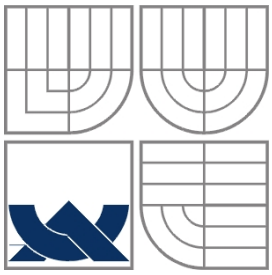
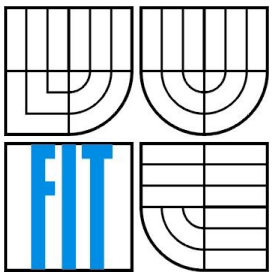


VYSOKÉ UČENÍ TECHNICKÉ v BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GRAFICKÉ INTRO 64KB S POUŽITÍM OPENGL

GRAPHICS INTRO 64KB USING OPENGL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ZDENĚK HEJL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HEROUT, Ph.D.

BRNO 2009

Zadání

1. Seznamte se s fenoménem grafického intra s omezenou velikostí.
2. Prostudujte knihovnu OpenGL a její nadstavby.
3. Popište vybrané techniky použitelné v grafickém intru s omezenou velikostí.
4. Implementujte grafické intro s použitím OpenGL, aby velikost spustitelné verze nepřesáhla 64kB.
5. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek pro prezentování projektu.

Abstrakt

Tato práce popisuje hlavní části implementace grafického intra s omezenou velikostí. Popsány jsou použité minimalistické techniky a způsob procedurální výroby textur a jednoduchých i složitých 3D objektů. Dále jsou vysvětleny způsoby načítání, úpravy a animace objektů a způsoby vytvoření použitých efektů.

Abstract

This document describes implementation of the graphical demo with limited size. Described are used minimalistic techniques and the methods of procedurally created textures and both simple and complex 3D objects. There are explained methods for loading, editing and animating objects and ways to create used effects.

Klíčová slova

grafické intro, grafické demo, OpenGL, textura, Perlinův šum, multitexturing, Bresenham, NURBS křivky, 3D objekt, 3D model, Catmull-Clark, animace, výšková mapa, skybox, hudba, syntetizátor, pasivní stereo

Keywords

graphics intro, graphics demo, OpenGL, texture, Perlin noise, multitexturing, Bresenham, NURBS curves, 3D object, 3D model, Catmull-Clark, animation, height map, skybox, music, synthesizer, passive stereo

Citace

Zdeněk Hejl: Grafické intro 64kB s použitím OpenGL, bakalářská práce, Brno, FIT VUT v Brně, 2009

Grafické intro 64kB s použitím OpenGL

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. A. Herouta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Zdeněk Hejl
5.5.2009

Poděkování

Dovoluji si poděkovat Ing. A. Heroutovi, Ph.D. za jeho odbornou pomoc a cenné rady. Dále bych rád poděkoval autorům použité hudby vystupujících pod přezdívkami kb a Quickyman a také demoskupině farbrausch za poskytnutí zvukového syntetizátoru a komprimačního nástroje.

Obsah

1 Úvod

1.1 Grafické intro

Cílem této práce je vytvořit grafické intro, občas též nazývané jako grafické demo. Grafické intro je vlastně naprogramované video, které má demonstrovat nějakou myšlenku a ukázat co nejobsáhlejší a nejdetailnější scény. Obvykle obsahují i řadu různých efektů a na pozadí těchto grafických ukázek často hraje hudba. pro většinu těchto programů platí, že jsou limitovány velikostí.

Fenomén grafického intra vznikl původně mezi počítačovými piráty, kteří přikládali do svých utilit krátké grafické programy jako svoji vizitku. Nejednalo se tedy o produkt s nějakým přímým užitkem, ale spíše o ukázku autorových dovedností. Z těchto grafických inter se časem stal samostatný obor, kterému se ve volném čase věnuje řada grafiků, zvukařů a vývojářů často tvořících skupiny, tzv. demoskupiny (demogroup). I přes přirozenou soutěživost mezi jednotlivými demoskupinami, mnozí autoři pomáhají ostatním například uvolněním jejich speciálních nástrojů (například zvukový syntetizátor, hudební skladby, komprimační nástroje...).

Existuje řada typů inter, některá mají za cíl přednést nějakou myšlenku či příběh, jiná mohou být abstraktní a snažit se o umělecký dojem. Některá intra mohou demonstrovat nějakou speciální použitou techniku, nebo fungovat jako ukázka k nějaké nové technologii. Taktéž se může jednat o ukázkové video k nějakému existujícímu produktu, například k filmu nebo hře. Vytvářejí se i dema, jejichž cílem je vygenerovat jediný obrázek, nebo dema tvořící kompletní počítačovou hru.

Intra se vytváří na mnoha různých platformách a operačních systémech. Většina inter se vytváří pro PC s operačním systémem Windows, ale vyskytují se i intra na konzole, PDA, mobilní telefony a dokonce i na některé oblíbené historické počítače či na některé nové programovatelné kalkulačky. Některá intra mohou vyžadovat i speciální hardware, například určitou grafickou kartu.

V tvorbě těchto dem se pravidelně pořádají soutěže o nejhezčí práce. Kategorie soutěží bývají obvykle rozděleny dle typu intra, cílové platformy a hlavně dle maximální povolené velikosti. Nejmenší mnou nalezená intra mají velikost pár desítek bajtů, ale nejvíce dem je vytvořeno do limitů 256B, 1kB, 4kB nebo 64kB. Existují ale i velikostně neomezená intra. Samozřejmě, do více prostoru se vejde více efektů.

1.2 Velikost intra

K dosažení těchto velikostí se obvykle používají různé minimalistické postupy a také různé formy komprese dat. Základním principem je, aby samotný program obsahoval co nejméně statických dat. Všude, kde je to možné, je snaha data vygenerovat nebo vypočítat programem, protože takový algoritmus obvykle zabírá podstatně méně místa. Například je velmi nevhodné, až téměř nemožné, vkládat do intra obrázky, 3D modely a hudbu v plné kvalitě. Obrázky a textury se obvykle vyrábějí procedurálně pouze pomocí několika jednoduchých algoritmů.

Některé objekty ve scéně je taktéž možno vyrobit procedurálně, často se k tomu používají různé matematické funkce. Některé modely však nelze nijak jednoduše generovat, proto se často tyto polygonální 3D modely nejprve vytvoří v některém externím 3D modelovacím programu a potom se přiloží do zdrojového kódu. Tato obvykle rozsáhlá statická data se často ukládají různými speciálními způsoby, které zmenší jejich velikost, popřípadě i data upraví tak, aby se později lépe komprimovala. Lze také implementovat některý algoritmus dělící 3D modely, který z hrubého, hranatého a málo zabírajícího modelu vytvoří pěkný vyhlazený model, který by při přímém vložení do programu zabíral řádově více místa.

Hudbu také není vhodné přikládat v běžné kvalitě, například ve formátu mp3, ale používají se speciální formáty, které obsahují v principu jen seznam jednotlivých not. k tomuto účelu jsou vytvořeny speciální formáty zápisu hudby, které buď zabírají málo místa, nebo jsou velmi dobře komprimovatelné. Na tyto speciální formáty existují i jednoduché přehrávače, takzvané syntetizátory, které je možné přímo přiložit do svého projektu.

Pro zmenšení velikosti je také možné komprimovat celou aplikaci. k tomu je dostupných několik komprimačních nástrojů, které z klasické aplikace vytvoří samorozbalovací archiv. Po zkomprimování a následném spuštění se nejprve celá aplikace rozbalí do paměti a následně spustí. Celý tento proces rozbalování je při použití v malých intrech pro uživatele prakticky nezaznamenatelný.

1.3 Téma intra

Cílem mého intra bylo zobrazit řadu objektů a efektů, které budou demonstrovat jednotlivé použité techniky, a zároveň budou tvořit krátký příběh nebo vyjadřovat nějakou myšlenku. Hlavní děj mého intra probíhá v zatopené krajině, která obsahuje základní zeleň jako je tráva, křoví a stromy. Pro úplnost scény a pro dojem opravdové krajiny jsou zobrazeny i mraky, voda, mlha a realističnosti přispívají i další efekty, jako například odraz ve vodě.

Snaha byla směřována k co nejjednoduššímu vytvoření realistických scén za použití minimalistických technik. S výhodou byla proto velmi často používána náhodnost, jelikož i příroda jeví prvky náhodnosti. Nejedná se však o úplnou náhodnost, ale o přesně definovanou náhodnost pomocí pseudonáhodných čísel, které jsou popsány například ve skriptech [1]. Takže i přesto, že jsou jednotlivé objekty i jejich rozmístění a natočení generovány náhodně, celá scéna se vždy vygeneruje úplně stejně.

Jelikož intro je v podstatě krátké video a samotná scéna krajiny je statická, výsledné intro by bylo bez nějakého výrazného dynamického prvku nudné. Pro oživení scény bylo proto třeba vymyslet a implementovat nějakou animaci nebo pohyblivý objekt. Proto byl do intra přidán létající drak, který jako hlavní hrdina intru přidá dostatečnou akčnost a nabídne řadu animací a dalších efektů. Je tedy možné vytvořit děj celého intra, ve kterém divák uvidí pohybujícího se draka, jak přistává, létá a dokonce i plive oheň.

2 Výroba intra

Moje intro se řadí do kategorie animované grafické intro a je určeno pouze pro Windows. Grafika intra se zobrazuje pomocí OpenGL a o vytváření okna a o interakce se stará winapi. Intro je procedurálně programováno v jazyce C, je vyvíjeno v multiplatformním opensource vývojovém prostředí Code::Blocks [2] a pro překlad je používán překladač GNU GCC [3]. Intro je kompilováno jako monolit, tedy jako jeden samostatný spustitelný .exe soubor, a splňuje limit 64KB, do kterých se musí vejít všechny grafické a zvukové prvky.

OpenGL

OpenGL je zkratka pro Open Graphic Library a jedná se o specifikaci platformně nezávislého aplikačního rozhraní pro programování 2D a 3D grafiky. OpenGL je nízkoúrovňové procedurální rozhraní, které umožňuje pomocí sady základních příkazů zobrazovat základní primitiva, jako jsou například body, úsečky, trojúhelníky, mnohoúhelníky apod. OpenGL funguje na principu klient-server, přičemž klient zasílá OpenGL serveru příkazy buď pro zobrazení primitiv nebo pro nastavení způsobu vykreslení těchto primitiv. Stav nastavení OpenGL je uložen v tzv. „OpenGL state machine“, který zajišťuje, aby se po odeslání příkazu nějakého nastavení (například nastavení barvy, textury), všechny následně vykreslené objekty zobrazovaly s tímto nastavením, dokud nedojde k další změně tohoto nastavení.

Jelikož je jazyk C i OpenGL multiplatformní a intro používá minimum platformně závislých funkcí, nemělo by být složité předělat samotnou grafiku intra pro jinou platformu. S hudbou by to bylo složitější, jelikož použitý zvukový syntetizátor je dostupný pouze pro Windows.

OpenGL okno

Prvním krokem ve vývoji grafického intra s omezenou velikostí je vytvořit kostru programu, která otevře prázdné okno použitelné pro vykreslování obsahu za pomocí OpenGL. Tato kostra programu musí používat co nejmenší počet knihoven a funkcí, které by zbytečně zvětšovaly velikost výsledné aplikace. Nakonec bylo intro upraveno pouze pro zobrazování v okně, jelikož přepínání do fullscreen a změna rozlišení působí rušivě. Pro snazší budoucí přenositelnost nebyla použita žádná dialogová okna s nastavením, nastavení programu je možné pouze přes parametry příkazové řádky. Velikost této aplikace s nachystanou OpenGL obrazovkou v mém případě zabírala 8kB, po kompresi jen 3,55kB.

Pro testovací účely bylo třeba vytvořit základní prostředky pro orientaci ve 3D prostoru, například možnost intuitivního pohybu ve scéně pomocí klávesnice a myši. Další pomůcky a vytvořené ladící nástroje budou popsány později.

Soubory projektu

Pro obsáhlost celkové aplikace bylo nutností rozdělit projekt do několika modulů. Každý takový modul obsahuje jeden zdrojový (.cpp) a jeden hlavičkový (.h) soubor. Seznam těchto modulů je stručně popsán v následující tabulce:

Modul	**Funkce**
animation	Animace, časová osa, naplánované události.
draw	Hlavní kreslící funkce. Kreslení scény, draka a efektů.
draw_texts	Vytvoření a tisk textu. Kreslení úvodní načítací stránky a závěrečných titulků.
debug	Ladící funkce a ovládání z klávesnice.
init	Inicializace dat a 3D modelů.
init_textures	Inicializace textur.
main	Hlavní program, vytvoření a nastavení okna.
math	Matematické funkce.
object	Funkce pro vytvoření a práci s 3D modelem.

Tabulka 1: Moduly projektu

Do projektu jsou však vloženy i hlavičkové soubory (.h) , které obsahující uložené 3D modely. Pokud obsahuje jeden model více souborů, jedná se o stejné modely, pouze různě deformované. To umožňuje pozdější animace.

3D model	**Soubory s modely**
Drak - hlava	drak_hlava, drak_hlava_fire
Drak - křídlo	drak_kridlo, drak_kridlo_let, drak_kridlo_slozene
Drak - noha	drak_noha, drak_noha_let
Drak - ocas	drak_ocas, drak_ocas_dole, drak_ocas_nahore, drak_ocas_vpravo
Drak - tělo	drak_telo, drak_telo_nadech

Tabulka 2: Hlavičkové soubory s uloženými 3D modely

Hudba je uložena ve dvou hlavičkových souborech, ostatní soubory jsou potřebné pro zvukový syntetizátor.

Soubor	Soubory s modely
music_ouverture.h	Hudba pro hlavní část intra. Délka: 2 minuty 52 sekund.
music_to_short.h	Hudba pro závěrečné titulky. Délka: 1 minuta 32 sekund.
libv2.lib	Externí knihovna zvukového syntetizátoru.
libv2.h	Externí knihovna zvukového syntetizátoru.
dsound.lib	Potřebná knihovna pro zvuk.

Tabulka 3: Seznam souborů nutných pro přehrávání hudby

2.1 Textury

Jelikož textury nelze kvůli velikosti předpřipravit a načítat, veškeré textury se vyrábí procedurálně při startu aplikace. Základem výroby téměř všech textur je princip Perlinova šumu. Většina funkcí vytvořená pro výrobu textur pracuje s jedním barevným kanálem a výsledná textura se po řadě jednoduchých úprav složí do 4 barevných kanálů, tedy včetně alfa kanálu. Ten umožní plnou průhlednost některých míst textury při použití alfa-testingu, nebo částečnou průhlednost při použití alfa-blendingu. Před samotným vyrobením textury pomocí OpenGL se definuje texturový filtr (GL_LINEAR, GL_NEAREST, nebo mipmaping).

Perlinův šum

Perlinův šum je postup pro generování textur, jehož autorem je Ken Perlin [4]. Cílem tohoto postupu je pomocí šumové funkce procedurálně vytvořit přirozeně vypadající textury. Princip spočívá ve vygenerování matice náhodných čísel, která se poté upravuje a sčítá v různém měřítku, v takzvaných oktávách. Nejprve se tedy vygeneruje jedna textura s náhodnými hodnotami a ta se poté v různém měřítku a s různými koeficienty sčítá do výsledné textury. Některé textury jsou předtím/potom vyhlazeny či jinak jednoduše upraveny, aby více odpovídaly jejich očekávanému vzhledu.

Obrázek 1: Ukázka textury

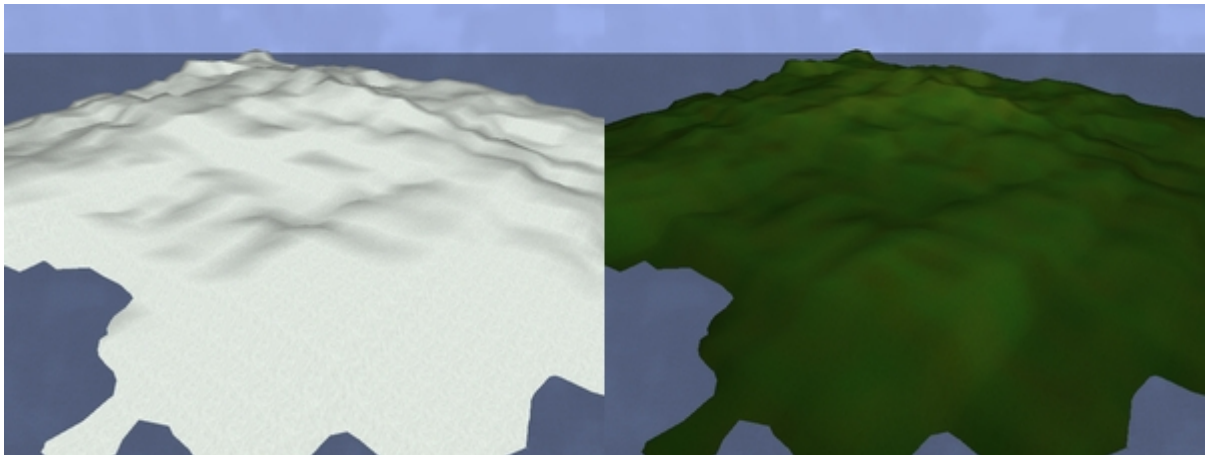
Multitexturing

Pro rozmanitost a nejednotnost textur některých objektů podporuje aplikace multitexturing. Možnost mulitexturingu je v OpenGL řešena pomocí rozšíření (extensions), což znamená, že tato funkce nemusí být dostupná na všech grafických kartách. Po rozsáhlém testování jsem však nenašel sestavu, ve které by multitexturing nefungoval.

V interních objektech, které uchovávají všechny potřebné informace o 3D modelech, je počítáno až s dvěmi texturami. Funkce zobrazující objekty tedy umožňuje zobrazit objekt bez textury, s jednou texturou nebo s dvěmi texturami pomocí multitexturingu. Přičemž je možné, aby textura byla definována přímo v objektu, nebo je možné ji zvolit před samotným vykreslováním objektu, což umožňuje jeden objekt zobrazit pokaždé s jinou texturou.

Multitexturing je v intru použit zatím pouze na terén. Místo použití multitexturingu by šlo obarvovat jednotlivá políčka terénu vlastním odstínem, ale to by způsobilo, že by byly jasně vidět přechody mezi jednotlivými políčky a celá mřížka terénu by se zvýraznila. S použitím multitexturingu se přispívá k větší realističnosti a hrany terénu se naopak opticky vyhlazují.

Na terén se tedy nanáší jedna textura stále dokola na každé „políčko“ jako tapeta a pomocí multitexturingu se vloží i druhá textura, která terénu dává různé odstíny. Textura s odstíny je zobrazena jednou přes celý terén. Díky většímu rozlišení než má samotný terén a díky lineárnímu filtrování textury dodává tato textura terénu plynulé změny odstínů od zelené až po žlutou a hnědou. Barvy textury s odstíny jsou tvořeny na základě výšky terénu s příměsí trochy náhodnosti. Na obrázku 2 vlevo je použita jen jedna základní textura, na stejném obrázku vpravo je přidána 2. textura s odstínem.



Obrázek 2: Multitexturing

Vyfocení scény do textury

Program umožňuje během průběhu animace fotit aktuálně zobrazené scény a vytvářet z nich textury. k vyfocení současného okna je použita OpenGL funkce *glReadPixels()*, která přečte zobrazené pixely přímo z nastaveného bufferu. Tato vyfocená data není možno použít k výrobě textury ihned, jelikož textury vyžadují čtvercové rozměry, které jsou mocniny dvou, a data z vyfoceného okna mohou mít obecně libovolné rozměry, jelikož rozměry okna lze za běhu měnit. Je tedy nejprve zjištěn potřebný rozměr textury a poté jsou do této textury data přenesena a vycentrována na střed. Ukládání stavu okna se uplatní pro zobrazování obrázků v závěrečných titulkách.

2.2 Trojrozměrné objekty

Pro jednotnou práci s mnoha 3D objekty byla vytvořena sada funkcí pracujících s definovanou strukturou univerzálního polygonálního modelu *TObject*, který obsahuje všechny potřebné informace o 3D objektu. V této struktuře jsou uloženy zvlášť jednotlivé vrcholy polygonální sítě a zvlášť jednotlivé polygony. Běžný 3D model je obvykle tvořen sítí polygonů, ve které jsou jednotlivé vrcholy použity pro více polygonů. Každý tento vrchol o třech souřadnicích lze proto uložit jen jednou a jednotlivé polygony tvořit z ukazatelů na tyto vrcholy. Toto rozdělení umožní pozdější jednodušší práci s objektem a navíc výrazně šetří operační paměť. Naopak přístup k těmto vrcholům může být o trochu pomalejší, jelikož je nutno přistupovat přes jeden ukazatel navíc oproti systému, ve kterém by se do jednotlivých polygonů ukládaly přímo souřadnice vrcholů.

Polygony i vrcholy jsou v objektu uloženy v dynamických nafukovacích polí, jejichž inicializace a realokace probíhá automaticky. Uvolnění objektů je však nutné provádět na konci programu pomocí funkce *Object_free()*.

Tyto objekty podporují pouze polygony se čtyřmi vrcholy, tudíž není možné vytvářet objekty například z trojúhelníků. To je na jednu stranu nevýhoda, na druhou stranu to přispívá k jednoduchosti algoritmů, lepší přehlednosti a v případě uložených modelů i k redukci uložených dat. Například krychle se dá popsat pomocí 6 polygonů, tedy 6x4=24 vrcholů, zatímco pomocí trojúhelníků je to 12x3=36 vrcholů.

Výroba objektů

Interní modely, ať už generované nebo načítané, se vytvářejí stejně – postupným přidáváním jednotlivých polygonů pomocí funkce *Object_add_polygon()*. Při vkládání každého polygonu dojde nejprve k vložení jednotlivých vrcholů do objektu funkcí *Object_add_point()*, která vrátí pořadové číslo nově vytvořeného nebo již uloženého vrcholu. Po uložení těchto pořadových čísel v polygonu se ještě pro polygon spočítá normála, takzvaná „face normal“, která označuje vektor kolmý na tento polygon.

Po vložení všech polygonů se pomocí funkce *Object_compute_normals()* spočítají normály pro jednotlivé vrcholy, které se používají pro kvalitnější Gouraudovo stínování. Vrcholové normály se spočítají jako průměr face normál všech polygonů, ve kterých je vrchol obsažen. Vrcholové normály jsou uloženy v nafukovacím poli stejné velikosti a stejně indexované jako je pole vrcholů, ke kterým tyto normály náleží. Z většiny objektů je také vygenerován takzvaný „display list“, který může urychlit zobrazování objektu.

Načítání objektů

Jelikož ne všechny objekty je možné jednoduše vygenerovat, umožňuje intro 3D objekty do projektu přikládat v hlavičkových souborech jako statická data. Objekt je v hlavičkovém souboru uložen ve dvou konstantních polích podobně, jako se ukládají objekty v paměti, tedy data modelu jsou rozdělena na pole vrcholů a pole polygonů. Tento princip šetří potřebnou paměť a ta je v tomto případě velmi důležitá, jelikož velikost uložených modelů může výrazně ovlivnit velikost celého intra. Jak v paměti, tak i v hlavičkových souborech jsou uloženy velikosti těchto polí. I zde tedy platí, že každý vrchol obsahuje 3 souřadnice a každý polygon obsahuje 4 použité vrcholy. Pro načtení objektu z hlavičkového souboru je třeba jen projít pole uložených polygonů a ty vložit do objektu pomocí již popsané funkce *Object_add_polygon()*.

Čísla použitých vrcholů jsou v polygonu uloženy jako *unsigned char*, který zabírá pouze 1 bajt. Použití typu *unsigned char* oproti nabízejícímu se typu *int*, který obvykle zabírá 4 bajty, nám sníží

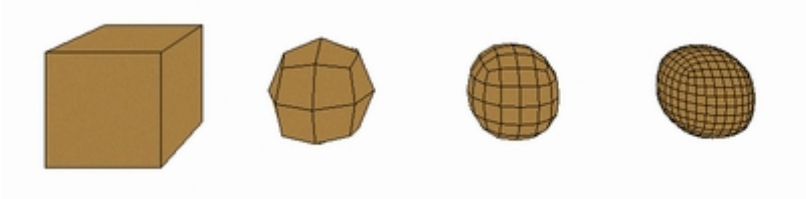
objem uložených dat v této sekci souboru na jednu čtvrtinu. Má to však výrazné omezení – objekt může obsahovat maximálně 255 polygonů. Tento limit je ale v tomto intru dostatečný, jelikož je použita technika dělení polygonálních modelů.

Souřadnice vrcholů jsou uloženy s přesností *float*, což jsou obvykle 4 bajty. Zde by se dalo výrazně ušetřit prostor. Pokud bychom se smířili s podstatně nižší přesností souřadnic, mohli bychom jednotlivé souřadnice uložit na 2 či dokonce na 1 bajtu. To by ale znamenalo degradaci modelu. Vzhledem k tomu, že použité modely jsou velmi malé a celková aplikace nepřesahuje požadovaný limit, nebyla tato optimalizace nakonec implementována.

Celkově je v hlavičkových souborech s uloženými modely ještě hodně potenciálu pro zmenšení velikosti. Možné by bylo přerovnat vrcholy a polygony za účelem pozdější lepší komprese, nebo data rovnou ukládat komprimované. Další možností by bylo ukládat polygony ve formátu tzv. „quad strip“, který pro první polygon potřebuje všechny 4 vrcholy, ale pro každý další navazující polygon stačí definovat pouze 2 vrcholy. Zbylé 2 se berou z toho předchozího. Tato metoda ukládání by pravděpodobně zmenšila velikost uložených dat až o desítky procent, ale uložit data do takového formátu by bylo komplikovanější. Ce lá polygonální síť modelu by se musela „rozřezat“ na jednotlivé pásy polygonů, které by šly takto uložit. Automatický převod klasické polygonální struktury na tento způsob není triviální, proto tento způsob uložení dat nebyl implementován a je zde zmíněn jako možnost pro budoucí vývoj.

Algoritmus Catmull-Clark

Algoritmus Catmull-Clark [5] spočívá v rozdělování polygonální sítě a systematickému průměrování vrcholů což umožňuje z hranatého modelu vytvořit pěkný zaoblený model. Tento algoritmus je možné používat rekurzivně, čímž lze vytvořit dokonale hladké povrchy. Avšak při každém zjemnění modelu se počet polygonů zvětší 4krát, což samozřejmě zvyšuje nároky na operační paměť a rapidně snižuje rychlost vykreslování těchto objektů. Proto je vhodné dělit model jen do požadovaných detailů. Díky tomuto algoritmu tedy není třeba ukládat dokonale hladké modely, stačí ukládat pouze hranaté kostry modelů, což ušetří značné množství místa při zachování dostatečné kvality modelů.



Obrázek 3: Ukázka dělení modelu krychle algoritmem Catmull-Clark

Algoritmus Catmull-Clark je implementován ve funkci *Object_divide()*, která vytvoří ze zadaného objektu nový, vyhlazenější objekt. Základní informace o algoritmu byly čerpány z popisu ve Wikipedii, otevřené encyklopedii [5]. Implementovaný algoritmus navíc rovnoměrně dělí i texturovací koordináty. To znamená, že při rozdělení jednoho polygonu na čtyři se na vytvořených polygonech nezobrazuje celá textura, ale jen odpovídající čtvrtina textury. Tím je docíleno, aby opakovaně rozdělený povrch nejevil známky tapetování textury. Textura však musí být dostatečně veliká, při malých rozměrech textury a při několikanásobném dělení povrchu může dojít k artefaktům díky zaokrouhlování málo detailní textury.

Vykreslení objektu

K vykreslování objektu je vytvořena funkce *Draw_object()*, která projde celý objekt a postupně pošle OpenGL k zobrazení nadefinované textury, texturovací koordináty, vrcholové normály a souřadnice samotných vrcholů. Vykreslování objektu je tedy možné buď přímo pomocí této funkce, nebo voláním display listu, pokud byl pro daný objekt vytvořen.

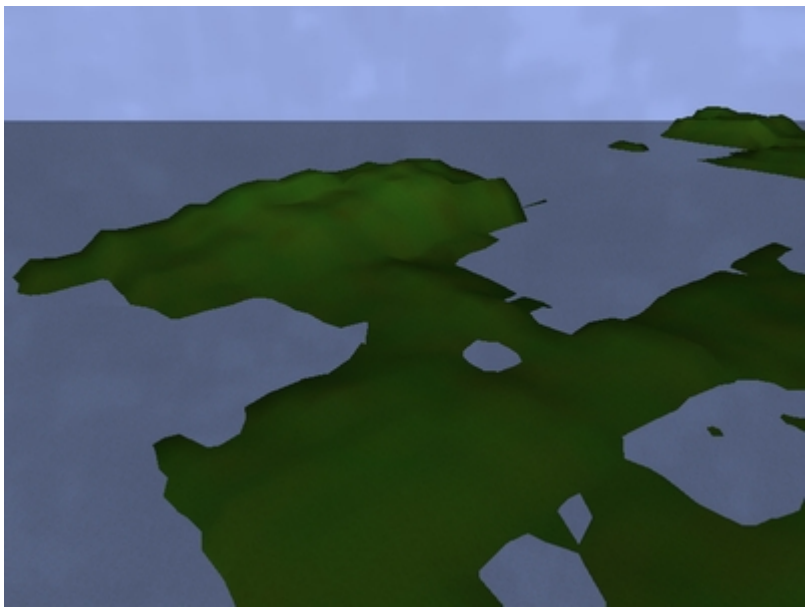
2.3 Objekty krajiny

Veškeré objekty scény jsou načteny nebo vyrobeny již při startu intra. Pro některé objekty jsou také vyrobeny jejich vyhlazenější verze. Po ukončení načítání se žádné nové objekty ani animace nevyrábí, již se pouze zobrazuje. Tím je zajištěna plynulost videa bez čekání na výrobu dat během animace, všechno čekání je ale nutné si přečkat na začátku intra.

Terén

Terén se tvoří ve funkci *Gen_terrain()* generováním pseudonáhodné výškové mapy, která se několikrát sčítá s rozdílným měřítkem, podobně jako výroba textur pomocí Perlinova šumu. Terén je tvořen čtvercovou sítí, která obsahuje jemně vlnitý povrch, ale i větší kopce a údolí. Celý terén je veliký pouze 63x63 políček, přičemž políčka, která jsou celá pod úrovní vodní hladiny, jsou rovnou vyřazena. Políčka protínající vodní hladinu se ponechávají celá. Odseknutí částí políček, která jsou pod úrovní vodní hladiny, probíhá až při vykreslování kompletní scény se všemi objekty pomocí OpenGL ořezávací funkce – *glClipPlane()*.

Krajní políčka čtvercového terénu jsou snížena pod vodní hladinu, aby terén nekončil „někde ve vzduchu“, ale tvořil pěkné ostrůvky. Bohužel, díky pevně dané velikosti terénu, jsou na jeho koncích zřetelné nepřirozené útesy. Tento nežádoucí jev by šlo eliminovat generováním nekonečného, nebo alespoň opakujícího se terénu. V současnosti je tento problém maskován nezobrazováním scény z úhlů, ve kterých je tento jev viditelný.



Obrázek 4: Členitý terén s vodní hladinou a mraky

Dále je přímo ve funkcí *Gen_terrain()* za pomocí dat z vytvořené výškové mapy generována textura s odstíny povrchu terénu a podle výšky terénu jsou pseudo-náhodně rozmístěny další objekty, které mají být postaveny na terénu. Tvorba textury a rozmísťování objektů přímo v této funkci sice není ideální, ale vzhledem k získanému zjednodušení se dá tolerovat.

Mraky

Dalším detailem, který přidává na realističnosti je zobrazení mraků. Mraky se zobrazují na pozadí scény díky dočasně vypnutému hloubkovému testování a zobrazováním mraků před vším ostatním. Nejprve se tedy vykreslí celé okno mraky a přes něj se poté překreslí scéna. Jen v místech, kde se nic nevykreslilo, zůstalo pozadí s mraky.

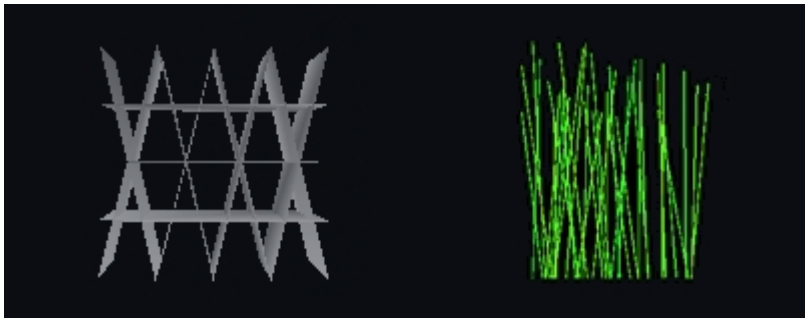
Mraky jsou mapovány na povrch koule, přesněji na jedenkrát zaoblenou krychli pomocí algoritmu Catmull-Clark. Přes zapnutou mlhu jsou však mraky méně vidět.

Tráva

Tráva se zobrazuje jen na rovnějších místech terénu a je tvořena z mnoha vhodně uspořádaných textur. Tato textura trávy se vytváří jednoduchým kreslením různě barevných úseček znázorňujících jednotlivá stébla trávy. Ke kreslení úseček do textury byl použit Bresenhamuv algoritmus pro rasterizaci úseček, jelikož je podle [6] nejefektivnější a nejpoužívanější. Jedna z dvaceti použitých textur v intru je zobrazena na obrázku 5 vpravo.

V intru byla použita technika seskupování trávy do větších trsů, se kterými se lépe pracuje a které se dají samostatně zobrazit. Částečně průhledné textury trávy bylo nutno rozmístit tak, aby

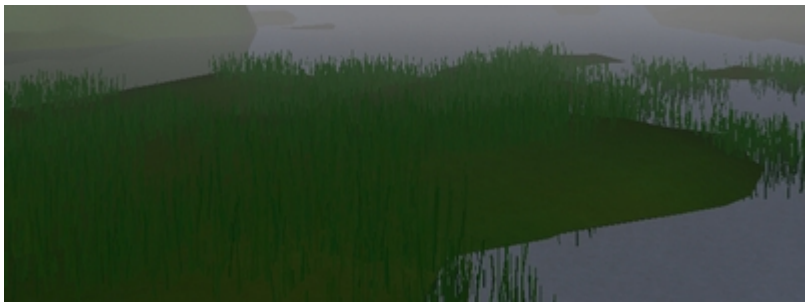
tráva vypadala přirozeně a z žádného úhlu kamery nešlo jednoduše rozeznat jednotlivé polygony. Samozřejmě bylo třeba hledat uspořádání s co nejlepším vizuálním dojmem a s co nejmenším počtem polygonů. Nakonec použité uspořádání používá celkem 11 polygonů kolmých na rovinu terénu. Přesné uspořádání těchto textur je vidět při pohledu ze shora na obrázku 5 vlevo. Toto uspořádání se ukázalo jako velmi efektivní při běžném procházení po povrchu terénu, ale není použitelné při pohledu na terén z vysoké výšky. V intru se však kamera nikdy nedívá z výšky přímo k zemi, proto je toto uspořádání dostatečné.



Obrázek 5: Vlevo polygony trávy, vpravo textura trávy

Zobrazované trsy trávy míří kolmo vzhůru, nenaklání se tedy podle terénu. Tak je to sice správně, jelikož tráva roste nahoru, ne kolmo na samotný terén, ale v případě zobrazení trávy v kopci vypadá rovnoběžná tráva velmi nepřirozeně. V intru je tento problém vyřešen generováním trávy jen na rovnějších plochách. Tento neduh by šel vyřešit neseskupováním trávy do zobrazitelných trsů, ale zobrazováním přímo samostatných textur trávy či přímo jednotlivých stébel. Takovým způsobem by se například dalo trávu i rozpohybovat a simulovat tak foukání větru.

Celkem je v intru vytvářeno 10 různých trsů trávy, přičemž každý obsahuje náhodné textury. Trsy trávy jsou na terénu rozmístěny náhodně a s různou rotací a místy se i navzájem překrývají. To vše ve výsledku působí velmi přirozeně.

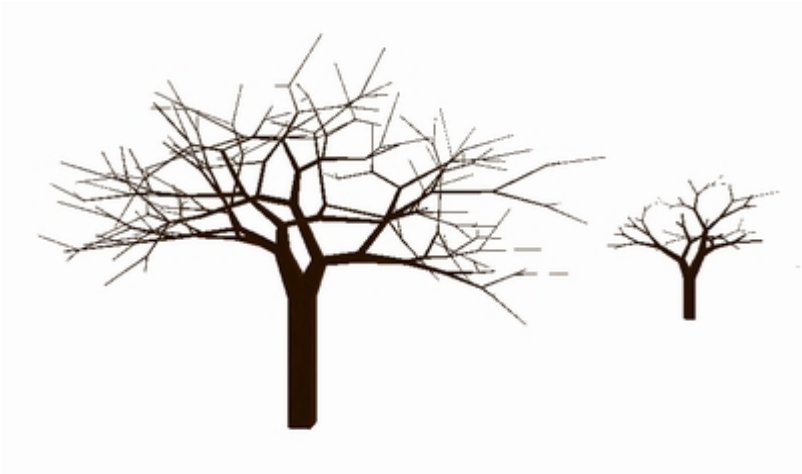


Obrázek 6: Trsy trávy

Stromy, křoví

Tvorba stromů a křoví se dělí na 2 části. Je třeba vygenerovat kmen s větvemi a také je třeba vygenerovat shluky listí na jednotlivých větvích, přičemž obě části jsou pro pozdější snazší úpravy uloženy ve vlastním objektu.

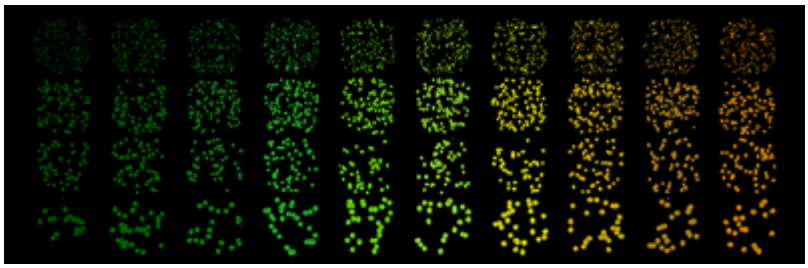
Stromy se generují ve funkci *Gen_tree()*, přičemž opakovaným zavoláním této funkce se pokaždé vytvoří trochu jiný strom. Generování větví stromu probíhá rekurzivně od kmene až k jednotlivým větvičkám dle předem stanovených parametrů pro daný typ stromu. V parametrech lze například nastavit tloušťku počátečního kmene, jak dlouhé bude mít strom větve, jak hodně bude strom košatý, jaké barevné schéma bude použito pro listí a několik dalších vlastností. Pomocí změny několika parametrů lze tedy vygenerovat výrazně odlišné typy stromů. Jen pomocí změny parametrů tato funkce vyrábí 3 různé typy stromů a dokonce 1 typ křoví. Všechny tyto 4 objekty jsou vyrobeny ve třech různých barevných schématech použitých pro listí. Celkem je tedy 12 výrazně odlišných stromů, které jsou na terénu rozmístěny různě otočené, takže divák nepozná, že se občas nějaký strom opakuje.



Obrázek 7: Kmen stromu

Při generování jednotlivých větví se na určité úrovni začnou generovat i shluky listí. Tyto shluky listí jsou jednoduché textury, obsahující náhodný počet jednotlivých lístků daného typu v daném barevném odstínu. Jednotlivé lístky jsou uspořádány tak, aby výsledná textura nevypadala hranatě, ale kulatě. Jakékoliv hrany nejsou v tomto případě žádoucí. U textur listí se využívá alfa-testing, díky kterému jsou nevyplněná místa v textuře průhledná. To umožňuje do jedné textury vygenerovat velké množství malých lístků, které tedy není třeba zobrazovat jednotlivě. Sada použitých textur s lístky je vyobrazena na obrázku 8, kde jsou v horizontálním směru vidět barevné

odstíny a ve vertikálním směru je vidět různá velikost a počet lístků. Pro lepší viditelnost jsou textury zobrazeny na černém pozadí.



Obrázek 8: Sada textur listí

Jelikož je zobrazování stromu relativně pomalé, jsou z výkonnostních důvodů vytvářeny zároveň dva objekty s listím. Jeden objekt listí obsahuje všechny shluky listí, tedy zhruba dvě textury na každou větvičku, druhý objekt listí obsahuje pouze zlomek těchto textur, ale s většími listy. Tyto dva objekty lze na jednom stromu přepínat a získat tak různě detailní (a různě náročnou) korunu stromu. Jednoduššího modelu listí lze využít při zobrazování vzdálených stromů, u kterých není potřeba nejvyšších detailů.

Obrázek 9: Kompletní strom

Při samotném zobrazování listí je vypnuto osvětlení, které by nevhodně osvětlovalo jednotlivé textury listí. Jak je vidět na obrázku 9, textury listí nejsou příliš patrné a listí se zdá být generováno jen kolem větviček.

Voda

Pro věrnější obraz se ve vodní hladině zrcadlí krajina a objekty nad vodou. Tento efekt je docílen zobrazením celé scény dvakrát – jednou reálně a podruhé jako zrcadlený odraz pod vodní hladinou.

Pro realističtější zobrazení vodní hladiny se na její povrch promítají dvě textury: textura sloužící jako barevný filtr pro vodní hladinu a textura symbolizující lehce špinavý vodní povrch, která se pohybuje a vytváří tak efekt tekoucí vody. Aby byl skrz tyto textury vidět odraz, jsou obě textury zobrazovány pomocí alfa-blendingu na konci programu.

Testováním na řadě diváků bylo zjištěno, že není třeba zobrazovat zrcadlení vždy a pro všechny objekty. Divák si sice zrcadlení všimne, ale později, pokud se ve scéně stále něco děje, již odrazy příliš nezkoumá. Proto není efekt zrcadlení z výkonnostních důvodu zapnut pro všechny objekty na scéně. Například drak se ve vodě neodráží. Toho si ale při shlédnutí intra všimne jen málokdo a proto se stejného ošizení scény využívá i při rychlých průletech scénou, kdy si člověk chybějících odrazů také nevšimne. V těchto rychlých scénách, konkrétně ve druhé polovině intra, se proto odraz vypíná úplně, což výrazně zrychlí zobrazování.

Obrázek 10: Zrcadlení terénu ve vodě

2.4 Model draka

Vytvoření modelu

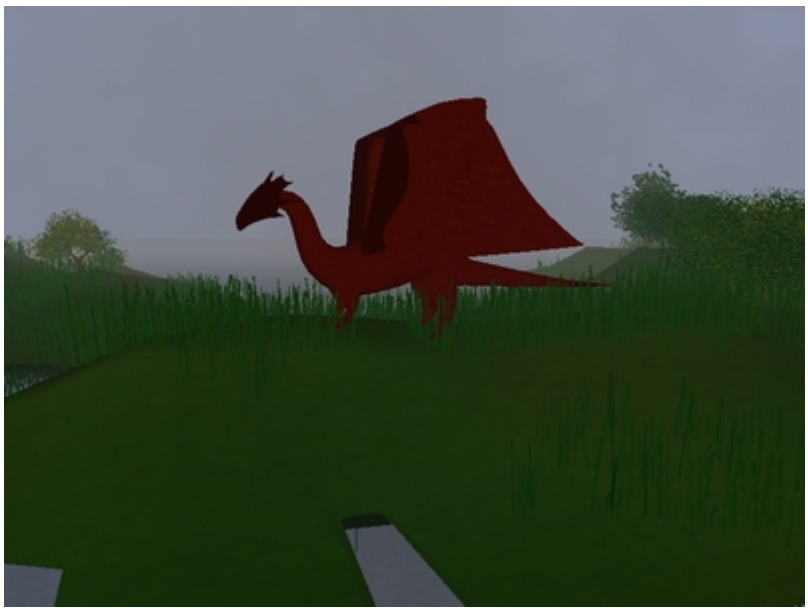
Model draka byl nejprve vytvořen jako celek v 3D modelovacím programu Cinema 4D [7]. Model byl poté rozřezán na několik kusů, aby byl jednodušeji animovatelný a jednotlivé díly byly exportovány do formátu „direct 3D“ s koncovkou „.x“. Tento formát byl ze všech možných vybrán pro svojí jednoduchost, jelikož pro jeho pochopení nebylo třeba studovat žádné další materiály. Je to

totiž textový formát obsahující samovysvětlující popisky dat a dokonce i komentáře. Navíc data v tomto formátu jsou uložena stejným způsobem, v jakém jsou ukládány v intru, tedy pole vrcholů a pole polygonů. Formát však obsahuje i spoustu dat, která nejsou pro intro potřebná, například normály, texturovací koordináty a specifikace materiálu. Tyto informace si intro samo dopočítává.

Pro jednoduché a hlavně rychlé vytváření potřebných hlavičkových souborů, které lze jednoduše přiložit do projektu, byl vytvořen samostatný konvertor. Tento konvertor naráz převede všechny modely použité v intru do formátu jazyka C a uloží je jako hlavičkové soubory přímo do složky projektu. Při změně modelu tedy stačí model znovu vyexportovat, spustit tento konvertor a překompilovat intro.

Uložení draka

Animace modelu draka byla docílena rozřezáním draka na několik části: hlava, tělo, ocas, křídla a nohy. Jelikož je drak symetrický, je v některých případech možné uložit pouze polovinu dat. Pro rekonstrukci celého draka bylo potřeba uložit jen jedno křídlo, jedna přední noha, polovina těla, polovina hlavy a ocas. Pro druhé křídlo a druhou přední noha se použijí stejná data jako první, jen se zrcadlí podle osy draka. Zadní nohy jsou vyrobeny ze stejných dat jako přední, jen jsou při načítání zvětšeny. Tělo s hlavou a krkem jsou uloženy rozpůlené podle osy draka a při načítání se druhá polovina dopočítá. Ocas musel být uložen celý kvůli možnostem jeho animace.



Obrázek 11: Drak

Rozpohybování draka

Základním úkolem pro animace draka bylo zajistit otáčení jednotlivých částí, aniž by docházelo k viditelným chybám, jako například oddělení křídel od těla při jejich mávání. Modely proto musely být mírně upraveny, aby vypadaly dobře ve všech fázích takovéto animace. Další úpravy částí modelu byly potřebné kvůli algoritmu Catmull-Clark, který modely trochu zmenší. Model draka umí takto pohybovat s křídly a všema nohama.

Při pouhém otáčení jednotlivých částí však drak působí velmi „dřevěně“. Proto byly implementovány i animace polygonálních dat modelů. Pro každou animaci je vytvořena řada velmi podobných modelů, které se od sebe liší jen posunutím některých bodů. Postupným střídáním těchto modelů dochází k animaci. Ukládat celu řadu těchto modelů by nebylo efektivní, proto se pro každou animaci ukládá pouze počáteční model a koncový model. Ostatní modely animace se lineárně interpolují z těchto dvou krajních modelů. Díky rozřezání draka na jednotlivé animovatelné části se v každém uloženém modelu nemusí opakovat části draka, které se neanimují.

Obrázek 12: Rozbalování křídel a vrtění ocasu

Drak kromě mávání křídly a pohybování nohama umí i deformovat křídla za letu, otevírat pusu při chrlení ohně a mávat ocasem několika směry. Drak také hýbe hrudníkem, čímž simuluje dýchání, a při přistání/vzletu sbaluje/rozbaluje křídla. Přidání jakékoliv další animace není problém, stačí vytvořit dvě verze stejného modelu, zavolat na ně funkci *Load_animation()*, která se postará o vytvoření všech mezikroků animace, a při zobrazování zajistit střídání vytvořených modelů. Jen je nutno dodržet podmínku, že dané dva modely pro animaci mohou být rozdílné jen v souřadnicích jednotlivých bodů. To je nezbytné pro správné dopočítání všech modelů a v případě nedodržení této podmínky by mohlo dojít k různým chybám. Při animaci jednotlivých částí se tedy uplatňují jak popsané rotace, tak i změny polygonální sítě modelu, čímž je docíleno větší živosti draka.

Obrázek 13: Animace křídel při letu

2.5 Animace

Jelikož má být výsledné intro animované, je nutné připravit prostředky pro animování celého videa. Důležitou roli hraje synchronizace s časem. Je totiž důležité, aby se děj intra pohyboval nezávisle na aktuální rychlosti zobrazování. Pokud by tomu tak nebylo, animace by se v některých místech mohla nepříjemně zrychlovat či zpomalovat. Základem přehrávání je tedy běžící reálný čas, přičemž je umožněno v libovolném čase jednoduše vyvolat předem definovanou událost, například animaci nějakého objektu.

Seznam všech naplánovaných událostí je definován ve funkci *Calenar()*, která se spouští na začátku zobrazování každého snímku a podle aktuálního času se dívá, jestli je třeba provést nějakou událost. Touto událostí může být výpočet, volání funkce či změna proměnné. Všechny událostí jsou pro přehlednost seřazeny dle času startu události a pomocí časových podmínek strukturovány do několika časových bloků, které urychlí procházení. Tento způsob je pro krátké intro s pár desítkami událostí dostatečný, avšak pro případné delší scény s daleko více událostmi by tento způsob byl neefektivní. Pro ně by bylo vhodné události zřetězit, například do časově seřazeného lineárního seznamu, a s postupem času intra se tímto seznamem jednosměrně pohybovat. Tím by se při každém zavolání testovala jediná časová podmínka a pouze pokud by byla splněna, testovala by se další atd. Oproti řešení s lineárním seznamem událostí se v mém případě testuje při každém snímku až desítky podmínek navíc.

Mezi nejčastější události patří nastavování nové hodnoty nějaké proměnné v přesně definovaném čase a výpočet pořadového snímku nějaké animace. Jelikož všechny animace modelů mají uloženy stejný počet modelů pro každou animaci, je možné pro všechny tyto animace používat jednotné funkce. Pomocí funkce *AnimateUp()* se provádí animace od prvního do posledního modelu a pomocí funkce *AnimateDown()* se provádí animace opačná. Tyto dvě funkce nastavují jedinou proměnnou, ve které je uloženo, který model dané animace se bude zobrazovat. Jedinými dalšími

parametry těchto funkcí jsou dvě proměnné, kterými se dá měnit čas začátku animace a doba trvání animace. To umožňuje snadný a velmi přehledný způsob jak definovat desítky různých animací.

Scénář

Před samotným režírováním animace je vhodné mít připraven scénář, podle kterého bude celé intro probíhat. Celé intro se dělí na 3 základní části – načítací obrazovka, hlavní obsah a závěrečné titulky. Z hlediska scénáře je třeba plánovat především obsahovou část, ve které probíhá to nejdůležitější.

Hlavní obsah začíná klidným průletem krajinou, při kterém se divák seznámí s terénem a postupně všemi typy objektů. Po pár průletech krajinou se hlavním objektem pozornosti stává drak. Tento drak se nejen sám pohybuje, ale je i animovaný – umí pohybovat jednotlivými částmi svého těla. Navíc, jak od něj naprostá většina diváků očekává, drak bude několikrát i plivat oheň.

Zatímco v první části se tedy zobrazují nepohyblivé objekty, druhá část je značně dynamičtější a jsou v ní prezentovány všechny drakovi „dovednosti“.

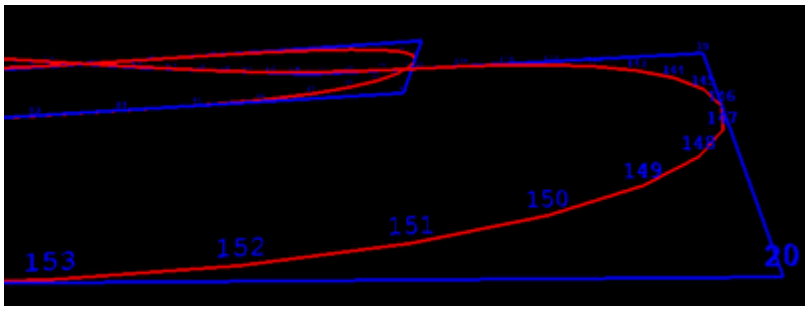
Pohyb po křivkách

Úplně základním prvkem animace je umožnění pohybu kamery a objektů ve scéně. Pohyb kamery by měl být také plynulý a co nejjednodušeji definovatelný. Pro splnění těchto podmínek je vhodné proložit dráhu kamery a rotace kamery do křivek. Křivky jsou výhodné především pro jejich nastavitelnou hladkost a pro malé paměťové nároky, jelikož při použití křivek se dá celá trajektorie pohybu definovat pouze několika řídícími body. Oproti pohybu jen po přímkách se s křivkami také lépe pracuje, jelikož není třeba řešit zaoblení hran a při libovolných změnách obvykle stačí měnit podstatně menší počet bodů.

V intru byly testovány dva typy křivek: Béziérovy křivky implementované rekurzivním algoritmem „de Casteljau“ a zjednodušené křivky NURBS nezohledňující váhové koeficienty řídících bodů. Křivky NURBS nakonec pro proložení drah a rotací zvítězily, jelikož poskytují daleko větší jednoduchost a komfort při jejich tvorbě a úpravách.

Samotný pohyb kamery i objektů spočívá v pohybu po krátkých úsečkách, které jsou vypočítány z několika řídících bodů pomocí křivek. Na těchto krátkých vypočítaných úsečkách dochází k lineární interpolaci souřadnic a rotací počátečního a cílového bodu. Zjištění aktuálních souřadnic a rotací pro zobrazení obstarává funkce *move()*, která dle aktuálního času vrátí interpolované hodnoty z aktuální úsečky. Pro pohyb kamery je v programu definováno a staticky uloženo 50 bodů, přičemž každý bod obsahuje 7 hodnot: 3 souřadnice kamery, 3 hodnoty rotací a 1 hodnotu určující čas. Z těchto 50 bodů je pomocí křivek vypočítáno 500 krátkých úseček, po kterých přímo probíhá pohyb kamery. Pohyb draka je uložen na pouhých 28 takových bodech. Rotace draka se ale počítají automaticky ze souřadnic a v tomto formátu jsou uloženy jen pro jednoduchost

a univerzálnost algoritmů. Rotace draka nejsou třeba vůbec ukládat, protože jdou spočítat. Drak se totiž vždy dívá ve směru pohybu – necouve, ani nedělá úkroky, takže se úhel natočení dá spočítat z dvou po sobě jdoucích bodů. Pro draka se počítá i úhel natočení při letu, například pokud drak zatáčí doleva, celý se přitom také nakloní doleva.



Obrázek 14: NURBS křivky

Animace draka

Drak se stává ústředním objektem druhé poloviny intra ihned po svém příletu, kdy několika přískoky zabrzdí a složí svá křídla. Při krátkém průletu kolem draka si divák může všimnout, že drak po celou dobu dýchá, což je simulováno pohybujícím se hrudníkem. Po rychlé prohlídce drak zamává ocasem, rozbalí křídla a s odrazem od země opět vzlétne. Při letu napne zadní nohy a s animovaným máváním křídel letí krajinou. Drak i několikrát otevře pusu a vychrlí oheň. V závěru intra pak opakovaným chrlením ohně spálí strom.

Pohled na draka je poskytnut ze všech možných úhlů díky rotaci kamery kolem samotného draka. Těžší částí tvorby výsledného videa bylo zajistit, aby byl drak stále vidět ve všech záběrech, a to i při pohybu kamery kolem draka. Potřebné souřadnice kamery byly nakonec získány následujícím způsobem:

1) Nejprve se nadefinovala trajektorie draka.
2) Vypnul se automatický pohyb kamery a povolil se pohyb pomocí klávesnice.
3) Celá animace se nastavila na manuální krokování po krátkém časovém intervalu (1s).
4) V průběhu krokování se pohybem pomocí klávesnice nastavila pozice kamery na požadované místo a s požadovanou rotací tak, aby byl vidět drak.
5) Požadované souřadnice a rotace se pro daný čas animace uložily stisknutím klávesy.
6) Uložené body se překopírovaly do zdrojových souborů jako souřadnice a rotace kamery.

2.6 Efekty

Oheň

Oheň, který drak chrlí, je tvořen částicovým systémem s 6 000 částicemi, které se pohybují ve všech 3 prostorových souřadnicích. V případě ohně chrleného drakem jsou částice usměrněny jedním směrem a to tak, že pro každou částici je vytvořen její směrový vektor, který se od hlavního směru plamene liší jen s drobnou náhodnou odchylkou. Při každém snímku se pak pro každou částici spočítají nové souřadnice právě přičtením jejího směrového vektoru vyděleného aktuálním počtem snímků za sekundu. To zajistí plynulý a stejně rychlý pohyb částice při různých rychlostech zobrazování i při změnách v rychlosti zobrazování.

Po spuštění efektu ohně jsou jednotlivé částice postupně zapojovány do efektu, čímž simulují opravdový let ohnivého sloupce. Jakmile částice dorazí až na definovaný konec, je resetována a začíná opět v počátku. Takto se to opakuje pro všechny zapojené částice, dokud nedojde k ukončení efektu. Po ukončení efektu všechny letící částice doletí a poté se již neobnovují. Oproti možnosti vypnout naráz zobrazovaní všech částic při skončení efektu působí toto řešení přirozeněji. Při tvorbě animace se ale musí počítat s tím, že po definovaném konci ohnivého efektu částice ještě chvíli letí.

Každá částice je tvořena jedním polygonem s texturou gradientního kola, která se zobrazuje pomocí alfa-blendingu při vypnutém hloubkovém testování. Vypnuté hloubkové testování má nevýhodu, že částice jsou vidět vždy, i když jsou za nějakou překážkou. Částice sice lze překrýt celým objektem jednoduchou změnou pořadí vykreslování, ale bez hloubkového testování není automaticky zajištěno nezobrazování částic, které jsou již například pod povrchem terénu. S tím je ale v intru počítáno a proud ohně je zobrazován jen tam, kde mu nestíní žádná překážka. Hloubkové testování má však výhodu, že při zobrazování poloprůhledných částic alfa-blendingem nemusíme řešit pořadí jejich vykreslování.

Obrázek 15: Částicový systém ohně z různých úhlů

Ohoření stromu

V závěru intra drak několikrát chrlí oheň na jeden definovaný strom, který jeví známky působení ohně. Při první ohnivé dávce celý strom i s listím postupně zčerná. Po druhé dávce změní listí svůj vzhled na hrubé seškvařené listí a při třetí dávce toto listí plynulým přechodem průhlednosti úplně zmizí a zbude pouze holý strom. Celkově je tedy tento efekt složen pouze z těchto tří jednoduchých efektů. Pro větší věrohodnost by mohlo pomoci, kdyby strom po prvním zažehnutí sám vzplanul. k tomu by byl třeba další částicový systém, který by mohl realističnost podtrhávat i zobrazováním kouře.

Mlha

Pro větší realističnost a pro maskování některých druhů optimalizací scéna obsahuje mlhu. Ta je generována pomocí vestavěných funkcí OpenGL, které stačí pouze zapnout. Mlha byla zapnuta s nastavením nejlepší kvality (GL_EXP2), jelikož nebyl zaznamenán výkonnostní rozdíl oproti méně kvalitním nastavením.

Texty

Intro podporuje zobrazování 3D textů, které umožňují do scény umístit barvené či texturované prostorové texty. V intru je ve funkci *Gen_font()* vytvořeno jedno základní písmo, přičemž pro zobrazování různých velikostí písma je použito zvětšování pomocí OpenGL funkce *glScalef()*. Texty však nakonec nebyly v hlavní části intra použity, protože se do zvoleného tématu příliš nehodí. Při zobrazování spousty textů by divák získal dojem obrovské obsáhlosti intra i když zkomprimované texty by zabíraly minimum prostoru. V mém intru je však pocitu obsáhlosti intra dosaženo tím, že intro každou chvíli představuje nějaký nový efekt a divák by další texty stejně nestíhal číst. Textů bylo však použito pro ladící účely a v závěrečných titulkách, ve kterých se zobrazí dlouhé bloky textu i s obrázky.

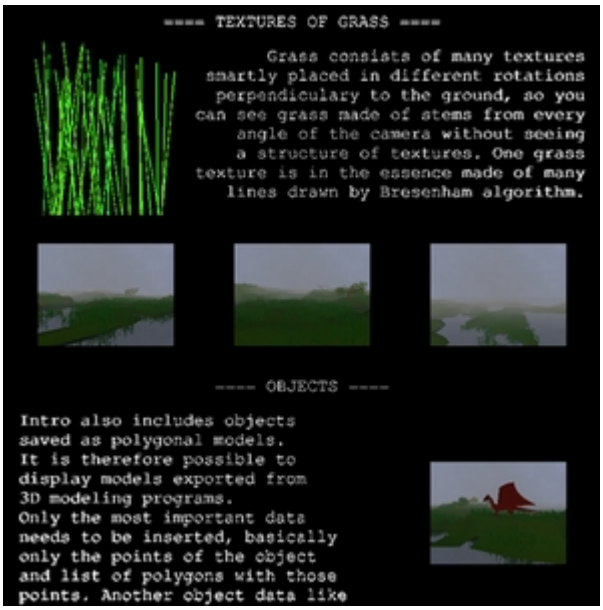
Závěrečné titulky

Po skončení hlavního děje celá scéna během několika sekund plynule tmavne až přejde k úplně černé obrazovce, ve které začnou pomalu vyjíždět závěrečné titulky. V závěrečných titulkách jsou nejen informace o autorovi a důvodu vzniku intra, ale pomocí textů a obrázků shrnuty hlavní použité techniky, objekty a efekty, které divák během intra mohl vidět.

Pro zobrazování textů použitých v závěrečných titulkách se používá funkce *Print()*, která přijímá různé formátovací řetězce, jako její vzor *printf(),* který se používá pro tisk do konzole. Funkce *Print()* však umožňuje i definovat zarovnání textu. Podporované je zarovnání doleva, doprava a na střed. Navíc za sebou funkce automaticky odřádkovává. Pro tisk delších bloků textu je však tato

funkce zapouzdřena ve funkci *Print_block()*, která zadaný blok textu rozdělí na jednotlivé řádky a přečte i jejich barvu, která může být speciálně uložena na začátku každého řádku. Pokud řádek začíná dvojtečkou, očekávají se za ní tři textově zapsané číslice 0-9, přičemž každá číslice odpovídá intenzitě jedné barevné složky, tedy popořadě červené, zelené a modré. To umožňuje vytvořit 1 000 různých barev, které pro zobrazování textu bohatě stačí.

V intru se zobrazují nejen různě zarovnané a barevné texty, ale i obrázky. Obrázky mohou být stejným způsobem zarovnávány a lze u nich i definovat, jestli se za nimi bude zalamovat řádek. Díky tomu lze obrázky i text zobrazovat vedle sebe do sloupců. Všechny obrázky v závěrečných titulkách jsou tvořeny z jednoho polygonu s jednou nanesenou texturou. Může se jednat o skutečně použitou texturu v intru nebo o texturu, která byla vytvořena v průběhu intra vyfocením celého okna.



Obrázek 16: Závěrečné titulky

Úvodní načítání

Součástí intra je i jednoduchá úvodní obrazovka, která informuje diváka o průběhu načítání a vytváření všech potřebných dat. Přestože je program hluboko zanořený v nějaké funkci vytvářející data, po každé krátké dokončené části se zavolá funkce *Draw_loading()*. *Tato funkce* překreslí informativní proužek o průběhu celkového načítání a zpracuje příchozí události, takže je možné si při načítání přizpůsobit velikost i umístění okna. Úvodní obrazovka by mohla obsahovat podstatně více, než jen obyčejný proužek, ale jelikož se nejedná o hlavní část intra, byla zvolena nejjednodušší a nejpřehlednější varianta.

Stereo

Celé intro je možné se správným vybavením sledovat v opravdovém 3D. Možnosti pro výrobu stereoskopického obrazu je více. Zjednodušeně řečeno se ale využívá buď princip, při kterém se každému oku zobrazuje vlastní obraz (například pomocí 3D brýlí), nebo se využívají barevné filtry a speciálně vytvořené obrázky zkombinované z obrazu pro levé a pravé oko.

Stereo je v intru řešeno pasivně, tedy celá scéna se zobrazuje dvakrát vedle sebe za použití OpenGL funkce *glViewport()*. Druhá možnost by byla využít stereo OpenGL režimu, ve kterém by se scéna vykreslovala také dvakrát, ale jen do bufferu, přičemž v nastavení grafického ovladače by se obraz nechal klonoval tak, aby každý monitor dostával svou verzi obrazu. Pasivní režim má výhodu, že jde spustit i bez druhého monitoru a lze v něm snímky jednoduše vyfotit a pak s nimi dále pracovat. Má ale nevýhodu, že je při spuštění intra nutné ručně nastavovat velikost okna. To by však šlo automatizovat, nebo alespoň zjednodušit načítáním vlastností okna pomocí parametrů programu.

Pro zajištění 3D obrazu je nutné okno intra roztáhnout přes dvě pracovní plochy na dvou monitorech tak, aby se na jednom monitoru zobrazovala levá scéna a na druhém pravá scéna. Poté je možné se na obraz dívat například přes speciální digitální 3D brýle, které každému oku zobrazují jeden monitor, tedy scénu určenou pro dané oko. Tyto dvě velmi podobné scény si lidský mozek spojí dohromady a vytváří tak efekt opravdové prostorovosti.

Tyto dva obrazy se liší trochu odlišným nastavením kamery. Pro výpočet nastavení těchto kamer se používá princip tzv. „Toe-in“ [8], který není úplně přesný, ale je nejjednodušší oproti principu tzv. „Off-axis“ [8]. Princip spočívá v zobrazení scény podobně, jako scénu vidí člověk, tedy levý obraz odpovídá levému oku a pravý obraz pravému oku, přičemž obě oči jsou od sebe kousek vzdálené. Oči se dívají stejným směrem a v ideálním případě by se pomyslné paprsky těchto očí měly sbíhat na právě pozorovaném objektu. Toho v intru není dosaženo, jelikož se cíle pozornosti přibližují a vzdalují. Proto se paprsky sbíhají v odhadované průměrné vzdálenosti. V ladící verzi je možné měnit vzdálenosti očí i úhel kamery, který ovlivňuje vzdálenost zaostření.

Stereo jde v programu zapnout spuštěním s parametrem „stereo“, nebo se do tohoto módu přepnout za chodu s klávesovou zkratkou „v“.

2.7 Zvuk

Pro minimalistické řešení nebylo možné použít například kvalitu mp3 souborů, ale sáhnout po méně paměťově náročném formátu. Použitelný by mohl být například formát MIDI, ve kterém je uložen v podstatě jen seznam not dané melodie. V tomto intru však používám formát V2m, který oproti MIDI zajišťuje větší možnosti a kvalitu zvuku. Pro interpretací tohoto formátu je třeba přehrávač,

takzvaný syntetizátor. Tento syntetizátor jsem nevytvářel vlastní, jelikož výroba vlastního by byla nad rámec této práce. V tomto intru používám externí volně dostupný syntetizátor „V2 synthesizer system“ [9] ve verzi 1.0. Tento syntetizátor poskytl pro ostatní tvůrce dem autor Tammo "kb" Hinrichs ze známé německé demoskupiny farbrausch [10].

Moje grafické intro doprovází hned dvě melodie. Jedna melodie se jmenuje „Ouverture“ od již zmíněného autora kb a druhá se jmenuje „To_short“ od autora s přezdívkou Quickyman. Implementace hudby do intra znamenalo přiložit do projektu knihovny syntetizátoru a v programu volat jednoduché funkce pro inicializaci, spuštění a ukončení přehrávání. Jediný problém bylo předat inicializační funkci data z hudebního souboru, jelikož hudební data musí být součástí aplikace. Z tohoto důvodu byl napsán konvertor pro převod souborů do hlavičkových souborů jazyka C. Dva hudební soubory jsou tedy překonvertovány do dvou konstantních polí a jsou zapsány v hexadecimálním tvaru ve dvou hlavičkových souborech.

Na 2 z asi 25 testovaných sestavách se intro při ukončení načítání celé zaseklo a bylo operačním systémem ukončeno. Bylo to způsobeno chybně nainstalovaným ovladačem zvukové karty, popřípadě úplně chybějícím zvukovým hardwarem. Z tohoto důvodu je implementována možnost spustit intro bez hudby zadáním parametru „nosound“ při spuštění programu.

2.8 Optimalizace a ladění

Pro rychlé zobrazování intra bylo nutno po celou dobu vývoje hledat nejefektivnější cesty k zobrazení jednotlivých detailů nebo efektů. Při nalezení dvou rozdílných metod pro zobrazení nějakého efektu obdobné kvality obvykle zvítězila ta rychlejší metoda.

Kvůli rychlosti výsledné animace se zobrazuje ve scéně jen to, co je vidět. Ostatní objekty, které ve scéně nebudou zobrazeny se pomocí jednoduchého výpočtu předem vyřadí a nezatěžují zbytečně grafickou kartu. Viditelnost objektů se v programu zjišťuje velmi jednoduchým, ale dostatečným způsobem. Zobrazují se všechny objekty v kouli, která je definována tak, že kamera je umístěna na povrchu této koule a střed koule je umístěn ve směru kamery v poloviční vzdálenosti, na kterou chceme vidět.

V některých případech došlo k úmyslnému snížení kvality obrazu pro zajištění vyšší rychlosti zobrazování. Příkladem může být vypínání odrazů ve vodní hladině, pokud se scéna pohybuje příliš rychle na to, aby si jich někdo všímal.

Další implementované snížení kvality zobrazení bylo implementováno u stromů pomocí techniky LOD (level of detail). Stromy obsahují různé úrovně detailů a pro vzdálenější objekty

se nepoužije plnohodnotný model, ale jen zjednodušená verze, která je méně náročná na zobrazení. Bohužel těchto úrovní detailů je málo a při postupném vzdalování objektu jsou vidět znatelné skoky.

Možnosti ladění

Pro samotnou výrobu intra byla použita řada funkcí a možností, které nejsou ve výsledném intru potřeba. Přesto ale některé tyto funkce ponechávám v kódu a pomocí parametrů překladu je umožňuji zahrnout do výsledného programu. Je tedy možné intro přeložit ve 2 verzích: normální verze a tzv. „debug“ verze, která značně rozšiřuje ovládání programu z klávesnice a umožňuje zobrazit řadu testovacích a pomocných objektů. V následující tabulce jsou popsány nejdůležitější klávesové zkratky použité v debug verzi. V normální verzi fungují pouze první dvě klávesové zkratky – ESC a V.

Kl. zkratka	**Funkce**
ESC	Ukončí celé intro.
V	Přepíná stereo zobrazení.
K, L	Zvětšuje a snižuje vzdálenost očí při stereo zobrazení.
O, P	Zvětšuje a snižuje úhel kamery při stereo zobrazení.
G	Zapíná a vypíná ladící mód. Při zapnutí zobrazuje pomocné objekty, texty, křivky atd. a při zastavené animaci povoluje otáčení kamery pomocí myši.
Šipky	Šipky nahoru, dolů, doleva a doprava otáčí kameru daným směrem.
W, S, A, D	Jednotlivé klávesy slouží pro pohyb ve scéně dopředu, dozadu, doleva a doprava.
R	Velmi zrychlený pohyb dopředu umožňující rychlé průlety scénou.
Mezerník	Pauza – pozastavení animace a pokračování v animaci.
C	Restartuje celou animaci. Funguje pouze při pozastavení animace.
M	Zapíná a vypíná multitexturing.
T	Zapíná a vypíná zobrazení stromů, křoví a trávy.
F	Zapíná a vypíná mlhu.
B	Tiskne do konzole čas animace, aktuální souřadnice a rotace kamery.
X	Vyfotí aktuální scénu a uloží ji do bitmap souboru ve složce intra.

Tabulka 4: Seznam použitých klávesových zkratek v ladící verzi programu

Při programování intra byly tyto klávesové zkratky často používány a urychlily tak samotný vývoj celého intra. Například zobrazením ladících objektů a pozastavením animace si lze prohlédnout, po jakých trajektoriích se pohybuje kamera i drak. Nejdůležitějším ladícím nástrojem tedy byl intuitivní pohyb scénou pomocí klávesnice a myši, který je koncipován podobně jako v akčních hrách. Pohybem po scéně je tedy možné vidět celý terén, všechny objekty i všechny vygenerované textury a to ze všech úhlů a v libovolném čase animace. Při ladění byla také často

využívána konzole, do které se tiskly ladící informace. Část těchto výpisu je v kódu stále ponechána v zakomentované podobě.

Zejména pro focení obrázků do této práce bylo pomocí funkce z [11] implementováno ukládání obrázků do souboru jako bitmapy. Pro samotné vyfocení scény i zde používá OpenGL funkce *glReadPixels()*. Pomocí tohoto focení a plynulého krokování animace bylo celé intro snímek po snímku přefoceno a složeno do klasického video souboru.

Parametry překladu a spuštění

Pro překlad jsou v projektu díky dvou parametrům definovány celkem 4 různá nastavení, která kombinují všechny možnosti parametrů. Jedním parametrem je „DEBUG“, jehož použitím se do projektu přiloží ladící funkce. V nastavení projektu je při použití parametru „DEBUG“ zobrazována pod oknem intra i okno s konzolí, do které se tisknou případné ladící informace. Druhým parametrem je „SOUND_ON“, který přeloží intro s hudbou. Pokud se tento parametr nepoužije, intro se přeloží kompletně beze zvuku, tedy jak bez použitého zvukového syntetizátoru, tak bez přiložených melodií. Zvuk jde ale u obou verzí explicitně vypnout spuštěním výsledného intra s předaným parametrem „nosound“. Druhý použitelný parametr příkazové řádky je příkaz „stereo“, který automaticky zapíná stereo zobrazení.

2.9 Velikost intra a komprese

Pro zmenšení velikosti intra bylo použito mnoho technik a postupů již při programování obsahu. Také byly do finální verze zahrnuty jen funkce, které byly opravdu potřeba. Místo bylo také ušetřeno nižší kontrolou správného chodu programu. Například byly z finální verze odstraněny některé kontroly logických podmínek, které ověřovaly, že programátor na nic nezapomněl.

Pro další zmenšení výsledného programu je však možné ještě nastavit speciální parametry překladu v překladači GNU GCC Compiler. Samozřejmostí je nevkládat do výsledného programu žádné ladící symboly ani profilovací informace. Navíc překladač podporuje přepínač „-s“, který by měl ořezat symboly z binárního souboru a tím zmenšit velikost, a přepínač „-Os“, který je zaměřen přímo na zmenšení velikosti. Zmenšování velikosti tímto způsobem by ale mohlo být na úkor výkonu. Při neoptimalizování pro velikost by se některé bloky kódu mohly naopak rozepsat na více místa a zrychlit provádění kódu. Při testování mého intra se ale toto tvrzení nepotvrdilo. Intro optimalizované pro velikost bylo zhruba stejně rychlé jako intro optimalizovaného pro rychlost. Pravděpodobně je to způsobeno tím, že při použití optimalizací pro velikost pomocí přepínače „`-Os`“ se podle [12] použijí i optimalizace pro rychlost „-o2“, které typicky nezvětšují velikost. Výsledky rychlosti a velikosti intra při různých nastavení kompilátoru jsou shrnuty v následující tabulce, která

porovnává čtyři základní nastavení kompilátoru: žádné optimalizace, optimalizace jen pro výkon, optimalizace jen pro velikost, optimalizace pro výkon i pro velikost.

	Verze se zvukem		**Verze bez zvuku**		**Ladící verze**	
Použitá optimalizace	**Velikost**	**FPS**	**Velikost**	**FPS**	**Se zvukem**	**Bez zvuku**
Žádná	257kB	21,31	139kB	20,18	269kB	150kB
Pro rychlost	197kB	25,81	108kB	26,61	210kB	121kB
Pro velikost	104kB	27,05	56,5kB	26,77	114kB	66,5kB
Pro rychlost i velikost	113kB	26,14	65,5kB	26,51	124kB	76,5kB

Tabulka 5: Výsledky různého nastavení kompilátoru

FPS značí průměrné snímky za sekundu na testovaném notebooku Acer Aspire 5651 (Intel Core Duo T2050 1.66GHz, 2GB RAM, nVidia GeForce Go 7600 256MB, Microsoft Windows XP). Zajímavé je, že na tomto notebooku dosahuje intro lepších FPS, než na některých výkonnějších sestavách. Způsobeno je to nejspíše tím, že intro bylo na tomto notebooku vyvíjeno a při výběru různých implementací byla vždy zvolena ta rychlejší varianta právě na tomto stroji. Proto by se dalo říci, že intro je pro tento stroj vyladěno lépe, než pro kterýkoliv jiný.

I přes zapnuté optimalizace se však velikost aplikace nevejde do požadovaného limitu 64kB. Tento limit splňuje pouze verze beze zvuku, která zabírá při maximální optimalizaci 56,5kB. Existuje však ještě jeden způsob dodatečného zmenšení velikosti - komprese. Pro celkovou kompresi binárních souborů existuje řada komprimačních nástrojů, které obvykle velmi výrazně zmenší velikost výsledného programu. Většinou fungují na podobném principu a využívají řadu různých technik a komprimačních algoritmů. Po spuštění takto zmenšeného programu se nejdřív spustí dekomprimační nástroj, který rozbalí do paměti výsledný program a poté jej spustí. Nevýhoda tohoto komprimovaní programů je v tom, že některé antivirové programy chápou tento způsob komprese jako nekalou praktiku a takto komprimované soubory označují za hrozbu a blokují jejich spuštění. Asi díky tomu se tento způsob komprese programů nepoužívá pro běžné aplikace. Pro jistotu je v příloze na DVD uložena jak zkomprimovaná tak nezkomprimovaná verze intra.

Pro komprimování jednotlivými nástroji byly testovány různé verze daných programů s obvykle několika různými nastaveními. Byla zkoušena i opakovaná komprese jednotlivými nástroji navzájem, ale daná operace se buď neprovedla, nebo nebylo intro spustitelné. V následující tabulce jsou shrnuty pouze nejlepší dosažené výsledky pro danou verzi nástroje. Komprimována byla finální verze se zvukem o velikosti 104kB.

Nástroj	**Verze**	**Finální intro [kB]**	**Zmenšeno na [%]**
Aspack [13]	2.2 demo	49,5	47,59
Upx [14]	3.03w	45	43,27
kkrunchy [15]	0.23a	39	37,50
kkrunchy [15]	0.23a2	37	35,58
WinUpack [16]	0.39e	40,7	39,13

Tabulka 6: Úspěšnost komprese různými nástroji

Nejúspěšnější nástroj pro komprimaci mého intra byl program kkrunchy ve verzi 0.23a2, který dokázal srazit velikost intra na pouhých 37kB. Tento komprimační nástroj byl vytvořen již zmíněnou demoskupinou farbrausch [10] a je používán pro komprimaci velké části dem v celé demoscéně.

3 Závěr

Výsledné intro o velikosti pouhých 37kB nabízí 3 minuty dlouhé video, ve kterém se prezentuje mnoho objektů a efektů, a 3 minuty závěrečných titulků se spoustou textů a obrázků z intra. Navíc na pozadí celého intra hraje hudba - dvě písničky s celkovou délkou téměř 4,5 minuty. Do limitu 64kB má intro velkou rezervu a díky všem připraveným nástrojům a funkcím by bylo možné intro rozšířit o značně množství dalších objektů, efektů a animací. Při chytrém návrhu, použití minimalistických technik a komprese nebyl limit 64kB omezující.

Díky vestavěným funkcím pro kreslení, křivky, textury a pro práci s 3D objekty umožňuje intro zobrazit a animovat téměř libovolnou scénu při přijatelné velikosti výsledné aplikace. Implementována byla řada jednoduchých i složitých algoritmů, přičemž jejich implementace není limitována pouze pro účely tohoto intra, ale po drobných úpravách by mohla být použita i do jiných grafických projektů.

Celkově jsem při tvorbě tohoto intra získal mnoho nových znalostí z oblasti tvorby 3D grafiky a především nové praktické zkušenosti s OpenGL a jeho rozšířeními. Také jsem se seznámil s tvorbou programů s limitovanou velikostí a možnostmi jejich komprese.

Další pokračování projektu bych viděl ve vylepšení současných detailů, především textur, materiálů a osvětlení. Další možností je tvorba nových objektů, animací či celých scén, aby se zaplnil celý limit a naplno se využily již implementované algoritmy. Funkčně by šlo intro rozšířit například o výpočet a zobrazování stínů, opravdovou kinematiku 3D objektů, různé způsoby deformace objektů a o nové efekty.

Literatura

Všechny zde zmíněné WWW stránky byly dostupné dne 1.5.2009 a 15.5.2009.

[1] Dr. Ing. Petr Peringer: Modelování a simulace – IMS - Studijní opora, Brno, FIT VUT v Brně. 19.11.2008

[2] WWW stránky – Code::Blocks, http://www.codeblocks.org/

[3] WWW stránky - GCC, the GNU Compiler Collection - Free Software Foundation (FSF), http://gcc.gnu.org/

[4] WWW stránky – Ken Perlin's homepage, http://mrl.nyu.edu/~perlin/

[5] WWW stránka – Wikipedia, the free encyklopedia, http://en.wikipedia.org/wiki/Catmull-Clark_subdivision_surface

[6] Ing. Přemysl Kršek, Ph.D.: Základy počítačové grafiky - IZG - Studijní opora, verze 0.9, Brno, FIT VUT v Brně

[7] WWW stránky - MAXON - The makers of CINEMA 4D and BodyPaint 3D http://www.maxon.net

[8] WWW stránka - Paul Bourke: Calculating Stereo Pairs, červenec 1999 http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/stereographics/stereorender/

[9] WWW stránka – Tammo Hinrichs: V2 synthesizer system, http://www.1337haxorz.de/products.html

[10] WWW stránky demoskupiny farbrausch, http://www.farbrausch.de/

[11] WWW stránka – How to save bitmap to file, 14.03.2007 http://sarathc.wordpress.com/2007/03/14/how-to-save-bitmap-to-file/

[12] WWW stránka - Optimize Options - Using the GNU Compiler Collection (GCC) http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

[13] WWW stránka - ASPACK SOFTWARE - Compression and Protection Tools http://www.aspack.com/

[14] WWW stránka – UPX: the Ultimate Packer for eXecutables http://upx.sourceforge.net/

[15] WWW stránka - Kkrunchy: pretty good executable compression http://www.farbrausch.de/~fg/kkrunchy/

[16] WWW stránka - Dwing's homepage - My Works - Compression http://wex.cn/dwing/mycomp.htm

Seznam ilustrací

Seznam tabulek

Seznam příloh