme pro nově vzniklé části a postupně zmenšujeme rozsah náhodných posunů. Tak pokračujeme, dokud není vyplněna celá výšková mapa.

Obdobě, posouváním výškových dat, funguje metoda náhodných poruch (metoda zlomů). V každém kroku algoritmu se zvolí přímka – řez výškovou mapou dělící data na dvě skupiny. Jedna skupina výškových dat se náhodně posune. Opakováním dostatečného počtu iterací vznikne chaoticky vypadající terén, který je potřeba dále upravit erozí či vyhlazením. Tato metoda se často používá při generování povrchu planet resp. textury jejich povrchu.

Perlinův šum je detailně popsán v kapitole 3.2.1, můžeme jej použít přímo jako výškovou mapu a ráz terénu ovlivnit parametry při generování tohoto typu šumu.

Metoda nanášení částic se podobá sněžení – přidáváme na kupku částice, dokud není terén kompletní. Začíná se s existující hustou mřížkou a přidání částice je zvýšení některého z bodů. Výběr bodu může být náhodný výběr s přesunem na nejbližší bod nebo přesunem na nejnižší či nejvyšší bod v okolí výběru. Od toho se také odvíjí vzhled výsledku. Tento algoritmus je vhodný na nížinné povrchy s mírnými změnami, např. souostroví.

Frekvenční syntéza je zajímavý postup, při kterém se frekvenční spektrum generovaného bílého šumu filtruje funkcí

$$\frac{1}{f^B},$$

kde f je frekvence a B má spojitost s fraktální dimenzí D podle vztahů:

$$B = 1 + 2H$$
$$D = 3 - H$$

Zpětnou rychlou fourierovou transformací FFT získáme amplitudy použitelné do výškové mapy. Generování šumu ve frekvenční oblasti probíhá poměrně jednoduchým způsobem, což je výpočetní výhoda. Nevýhodou je poměrně vysoká periodičnost povrchu [4].

Multifraktální metody se většinou používají na již existující výškovou mapu terénu a zajistí jeho větší realističnost. Zavádějí totiž více různých fraktálních dimenzí výsledného povrchu, např. v nížinách je běžná menší členitost povrchu než v horách na skalách. Algoritmus je rovněž velmi rychlý oproti ostatním metodám, ale náročnější na implementaci [4].

L – systémy

Lindenmeyerovy systémy představují asi nejlépe propracovanou formální teorii modelování rostlin, a to včetně jejich růstu. Názorný, avšak zjednodušený popis problematiky lze nalézt v literatuře [1].

L-systém je v podstatě gramatika, jejíž terminály se interpretují geometricky. Jako existuje více druhů gramatik, tak existuje i více druhů těchto systémů. Determinismus není nutnou podmínkou pro fungování systému, nedeterministická přepisovací pravidla umožňují reagovat nějakým způsobem např. na prostředí, ve kterém rostlina roste (detekce kolizí, ohodnocení životních podmínek apod.) výběrem různého rozvoje stejných nonterminálů.

Pro objekt rostliny je tedy potřeba definovat gramatiku, určit přepisovací pravidla a geometrickou reprezentaci jednotlivých symbolů a zvolit počáteční řetězec. K zobrazení výsledku se používá tzv. želví grafika. Želva představuje kreslící nástroj a jednotlivé symboly reprezentují příkazy pro její otáčení a pohyb. Zavedením závorek ve formě zásobníku geometrických stavů (podobně jako zásobník transformačních matic v OpenGL) získáme velmi mocný nástroj pro kreslení.

Jednoduchá gramatika pro květinu by mohla vypadat následovně:

$$G = (N, \Sigma, P, S),$$

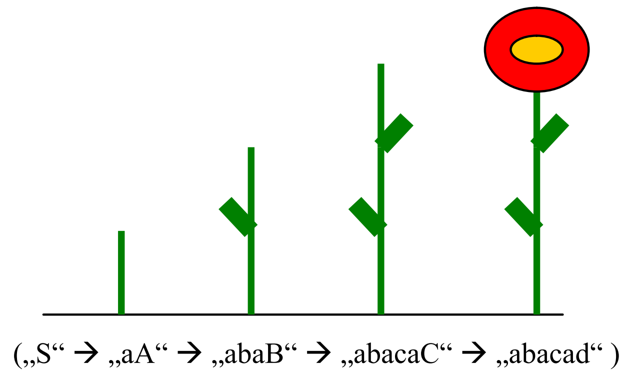
kde

$$N \in \{S, A, B, C\},\ \Sigma \in \{a, b, c, d\}$$

a množina přepisovacích pravidel P obsahuje:

$$S \rightarrow aA$$
$$A \rightarrow baB$$
$$B \rightarrow caC$$
$$C \rightarrow d$$

Pak ***S*** představuje pomyslné semínko, ***a*** je stonek rovně nahoru, ***b*** je list doleva, ***c*** je list doprava, ***d*** je květ. Počáteční řetězec „*S*“ se postupně rozvíjí podle pravidel, jak demonstruje obrázek 2.

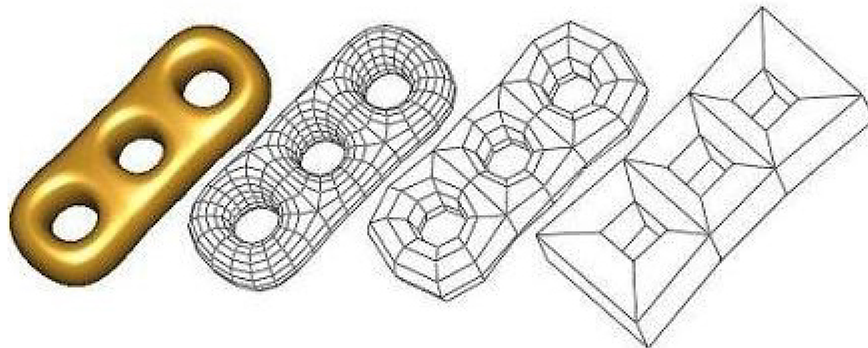


Obrázek 2: Ilustrace grafické reprezentace přepisovacích pravidel

3.1.2 Dělení ploch

Dalším způsobem, jak ušetřit cenné bajty či kilobajty zabrané trojrozměrným modelem ve statických datech, je uložení hrubé kostry a pozdější aplikace algoritmu dělení ploch.

Navzdory tvrzení z úvodu kapitoly o uchování 3D dat lze opravdu vypustit některé zbytečné vrcholy a snížit tak drasticky počet polygonů modelu. Je však potřeba udělat to tak chytře, aby po dělení ploch opravdu vzniknul námi zamýšlený objekt. Nejedná se tedy opravdu o bezmyšlenkovité vynechávání vrcholů, ale předem promyšlenou úpravu dat s cílem připravit minimalistickou kostru pro další zpracování za běhu programu.



Obrázek 3: Postupné dělení ploch zprava doleva

3.1.3 Způsob uložení v paměti

Prostým uložením struktury dat do paměti plýtváme zbytečně místem, aniž bychom si toho všimli. Proto je nutné věnovat zvýšenou pozornost způsobu, jakým připravený a pracně minimalizovaný model uschováme.

Je obvyklé, že překladače zarovnávají paměť na sudé adresy případně i na 4B apod. Pokud máme strukturu objektu sestávající se z mnoha položek, u každé dojde k zarovnání byť o jeden bajt, znamená to citelnou ztrátu. Naštěstí (pro tento projekt) se dá zarovnání paměti vypnout, a tím ušetřit.

Případně se struktura navrhne tak, aby byla přímo zarovnaná, čímž se účelně vyplní celý paměťový prostor jí přidělený.

Pokud zamýšlíme data komprimovat, můžeme se pokusit strukturu nějakým způsobem optimalizovat pro tuto akci. To znamená sdružovat dlouhé úseky stejných dat nebo mnoho periodicky opakujících se vzorů dat. Pokud např. uchováváme dvoubajtová celá čísla, horní bajt může často obsahovat velké množství nul. Pokud v takovém případě ukládáme odděleně nejdříve všechny spodní bajty a poté všechny horní bajty za sebe, dostáváme dlouhou řadu nul.

V neposlední řadě existuje technika pro redukci bitové délky jednotlivých souřadnic vrcholů modelu. Spočívá v převedení reálných čísel s plovoucí řádovou čárkou na celá čísla s menším počtem bitů. Model se uzavře do jednotkové krychle, takže maximální souřadnice modelu odpovídají souřadnici 1 v krychli. Hrany krychle se dále rozdělí na 256 částí (v případě převodu na 1B celé číslo) a jednotlivým vrcholům modelu se přiřadí místo původních souřadnic odpovídající čísla 0-255.

3.2 Procedurální textury

Hlavní nevýhodou bitmapových (či správně podle OpenGL pixmapových) textur je velký prostor, který zabírají. Procedurální textury tímto neduhem netrpí. Jedná se o funkce, které pro zvolený bod vrátí hodnotu textury v něm, není třeba tedy pamatovat si předem vše (což by např. u trojrozměrných textur znamenalo enormní množství dat). Navíc nemají takto definované textury rozlišení omezené diskrétním rastrem, mohou tedy zobrazovat libovolně malé detaily a zobrazení zůstává přesné a hladké i bez interpolací. Důležitým aspektem je také fakt, že parametrizované funkce generují celou třídu textur.

Typicky používaný způsob vytváření je kombinace několika jednoduchých funkcí, které vytvoří základní vzor, a následné přidání šumu, který dodá realistický vzhled. Přidání šumu většinou nespočívá v prostém přičtení náhodných hodnot do textury, ale v náhodném posunu bodu, pro který texturu počítáme. Jinými slovy, parametrem procedurální funkce pak není právě obarvovaný bod, ale bod o náhodný posun vedle, z čehož plyne jeho jiné, takto náhodně změněné, obarvení.

3.2.1 Využití šumu

Generátor náhodných čísel produkuje bílý šum (mající rovnoměrně zastoupeny všechny frekvence spektra), je tedy zcela chaotický. Pro textury má však hned několik nevýhod. Především není frekvenčně omezený, proto bude docházet k aliasingu, a dále u takovéhoto generování není žádný parametr, kterým bychom ovlivnili výsledek. Pro potřeby texturování totiž potřebujeme generátor parametrizovat tak, aby sice produkoval náhodné hodnoty jednotlivých bodů, ale zároveň aby pro stejné vstupní parametry vracel vždy stejné hodnoty (parametrem je např. právě bod v prostoru).

Na šumové funkce pro účely procedurálního texturování tedy klademe několik hlavních podmínek: závislost na vstupu, frekvenční omezení (známá maximální frekvence a fakt, že při malé

změně vstupního parametru dojde k malé změně i ve výstupní hodnotě), neperiodičnost (nebo alespoň ne na první pohled patrná periodičnost) a zvolený rozsah hodnot (to se ale případně dá přepočítat i později). Existuje tedy několik používaných metod a typů výroby šumu.

3.2.1.1 Mřížkové šumy

Mřížkové šumy jsou nejrychlejší. Za svůj název vděčí své struktuře – jsou popsány mřížkou resp. mřížkami, tedy tabulkou náhodných např. 256 hodnot a permutační tabulkou stejného počtu prvků. Permutační tabulka zajišťuje náhodnost čísel a zároveň vlastnost, že při opakovaném zavolání je vrácena vždy stejná hodnota. Při aplikaci šumu se provádí interpolace mezi hodnotami mřížky známými jen v diskrétním prostoru celočíselných indexů.

Nevýhodou je poměrně velká „mřížkovitá“ pravidelnost výsledku, což se snaží řešit gradientní šum. V uzlech mřížky nejsou přímo hodnoty, ale gradienty textury v daném místě, tedy derivace hodnot. Na základě známých gradientů lze následně lépe interpolovat hodnoty. Nejlepší výsledky se dosáhnou kombinací obou zmíněných přístupů.

Perlin Improved Noise je asi nejrozšířenějším mřížkovým šumem, u kterého nejdříve proběhne gradientní interpolace na základě údajů v mřížkách a poté lineární interpolace výsledných dat [5].

3.2.1.2 Spektrální syntéza

O spektrální syntéze šumu již byla řeč v souvislosti s generováním terénu v kapitole 3.1.1.1.

Metoda využívá faktu, že signál lze převést fourierovou transformací do frekvenční oblasti a následně vrátit zpět inverzní fourierovou transformací. Signál se tedy jednoduchým způsobem (prostá náhodná čísla) generuje ve frekvenční oblasti a následně se převede zpět do časové oblasti, čímž vznikne zajímavý šum.

3.2.1.3 Fraktální sumy

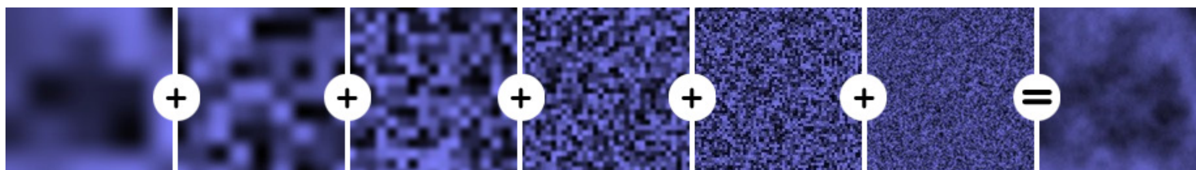
Jedná se o sumu několika šumů v různých frekvencích. Na této myšlence je založen nejznámější šum v počítačové grafice, proto mu věnujeme samostatnou kapitolu – Perlinův šum.

3.2.2 Perlinův šum

Ken Perlin se šumem začal zabývat v 80. letech v souvislosti s prací na vývoji prvního filmu s využitím počítačové grafiky, kterým byl TRON (natočený 1982). Bylo potřeba velkého množství detailních textur, ale nedostatek operační paměti tehdejších počítačů znemožňoval jejich načítání z obrázků. Navíc byly objekty scén reprezentovány objemově. To vedlo autora k vytvoření první šumové funkce plnící trojrozměrný prostor [7].

Stejně jako přírodní jevy má Perlinův šum fraktální povahu – různě jemné složky pro různé úrovně detailů. Vzniká několikanásobným sčítáním stejné šumové funkce s různými frekvencemi,

tzv. *oktáv*. Každá následující přidaná funkce má totiž obvykle poloviční amplitudu a dvojnásobnou frekvenci, stejně jako je tomu u oktáv v hudbě.



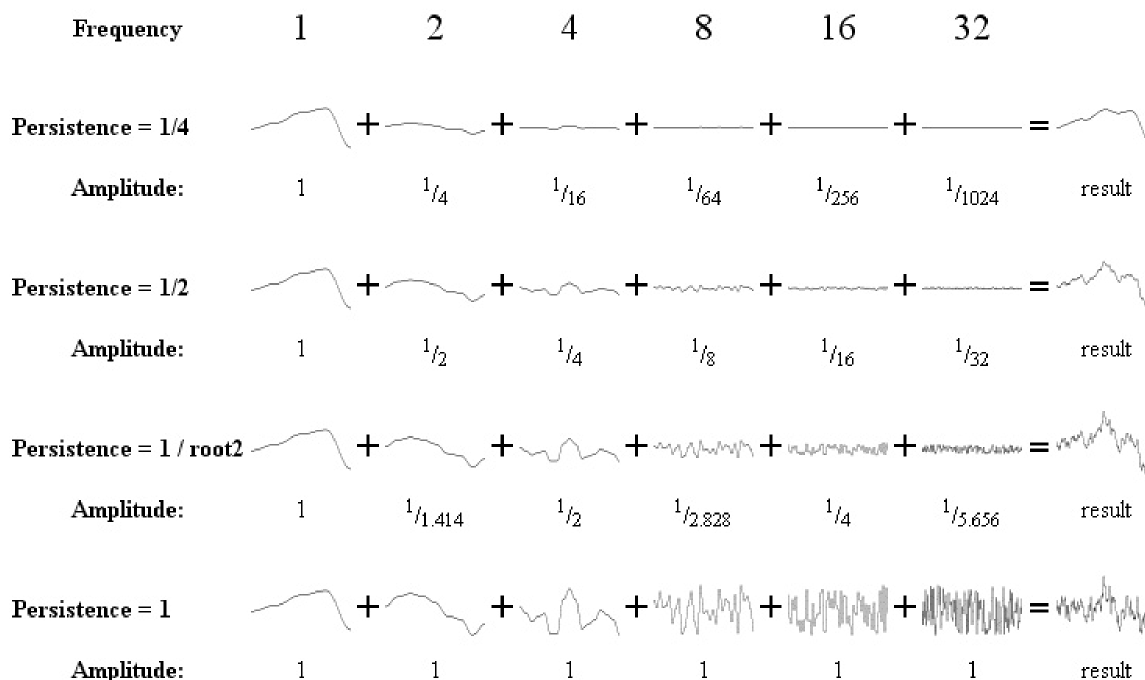
Obrázek 4: Demonstrace skládání Perlinova šumu [6]

Frekvence resp. amplituda však nemusí být pouze dvojnásobná resp. poloviční, můžeme volit i jiné hodnoty a tím určovat odlišný ráz výsledného signálu. Nastavení Perlinova šumu pomocí stanovení amplitud a_i a frekvencí f_i pro jednotlivé oktávy i je zbytečně složité a proto se zavedl parametr *perzistence* p a následující rovnice:

$$f_i = 2^i$$

$$a_i = p^i$$

Stačí tedy pouze specifikovat perzistenci a počet oktáv.



Obrázek 5: Demonstrace vlivu parametru perzistence na výsledný šum [6]

Oktáv můžeme sčítat libovolný počet, čímž dosahujeme stále jemnějších detailů šumu. Při zobrazení výsledku do rastru je jasné, že od určité frekvence jsou změny menší než 1 bod a nemá už cenu další detaily zavádět.

Skládání jednotlivých oktáv můžeme tedy zapsat jako

$$perlin_noise(x,y,z,p,n) = \sum_{i=0}^{n-1} p^i * noise(2^i x, 2^i y, 2^i z)$$

Pokud výpočet šumu v sumě uzavřeme do absolutní hodnoty, získáme turbulenci, která se využívá např. při simulaci mramoru. Viz [1] a [6].

3.3 Vizuální efekty

Jako ve filmu, i grafické intro potřebuje efekty k dokreslení děje. Jsou to především rozmazání, záření, čočkové efekty a další filtry známé z 2D zpracování obrazu. V OpenGL však nebudeme přistupovat přímo k jednotlivým pixelům do obrazového bufferu, abychom filtry aplikovali. Je to příliš pomalé a existují rychlejší řešení nastíněných problémů. Níže uvedený výčet efektů není zdaleka úplný, jedná se spíše o příklady některých nejběžnějších.

Popíšeme také speciální techniky zobrazení přírodních jevů. Modelování některých objektů může být problém – jejich tvar je příliš členitý, nespojitý nebo se rychle mění. Jako příklad lze uvést oheň a výbuchy, mraky jisker, mlha, dým, sníh a déšť či celé ekosystémy. Tyto úkoly řeší *částicové systémy (particle systems)*.

3.3.1 Částicové systémy

Jak již z názvu vyplývá, jedná se o soubor velmi malých prvků (částic), které mají určitou grafickou reprezentaci a jejichž parametry se mění v čase. Mezi vlastnosti takové částice tedy typicky patří poloha, vektor rychlosti a zrychlení, čas života, údaje o vzhledu (barva, tvar) apod.

Částice vznikají v určitém místě, po uplynutí definované délky života buď zanikají nebo emitují další, při pohybu mohou reagovat na kolize s okolím, mohou představovat jiskru (jako obyčejný bod) nebo např. jednoduchý model motýla.

Celý systém po nastavení pak funguje v několika málo krocích. Při každém překreslení se aktualizuje poloha částic, vytvoří se nové částice resp. smažou staré, novým částicím se při vzniku přiřadí vlastnosti odpovídající jejich vzniku a nakonec se celý systém vykreslí.

3.3.2 Radial Blur

Kruhové rozmazání (anglicky *radial blur*) rozostří obraz v kruzích směrem od jeho středu ven. Běžně se používá při úpravách fotografií pro rozmazání pozadí a zdůraznění hlavního objektu. Ve sportovní fotografii také umožňuje přidat dodatečně dynamičnost snímku.

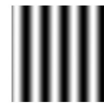
Asi nejslavnější aplikace tohoto filtru můžeme spatřit v seriálu Akta X, kde je použit rozmazání záře kolem tmavé postavy, čímž evokuje jakési vtahování do tunelu, nebo zoom snímku.

Obrázek 6: Ukázka filtru kruhového rozmazání

3.3.3 Plasma

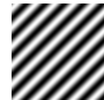
Především ve starých intrech byl velmi populární efekt plasmy. Zde je uveden jen pro úplnost, protože jeho použití nevyužije hardwarovou akceleraci, podobně jako tradiční přístup k 3.3.2.

Efekt je docílen kombinací barevných palet a jednoduchých funkcí parametrizovaných konstantami. Např.



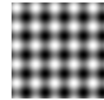
$$\sin(x)$$

pro generování postupných svislých pruhů,



$$\sin(x+y)$$

pro pruhy pod úhlem 45°,

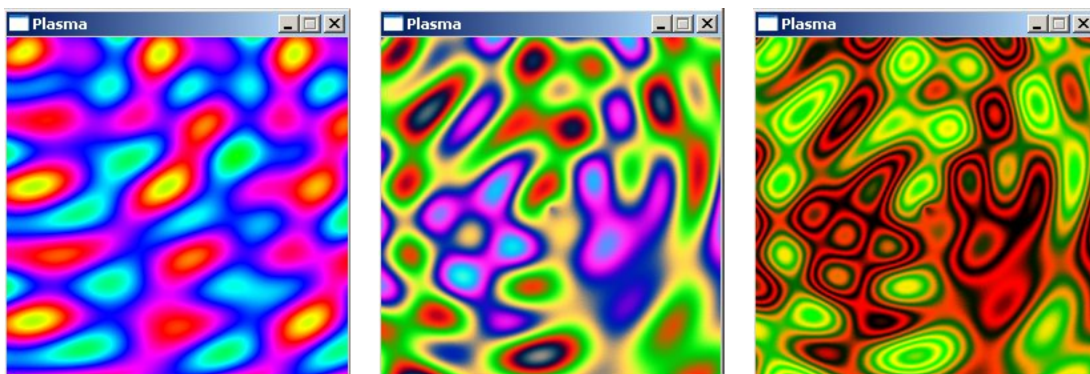


$$\sin(x)+\sin(y)$$

pro mřížku. Vzdálenost bodu od středu obrazu se dá vyjádřit jako

$$\sqrt{(x-\frac{width}{2})^2+(y-\frac{height}{2})^2}\,.$$

Pokud tento vztah použijeme jako argument funkce sinus, dostaneme soustředné kruhy.



Obrázek 7: Ukázka barevné plasmy [9]

Více informací včetně obrazových příkladů a zdrojových kódů lze nalézt v dostupné literatuře na internetu [9].

4 Techniky ozvučení intra

Jak již bylo řečeno výše, grafické intro je multimediální prezentací, potřebuje tedy i zvuk. To jej posouvá z obyčejné animace (běžně viděnou na webových stránkách) blíže k filmovému zážitku a subjektivně určitě ovlivní celkový dojem.

Existuje mnoho zvukových formátů a způsobů komprese, nabízí se tedy otázka, jak a čím tedy intro vlastně ozvučit.

Skladba ve formátu wav má průměrně několik desítek megabajtů, po kompresi mp3 ve srovnatelné kvalitě se dostaneme maximálně na jednotky megabajtů. To je stále o několik řádů větší, než celý zamýšlený program. S drastickou ztrátou kvality při kompresi na 8kbps se velikost mp3 skladby dostane ke stovkám kilobajtů a při pokusu o kompresi se zmenší jen o něco málo. Obecně zakódovat celou písničku pod 100KB pravděpodobně neuspějeme. Proto je potřeba najít vhodnější způsob uchování zvukové stopy.

4.1.1 MIDI

Asi nejstarším, ale hlavně nejjednodušším a paměťově velmi výhodným způsobem je použití MIDI. Jako každé řešení, má i toto své klady a zápory i specifické využití.

Pokud zabrousíme na Wikipedii, konkrétně viz [3], dočteme se, že: „**MIDI** (**Musical Instrument Digital Interface**) je mezinárodní standard používaný v hudebním průmyslu jako elektronický komunikační protokol, který dovoluje moderním digitálním hudebním nástrojům, počítačům i dalším přístrojům komunikovat v reálném čase.“

Tento formát je zjednodušeně založen na takové filosofii, že není potřeba uchovávat kompletní nahrávky skladeb, stačí si pamatovat noty a každý si je už přehraje sám. To s sebou nese znatelnou úsporu místa a možnost kreativního vyžití bez skutečných nástrojů – každý si může poskládat vlastní skladbu s pomocí např. klávesnice a programu a následně si ji přehrát.

V uložené informaci nejsou obsaženy jen noty, ale i nastavení nástrojů, časování a podobná metadata zajišťující vše potřebné pro přehrání na cílovém systému.

Bohužel, slabé místo celého konceptu je právě v cílové interpretaci MIDI dat. Ta do velké míry kvalitativně závisí na tom, jak který hardware data přehraje. Kvalita tedy většinou zdaleka neodpovídá skutečné hudbě a na první poslech je patrné, že se jedná o MIDI skladbu. Při tvorbě intra je kladen důraz na výsledné zpracování a je tedy žádoucí určitá zaručitelná míra kvality.

Poněkud odlišná situace je u grafických inter s omezením do 4KB. Tyto produkty jsou většinou kvůli velmi přísnému omezení úplně bez zvukového doprovodu a MIDI je v podstatě jediný způsob, jak nějaké ozvučení přidat. Proto je v této kategorii použití MIDI zvuku běžné a na rozdíl od větších inter akceptovatelné hodnotící komunitou.

4.1.2 Hudební moduly

Hudební moduly jsou dalším způsobem, jak ozvučit grafické intro, a existují v mnoha různých souborových formátech.

Koncept je pro všechny formáty hudebních modulů stejný. Podobně jako MIDI obsahuje hudební modul informace o nástrojích, časové údaje a nastavení, kdy má co hrát a jakým způsobem. Odstraňuje však základní nedostatek MIDI formátu, kterým je nekonzistentní výstup při přehrávání na různých počítačích.

Hudební modul totiž navíc obsahuje hudební vzorky (*audio samples)* jednotlivých nástrojů, které se využívají k syntéze zvukového výstupu na straně přehrávače. Pokud tedy pomineme různé interpolační chyby a šum, zní skladby vždy stejně na všech systémech.

Hudební moduly, také *tracker modules*, a jejich tvorba, *tracking*, vděčí za své názvy prvnímu programu pro jejich tvorbu, který se jmenuje *Ultimate SoundTracker*, vytvořenému již v roce 1987. On a jeho klony obsahovaly v základní obrazovce uživatelského rozhraní několik nezávisle upravovatelných zvukových stop, *tracks*, odtud tedy *tracker*.

V 90. letech byl tento systém velmi populární a existuje kolem něj celá komunita vývojářů a tvůrců. I v současné době řada dem stále využívá této široké komunity a hudbu zajišťuje tímto způsobem, populárnější se však staly plnohodnotné syntetizéry zvuku.

4.1.3 Syntetizátory

Posledním stupněm v evoluci různých způsobů ozvučení grafických inter je přímá syntéza zvuku. Signál z různých jednoduchých generátorů prochází funkcemi, které jej modifikují. Princip je stejný jako v reálném světě, kdy např. signál ze snímače elektrické kytary prochází několika krabičkami, kde je filtrován, zpožďován apod. Výsledný signál je přiveden na výstup.

Stačí pak mít implementovány v knihovně tyto „krabičky“, znát jejich nastavení a znát parametry generátorů v čase, zbytek se již dopočítává průběžně sám.

Představitelem této skupiny je program Jeskola BUZZ, což je první modulární syntetizér. Jednotlivé moduly (*buzz machines*) představují ony „krabičky“ a „generátory“ a lze jak nalézt velké množství již existujících nebo tvořit vlastní. Program má grafické rozhraní umožňující pohodlné nastavení a propojení jednotlivých komponent. Více informací lze nalézt na odpovídajícím internetovém portálu [10].

Každý modul použitelný v grafickém intru se tedy skládá z *buzz machine* a odpovídající statické knihovny k programu.

5 Nástroje při vývoji intra

5.1 Komprese spustitelných souborů

5.1.1 O co jde

Komprese spustitelných souborů byla populární zejména v dobách malých disků a disketových mechanik. Umožňovala uživatelům šetřit místem a zároveň odbourávala potřebu před každým spuštěním programu provádět ruční dekompresi. Je také jednou z metod, jak ztížit reverse engineering programů, bohužel pomáhá i virům a jinému malware zamaskovat svou přítomnost.

Principiální funkce je podobná samorozbalovacím archivům. Exe soubor je zabalen a je k němu připojen kód pro dekompresi. Ten se aktivuje při spuštění souboru, dojde k rozbalení původního obsahu do paměti a teprve poté ke spuštění programu. Některé nástroje rozbalují namísto do operační paměti na disk.

Kompresní poměr je obvykle výrazně lepší než při použití klasických nástrojů pro pakování, protože tyto nástroje nejsou optimalizované pro spustitelný kód, leč pro data.

Časové prodlevy při výše zmíněných operacích jsou při dnešním výkonu počítačů neznatelné. Na výkon má neblahý vliv swapování při použití vitrtuální paměti, obzvláště, pokud je aplikace spouštěna vícekrát – vždy dojde k nové alokaci paměti pro dekompresi a dochází k zbytečným přesunům dat mezi operační pamětí a diskem, i když velké množství rozbalených dat není aktuálně potřeba a bylo by možné je znovu načíst z archivu.

Problémy mohou nastat také u antivirových kontrol, které mohou spustit planý poplach. Více informací lze nalézt na [11].

5.1.2 Použití v demoscéně

Pro účely využití v počítačových intrech doznaly tzv. *exe packers* jistých změn zajišťujících extrémní výkony v oblasti jak kompresního poměru, tak rychlosti.

Existuje nejmíň na dvě desítky různých programů a jednotlivé z nich jsou specificky určeny pro konkrétní cílovou velikost intra. Nástroj určený pro nejmenší velikosti nelze příliš úspěšně použít pro větší soubory kvůli jeho nárokům závislým na objemu dat. Obecně platí, že čím specifičtější použití, tím vyšší výkon a naopak.

Systémy se tedy liší použitými kompresními algoritmy s ohledem na velikost rozbalovacího kódu přidaného k samotné aplikaci, rychlostí dekomprese, která musí odpovídat reálnému času a množstvím zabrané paměti, kterou daný algoritmus potřebuje ke své práci.

5.2 Demo Tools

5.2.1 Teorie

Pojem *demo tool* je dalším z řady speciálních názvů obyčejných věcí kolem inter a demoscény vůbec. Jedná se v podstatě o nástroj nebo sadu nástrojů pro intuitivní tvorbu dema, tedy nástroj pro grafiky, animátory, zvukaři či scénáristy, pokud se tým skládá z těchto oddělených rolí. Jde hlavně o pohodlné a interaktivní ovládání jednotlivých proměnných celé prezentace. Pracovní prostředí je podle rozsahu a přesného účelu konkrétního nástroje od jednoduchých editačních oken proměnných až po komplexní GUI podobné CAD systémům či 3D grafickým editorům.

Demo Tool umožňuje specializaci jednotlivých tvůrců dema – každý člen týmu nemusí být programátor. V extrémním případě nikdo v týmu nemusí umět programovat a s využitím nástrojů třetích stran mohou vytvořit obstojné intro. Tím už se však dostáváme mimo téma, kterým je nahlížení právě pod pokličku problematiky.

Dalšími výhodami jsou rychlejší sestavení celku podle konkrétních představ díky okamžité odezvě změn a také rychlejší tvorba nových produkcí, protože výpočetní jádro zůstává stejné (případně doznává pouze minimálních změn) a mění se použitá data – modely, animace, hudba.

Každá skupina zabývající se tvorbou na soutěžní úrovni má vlastní sadu nástrojů. Některé jsou přísně střeženým tajemstvím, některé jsou veřejně dostupné, a to např. včetně ukázkových projektů známých dem. Je vhodné mít při kompozici výsledné animace alespoň nějaké základní nástroje, které ušetří mnoho času ladění.

5.2.2 Realizace

Pokud se nejedná o programy na externí zpracování dat, převody formátů apod., má demo tool podobu statické knihovny připojené k projektu. Při ladění pak poskytuje veškerý komfort, následně je odstraněn a ve finální verzi intra nezabírá zbytečně žádné místo.

6 Návrh a realizace aplikace

6.1 Nastavení překladače a linkeru

Prvním krokem při jakékoliv tvorbě programů by mělo být správné nastavení kompilátoru, čímž zajistíme nejen vypisování či potlačení varovných zpráv, kontrolu dodržení norem, ale také správné optimalizace kódu, ať už nám jde o rychlost, nebo o velikost výsledku.

Při psaní intra s omezenou velikostí toto platí dvojnásob a bez správných přepínačů nemá význam se o něco pokoušet. Špatné nastavení má za následek zbytečné úniky cenných kilobajtů, které jsme pracně ušetřili jinde.

Níže popsané záležitosti se týkají Microsoft Visual Studia 2005, principy jsou však obecně platné a jednotlivé překladače a linkery mají řadu shodných nastavení.

6.1.1 Knihovny

Hlavní myšlenkou malé aplikace je nalinkování co nejmenšího počtu komponent. Obecné knihovny obsahují množství funkcí, které v projektu vůbec nevyužijeme.

V základním nastavení je k programům automaticky připojena běhová knihovna *C run-time library (CRT)*, která zabere 30-40 kB, tedy vůči cílové velikosti projektu velmi mnoho. Při jejím vynechání sice přijdeme o několik užitečných funkcí a program potřebuje úpravu pro správný běh, chybějící funkce však není problém dopsat, případně využít ekvivalenty některých z nich z rozhraní WinAPI. Stejné rozhraní se stará spolu s OpenGL i o zobrazení okna, tedy CRT není v podstatě potřeba.

K zákazu automatického připojování všech implicitních knihoven slouží přepínač `/NODEFAULTLIB` (Ignore All Default Libraries) u nastavení parametrů linkeru.

Ihned po vypnutí CRT linker hlásí, že nenalezl funkci *WinMainCRTStartup*. Je to proto, že jsme spolu s knihovnou odstranili vstupní funkci do programu, která měla za úkol inicializovat vše potřebné, spustit konstruktory globálních objektů apod. Teprve odtud by se volala funkce *main()* nebo *WinMain()*. Tento problém lze řešit dvěma způsoby. Buďto je možné explicitně změnit vstupní proceduru programu pomocí přepínače `/ENTRY:function`, nebo si nadefinujeme vlastní funkci shodného prototypu:

```
int WinMainCRTStartup(HINSTANCE hInstance);
```

Více informací na toto téma lze nalézt na odpovídajících stránkách Microsoftu [12], nebo v dalších článcích [13] a [14].

6.1.2 Chybějící funkce

Jak již bylo zmíněno dříve v textu, odpojením CRT knihovny od projektu přijdeme o několik užitečných funkcí. Především budeme potřebovat matematické funkce, jako jsou goniometrické funkce sinus a cosinus, absolutní hodnota, odmocniny a mocniny.

Nejvhodnějším způsobem jejich nahrazení je využití assembleru a zápis přímo odpovídajících instrukcí FPU, například takto:

```
float sqrt(float x)
{
        __asm fld x;  // Push a float number to the stack top
        __asm fsqrt;  // Replace the value on the top with sqrt
        __asm fstp x;// Pop a float number from the stack top
        return x;
}
```

Obdobně lze postupovat v případě dalších matematických funkcí potřebných při výpočtech. Vyčerpávající manuál instrukcí FPU lze nalézt na webových stránkách [16].

Další chybějící funkce se týkají správy paměti – potřebujeme *malloc()* a *free()*. Naštěstí funkce se stejným účelem poskytuje i WinAPI. Přehled některých ekvivalentů ukazuje tabulka 1. Další informace a kompletní seznam ekvivalentních funkcí jsou k nalezení na webu Microsoftu [17].

Standardní funkce	Win32 ekvivalent	Standardní funkce	Win32 ekvivalent
malloc	GlobalAlloc	isalpha	IsCharAlpha
free	GlobalFree	isalnum	IsCharAlphaNumeric
memcpy	CopyMemory	islower	IsCharLower
memset	FillMemory, ZeroMemory	isupper	IsCharUpper
memmove	MoveMemory	fopen	CreateFile
strcpy	lstrcpy	fclose	CloseHandle
strcat	lstrcat	fread	ReadFile
strlen	lstrlen	fwrite	WriteFile
strcmp	lstrcmp	fseek	SetFilePointer
toupper	CharUpper	sprintf	wsprintf
tolower	CharLower	vsprintf	wvsprint

Tabulka 1: Standardní funkce a jejich ekvivalenty ve WinAPI

6.1.3 Sloučení sekcí

Program je vnitřně dělen do několika sekcí – pro data, konstanty, kód. Každá z nich pak podléhá zarovnávání 4kB, případně 512B (viz Optimalizace kódu). Sloučením několika sekcí do jedné dojde k zarovnání pouze jednou a tím k ušetření místa.

Sloučení se provádí jako všechny přepínače zadáním */MERGE:[from]=[to]* do položky Command Line Options, případně pomocí pragma příkazů takto:

```
#pragma comment(linker, "/MERGE:.rdata=.data")
#pragma comment(linker, "/MERGE:.text=.data")
```

Pokud chceme mít čisté výpisy při linkování, můžeme potlačit výpis varování spojených se sloučením sekcí přepínačem */ignore:4254*.

Více informací o tomto tématu lze najít v dokumentu [15].

6.1.4 Optimalizace kódu

Následující čtyři nastavení zajistí hlavní část velikostní optimalizace:

/O1 (Minimize Size) a /Os (Favor Small Code)

Překladač se snaží vyrobit co nejmenší kód na úkor rychlosti při přepisu procesů do strojového kódu, pokud je možnost volby z více variant.

/Zp[1|2|4|8|16] (Struct Member Alignment)

Zarovnání jednotlivých položek struktur je nastaveno na 8B, což může způsobovat problémy při nevhodně velkých datech uložených do struktur ve velkém počtu (v našem případě informace vztahující se k vrcholům modelů).

/OPT:REF (Eliminate Unreferenced Data)

Toto užitečné nastavení za nás automaticky odstraní všechny nevyužité věci v kódu. V Release verzi překladu je již přepínač ve správné poloze.

/OPT:NOWIN98 (Optimize for Windows 98)

Microsoft Visual Studio používá implicitně zarovnání souboru na 4kB. Tímto nastavením můžeme tuto velikost změnit na 512B. Důsledkem bude pouze zpomalený start programu na Windows 98.

6.1.5 Další přepínače

/GS- (Buffer Overruns Check)

Pro kontroly zabraňující přetečení zásobníku a např. přepsání návratové adresy jsou do kódu přidány četné kontroly. Nic takového by se v tomto projektu ale dít nemělo, proto můžeme kontroly vypnout.

/GR- (Run-Time Type Information)

Funkce přepínače se využívá při určování typů objektů za běhu programu, při jejich přetypovávání a polymorfismu. Takové věci nejsou využity a proto nemají význam.

/Ehsc, /Eha

Tyto parametry zapínají kontrolu vyjímek a jeden z nich je přednastaven. V projektu však žádné vyjímky použity nejsou, proto můžeme toto take vypnout.

6.2 Model scény

Hlavní téma tohoto projektu je noční osvětlená metropole. Po vytvoření zobrazovacího jádra aplikace bylo potřeba se rozhodnout, jakým způsobem do programu dostat model velkého a rozmanitého města. Existují dva hlavní přístupy v tvorbě a uchovávání modelů scény v grafickém intru.

První možností je vytvořit objekty v externím programu, zajistit konverzi na vhodný formát a posléze vše uložit do projektu mezi statická data. Tento způsob na jednu stranu umožňuje stoprocentní kontrolu nad modelem, propracované detaily přesně podle požadavků, ovšem na druhou stranu je skladování spojeno s několika problémy.

Zaprvé je potřeba implementovat techniky pro úsporné uložení dat, jak popisuje kapitola 3.1, tedy kombinace pečlivého a vhodného rozdělení jednotlivých datových položek do struktur, změna typové velikosti jednotlivých souřadnic, případně příprava modelu pro aplikaci algoritmu dělení ploch. Pokud máme potřebu většího množství objektů ve scéně, je na zvážení, zda jít touto cestou. Tak se dostáváme k dalšímu problému. Pro generování scény s mnoha podobnými, avšak v zásadě odlišnými objekty musíme řešit jejich výrobu. Ta se dá realizovat buďto výrobou několika základních verzí a jejich následnou parametrizací, nebo naivně ruční zdlouhavou prací, na jejímž výstupu je neúměrné množství dat.

Protože bylo v plánu vytvořit rozlehlou scénu a detaily v nočním šerosvitu nejsou až tak patrné, stačí pro reprezentaci budovy jednoduché tvary a sada textur. Je tedy možné jít druhou cestou tvorby objektů – zadat parametry, hranice a nechat většinu práce na generátoru pseudonáhodných čísel. Velkou výhodou je, že tímto přístupem snadno získáme potřebnou pestrost, které by se jinak dosahovalo jen stěží.

6.2.1 Generátor pseudonáhodných čísel

Tato komponenta řídí většinu vzhledu intra, proto je vhodné se zmínit o její funkci.

Pro generování náhodných čísel na počítači existuje celá řada algoritmů. Plně náhodná čísla se dají získat jedině z fyzikálních dějů okolí, například čas příchodu paketů ze sítě, prodlevy stisků kláves, kolísání otáček disku v důsledku turbulencí apod. Pro jednoduché použití stačí generovat čísla

pseudonáhodná pomocí generátoru, který na základě něčeho výsledky počítá a má určitou (dostatečně velkou) periodu.

Nejčastěji jsou generátory postaveny na principu lineárního kongruentního generátoru (Linear Congruential Generator), který je definovaný následujícím vztahem:

$$x_{i+1} = (ax_i + b) \bmod m ,$$

kde operace mod je zbytek po celočíselném dělení, *a*, *b* a *m* jsou vhodně zvolené konstanty a x_0 je počáteční nastavení generátoru, tzv. *semínko (seed)*.

Výsledkem je množina prvků, pro kterou platí, že

$$0 \le x_i < m ,$$

a proto je pro získání základního generátoru rozsahu <0,1) potřeba vydělit výsledky modulem *m*.

Statistické vlastnosti výstupu jsou dány zvolenými konstantami v rovnici – lze je nalézt v literatuře např. [18] a [19]. Pro úplnost uvádím některé hodnoty v tabulce 2.

Zdroj	a	b	m
RAND	69069	1	2^{32}
Borland C/C++	22695477	1	2^{32}
GNU Compiler Collection	69069	5	2^{32}
Microsoft Visual C/C++	214013	2531011	2^{32}
Park-Miller random num. gen.	16807	0	2^{31}-1

Tabulka 2: Některé hodnoty konstant lineárního kongruentního generátoru pseudonáhodných čísel

Generátor použitý v tomto projektu odpovídá prvnímu řádku tabulky 2. Je jednoduchý, rychlý a má dlouhou periodu 2^{32}. Inicializace je provedena při každém spuštění stejným semínkem, proto jsou výsledná čísla náhodná, ale zároveň vždy stejná. Z toho vyplývá, že i model města, který generátoru využívá, bude vypadat vždy stejně.

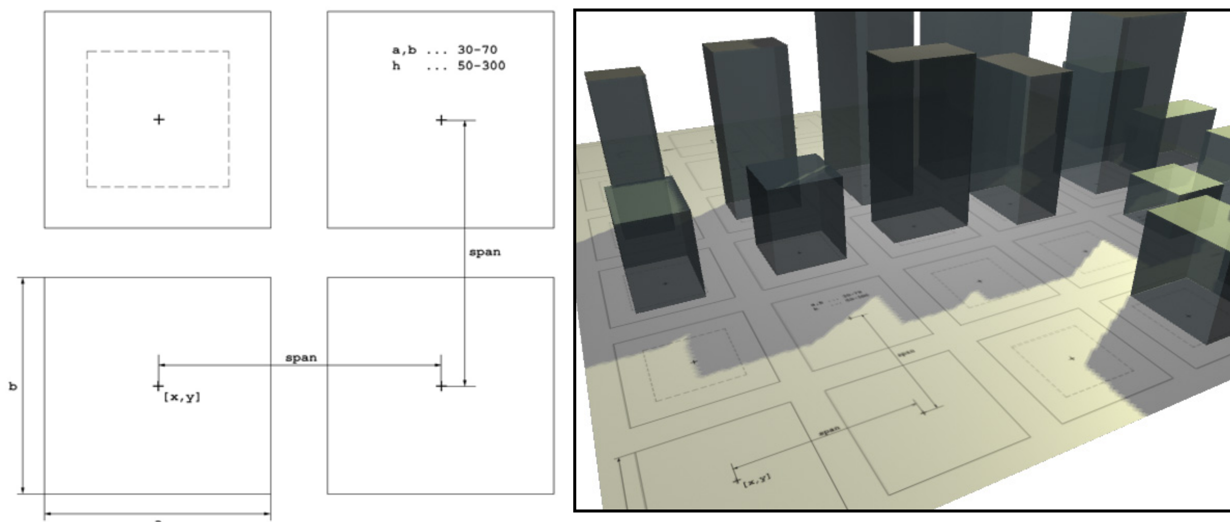
Funkce Random vrací náhodná čísla z intervalu <0,1):

```
static unsigned long x_i = 12345;
double Random(void)
{
      X_i = x_i * 69069L + 1;
      return x_i / ((double)ULONG_MAX + 1);
}
```

6.2.2 Plán města

Město je vytvořeno v pravoúhlé síti ulic podobně jako např. Manhattan. Jednotlivé budovy mají vyčleněn svůj prostor, v rámci něhož se pohybují generované rozměrové parametry. S rostoucí

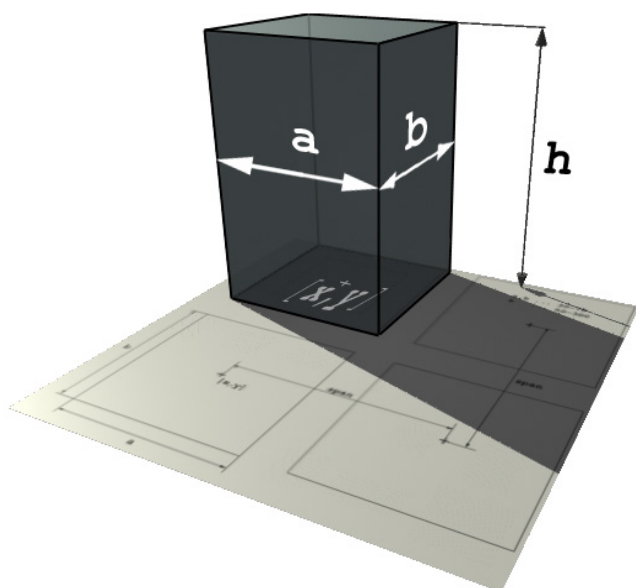
vzdáleností od centra klesá výška budov, aby bylo zachováno klasické panorama s mrakodrapy uprostřed.



Obrázek 8: Ilustrace uspořádání do pravidelné pravoúhlé sítě s velikostním limitem půdorysů budov

6.2.3 Budovy

Budovy ve městě jsou tvořeny jednoduchou texturovanou geometrií.



Obrázek 9: Ilustrace modelu budovy

Základním tvarem budovy je kvádr parametrizovaný výškou, šířkou ve dvou zbývajících osách a pozicí. Další proměnné ovlivňují typ osvětlení (může být rozsvíceno buďto vždy celé patro nebo osamělá okna zvlášť, viz kapitola 6.2.3) a míru osvětlení (tzn. kolik oken resp. pater svítí).

V neposlední řadě se také vybírá vhodná textura fasády domu, která ovlivňuje vzhled jednotlivých oken – jejich tvar, velikost a rozmístění. Na výběr je ze zhruba 7 typů, jak bude dále popsáno v příslušné kapitole o texturách.

Jednotlivé parametry, kromě pozice, jsou ovlivněny generováním pseudonáhodných čísel a dále upraveny do požadovaného rozsahu.

6.2.3.1 Rozsvícená okna

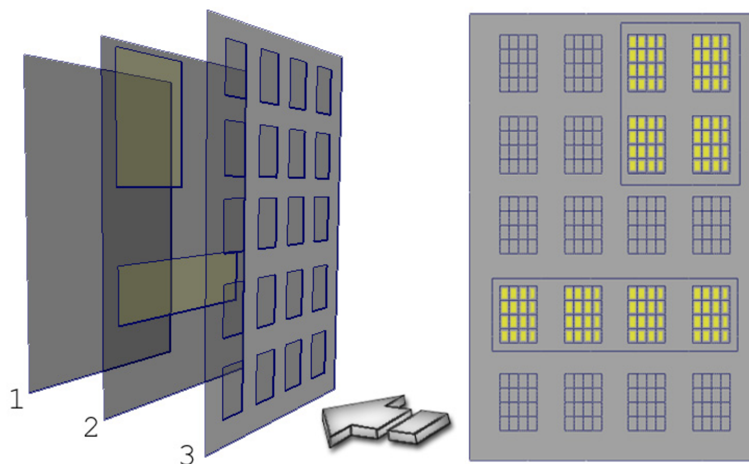
Zajímavým problémem bylo řešení tvorby svítících oken. Nabízí se dva naivní přístupy – okna reprezentovat kompletně texturou, případně kompletně se spolehnout na geometrii.

V prvním případě narazíme záhy na problém, že pro dosažení potřebné pestrosti budov ve městě by bylo potřeba negenerovat velké množství různých textur. Například pokud chceme mít pouhých pět typů tvarů oken, pět různých pozic rozsvícených oken a pět úrovní počtu rozsvícených oken na budovu, dostáváme se k číslu 125, a to je moc.

Pokud se rozhodneme zcela opustit myšlenku textur a začneme okna reprezentovat přímo geometrií, brzy zjistíme, že nárůst polygonů ve scéně je příliš dramatický. Pro každé patro s 10 okny na jednu stranu potřebujeme 40 obdélníků, pokud počet násobíme průměrným počtem pater a počtem budov ve městě, docházíme opět k nevhodně vysokým číslům.

Mé řešení kombinuje oba přístupy do jednoho, a nejen snižuje zátěž snížením počtu nutných polygonů a objem práce s tvorbou textur, ale také zvyšuje možnosti úprav při generování – není omezeno určitým počtem pevně stanovených pozic oken, nevyžaduje opakování stejných vzorů.

Je vykreslen vnější plášť budovy, na kterém je nanesena textura oken. V místě okna je textura průhledná a prosvítá další vrstva, kterou jsou svítící oblasti jakožto pozadí rozsvícených oken. To poskytuje prostor pro rozsvícení najednou celé větší oblasti oken nebo celého patra pouze jedním větším polygonem. Poslední vrstvou je černý podklad, který tvoří výplň temných zhasnutých oken, aby nebylo skrze dům vidět.



Obrázek 10: Ilustrace třívrstvého modelu (vlevo 3D pohled, vpravo pohled po šipce)

Toto řešení tedy umožňuje šetřit polygony při zobrazení velkého množství oken, zároveň poskytuje možnost měnit počet a pozici rozsvícených oken u každého domu bez omezení. Ve výsledku vypadá vše náhodněji a přirozeněji, a to s přijatelným výkonem.

6.3 Generované textury

Jak již bylo zmíněno dříve v textu, ukládání textur mezi statická data nepřipadá vzhledem k velikostnímu omezení aplikace v úvahu. Rychlým výpočtem zjistíme, že obrázek v rozlišení 512x512 pixelů v módu RGBA zabere 1MB. Veškeré potřebné textury je tedy potřeba vytvořit až za běhu programu.

Pro tento projekt jsou důležité hlavně textury zdí budov s různými okny a vytvoření oblohy.

6.3.1 Textury budov

Vzhledem k velkému počtu budov ve scéně jsou veškeré detaily přeneseny do textur. Ty se snaží zajistit víceméně realistický vzhled budov jak při pohledu z dálky, tak při bližším zkoumání. Píšu „víceméně“ proto, že stačí slušný vzhled na první pohled. Pokud je budova zabírána kamerou z blízka, nikdy to není tak dlouho, aby mohl divák detailně sledovat jednotlivé cihly apod.

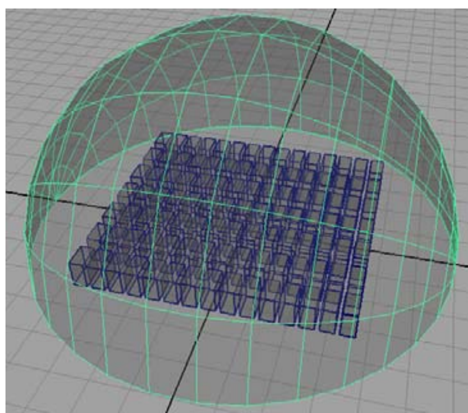
Základem fasády budov je jednoduchá textura cihel. Na té jsou pak naneseny jednotlivé tvary oken včetně jejich korektního rozmístění. Vše je potřeba nějak skloubit s rozsvěcovacím systémem, který je popsán v kapitole 6.2.3.1. Proto jsou textury tvořeny v určitém měřítku, konkrétně jedna pokryje vždy plochu 16x16 metrů plochy budovy. Tím je zajištěna téměř přesná korespondence pater a otvorů oken při aplikaci „světel“ (viz 6.2.3.1).

Spolu s rozměry a osvětlením jsou textury povrchu budov třetím stěžejním pilířem rozmanitosti města.

6.3.2 Noční obloha

Byly dvě možnosti, jak realizovat noční oblohu. Pokud nepočítám úplnou tmu, mohly by na nebi svítit hvězdy. Pro větší dramatičnost však nad městem v animaci letí mraky. Tento zdánlivý detail je výsledkem delšího procesu.

Skydome



Obrázek 11: Skydome kolem scény

Při aplikaci „pozadí“ v 3D scénách, tedy např. nevýznamné okolí dějiště nějaké hry nebo obloha, se využívá mapování textur s tímto motivem na geometrický útvar. Tím může být krychle nebo koule resp. polokoule. Celou scénu pak tento objekt obepíná a na něm je nanesen obrázek.

I když se na první pohled zdá, že 6 stěn krychle bude výhodnější, brzy zjistíme, že je pro správné

zobrazení nutné upravit a deformovat textury tak, aby nebyly vidět hrany mezi jednotlivými stěnami.

Navíc je nastaven gradient opacity tak, že textura směrem od nejvyššího místa k obzoru bledne až do průhlednosti. Mraky tedy plynou ze tmy opět do ztracena. To by také muselo být ošetřeno u krychle jinak, vzhledem k různé vzdálenosti jednotlivých úseků stěn od pozorovatele. Zvolil jsem tedy polokouli.

Mraky

Textura mraků je vytvořena na principu perlinova šumu, o němž již byla zmínka v kapitole 3.2.2.

Místo výpočtů každého pixelu „ručně“ podle uvedených vztahů, tzn. interpolace a sčítání, využijeme grafického hardware, aby vše udělal za nás. Způsob, jakým je toho docíleno, je prostý:

- Začneme nagenerováním nejvyšší frekvence šumu do textury.
- Texturu na skydome nanášíme několikrát po sobě se zapnutým mícháním barev.
- Při nanášení každé další vrstvy vždy redukujeme průhlednost na polovinu předešlé.
- Při nanášení každé další vrstvy také měníme souřadnice textury tak, abychom ji natáhli na dvojnásobnou velikost. Zároveň máme zapnutou automatickou lineární interpolaci textur.
- Začínáme s původní velikostí textury (tedy s nejvyšší frekvencí šumu) a s nejvyšší průhledností (tedy nejnižším přírůstkem při pozdějším sčítání).

Při uvedeném procesu se textura roztahuje, čímž se snižuje frekvence šumu na ní, zároveň dochází k hardwarové interpolaci, tedy se šum zahlazuje přesně tak, jak má. Jednotlivé vrstvy mají různou průhlednost, což odpovídá amplitudě jednotlivých frekvencí ve sčítací rovnici u perlinova šumu (viz 3.2.2). O sčítání všech barev se stará hardwarová jednotka odpovědná za míchání barev.

Obrázek 12: Mraky nad městem přecházejí směrem k obzoru do ztracena

6.4 Animace

Hlavním úkolem intra je: „Aby se tam něco hýbalo!“ V tomto projektu létá nejvíce kamera, ale kolem ní se dějí i další události, jako zatmívání při přechodech střihu, zobrazování textu apod. Vše se musí řídit časem, a to je první problém, který je potřeba řešit. Dále je nutné někde uchovat záznam o jednotlivých kamerách a událostech.

6.4.1 Časovač

Čas může při nevhodné implementaci na každém počítači plynout jinak rychle. Pokud se spolehneme na fixní konstantu mezi dvěma voláními vykreslovací funkce na jednom stroji, jinde toto platit nemusí. Pak je sice vykresleno každé pomyslné filmové okýnko, ovšem mění se rychlost pohybujících se věcí v reálném čase v závislosti na výkonu počítače, a to nechceme.

Proto je potřeba implementovat vlastní časový čítač, kterého se budeme ptát, kolik času že to uplynulo od posledního volání funkce, a podle naměřených hodnot pak upravíme míru předem zvolených relativních posunů či rotací objektů za sekundu. Tímto se docílí toho, že budou posuny objektů v animaci za určitý čas vždy konstantní, tedy nebudou záviset na rychlosti počítače ani aktuální zátěži procesoru.

Implementace časovače včetně jednoduché animace může vypadat například takto:

```
void AnimUpdate()
{
      static double lastTime =-1.0; // cas predchoziho volani funkce
      double deltaT;                // cas od minuleho volani funkce

      // INICIALIZACE
      if (lastTime == -1.0) {       // prvotni inicializace
            lastTime = getTime();   // vezmi aktualni cas
            deltaT = 0.0;
            Globals::time_start = lastTime;
            return;
      }

      // pocitej cas od predchoziho volani funkce
      double currentTime = getTime();     // vezmi aktualni cas
      deltaT = currentTime - lastTime;    // spocitej deltu
      lastTime = currentTime;             // nastav novy lastTime

      ...
      // animace:
      float speed = 0.01;      // vysledna velicina = rychlost * deltaT
      myCubeTranslate += speed * deltaT;
      return;
}
```

6.4.2 Kamery a události

Pro uchování trajektorií kamer jsem zvolil sadu úseček a jednosměrně vázaný spojový seznam. Jedná se o poměrně jednoduchý aparát umožňující zadat postupně pohyb kamer v průběhu celé animace – vždy počáteční a koncovou polohou kamery a bodu, na který se dívá, plus odpovídající časy. Mezi jednotlivými klíčovými hodnotami funguje lineární interpolace.

Výhoda řešení je hlavně jednoduchost, přímočarost funkce, přesné polohování kamery v konkrétním zajímavém čase a pohodlné přidávání. Nevýhodou je pak mírně nepřirozený pohyb při ostré změně směru průletu, což lze obejít přidáním většího množství úseček na takovou zatáčku, čímž se průlet vyhladí.

Pro uchování seznamu kamer jsou použity tyto struktury:

```
typedef struct tcamerapath
{
	Vector3f start;
	Vector3f end;
} tCameraPath;

typedef struct vector3f
{
	float X;
	float Y;
	float Z;
} Vector3f;
```

```
typedef struct tcamera
{
	tCameraPath eyePath;
	tCameraPath centerPath;
	Vector3f eye;
	Vector3f center;
	Vector3f up;
	double startTime;
	double endTime;
	(struct tcamera)* next;
} tCamera;
```

Události jsou rozděleny na více typů a uloženy obdobně jako kamery. Kromě typu a příslušných parametrů uchovávají pak pouze okamžik, ve kterém se mají stát. Při animaci se při překreslování scén testuje, zda nedošlo ke změně kamery nebo nějaké události. Pokud ano, je tato akce realizována a příslušný záznam je vyřazen ze seznamu. Tak nemůže dojít k „propásnutí“ nějakého časového okamžiku – pokud již před několika milisekundami čas další plánované akce uplynul, akce se stejně spustí, protože se podle seznamu ví, že ještě neproběhla.

6.5 Efekty

Rozhodl jsem se vyčlenit samostatnou kapitolu věnující se záležitostem, které jinak příliš nezapadají ani do jedné kategorie již zmíněných aspektů intra, ale o kterých by bylo vhodné se zmínit.

Iluze pohybu spoléhá na nedokonalost lidského oka, které při dostatečné frekvenci přestane vnímat jednotlivé obrázky a mozek vytvoří z příchozích dat animaci. Je tedy stěžejní dosáhnout dostatečného počtu překreslení za sekundu, ideálně 30 snímků (např. televize v normě PAL vysílá 25, i při 20 ještě pozorovatel nepozná rozdíl). Starší kamery sice údajně používaly nižší počet snímků za vteřinu, někdy 16, ruční kamery dokonce i kolem 10, avšak pokud klesne počet snímků pod 15, nedá se již opravdu hovořit o plynulé animaci, je tedy nutné mít na paměti rychlost překreslování. Ta trpí hlavně používáním efektů, které je potřeba většinou vypočítat pro každý snímek zvlášť.

6.5.1 Text

K zobrazení textu je potřeba vytvořit nejdříve příslušný font, který je potřeba dále nějak předat do vykreslovacího řetězce ve formě OpenGL příkazů. Naštěstí existuje skupina funkcí ve Windows implementaci OpenGL, která se tomuto problému věnuje.

K vytvoření fontu slouží funkce CreateFont(), jejíž specifikace vypadá následovně:

```
HFONT CreateFont(
    int nHeight,                    // height of font
    int nWidth,                     // average character width
    int nEscapement,                // angle of escapement
    int nOrientation,               // base-line orientation angle
    int fnWeight,                   // font weight
    DWORD fdwItalic,                // italic attribute option
    DWORD fdwUnderline,             // underline attribute option
    DWORD fdwStrikeOut,             // strikeout attribute option
    DWORD fdwCharSet,               // character set identifier
    DWORD fdwOutputPrecision,       // output precision
    DWORD fdwClipPrecision,         // clipping precision
    DWORD fdwQuality,               // output quality
    DWORD fdwPitchAndFamily,        // pitch and family
    LPCTSTR lpszFace                // typeface name
);
```

Jak víme, v OpenGL je obvyklá praxe taková, že font je představován sadou zobrazovacích seznamů, tzv. *displaylistů*. Pro každé písmeno je vytvořen jeden. Můžeme pak pro výpis řetězce s výhodou využít funkce:

```
void glCallLists(GLsizei n, GLenum type, const GLvoid *lists);
```

Další krásnou vlastností tohoto řešení je fakt, že displaylist může obsahovat jakoukoliv sadu příkazů, tedy je v podstatě jedno, jak font vypadá, nebo jestli kreslíme 2D nápis do zobrazovací roviny či 3D písmena do scény.

K vytvoření toho správného zobrazovacího seznamu nám poslouží právě zmiňovaná sada funkcí WGL. Seznam všech funkcí, jak jej ukazuje tabulka 3, včetně jejich popisu lze nalézt v dokumentaci Microsoft Developer Network.

wglCopyContext	wglGetProcAddress
wglCreateContext	wglMakeCurrent
wglCreateLayerContext	wglRealizeLayerPalette
wglDeleteContext	wglSetLayerPaletteEntries
wglDescribeLayerPlane	wglShareLists
wglGetCurrentContext	wglSwapLayerBuffers
wglGetCurrentDC	wglUseFontBitmaps
wglGetLayerPaletteEntries	wglUseFontOutlines

Tabulka 3: Seznam funkcí WGL

Konkrétně nás zajímají tyto dvě funkce:

```
BOOL wglUseFontBitmaps(
      HDC   hdc,          // device context whose font will be used
      DWORD   first,      // glyph that is the first of a run of glyphs to
                          // be turned into bitmap display lists
      DWORD   count,      // number of glyphs to be turned into display lists
      DWORD   listBase    // specifies starting display list
);
BOOL wglUseFontOutlines(
      HDC   hdc,          // device context of the outline font
      DWORD   first,      // first glyph to be turned into a display list
      DWORD   count,      // number of glyphs to be turned into display lists
      DWORD   listBase,   // specifies the starting display list
      FLOAT   deviation,  // specifies the maximum chordal deviation from
                          // the true outlines
      FLOAT   extrusion,  // extrusion value in the negative z direction
      int   format,       // specifies line segments or polygons in
                          // display lists
      LPGLYPHMETRICSFLOAT   lpgmf // buffer to receive glyph metric data
);
```

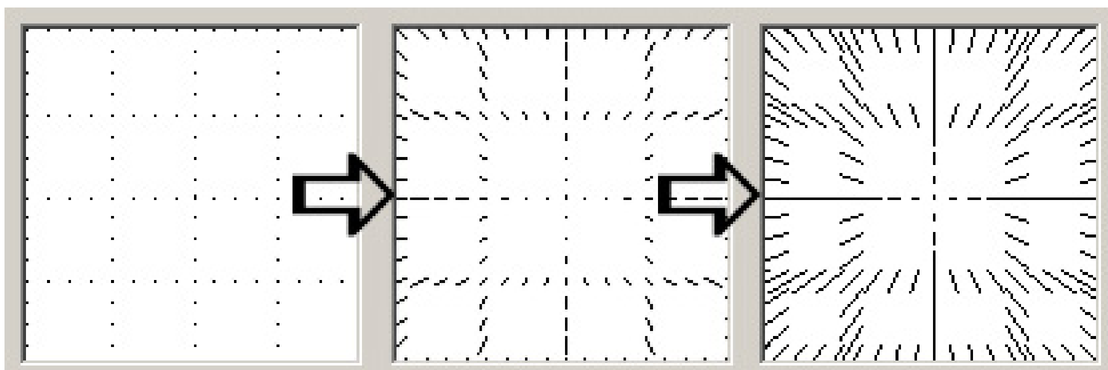
Obě funkce pracují nad aktuálním device-contextem a aktuálně zvoleným fontem. První zmiňovaná vytvoří zobrazovací seznam s bitmapami jednotlivých znaků, druhá pak s 3D reprezentací jednotlivých znaků se souřadnicemi vrcholů v plovoucí čárce. Druhá funkce, která nás bude zajímat v projektu více, pracuje pouze s TrueType fonty.

Protože zmíněné funkce generující zobrazovací seznamy pracují s aktuálním fontem, musíme tento font při použití většinou změnit na námi požadovaný. Proto je vhodné, abychom ten stávající uschovali a po provedení všech akcí opět vrátili zpět:

```
HFONT oldFont = (HFONT)SelectObject(Globals::hDC, newFont);
...
SelectObject(Globals::hDC, oldFont);
```

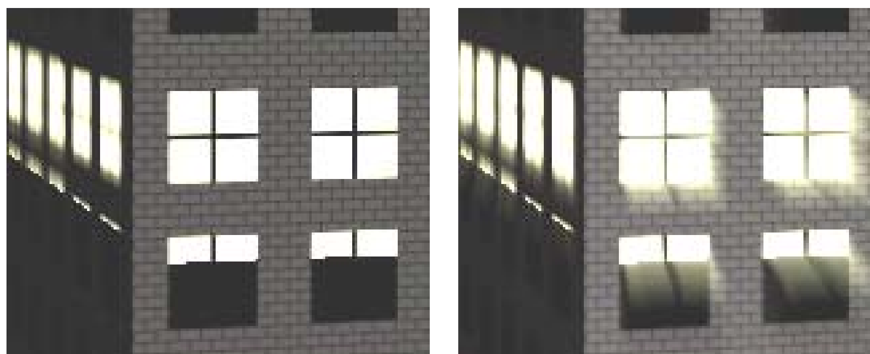
6.5.2 Kruhové rozmazání

O principu kruhového rozmazání již byla řeč v kapitole 3.3.2. V intru tento efekt nebudeme využívat k oddělování pozadí od hlavního motivu, což je jeho obvyklé použití, ale pomůže nám při vykreslování rozsvícených oken.



Obrázek 13: Funkce kruhového rozmazání

Obrázek 13 ukazuje klasickou funkci filtru. Můžeme si všimnout, že směr paprsků se velmi podobá světlu, jak bychom jej očekávali vycházet ze svítícího okna budovy. Použijeme tedy tohoto principu, aplikujeme filtr pouze na části obrazu, kde se rozsvícená okna vyskytují a výsledkem bude velmi pěkný efekt záře směrem z pokojů na ulici.



Obrázek 14: Kruhové rozmazání aplikované pouze na rozsvícená okna (aktivní vpravo)

Hardwarová akcelerace nám příliš nepomůže, kdybychom problém řešili tradiční cestou – filtrací 2D dat obrazového bufferu. Proto využijeme několika vlastností OpenGL k malému triku.

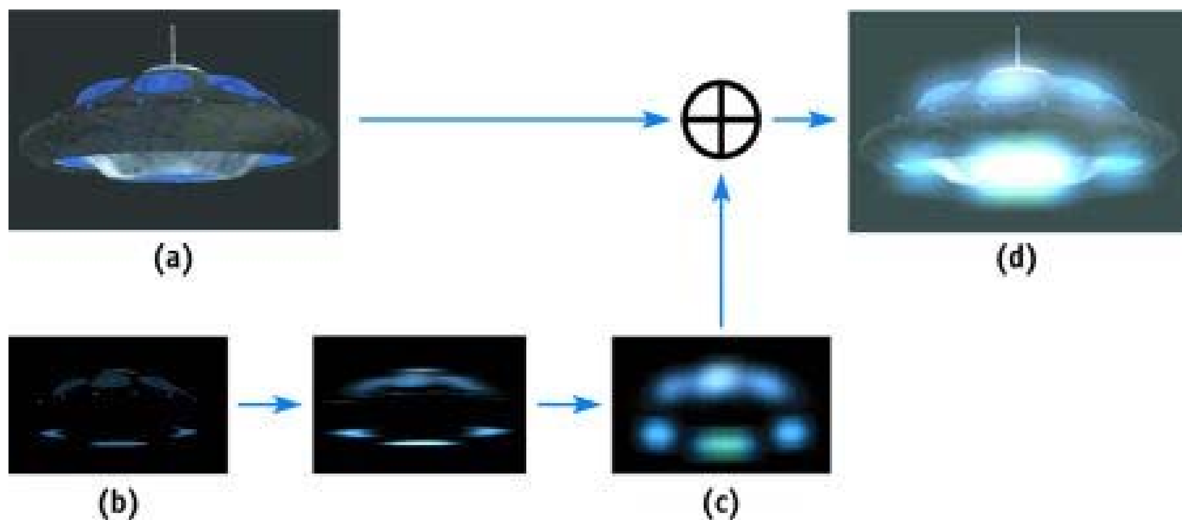
Scénu si vykreslíme do textury, kterou podvzorkujeme a namapujeme na sprite velikosti původní obrazovky. Textura se přirozeně roztáhne na námi požadovanou velikost a implicitní lineární interpolace se postará o rozmazání. Podvzorkování je realizováno prostým dočasným zmenšením rozlišení obrazovky, což vede k renderování menšího obrazu.

Tento postup opakujeme několikrát pro různé podvzorkování, nakonec zobrazíme jednotlivé sprity s texturami a sníženou opacitou, čímž zajistíme jejich smíchání. Vykreslíme objekt a výsledkem je iluze kruhového rozmazání scény. Čím více kroků použijeme, tím hladší je samozřejmě celkový dojem efektu.

6.5.3 Záře

Při zobrazení scén na počítači je množství světla vnímané okem limitováno svítivostí obrazovky, takže jediným způsobem, jak vnímat silné zdroje světla je podle jejich záře. Tu je potřeba vyrobit „na zakázku“ uměle, protože na rozdíl od reálného světa, kde jsou podobné záře způsobeny odrazy světla a jeho rozptylem, v OpenGL nic podobného nenajdeme. Objekt, kolem kterého je záře, automaticky vnímáme jako světlejší. Proto se tento efekt výborně hodí na zdůraznění osvětlení, pohledy skrze prosluněné okno nebo na záři neonů velkoměsta.

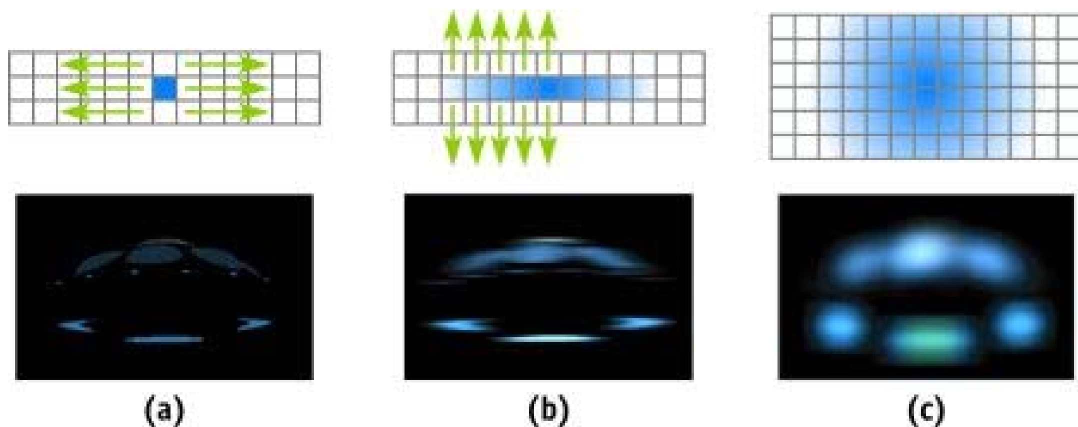
Efekt záře můžeme spatřit v mnoha počítačových hrách (např. Sprinter Cell, Halo 2) a také filmech (např. již zmiňovaný Tron, nebo Hledá se Nemo).



Obrázek 15: Kroky k vytvoření real-time efektu záře [20]

Princip efektu popisuje názorně obrázek 15. Scéna je vykreslena normálním způsobem (a), zároveň je vykreslena tak, že jsou vidět jen zdroje záření (b). Ty jsou pak rozmazány (c) a výsledek je sloučen s původním obrázkem. Vzniká výsledný obraz s efektem záře (d).

K rozostření lze použít standardní metody, jako konvoluční filtr. Ten se však v OpenGL objevuje pouze v rámci rozšíření v GL_ARB_imaging. Protože jsem se chtěl vyhnout co nejvíce omezujícím rozšířením, rozhodl jsem se použít jinou metodu rozmazání. Spočívá v postupném roztažení obrazu ve dvou směrech, čímž se napodobí gausovské rozmazání, jak ukazuje obrázek 16. Nejdříve je provedeno rozmazání v jedné ose (a), dočasný mezivýsledek je pak rozmazán v druhé ose (b), čímž vznikne kýžený produkt (c).

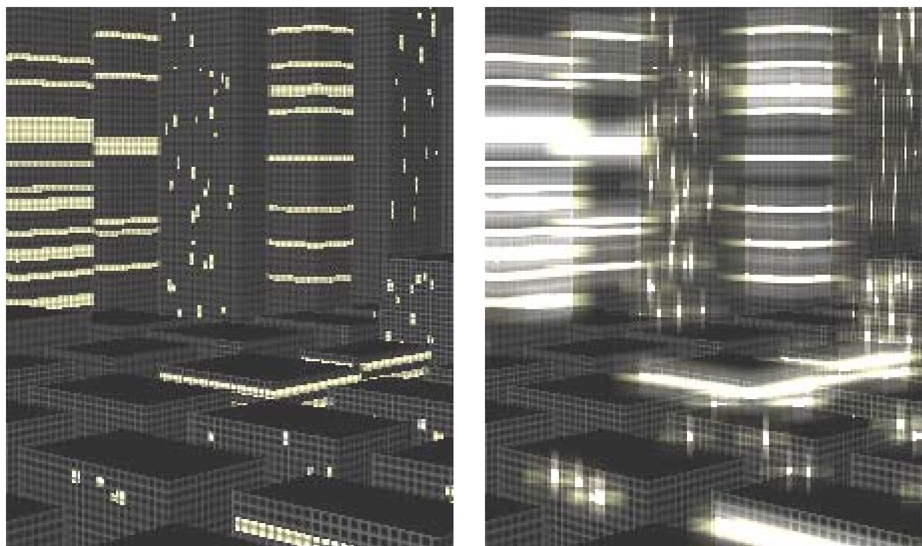


Obrázek 16: Rozostření záře [20]

V intru bylo potřeba efekt aplikovat na světla ve městě. Výhodou je, že mohu vykreslit pouze světla, a tím přeskočit problémy s detekcí světlých oblastí obrazu pro účely tvorby záře. Využívám tedy renderingu do textury, kterou následně roztáhnu podle výše zmíněných principů a vše smíchám s původním obrazem za použití rovnice

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE);
```

Výsledkem je scéna, která vypadá velmi energicky. Světla září do noci s téměř komiksovou intenzitou, a to je přesně ten efekt, který diváka upoutá svou zajímavostí při pohledech na město z dálky.



Obrázek 17: Efekt záře (aktivní vpravo)

6.5.4 Vodní hladina

Při sledování města z dálky se člověk neubrání otázce, co je mezi městem a pozorovacím místem. Odpověď nabídneme ve formě odrazu scény ve vodní hladině. Efekt je to jednoduchý, vyzkoušeli jsme si jej již v Základech počítačové grafiky na konci druhého ročníku, avšak vyžádá si určitý výkon navíc vzhledem k tomu, že je potřeba scénu vykreslovat dvakrát.

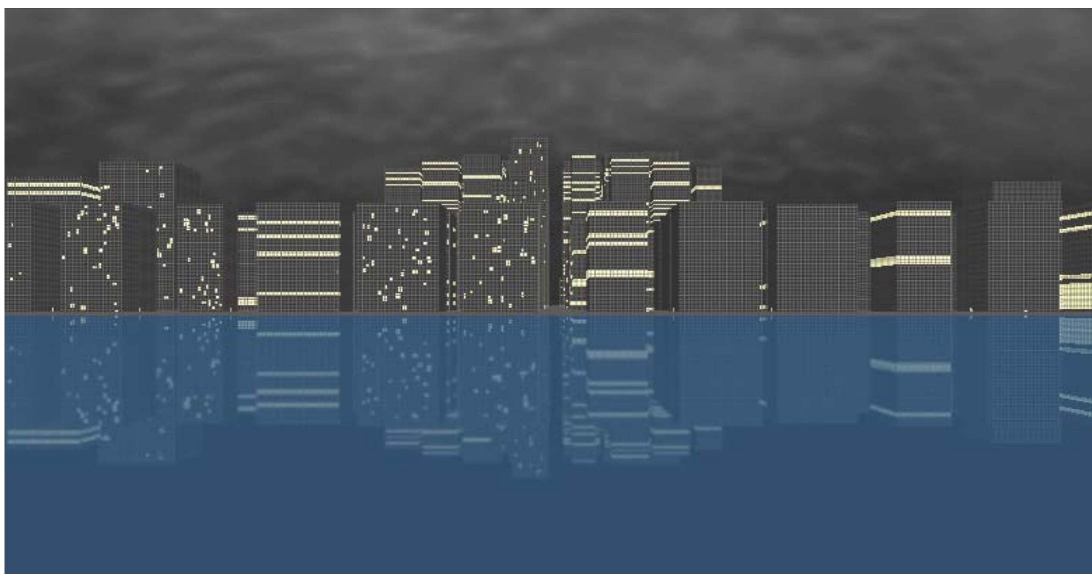
Vše probíhá v několika krocích. Nejdříve vykreslíme odlesk na hladině, poté hladinu a nakonec hlaví scénu. V prvním kroku tedy musíme převrátit kameru podle vodorovné roviny, čehož snadno dosáhneme příkazem

```
glScalef(1.0f, -1.0f, 1.0f);
```

Pro případ, že by něco při tomto vykreslení vyčnívalo nad zamýšlenou hladinu, nastavíme ořezovou rovinu na rovinu hladiny

```
double eqr[] = {0.0f,-1.0f, 0.0f, 0.0f};
glClipPlane(GL_CLIP_PLANE0, eqr);
glEnable(GL_CLIP_PLANE0);
```

Pak stačí jen vrátit převrácený pohled zpět do normálu, vykreslit poloprůhlednou vodní hladinu, vypnout ořezy a vykreslit skutečnou scénu.



Obrázek 18: Výsledek po zobrazení vodní hladiny

6.6 Hudba

Hudební doprovod pouze dokresluje atmosféru celého projektu, není však hlavním předmětem této práce. Proto jsem pro přehrávání hudby použil syntetizér skupiny *Farbrausch* s názvem *V2 Synthesizer System*, aktuálně ve verzi 1.5, který byl použit v jejich slavných intrech *.the .product*, *poemtoahorse*, *Candytron*, *Flybye* a dalších produkcích.

Syntetizér je dodáván ve formě statické knihovny spolu s pluginy VSTi, Buzz Machine a rozšířením přehrávače WinAMP. Obsahuje kromě hudebních modulů také přístup k hlasovému syntetizéru Windows.

Bohužel není dostupná téměř žádná dokumentace, jak bývá v produkci z oblasti demoscény špatným zvykem, pouze lze najít ukázku použití, z níž je i skladba hrající v mé animaci. Jejím autorem jakož i autorem systému je Tammo "kb" Hinrichs.

6.7 Komprese výsledného programu

Finální produkt je hotov, ovšem při pohledu na velikost výsledku nemůžeme být spokojeni. S připojenou knihovnou pro zvuk a skladbou je výsledná velikost po všech optimalizacích **2 719 744 B**. Na první pohled to může být poněkud šokující zjištění, není však důvod k panice.

Jak již bylo popsáno v kapitole 5.1, existují kompresní nástroje pro spustitelné soubory, které cíl zabalí a přitom nechají transparentně spustitelný jako dříve. Cenou je určitý výkon procesoru a zabraná určitá část operační paměti.

Protože je nástrojů větší množství, každý bývá určen pro jiný druh cílových aplikací. Obecně se dá říci, že čím obecněji navržený systém, tím horší kompresní poměr. To je spojeno s faktem, že je

velký rozdíl v softwaru specializovaném na řádově několikabajtové aplikace, několikakilobajtové či megabajtové. Především se liší použité algoritmy, ale také velikost dekompresního kódu. U programu, který má 1,5MB si 60kB přidaného „rozbalovače“ ani nevšimneme, ale u intra, které má být omezeno 64kB je to téměř 100 % jeho vlastní velikosti.

Udělal jsem srovnání několika dostupných systémů a výsledky ukazuje tabulka 4. Na prvních místech podle očekávání končí systémy, které byly přímo vytvářeny pro oblast grafických inter s omezením 64kB a jsou proto na tuto velikost souborů navrženy a vyladěny. Ani nejznámější univerzální systém, kterým je UPX, nezůstává o mnoho pozadu (o 4608B více než vítěz), v relativním měřítku k výsledku z kkrunchy je však o celých 24 % větší.

Název komprimačního nástroje	Výsledná velikost (B)
kkrunchy (v0.23alpha)	18944
MEW (v11 SE)	20677
FSG [Fast Small Good] (v2.0)	21265
XPack (v0.97)	21962
UPX - the Ultimate Packer for eXecutables (v3.03)	23552
PECompact (v2.82)	25088
ASPack (v2.12)	36352

Tabulka 4: Srovnání EXE kompresorů – výchozí velikost: 2 719 744 B

Na závěr přidávám pro zajímavost v tabulce 5 rozbor velikostí hlavních komponent projektu, a to z hlediska celkové velikosti výsledného spustitelného souboru (vždy pro komprimovaný a nekomprimovaný soubor) a v posledním řádku z hlediska zabrané operační paměti. Údaje jsou svou přesností spíše orientačního charakteru, což je také význam tohoto měření – dát čtenáři představu o tom, „co co stojí“ v oblasti paměti.

Procentuální vyjádření je mírně zkreslené nepoměrem velikostí knihovny V2 a zbytku programu. Je velmi zajímavé sledovat na jednotlivých součástech, jak dobře se jsou schopny zabalit. Z hodnot vyplývá, že s větším množstvím statických dat, ale vhodně uspořádaných, nemají kompresní algoritmy žádný problém a dokáží téměř zázraky.

Velikost součástí		Správa okna	Knihovna V2	Hudební data	Generátory textur a modelů
Nepakovaný	(kB)	3,5	2615	25,5	8
	(podíl)	0,13 %	98,6 %	0,96 %	0,3 %
Pakovaný	(kB)	5,5	7	1,5	3
	(podíl)	32,4 %	41,2 %	8,8 %	17,6 %
Operační paměť	(kB)	5640	192	3524	4948
	(podíl)	39,4 %	1,3 %	24,6 %	34,6 %

Tabulka 5: Velikosti hlavních komponent projektu na disku v nepakované a pakované podobě a dále v operační paměti. Údaje jsou vyjádřeny v kilobajtech a v procentech celkově zabrané paměti aplikací

Závěr

Předchozí text seznamuje čtenáře s fenoménem grafického intra s omezenou velikostí. Pojednává uceleně o podstatě problematiky, zmiňuje se krátce o historii a dále popisuje vybrané techniky a principy, které byly využity v implementaci praktické části.

K danému tématu v podstatě neexistuje tištěná literatura, jediná kniha pojednávající konkrétně o demoscéně, *Demoscene: The Art of Real-Time*, je stará 8 let, což v této rychle se rozvíjející oblasti představuje období srovnatelné se středověkem dějin. Proto bylo potřeba přečíst velké množství pramenů na internetu, uvedené algoritmy pak ověřovat v knihách věnujících se OpenGL či počítačové grafice obecně. Pro práci však byly z velké většiny použity články z univerzitního prostředí, konferencí, případně odborných internetových portálů, což zajišťuje dostatečnou úroveň, která se na internetu jinak hledá těžko.

Při řešení práce jsem prostudoval knihovnu OpenGL a její nadstavby. Projekt částečně navazuje na znalosti získané v několika grafických předmětech v průběhu studia, zejména „Počítačová grafika“ a „Pokročilá počítačová grafika“ (přehled o funkcích a omezeních OpenGL) a „Výtvarná informatika“ (přehled o technikách tvorby digitálního umění).

Další pokračování projektu by mohlo mít podobu vytvoření jiného motivu a příběhu animace na existujícím výpočetním jádře, případně by šlo na tomto jádře postavit grafické uživatelské rozhraní tak, aby aplikaci mohli upravit lidé jiných oborů bez znalostí programování – grafici, hudebníci, animátoři.

Výsledkem je kromě této technické zprávy také ukázková aplikace splňující všechny náležitosti grafického intra – jak velikostní limit, tak multimediální animovaný obsah. Na základě práce byl rovněž vypracován soutěžní příspěvek do Student EEICT 2008, který byl vydán ve sborníku 14. ročníku uvedené konference.

Literatura

[1] ŽÁRA J., BENEŠ B., SOCHOR J., FELKEL P. *Moderní počítačová grafika*. Brno: Computer Press, a.s., 2004.

[2] SHREINER D., WOO M., NEIDER J., DAVIS T. *OpenGL Průvodce programátora*. Překlad Jiří Fadrný. Brno: Computer Press, a.s., 2006.

[3] *Musical Instrument Digital Interface*. Technické specifikace a informace. MIDI Manufacturers Association, 1995-2008.

[4] GAVIN S., MILLER. P. *The Definition and Rendering of Terrain Maps*. Computer Graphics (Proceedings of SIGGRAPH 86), strana 39-48, 1986.

[5] PERLIN, K. *Improving Noise*. ACM Transactions on Graphics (SIGGRAPH 2002), svazek 21(číslo 3), 681-682, červenec 2002.

[6] ELIAS, H. *Perlin Noise*. Dokument dostupný na URL http://freespace.virgin.net/hugo.elias/models/m_perlin.htm (květen 2008).

[7] PERLIN, K. *Making Noise*. GDCHardCore gamers workshop, San Francisco, prosinec 1999.

[8] *Parties*. Archiv odborného portálu pouët.net, 2000-2008.

[9] VANDEVENNE, L. *Plasma*. 2004. Dokument dostupný na URL http://student.kuleuven.be/~m0216922/CG/plasma.html (květen 2008)

[10] *Buzz machines –What is BUZZ?*. Dokument dostupný na webu produktu http://www.buzzmachines.com/whatisbuzz.php (květen 2008)

[11] *Executable compression*. Dokument dostupný na URL http://en.wikipedia.org/wiki/Executable_compression (květen 2008)

[12] PIETREK M. *Under The Hood*. Microsoft Systems Journal, říjen 1996.

[13] MINTUS P. *Creating Small Win32 Executables - Fast Builds*. Hailstorm Papers, květen 2003. Dostupný na URL http://www.hailstorm.net/papers/smallwin32.htm (květen 2008)

[14] CATCH J. *Techniques for reducing Executable size*. Dokument dostupný na URL http://www.catch22.net/tuts/minexe.asp (květen 2008)

[15] WILSON T. *Aggressive Optimizations for Visual C++*. The Code Project, leden 2000.

[16] FILIATREAULT R. *Simply FPU.* The MASM Forum, 2003.

[17] *Win32 Equivalents for C Run-Time Functions*. Microsoft MSDN, ID:99456, listopad 2006.

[18] PERINGER P. *Modelování a simulace, studijní opora*. Brno, 2006.

[19] *Linear congruential generatorLinear congruential generator*. Dokument dostupný na URL http://en.wikipedia.org/wiki/Linear_congruential_generator (květen 2008)

[20] JAMES G., O'RORKE J. *Real-Time Glow*. Článek na odborném portálu Gamasutra, květen 2004.

Příloha 1. Užitečné odkazy

http://www.pouet.net/ - nejrozsáhlejší archiv dem, demo-skupin a akcí, aktuality

http://www.scene.org/ - aktuality ze světa demoscény, archivy

http://www.scene.cz/ - český portál demoscény

http://www.opengl.org/ - domovské stránky OpenGL

http://www.linuxdemos.org/ - portál věnující se linuxové demoscéně

http://www.scenemusic.eu/ - portál věnující se hudebním doprovodům dem

Seznam příloh

Příloha 1. – Užitečné odkazy.

Příloha 2. – CD s programem