VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

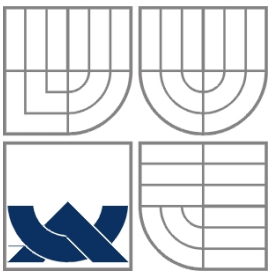
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GRAFICKÉ INTRO 64KB S POUŽITÍM OPENGL

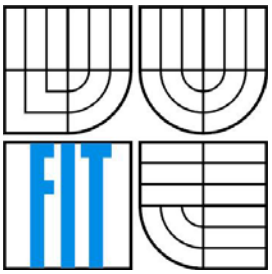
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE TOMÁŠ DROBISZ
AUTHOR

BRNO 2007



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

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ DROBISZ

VEDOUCÍ PRÁCE
SUPERVISOR

STANISLAV SUMEC, ING., PH.D.

BRNO 2007

Abstrakt

Bakalářská práce popisuje techniky použité při vytváření grafického intra, jejich výhody a nevýhody při dosahování limitů omezení výsledných spouštěcích souborů, dále pak historii demoscény, seznámení s knihovnou OpenGL. Výsledkem práce je implementace grafického intra se spustitelnou verzí nepřekračující limit 64kB.

Klíčová slova

Grafické demo, grafické intro, omezená velikost, OpenGL, generování 3d objektů, generování textur, částicový systém, osvětlení, multitexturing, pohyb založený na čase

Abstract

Bachelor´s thesis describes techniques that are used in graphic intro making, their advanteges and disadvantages in getting below size limit of executable file, history of demoscene, introduction to OpenGL. The issue is graphic intro implementation with single executable file with size below 64kB.

Keywords

Graphic demo, graphic intro, size limit, OpenGL, 3d object generating, texture generating, particle system, lighting, multitexturing, time-based move

Citace

Tomáš Drobisz: Grafikcké intro 64kB s použitím OpenGL, bakalářská práce,
Brno, FIT VUT v Brně, 2007

GRAFICKÉ INTRO 64KB S POUŽITÍM OPENGL

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Stanislava Sumce, Ph.D.
Další informace mi poskytl Ing. Adam Herout, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Drobisz
13.5.2007

Poděkování

Rád bych poděkoval Ing. Adamu Heroutovi, Ph.D., a Ing. Stanislavu Sumci, Ph.D., za pomocné rady při vypracovávání mé bakalářské práce.

Obsah

Úvod

1 Grafické demo a intro

Pod pojmem DEMO (zkratka od “demonstration” - demonstrace, INTRO od “introduction” – představení) si mnoho lidí představí bezplatnou ukázku daného programu, ve většině případů dané hry, a ve většině případů také komerčního charakteru. Na rozdíl od grafického dema, které znamená část volně šiřitelného softwaru nekomerčního použití. Tento program prezentuje pouze grafické a zvukové programovací techniky a schopnosti programátorů. Grafické dema jsou lineární a neinteraktivní - uživatel se na ně může dívat jako na film a nemá žádnou možnost ovlivnit jeho děj. Rozdíl oproti skutečnému filmu je v tom, že po spuštění dema se celá scéna nejdříve počítá a následně vykresluje, kdežto film pouze přehrává uložená data. Tvůrce takového dema má volnou ruku k rozhodnutí, co a jak ukáže a co k tomu použije, např. jednoduchý příběh nebo jen nějakou kombinaci zajimavých tvarů a barev, demo může být vtipné, vážné, vše záleží pouze na tvůrci, neexistuji žádná pravidla, co se týče obsahu.

Obr. 1: Pohled zachycený z grafického dema

2 Téma a cíle

V tomto demu jsem se pokusil nastínit obraz výřezu části krajiny se stavbou uprostřed, konkrétně domem, a při průletu kamerou kolem zobrazit nejdříve dům a potom část interiéru domu. Téma jsem si vytyčil celkem jasně a jednoznačně, a tudíž by neměl být problém s implementací. Bohužel to nebyl úkol tak jednoduchý, jak se na první pohled zdá, hodně práce spočívá ve výpočtech souřadnic základních stavebních bodů, dům musí mít přesně definovaný tvar, a tam se statickým datům nevyhneme. Mohli bychom je sice spočítat nějak náhodně, ale pak bychom se daleko vzdálili od původního plánu realistického domu.

3 Demoscéna

Demoscéna [10][11] by se dala popsat jako společnost tvůrců a příznivců vytváření grafických dem, snažící se ukázat, že počítače mají i jiné využití, než jak ho většina uživatelů chápe, tedy buď práci, hry nebo filmy, ale také k prezentaci svých nápadů, pocitů, životního stylu, a v neposlední řadě i ukázkou, že ten počítač sloužící převážně na psaní e-mailů dokáže zobrazit mnohem více než jen text. Zvlášní větví dem jsou pak dema technologická, která více než odhalit duši programátora slouží k odhalení výkonu a technologické vyspělosti grafické karty a/nebo procesoru, ale tato dema nepatří mezi produkty demoscény.

Demoscéna jako taková vznikala od 80. let, kdy při "cracknuti" nějaké hry apod. daná skupina se pod tento čin podepsala jednoduchým programem, který zobrazil její název, popř. jména členů, a na pozadí běžela jednoduchá grafická ukázka nebo efekt, a vše doplňovala jakási jednoduchá hudba či spíše zvuky. Tzv. crack-intro. Ke konci 80. let se od ilegální části (cracking) odděluje legální část - základ demoscény, jak ji známe dnes. Samozřejmě, že můžeme najít pár jedinců aktivních v demoscéně i v cracking-scéně. Hlavním produktem dnešní demoscény, které máme možnost vidět, jsou grafická dema.

Členem demoscény se může stát kažký, kdo má chuť se jakkoli podílet na této tvorbě, ať už se jedná o zkušeného programátora nebo začátečníka. Vítání jsou všichni, kteří chtějí prezentovat své výtvory a přitom se přiučit a inspirovat nápady ostatních. Tvorba grafických, nebo lépe řečeno multimediálních dem, může znamenat i nabídku zaměstnání právě v závislosti na předvedených zkušenostech, nápadech a dovednostech.

3.1 Demo, intro, nebo dentro?

Pojem intro pochází z první crack-scény - crack-INTROduction a většinou nemělo omezenou velikost. Dnes je intro většinou velikostně omezeno na jeden *.exe soubor do velikosti 64kB, což je asi nejběžnější, ale existují i kategorie 4kB a dokonce 256B. Dentro je kombinací mezi DEmo a iNTRO, z vlastností intra má limit na jeden *.exe soubor a z vlastností dema větší velikost (např. limit do 256kB nebo i bez limitu). Dentro také může být použito k představení finální verze dema. Dnes se používá označení demo nebo intro, u kterého se přidává velikostní limit, takže se můžeme setkat s pojmy Demo64kB i Intro64kB a obojí prakticky znamená to samé. Rozdíl je pak jen při technologických demech, kde se velikost pohybuje řádově v MB až stovky MB, a tato dema, ačkoli jsou volně šiřitelné, provází většinou komerční produkty (např. grafické karty), a čím novější demo, tím novější hardware potřebujete k jeho spuštění (což je také jejich účel). Těmto demům se pochopitelně nedává velikostní limit a nejsou produktem demoscény, ale profesionálních programátorů, kteří mají přesně popsáno, co a jak naprogramovat, aby všichni viděli technologickou náročnost dema a technologickou vyspělost daného hardware.

3.2 Platformy pro tvorbu

Tvorba demoscény pokrývá mnoho platforem, prakticky se dají naprogramovat dema pro jakýkoliv funkční kus hardware, počínaje prehistorickými kousky jako amiga, atari, c64, zx spectrum, přes herní konzole jako dreamcast, gameboy, playstation, xbox, dále mobilní telefony a pda, dokonce existují dema i pro grafické kalkulátory texas instrument. A samozřejmě nesmíme zapomenout asi na nejrozšířenější a nejpoužívanější: PC. Nezáleží ani na použitém operačním systému, existují dema pro ms-dos, windows i unix a další. Aktivita na daných platformách se odvíjí od momentálního rozšíření HW a dostupnosti vývojařského software. Ačkoli možnosti pro vývoj dem je mnoho, dnes jednoznačně převládá PC/windows.

3.3 Smysl a motivace

Vytváření dem můžeme považovat za koníčka jako cokoliv jiného, jen s rozdílem, že např. sbírání známek už nikoho nepřekvapí, zatímco vytváření dem není tolik rozšířené pro veřejnost a nároky na provoz takové záliby nekončí jen u finančních možností, jako např. při sbírání známek nebo čehokoliv dalšího. Nejblíže má asi k modelování, kde jsou kromě času a chuti potřeba i určité schopnosti, a v případě programování dem i vědomosti. Čili smysl tvorby dem je zábava, učení, prezentace svých nápadů elektronickou formou a v neposlední řadě i účast na různých soutěžích a párty, podobně jako na sběratelských burzách nebo zájmových kroužcích. Účastníci si vyměňují zkušenosti, předvádějí svá díla, spojují své síly a schopnosti na větších a složitějších projektech.

3.3.1 Umění?

Otázka, zda je demo forma umění či nikoliv, je dosti častá a také velmi sporná. Odpověd můžeme přirovnat k otázce, zda se za umění dají považovat knihy nebo filmy - některé ano, některé ne.

3.4 Limity a omezení

Limitů pro tvoření grafických dem je mnoho, od limitů řádu několika bytů po stovky kilobytů nebo dokonce megabytů. V této práci jsem se zaměřil na limit 64kB, který je jeden z nejběžnějších. Samozřejmě, že rady a postupy jsou použitelné i pro jiné limitové kategorie.

3.4.1 64kB - málo?

Když jsem poprvé viděl limit 64kB, jako osoba nezaujatá v tvorbě dem, jsem si pomyslel, že 64kB je naprosto nedostačující. Např. disketa má kapacitu 1.44MB, což je přibližně 22-násobek 64kB limitu, a všichni přece ví, že disketa je svou kapacitou v dnešní době naprosto nedostačující. Síla intra/dema tkví v tom, že tato aplikace obsahuje minimum statických dat, žádné nepoužité knihovny nebo funkce, naprosto nic, co se přímo nepoužije. Na každou věc (texturu, objekt atd.) si napíšeme vlastní funkci přesně podle našich požadavků, a tato funkce zabírá mnohonásobně méně místa, než kolik dat je schopna vyprodukovat. Výsledný renderovaný film nahraný do formátu *.avi (apod.) může mít kapacitu desítky až stovky MB, zatímco aplikace, která tento film vytvořila, zabírá např. zmiňovaných 64kB. A další výhoda je bezpochyby kvalita: renderovaná scéna má stejnou (vysokou) kvalitu v jakémkoliv rozlišení (pokud tedy používáme vektorovou grafiku), narozdíl od videa ve formátu *.avi, kde se při zvětšení (i zmenšení) rozlišení kvalita zhoršuje. Naopak nevýhoda je, že na starších nebo slabších počítačích nemusí jet grafické demo plynule, některé techniky nemusí být podporovaný (např. multitexturing), proto se demo objeví v horší kvalitě nebo se nespustí vůbec.

3.4.2 Jak dosáhnout 64kB

Dosáhnout limitu 64kB není tak složité, jak by se na první pohled mohlo zdát. Základem je si uvědomit, že největší velikost mají statická data (textury, objekty), které načítáme z externích souborů. Jelikož chceme pouze jeden *.exe soubor, není dobrý nápad je používat. Ačkoli výsledný *.exe soubor může být menší než 64kB, stačí jediná textura formátu *.bmp a lehce se přehoupneme daleko přes limit. Také máme možnost přenést data z textury přímo do proměnné ve zdrojovém kódu a použít je odtud, avšak tím pouze zajistíme výsledek v 1x *.exe ale určitě ne pod 64kB, zvlášť když zvážíme, že budeme určitě potřebovat více než 1 texturu, aby naše demo nevypadalo příliš jednotvárně. Načítání objektu je tentýž případ. Samozřejmě, že se statickým datům nevyhneme,

pokud nechceme abstraktní demo, které si vystačí s několika náhodnými proměnnými, ale snažíme se jim vyhnout, a to v případě textur a objektů jde velmi lehce. Ve výsledku máme uloženy jen ty nejdůležitější hodnoty, ze kterých jsme schopni odvodit a dopočítat ostatní pomocí funkcí, které zabírají jen zlomek velikosti dat, které vyprodukují. Další důležitá věc, která nám bude bránit dosáhnout našeho limitu, jsou knihovny. Musíme si určit, které knihovny budeme potřebovat, a podle toho nastavit překladač (některé překladače si automaticky přidávají mnoho dalších věcí, o kterých ani nemusí programátor vědět, a které se vůbec nepoužijí). Pokud potřebuji z dané knihovny jen jednu či dvě jednoduché funkce, je lepší si je naprogramovat sám, než používat předdefinované, které s sebou v knihovně berou i spousty dalších, které nepotřebujeme. Také musíme dát pozor na to, který překladač ve výsledku použijeme. Samozřejme to nemá až tak velký vliv, když je správně nastaven, ale pár kB bychom ušetřit mohli, a to v našem limitu 64kB je docela velká část. Mezi tři nejdůležitější pravidla k získání co nejmenší velikosti dema tedy patří :

1) Textury generujeme – nenáčítáme z *.jpg/bmp/gif atd. Tohle je jedna z nejdůležitějších věcí, která se v demu nesmí objevit.
2) Objekty také generujeme – žádné soubory z 3dmax či jiných aplikací.
3) Knihovny používáme jen ty nejdůležitější – snažíme se co nejvíce funkcí napsat sami. To je vlastně i smysl tvorby – nikoho pravděpodobně neohromíme několika standartními funkcemi. Hodnotí se i originalita.

Pokud tyto tři pravidla dodržíme, limit 64kB je již na dosah. Pokud si pohrajeme i s ostatními problémy, jako vhodnost překladače, či sám zdrojový kód, záleží už jen na nás, co všechno do těch 64kB ”natlačíme”. V našem případě se výsledná aplikace po zkompilování do limitu 64kB nevešla, ovšem s tímto výsledkem se počítá. Existují nástroje, které jsou schopny upravit *.exe soubor na mnohem menší velikost.

4 OpenGL

4.1 Co je to OpenGL

Open Graphic Library [1] byla navržena firmou Silicon Graphics Inc. jako aplikační programové rozhraní k akcelerovaným grafickým kartám a grafickým subsystémům. Předchůdcem této knihovny byla programová knihovna IRIS GL (Silicon Graphics IRIS Graphics Library). OpenGL byla navržena s důrazem na to, aby byla použitelná na různých typech grafických akcelerátorů a aby ji bylo možno použít i v případě, že na určité platformě žádný grafický akcelerátor není nainstalován - v tom případě se použije softwarová simulace. V současné době lze knihovnu OpenGL použít na různých verzích unixových systémů (včetně Linuxu a samozřejmě IRIXu), OS/2 a na platformách Microsoft Windows.

OpenGL je programové rozhraní grafického hardwaru. Tvoří ho asi 200 příkazů v jádře a dalších přibližně 50 v knihovně utilit, které se používají pro určení objektů a vytvoření interaktivních trojrozměrných aplikací. Neobsahuje žádné příkazy k práci s okny, ani příkazy pro definování trojrozměrných objektů. K jejich vytvoření jsou k dispozici jednoduché příkazy pro definování bodu, přímky a polygonu. Právě díky těmto vlastnostem je nezávislé na použitém hardwaru a použitelné na mnoha platformách.

4.2 OpenGL nebo D3D

V dnešní době je se stále více používá rozhraní DirextX. Tedy pokud se jedná o komerční produkty. OpenGL se naopak více používá u produktů freeware, na čemž jistě mají zásluhu i vývojáři a programátoři používající jiný operační systém než Windows. Nezávislost OpenGL upřednostňuje jeho použití na unixových systémech i na např. MacOS. Naopak výhoda DirectX je v ucelenosti balíku pro vývojáře, kteří mají k dispozici prostředky pro grafiku, zvuk, sítě, prakticky vše co potřebují. Další rozdíl je přidávání nových funkcí. V případě OpenGL je to praktikováno pomocí extensions, v případě DirectX je vydána nová verze nebo podverze. Rozdílů je mnohem a mnohem více, nelze však přímo říci, zda je lepší OpenGL nebo DirectX. Špatně napsaná aplikace bude špatná, ať už bude používat OpenGL nebo DirectX, a to samé platí i pro opak.

5 Použité programovací techniky

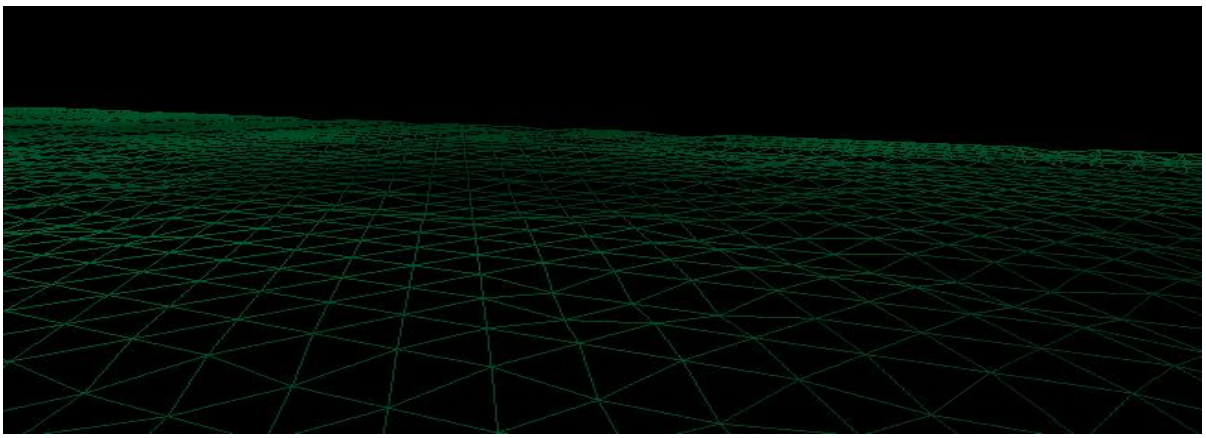
Zde se pokusím nastínit některé postupy a funkce, pro lepší pochopení principů. Nebudu zde rozebírat zdrojový kód, spíš se budu snažit, aby po pochopní textu nedělalo nikomu problém dané funkce napodobit.

5.1 Terén

Pro generování terénu existuje mnoho a mnoho technik, nejjednodušši by bylo načíst předdefinovaný, ale jak jsem se zmínil dříve, chceme si jej vygenerovat. Protože terén v tomto případě nehraje velkou roli, můžeme použít jednu z nejjednodušších technik: výškovou mapu.

5.1.1 Výšková mapa

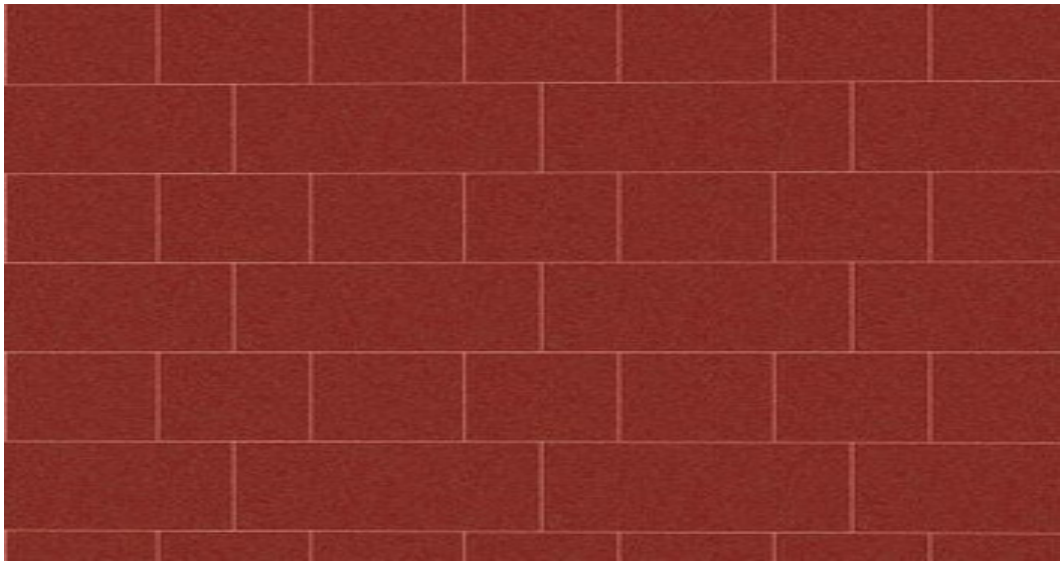
Výšková mapa [6] je v podstatě pole (nebo také tabulka) hodnot, které udávají výšku, tzn hodnotu Y. Hodnoty X a Z jsou spočteny z pořadí bodu v poli uspořádaného po řádcích za sebou. Pro hodnoty výšky jsem zvolil náhodné hodnoty upravené do menšího rozmezí, abych zabránil větším výškovým rozdílům a povrch zůstal jemně zvlněný. Algoritmus je celkem jednoduchý a nenáročný. Pokud bychom se ale zaměřili čistě na terén, pouhé náhodné hodnoty by jistě nebyly dostačující a museli bychom použít složitější rovnice, aby přechody mezi jednotlivými body v poli byly plynule a celkové uspořádání odpovídalo cíli, tedy např. kopcům, horám, nebo jen lehce prohloubenému údolí. Výškové mapy se většinou jen načítají z připraveného souboru. Čím větší pole bodů použijeme, tím detailnější povrch vznikne, avšak rychlost vykreslování se sníží. S tím musíme počítat, abychom předešli vysokým výpočetním nárokům, zvlašť, když vykreslení vysoké detailnosti terénu není náš primární cíl, a tudíž by jeho vykreslení mělo být co nejméně náročné.



Obr. 2: Výšková mapa, viditelné jsou malé výškové rozdíly

5.2 Textury

Textura [2] slouží k popisu vlastností daného povrchu. Pomocí ní můžeme vnímat barvu objektu, odraz světla na povrchu, průhlednost povrchu, pomocí změny normálového vektoru textury můžeme změnit optický tvar povrchu, aniž bychom měnili geometrii objektu. Texturu (v demu používám 2D) můžeme většinou chápat jako obrázek, který přiložíme na určitý objekt. Textura je reprezentována polem hodnot, které obsahují hodnoty pro červenou, zelenou a modrou složku barvy, a dále hodnoty průhlednosti (hodnota alfa). Každá trojice (resp. čtveřice) hodnot znamená jeden texel, tzn. jeden bod v textuře. Celé pole je velmi podobné poli pro výškovou mapu, jen v jednom rozdílu: výšková mapa obsahuje pro každý bod pouze jednu hodnotu odstínu šedi. Textury můžeme dále klasifikovat na základě jejich rozměrů. Jak už jsem se zmínil, v demu používám pouze 2D textury, které se mapují na povrch těles. Textury 1D se používají většinou jako vzorek pro generování přerušovaných křivek nebo pro opakující se podélné vzorky (např. přechod pro chodce). 3D textury, zvané někdy jako solid (=objemové), se používají ve spojení s objekty, které mají vypadat jakoby vyřezané z jednoho kusu materiálu (např. dřeva, mramoru apod.). 4D textury se používají pro animaci trojrozměrných textur. Problém, který nás v demu zajímá, je jejich velikost. Abychom i přesto mohli textury použít, odstraníme tento nedostatek jednoduchým řešením: textury vygenerujeme až po spuštění programu.

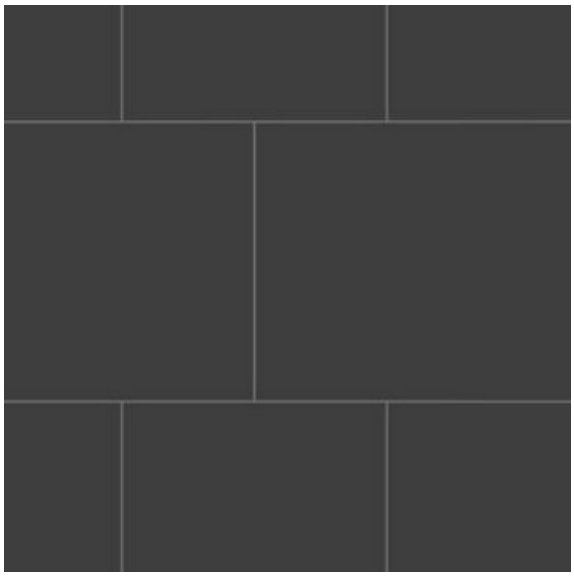


Obr. 3: Textura cihelné zdi, mapována několkrát vedle sebe a za sebou, mutitexturing původního vzorku cihly (Obr. 4) v kombinaci s červeně zašuměným vzorkem (Obr. 5), pohled z mírné vzdálenosti

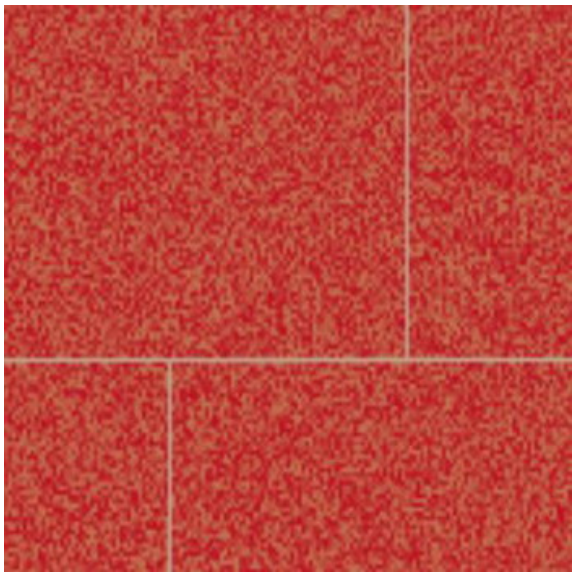
5.2.1 Generování textur

Generování textur není nic jiného než naplnění pole správnými hodnotami. Generováním textur jednoduše (někdy i docela složitě) vytvoříme opakující se vzory (např. cihlová zeď). Velké možnosti představuje generování zašuměných textur. Pro většinu objektů v demu začíná proces generování naplněním pole náhodnými hodnotami, které se poté upravují. Na tyto funkce jsou kladeny zvláštní požadavky: funkce musi být invariantní vzhledem k otočení, k posunutí, musí být spojitá, opakovatelná a mít omezené frekvenční spektrum. Splnění prvních dvou požadavků zaručí, že textura bude pořád stejná i při různém namapování a natočení. Opakovatelná znamená, že při zavolání funkce s parametrem x bude vracet vždy stejnou hodnotu y. My si samozřejmě vygenerovanou texturu nejdříve uložíme (limit pro velikost použité paměti po spuštění nemáme) a až potom ji použijeme, jako kdybychom ji předtím načetli odkudkoliv (ze souboru).

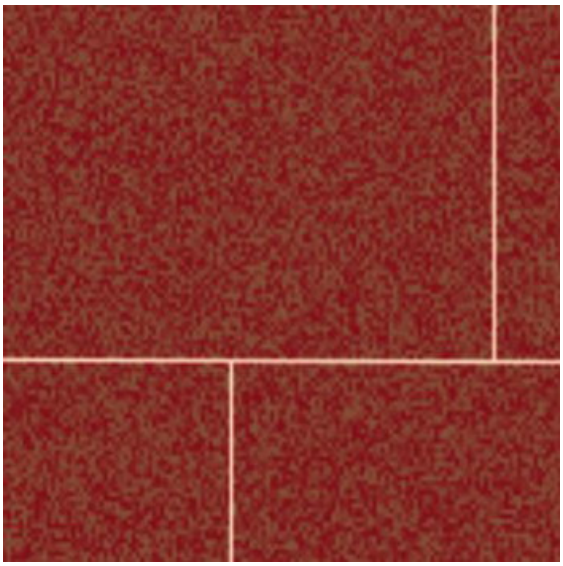
5.2.1.1 Příklady generování textur



Obr. 4: Původní textura vzorku cihly



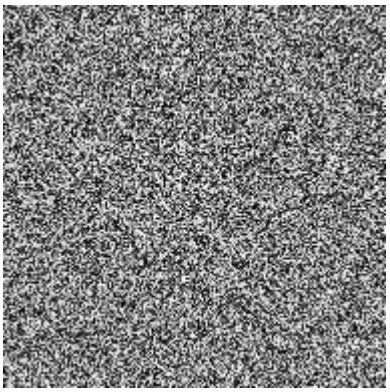
Obr. 5: Zašuměná textura s použitým vzorkem



Obr. 6: Multitexturing, kombinace Obr. 4 a Obr. 5, pohled z velmi malé vzdálenosti

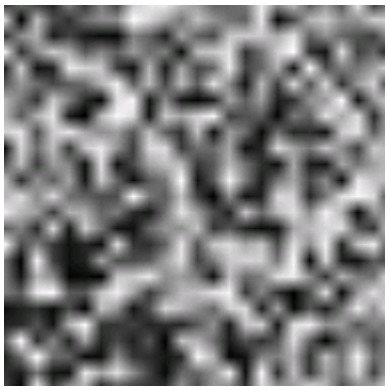
5.2.2 Perlinův šum

Perlinův šum [9] je příkladem generování náhodných textur. Základním prvkem je pole hodnot naplněných pomocí funkce Random(), které se poté jedním nebo více cykly prochází a pomocí několika funkcí se vytváří výsledný obraz.



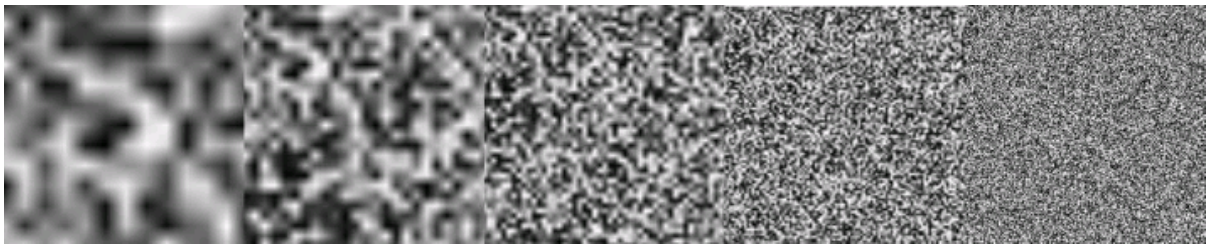
Obr. 7: Textura vygenerovaná pomocí random()

Obvykle použijeme funkci, která přidělí každému texelu odstín podle jeho čtyř sousedů. Výsledkem je pořád náhodně vypadající textura, teď již ovšem nejsou náhodné samotné body, ale můžeme sledovat jakési slučování do podobně zbarvených skupin.

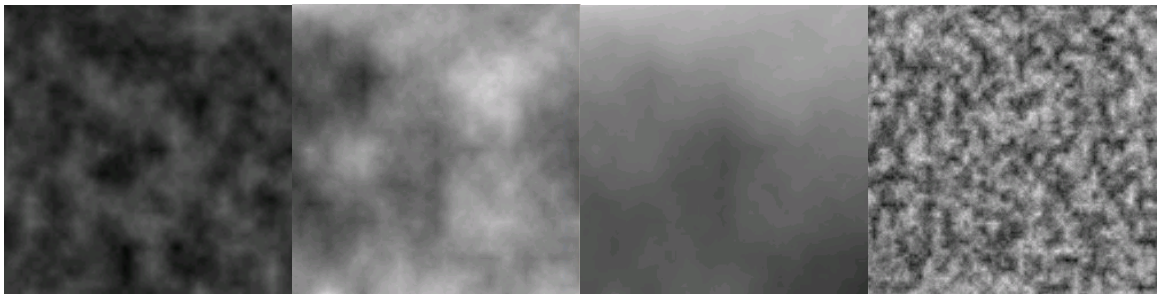


Obr. 8: Pokud bychom texturu neviděli blíže, interpolaci texelů se sousedy bychom neviděli

Při dalších úpravách použijeme funkci, která průměruje několik vzorků textury v různých rozlišeních. Již nevidíme malé nepravidelné shluky texelů, ale docela rozsáhlé plochy. Výsledek je samozřejmě ovlivněn parametrem udávající velikost a počet vzorků, se kterými počítáme. Při generování samozřejmě můžeme každou složku výsledné barvy upravit podle potřeby, v našem demu např. výsledná textura oblohy.

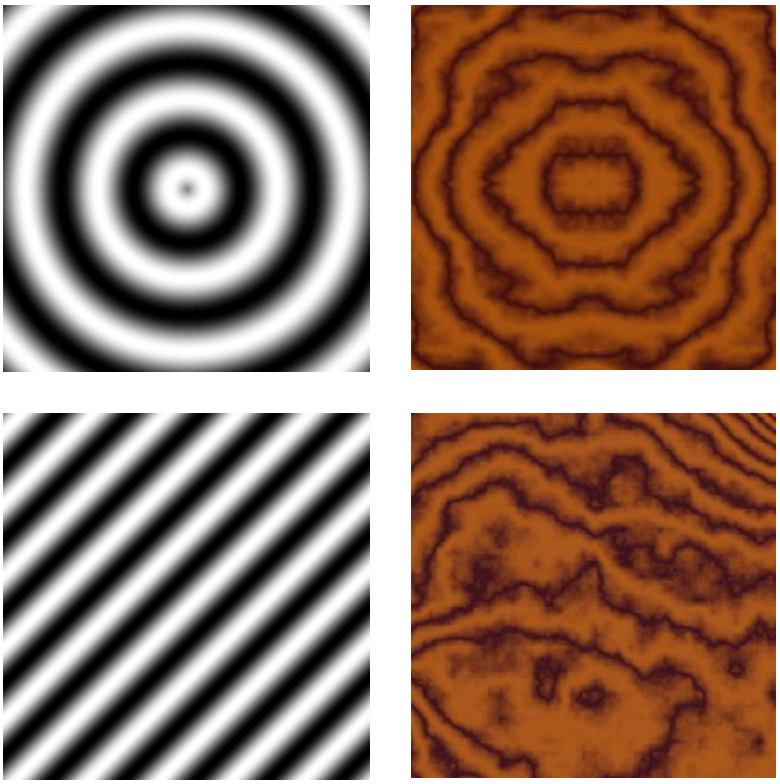


Obr. 9: Vzorky s různým rozlišením pro sečtení



Obr. 10: Příklady výsledků

Zajímavou obměnou počátečních náhodných hodnot na pravidelné můžeme dostat mnohem zajímavější textury. Vlevo původní pole, vpravo pak možné výsledky.



Perlinův šum můžeme použít i při tvorbě 3d textur a dokonce i k jejich animaci.

5.2.3 Mapování textur

Proces mapování textur znamená nanášení (v našem případě 2D) textury na povrch daného objektu. Složitost nanášení záleží na třech základních faktorech :

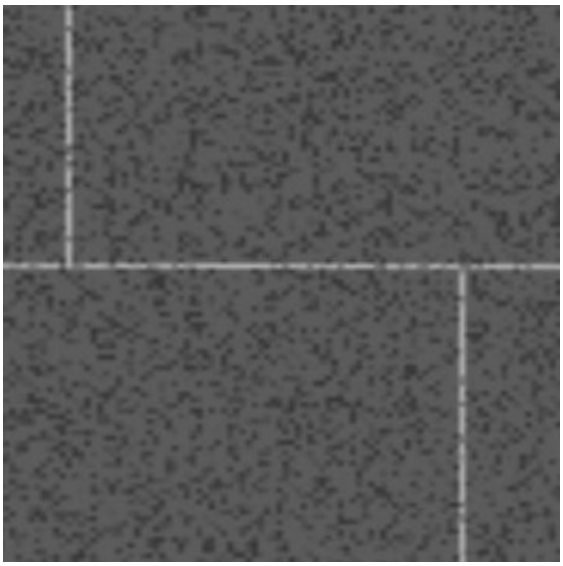
1) definice textury
2) tvar tělesa
3) mapované veličině.

5.2.4 Multitexturing

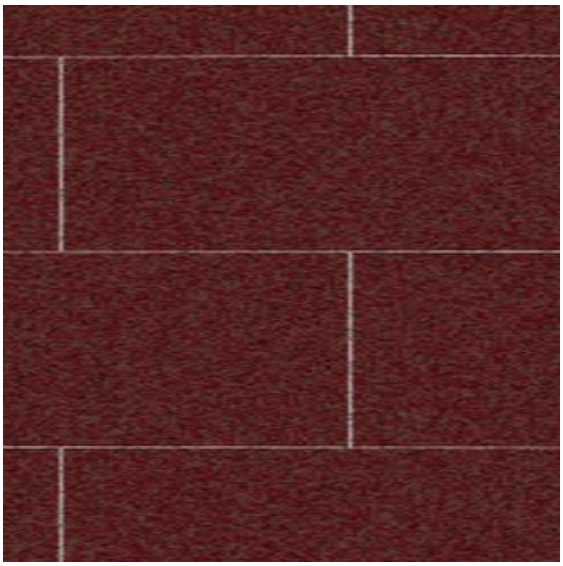
Multitexturing je jednou z důležitých otázek při tvorbě velké části grafických aplikací. Hlavním důvodem je nekompatibilita s některými, většinou staršími, grafickými čipy. Naštěstí v dnešní době je prakticky každý počítač, na kterém bychom mohli chtít danou aplikaci použít, vybaven grafickou kartou, která multitexturing podporuje. Z tohoto důvodu jsem se rozhodl multitexturing použít, ačkoli by bylo možné se mu vyhnout např. pomocí funkce, která by dané textury "složila" dohromady ještě před vykreslením. Pokud ale používáme multitexturing ve více kombinacích jednoduchých textur nebo vzorků, počet výsledných textur by se, při skládání mnoha kombinací, před

vykreslením vysoce zvledl. To, že textury generujeme až za běhu programu, má za následek, že nám nezabírají drahocenné místo, ale jen místo v operační (nebo grafické) paměti při běhu aplikace, na kterou nemáme žádný limit. A protože rozdíl v rychlosti vykreslování mezi použitím a nepoužitím multitexturingu jsem nezaznamenal, použití multitexturingu již nic nebránilo.

5.2.4.1 Příklady multitexturingu



Obr. 11: Multitexturing s použitím 2 textur, původního vzorku cihel a lehkého tmavého šumu



Obr. 12: Multitexturing s použitím 3 textur, kombinace šedé zašuměné zdi (Obr. 11, tzn. kombinace 2 textur) a červeně zašuměného vzorku cihly (Obr. 5)

5.2.4.2 Detail mapping

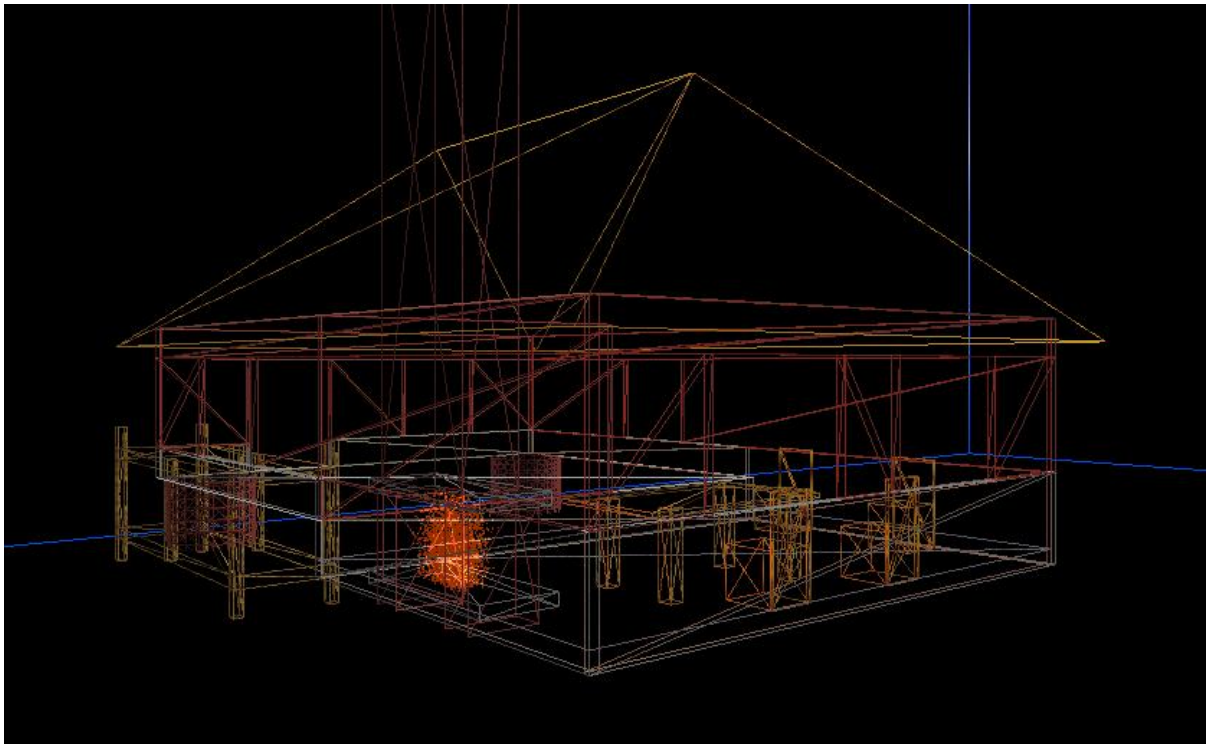
Detail mapping je ukázkou využití multitexturingu. Vygenerujeme dvě textury, obě o celkem malém počtů texelů (není ovšem nutné) – nízkém rozlišení. První z nich namapujeme na objekt tak, že každý jeden texel je dobře viditelný a zabírá poměrně mnoho pixelů (při pohledu kolmo), a celá textura může dokonce vypadat velmi "zubatě". Druhou texturu namapujeme tak, že ji necháme opakovat se ve velkém množství vedle sebe a za sebou, aby pokryla celou plochu předchozí textury, a toto opakování způsobí, že každý texel je velmi malý, a proto má druhá textura opačný efekt než první. Výsledek je velmi uspokojivý, při správném zvolení textur nejsou viditelné pravidelné vzory a výsledná textura vypadá dobře i z poměrně malé vzdálenosti, navíc se zbavíme opakovaného vzorku, např. tráva přece nemusí vypadat jako zelená tapeta.

Obr. 13: Povrch travnaté části

Obr. 14: Povrch travnaté části s použitím detail mapy

5.3 Objekty

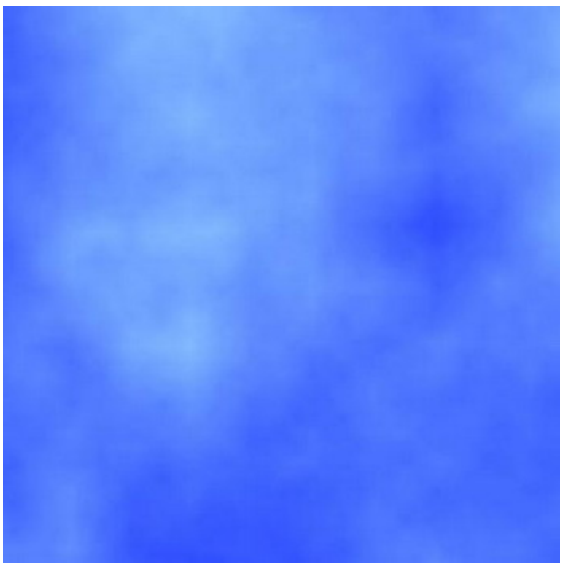
Objekty se opět, jako všechno ostatní, snažíme generovat, můžeme sice použít předdefinované, ale opět zde narazíme buďto na problém velikosti, dalšího souboru nebo další knihovny potřebné pro jeho načítání. Všechny objekty v demu jsou poskládány z ploch, které určují tvar a souřadnice pro nanesení textur. Plochu tvoří minimálně 3 body, větší plochy se skládají z několika až mnoha menších, ale vždy je tou nejmenší jednotkou trojúhelník. Dokonce i kulaté či zaoblené plochy se skládají z trojúhelníků. Bohužel se zde nevyhneme pevnému ukládání souřadnic bodů, naštěstí se většina bodů používá v demu více než jednou, a i na tom můžeme trochu místa ušetřit. Pro téměř všechny objekty používám GL_TRIANGLE_STRIP, který sdílí body : pro vykreslení 1 trojúhelníku je potřeba tří bodů, pro každý další pouze jeden bod, protože se použijí poslední dva body předchozího trojúhelníku. Mohli bychom použít i GL_QUAD (resp. GL_QUAD_STRIP), setkal jsem se však s problémem, kdy se při rozdělení čtyřúhelníku na trojúhelníky vykreslovaly výsledné trojúhelníky náhodně, a pokud nebyly všechny 4 body v jedné rovině, dosáhli jsme škaredého efektu náhodného problikávání daných plošek při každém dalším překreslení scény, tzn. dělící úsečka se měnila mezi pozicemi 1-3 bod a 2-4 bod.



Obr. 15: Méně přehledný obrázek, nicméně lze spatřit, že objekty jsou většinou tvořeny kvádry různých rozměrů

5.3.1 SkyBox

Skybox je technika, lépe řečeno objekt, používající se k modelování jednoduché oblohy. Základ tvoří krychle, ale klidně to může být i koule nebo válec. Skybox, tedy v našem případě krychle, se vykreslí jako první ve scéně, a co je hlavní – s vypnutým testováním hloubky (GL_DEPTH_TEST). Tím docílíme toho, že všechny následující objekty se vykreslí jakoby před ním, ikdyž podle porovnání souřadnic by následující objekt neměl být vidět. Proto má skybox většinou rozměry menší, tím že jsme deaktivovali při jeho vykreslení testování hloubky, se bude tvářit jako nejširší a nejdelší objekt ve scéně. Stačí na něj nanést vhodnou texturu a máme nebe na pozadí. Můžeme ještě přidat funkci, která při pohybu kamery bude skybox otáčet, ale nebude s ním pohybovat – stejně jako my v dálce vidíme nebe a při otáčení hlavou se nám pohybuje po horizontu, ale pokud půjdeme přímo rovně, žádný pohyb směrem k nám nebo od nás se nekoná. Dále můžeme při jeho vykreslení vypnout osvětlení (GL_LIGHTING) nebo použít speciální zvlášť pro něj – některé zdroje světla ve scéně by velmi rychle mohly odhalit skutečný tvar naší oblohy.



Obr. 16: Textura Skyboxu

Obr. 17: Přechod Skyboxu a vodní hladiny

5.3.1.1 Mlha

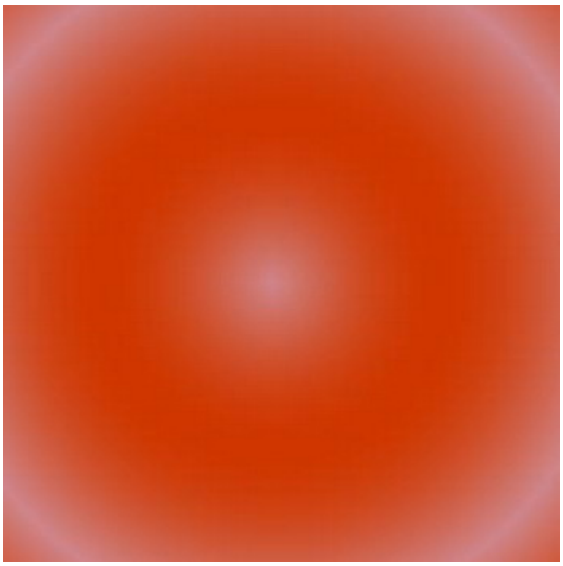
V tomto demu používám volumetrickou mlhu [7], ovšem její použití je poněkud sporné, nicméně svůj účel plní. Volumetrická mlha se částečně chová jako 3d textura, můžeme určit její polohu v prostoru, navíc však její hustotu v různých bodech, a právě o to se zde jedná. Mlha je namapována pouze jako 2d, a to na stěnách skyboxu, kde nejhustější část je na spodní ploše a směrem vzhůru se ztrácí, a tím zabraňuje, aby textura představující oblohu tvořila oblohu i pod vodní hladinou nebo nepatřičně do vodní hladiny zasahovala.

Obr. 18: Textura Skyboxu v kombinaci s mlhou

Obr. 19: Přechod vodní hladiny a Skyboxu

5.3.2 Částicový systém

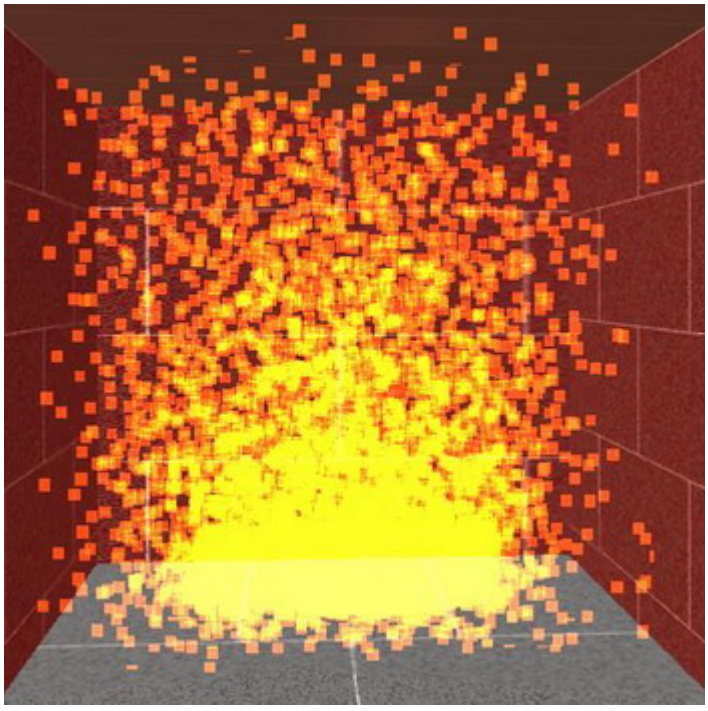
Částicové systémy [4] slouží k reprezentaci objektů, které nemají nebo mají jen těžce možnost definování pomocí vektorů, většinou se jedná o objekty typu oheň, voda, sníh, dokonce i hejno hmyzí populace. Jestliže celý částicový systém bereme jako jeden objekt, pak ten je tvořen mnoha a mnoha menšími dílky, stejně jako kouř je tvořen malými obláčky dýmu nebo voda malými kapkami. Výhoda spočívá v tom, že tyhle malé částice již můžeme lehce definovat a jejich umístění popř. pohyb určujeme u každé zvlášť tak, abychom vyvolali dojem náhodnosti a přirozenosti. V demu používám částicový systém při animaci ohně, kdy každá malá částice po vykreslení vykonala pohyb vzhůru a zanikla, a jako její náhrada se vytvoří částice nová, která absolvuje podobnou cestu a je zase nahrazena novou. Pohyb je počítán z náhodně vygenerovaných hodnot upravených podle potřeb měřítka a rychlosti pro dosažení objektu co nejvíce se podobající skutečnému ohni.



Obr. 20: Textura částice

5.3.2.1 Point Sprites

Původně vykreslované čtverce reprezentující častice jsem nahradil pomocí rozšíření z GL_EXT vykreslováním point sprites. Pro udání polohy se předává grafické kartě pouze jeden bod (střed částice), kolem kterého se částice vykreslí. Nevýhodou je právě vykreslování pomocí středu – pokud je částice na kraji obrazovky a její střed již není v záběru kamery, částice se nevykresluje a tím s ní mizí i část, která ještě viditelná byla. Výhodou je 4krát menší počet předávaných vertexů a tím pádem možné navýšení počtu částic k dosáhnutí lepší kvality výsledneho efektu a dále se částice natáčejí směrem ke kameře, takže je uvidíme ze všech směrů stejně.



Obr. 21: Výsledný efekt ohně, částice mají stále tvar čtverce, ale při pohybu není jejich tvar dobře viditelný a efekt je dostačující

5.4 Pohyb kamery

Pro pohyb kamery máme na výběr ze dvou řešení. První z nich je otáčení celou scénou upravováním transformační matice. Ja jsem použil způsob druhý, a to funkci GluLookAt(), která je součástí knihovny GLU [8], kterou při implementaci dema používám. Parametry této funkce jsou 3 vertexy : souřadnice pozice kamery, souřadnice bodu, na který se díváme a směrový vektor označující směr vzhůru, ktery slouží k naklánění kamery do stran. Tím, že nám stačí určit pouhé tři body pro určení pozice kamery se situace značně zjednodušuje. Několika jednoduchými rovnicemi přímek nebo křivek dostaneme plynulý pohyb, jak bodu pohledu tak bodu cílového. Velmi povedený pohyb poskytuje použití beziérových křivek. Mé řešení je přesto odlišné. V demu používám funkci, která určí pouze počáteční body polohy a cíle pohledu, které se pomocí přičítání nebo odčítání rychlostí vynásobenou velikostí uplynulým časem od předchozího volání funkce postupně mění, avšak pouze po dobu předem určenou. Funkce porovná rozdíl času od spuštění programu s časem aktuálním a podle toho určí, kterým směrem se kamera bude pohybovat a otáčet. Jediný problém je v editování, kdy změněním i jen jediného parametru nám změní všechny následující pohyby, proto musíme nastavovat pohyb kameru postupně od počátku.

5.4.1 Pohyb založený na čase

V dobách počítačů řady 286 a 386, kdy prakticky všechny stroje byly přibližně stejně výkonné, se pohyb všech objektů ve scéně počítal na pricipu přičítání hodnoty v každém dalším snímku. S příchodem procesorů Pentium se však starší hry pohybovaly velice rychle, příčinou bylo právě počítání založeně na rychlosti vykreslování, která s příchodem nových procesorů rychle rostla. Abychom zabránili těmto rozdílům, vzdálenost objektů mezi dvěma snímky násobíme rychlostí a hodnotou času, který uběhl od minulého vykreslení. Tím nezaručíme plynulý pohyb, pouze způsobíme to, že objekt se z bodu A do bodu B přesune stejně rychle, ať již k tomu bude mít deset či tisíc snímků.

5.5 Světla

Pro osvětlení scény [3] máme opět na výběr z několika různých řešení. Fyzikální modely, které jsou založeny na modelování vlnově-částicového chování světla, můžeme hned vynechat, protože jsou pro běžné použití příliš výpočetně náročné, a proto se využívají jen pro specifické účely. Jedno z dlaších řešení, které mě napadlo, je definování a generování light map, v podstatě textur, které by multitexturingem nebo připočtení do výsledné textury při generování změnily odstíny textur tak, že by to ve výsledku ukázalo tmavší plochy tam, kde bychom chtěli stín a světlejší tam, kde bychom chtěli světlo. Tento způsob dle mého názoru je při složitějších scénách velmi zdlouhavý na

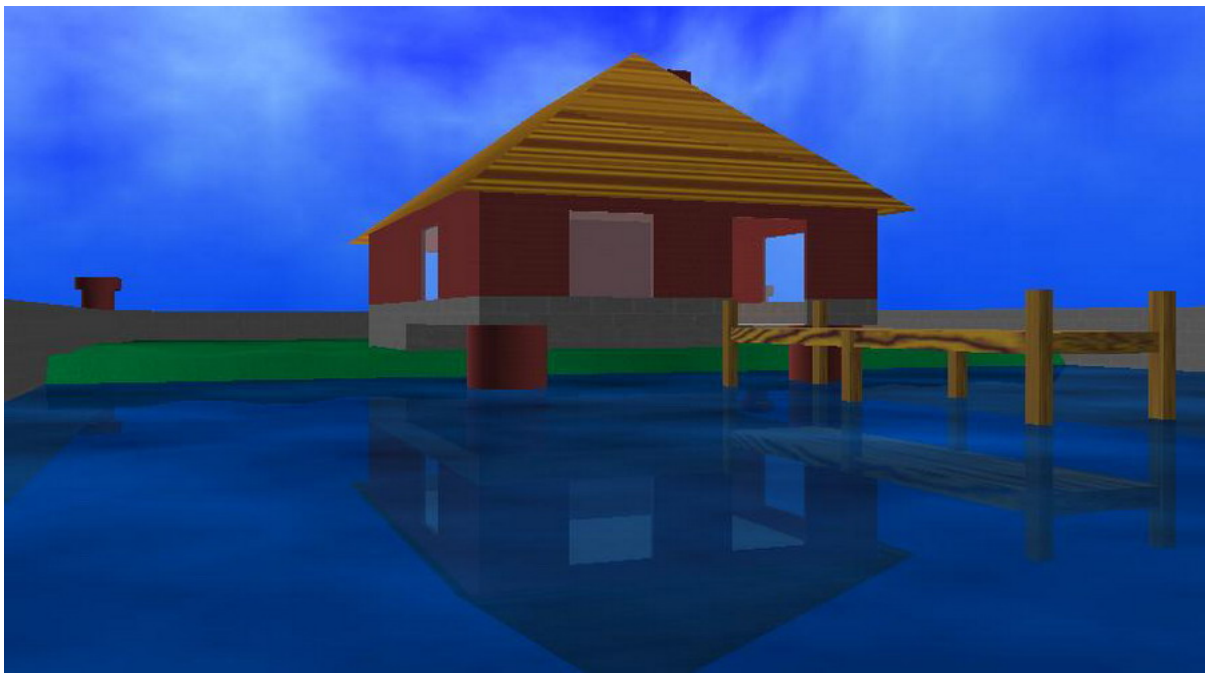
implementaci. Zaměřme se na empirické modely, které v mezích dobré rychlosti počítání nabíjezí slušnou kvalitu. Proto jsem zvolil Phongův osvětlovací model. Při počítání barvy každého bodu povrchu přičteme vliv ambientního (okolního) světla a za každý světelný zdroj se započítá zrcadlová a difuzní složka světla. Intenzita difuzní složky závisí na úhlu dopadu světla na osvětlovanou plochu, intenzita zrcadlového světla závisí na úhlové odchylce odraženého světla od pozorovatele. Osvětlovací model se často využívá v kombinaci se stínováním. Stínování je proces, při kterém se ze znalosti barvy povrchu pomocí normálového vektoru určí výsledná barva jednotlivých bodů povrchu. Měl jsem na výběr z několika možných typů. Konstantní stínování (flat shading), Goraudovo stínování a Phongovo stínování. Při konstantním stínování se všechny body plochy vykreslí stejným odstínem, a díky tomu můžeme vidět hraniční úsečky. Mnohem lépe vypadá Phongovo stínování, ale je náročnější na počítání. Goraudovo stínování je pro nás ideální. Spočítají se nejdříve vrcholy ploch a pomocí nich potom barva každého bodu plochy, a proto jsou přechody mezi plochami plynulé. Nejdříve musíme u všech vertexů nastavit normálové vektory. Protože v demu používám většinou hranaté objekty, nastavení normál z velké části pouze ve směru jedné z os není nic složitého. Částečně jsem si pomohl při definování normál tak, že normály povrchu trávy jsem nastavil všechny kolmo vzhůru. Odchylky ploch jsou minimální a proto tímto ulehčením se nepřipravíme o výsledný efekt, který by byl při počítání skutečných normál prakticky stejný.

Obr. 22: Osvětlení vnitřní části bodovým zdrojem oranžového světla umístěného na pozici vykreslování ohně

V aplikaci definuji tři světla s přislušnými parametry. První světlo je směrové bílé, paprsky rovnoběžně osvětljí celou scénu, další světlo je opačného směru, avšak poněkud slabší, to proto, abych zabránil vysokým rozdílům mezi osvětlenými a neosvětlenými plochami. Poslední světlo je bodový zdroj, jehož pozice je v místě vykreslování ohně, barvy oranžové a osvětluje jen vnitří část domu. Při vykreslování průhledných ploch jako jsou okna nebo hladiny vody, je osvětlení vypnuto.

5.6 Zrcadlení

Použití zrcadla [5] v demu představuje odraz domu na vodní hladině. Řešeno je to tou nejjednodšší cestou. Odraz ve skutečnosti není přímý odraz, ale pouze podruhé vykreslené objekty s opačnou souřadnicí výšky. Průhlednost hladiny je zapnutá a vše je vykresleno ve sledu odraz – spodní hladina – horní hladina – originální objekt. Hladiny jsou ve skutečnosti dvě, abychom jejich otáčením vyvolali dojem pohybující se vody, což vypadá podstatně lépe než kdybychom použili pouze jednu. Stencil buffer není nutné použít, kamera se nedostane do pozice pod hladinu ani jinde, kde bychom viděli např. odraz, který by se vlastně od ničeho neodrážel.



Obr. 23: Odrazy objektů na vodní hladině

6 Velikost výsledného souboru

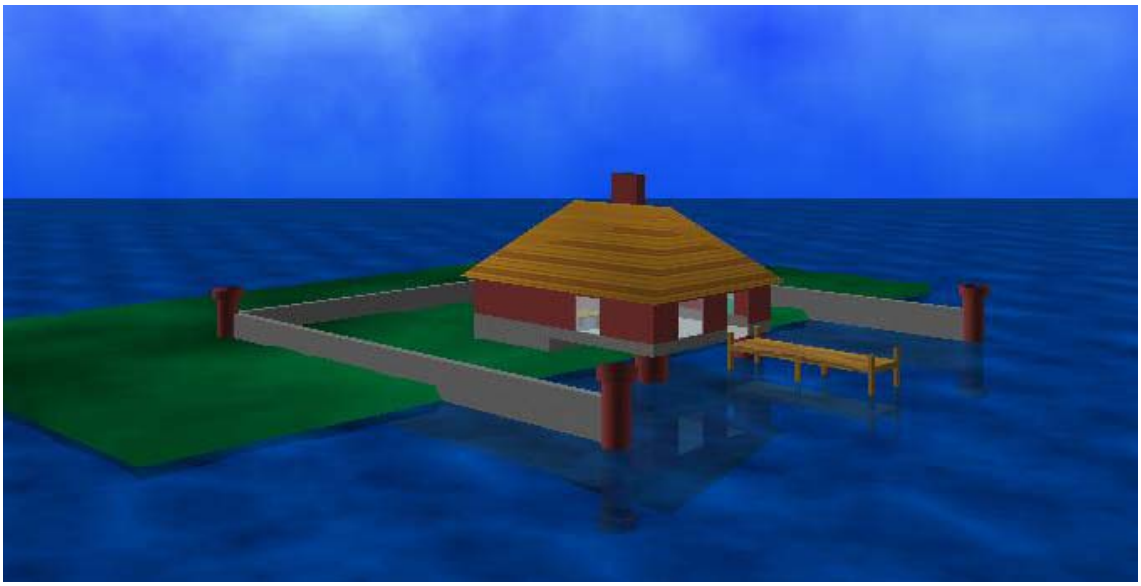
Velikost výsledné aplikace daleko přesáhla limit 64kB, ale jak jsem se již zmínil, neznamená to problém. Pomocí programů Strip a Upx jsme zmenšili jeho hodnotu na více než dostatečnou.

Původní velikost	Strip	Upx	Strip+upx
204 669 B	168 960 B	57 725 B	22 016 B

Jak je viditelné z tabulky, nejvhodnější kombinace je nejdříve použít program Strip, který je součástí programátorských nástrojů, v mém případě balíku Dev-C++ [13]. Program Strip odebere z aplikace nepotřebné části jako např. tabulku symbolů. Lze jej aplikovat nejen na aplikace, ale i např. na sdílené knihovny.

Druhý krok je použití programu Upx [12], nyní již ve verzi 3,0. Upx je volně šiřitelný, vysoce výkonný nástroj pro kompresi binárních dat. Nabízí vynikající kompresní poměr ve spojení s velmi rychlou dekompresí. Jeho výhodou je, že přidá do aplikace samorozbalovací kód, který po spuštění rozbalí aplikaci do systémové paměti, odkud je spuštěna s téměř nulovým zpožděním.

Tímto jsme dosáhli limitu pod 64kB a cíl práce je splněn.



Obr. 24: Záběr na hotovou scénu

7 Pokračování a rozšíření v budoucnu

Demo, které jsem vytvořil pro tuto práci samozřejmě nelze srovnávat s prakticky ”profi” výtvory některých skupin demoscény. Proto bych alespoň trochu naznačil možné pokračování práce na zdokonalení tohoto dema, jelikož jsme se k limitu 64kB nepřiblížili podle očekávání, možnosti pro rozšíření dema jsou poměrně velké. Ne všechny vylepšení a nápady nutně znamenaji nárust velikosti výsledného souboru. Některé techniky pouze vylepšují a nahrazují stávající kód nebo jeho zápis, některé naopak další kód přidávají, na což nemůžeme zapomenout a musíme počlivě hlídat velikost neboť to je prakticky jeden z primárních cílů. Naneštěstí pravděpodobně nebude použití všech návrhů možné právě kvůli limitu 64kB, což je ale jaksi předpokládáno, neboť kdyby bylo možné použít vše, bylo by to snad až moc jednoduché a bez výzvy pro jakoukoliv soutěž.

7.1 Technické úpravy

7.1.1 Vertex buffer object

V prvé řadě bych doporučil použití VBO (rozšíření podobné vertex arrays) místo používání GL_BEGIN / GL_END, což znatelně poznáme na rychlosti vykreslování. Vertex buffer ukládá data do paměti grafické karty, která je mnohem rychlejší než systémová pamět. Navíc se vyhneme opakovanému definování vrcholů a tím opět snížíme nároky na výkon. Další množství dat, které potřebujeme při práci s vrcholy, jako jsou souřadnice textur, normály, barvy, souřadnice mlhy atd., se ukládají do dalšího pole nebo polí a jsou volány pomocí jedné funkce namísto několika při stávajícím způsobu. Pro každý typ dat můžeme mít samostatné pole nebo použít prokládane pole, kde jsou uloženy všechny potřebne data.

7.1.2 Více zrcadel

Použití zrcadel je velmi pěkný efekt, ať už jako čistě zrcadlo nebo jen odlesk na lesklých plochách. K tomu nám pomůže stencil buffer, který určuje, které pixely budou použity k vykreslení odrazu obrazu. Vlastně se nejedná tak přímo o odraz, ale o vykreslení zrcadlených objektů pomocí zrcadlového otočení a jejich ořezání pomocí stencil bufferu pouze do plochy zrcadla. Při kulatých zrcadlech a dalších podobných ploch můžeme použít enviroment mappig, tzn. vygenerujeme si danou scénu do textury podle kamery vycházející z daného bodu a směrem od dopadu paprsku, a tu namapujeme na objekt zrcadlově převrácenou.

7.1.3 Stíny

Použití stínů by pozvedlo vizuální kvalitu o další stupeň výše. V plánu pro rozšíření obsahu se zmiňuji o střídáni den/noc, při čemž by stíny při pohybu Slunce po obloze nebo při hořícím ohni byly patřičně využity. Mohli bychom použít light mapy (pevné stíny přímo v texturách), nebo pro pohyb stínů použít mnohem vhodnější dynamické stíny. To dovoluje vrhat stín prakticky kamoliv, ale také přichází s dalšími nároky na výpočetní výkon.

7.1.4 Hudba a ozvučení

Další věc, kterou velká část dem obsahuje, je hudba. Jak je vidět, nebo lépe řečeno slyšet, v demu není jediný zvuk. Zvuk není o nic méně důležitou částí kvalitního dema než obraz. V dobré kombinaci obrazu a zvuku se ze dvou prakticky nepoužitelných části může stát velmi silná kombinace. Vidíme sice, že demo spíše existuje jako obraz bez zvuku než jako zvuk bez obrazu, ale to zatím jen z důvodu, že demo nelze považovat za finální verzi, která po dokončení snad bude mít i šance dostat se do některé ze soutěží. Toto demo je vlastne jakési demo dema.

7.2 Obsahové rozšíření

V rámci obsahu máme na výběr mnohem více možností vylepšení, a to právě z důvodu, že téma můžeme rozvinout jakýmkoliv směrem. Chceme-li klidný dům, máme ho, chceme-li na něj přivolat povodeň nebo meteor, můžeme. Moje vize spočívá spíše v klidnějším duchu. Rozšíření o několik dalších domů různých tvarů a barev, udělat celou scénu veselejší. Dále přidání stromů a zeleně, efekt pro ”vítr” popř. ”gravitaci”. Pro ukázku změny světel, barev a stínů by nejlépe posloužilo střídání dne a noci, popř. změna roční doby, která by měla za úkol i využití dalších částicových systému, jejichž efekt je velmi působivý (déšť, sníh, padající listí atd.).

8 Závěr

Cílem práce bylo vytvořit grafické demo s velikostí, která nebude přesahovat 64kB. Tento cíl se mi splnit podařilo, velikost je přibližně 22kB, tudíž rezerva pro pokračování je dosti veliká. Bez pomocí kompresních programů bychom se do limitu nevešli. Velkou zásluhu na tom má část kódu, která generuje a následně kreslí dům. Ten obsahuje mnoho důležitých bodů a částí, bez kterých by jeho správné vykreslení nebylo možné.

Interiér domu je sice poněkud strohý, jeho detailnější propracování by ovšem bylo mnohem náročnější. S původní představou domu má tento dům společného málo, jeho tvar a vzhled se měnil v závislosti na co nejjednodušším a nejpraktičtějším zápisu, a hlavně na výsledném grafickém efektu, aby demo pokud možno nebylo nudné.

Ačkoli to tak nevypadá, naprogramování nebyla na tomto demu nejtěžší práce. Mnohem složitější bylo získávání informací o daných technikách a následně jejich pochopení. Pravděpodobně žádná ze skupin demoscény nezveřejňuje své zdrojové kódy, proto musíme na mnoho věcí přijít sami postupným zkoušením a testováním, kdy mnohdy z původních teoretických návrhů nebylo po otestování použito nic, protože praktická zkouška nás přivedla na mnohem lepší řešení.

Literatura

[1] Shreiner D., Woo M., Neider J., Davis T. *OpenGL průvodce programátora.* Adison Wesley, Computer Press, Brno 2006

[2] Žára J., Beneš B., Sochor J., Felkel P. *Moderní počítačová grafika, strana 379 - Textury,* Computer Press, Brno 2004

[3] Žára J., Beneš B., Sochor J., Felkel P. *Moderní počítačová grafika, strana 319 - Světlo,* Computer Press, Brno 2004

[4] WWW stránky. NeHe OpenGL Tutoriály. Částicové systémy. http://nehe.ceskehry.cz/tut_19.php

[5] WWW stránky. NeHe OpenGL Tutoriály. Odrazy. http://nehe.ceskehry.cz/tut_26.php

[6] WWW stránky. NeHe OpenGL Tutoriály. Generování terénů. http://nehe.ceskehry.cz/tut_34.php

[7] WWW stránky. NeHe OpenGL Tutoriály. Volumetrická mlha. http://nehe.ceskehry.cz/tut_41.php

[8] WWW stránky. OpenGL Návody. http://www.root.cz/serialy/opengl-a-nadstavbova-knihovna-glu/

[9] WWW stránky. Perlin noise. http://student.kuleuven.be/~m0216922/CG/perlinnoise.html

[10] WWW stránky. Demoscene FAQ. http://tomaes.32x.de/text/pcdemoscene_faq.txt

[11] WWW stránky. Česká demoscéna. http://scene.cz/

[12] WWW stránky. Ultimate Packer for eXecutables. http://upx.sourceforge.net/

[13] WWW stránky. Dev-c++. http://www.bloodshed.net/devcpp.html

Seznam příloh

Příloha 1. CD