VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GRAFICKÉ INTRO 64KB S POUŽITÍM OPENGL

GRAPHICS INTRO 64KB USING OPENGL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ANTONÍN MINAŘÍK**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Minařík Antonín**
Obor: Informační technologie
Téma: **Grafické intro 64kB s použitím OpenGL**
Graphics Intro 64kB Using OpenGL
Kategorie: Počítačová grafika

Pokyny:

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 a krátké video pro prezentování projektu.

Literatura:

- dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3, experimenty směřující k vyřešení bodu 4.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Herout Adam, prof. Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Práce řeší tvorbu grafického intra s použitím OpenGL, jehož velikost nesmí přesáhnout 64 kB. Popsané a použité techniky zahrnují: Phongův osvětlovací model, mlhu, skybox, generování textur pomocí šumu a shadow mapping. Intro se odehrává ve velkoměstě, a jsou popsány techniky využité pro generování všech prvků města. Dále jsou popsány metody snížení velikosti aplikace a systém pro animaci kamery.

Abstract

This thesis deals with creating a graphics intro, which's size has to be below 64 kB. Described and used techniques includes: Phong reflection model, fog, skybox, generating textures from noise and shadow mapping. Intro is placed in a city and techniques of generating elements of the city are described. In addition, there are described methods for reducing file size and system for camera animation.

Klíčová slova

grafické intro, 64kB, OpenGL, město, generování, Catmull-Rom, skybox, shadow mapping, šum, mlha, kamera, procedurální generování, Phongův osvětlovací model

Keywords

graphics intro, 64kB, OpenGL, city, generating, Catmull-Rom, skybox, shadow mapping, noise, fog, procedural generation, Phong reflection model

Citace

MINAŘÍK, Antonín. *Grafické intro 64kB s použitím OpenGL*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Herout Adam.

Grafické intro 64kB s použitím OpenGL

Prohlášení

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

.....
Antonín Minařík
17. května 2017

Poděkování

Rád bych poděkoval vedoucímu prof. Ing. Adamu Heroutovi, Ph.D. za cenné rady a připomínky.

Obsah

1 Úvod **2**

2 Fenomén grafického intra s omezenou velikostí **3**

3 Grafické techniky a techniky generování obsahu **4**
3.1 OpenGL 4
3.2 Osvětlovací model 5
3.3 Mlha 6
3.4 Catmull-Rom spline 7
3.5 Procedurální generování 7
3.6 Šum 8
3.7 Skybox 9
3.8 Shadow Mapping 10
3.9 Hudba 10

4 Návrh generování města **13**
4.1 Budovy 13
4.2 Bloky budov 13
4.3 Město 15
4.4 Silnice 15
4.5 Textury 17
4.6 Skybox 17
4.7 Kamera 18

5 Implementace **20**
5.1 OpenGL 20
5.2 Načítání OpenGL funkcí 20
5.3 Použité knihovny 21
5.4 Hudba 22

6 Metody snížení velikosti aplikace **23**
6.1 Nastavení překladače 23
6.2 Programy pro zabalení spustitelného souboru 24

7 Vyhodnocení **25**

8 Závěr **28**

Literatura **29**

Kapitola 1

Úvod

Tato práce se zabývá tvorbou grafického intra s omezenou velikostí za pomoci OpenGL. Popisuje grafické techniky a techniky generování obsahu využité pro tvorbu intra a využitelné i pro jiné, zejména grafické, aplikace. Dále se text věnuje metodám pro snížení velikosti aplikace.

Grafické intro je spustitelný program představující schopnosti autora či autorů. Může se jednat o programování, design, skladatelství, umělecké cítění, protože každá z těchto činností je do nějaké míry pro tvorbu intra potřeba. Po spuštění výsledného programu je uživateli prezentován audiovizuální zážitek trvající obvykle několik minut, vykreslovaný v reálném čase. Některá intra si dávají za cíl dosáhnout velikosti spustitelného souboru menší než určitá hranice, a tím podtrhnout autorovy schopnosti. Intro řešené v této práci cílí na velikost nižší než 64 kB.

Intro se zaměřuje na generování sci–fi velkoměsta obsahujícího několik typů budov a mezi nimi silnice. Dále v něm bude přirozeně vypadající obloha a okolí ztrácející se v mlze.

Kapitola 2

Fenomén grafického intra s omezenou velikostí

Grafická intra se začala objevovat s příchodem domácích počítačů a softwarového pirátství. Crackeři prolamovali ochrany softwaru (především počítačových her) a nelegálně ho šířili. K takovému softwaru začali přidávat vlastní programy (intra), které je měli představit a proslavit mezi uživateli software. Postupně začali crackeři a skupiny crackerů soutěžit, kdo udělá technologicky vyspělejší a zajímavější intro, až to vedlo ke vzniku jednotlivců i skupin tvořících pouze intra.

Subkultura zabývající se tvorbou inter se začala nazývat demoscéna. Lidé z ní začali pořádat tzv. demoparty, na kterých se autoři inter a další zájemci scházejí a předávají si zkušenosti. Na těchto akcích se také konají soutěže a intra vytvořená speciálně pro ně jsou hodnocena v různých kategoriích. Od 90. let 20. století vznikly desítky těchto událostí po celém světě. Obvykle se pořádají každý rok, a autoři inter usilovně pracují, aby mohli předvést co dokáží.

Intra se dělí do kategorií v závislosti na velikosti spustitelného souboru a platformě. I dnes vznikají intra pro platformy jako Amiga, Commodore 64 nebo ZX Spectrum. Tyto počítače oproti dnešním PC disponují malou pamětí a výpočetními schopnostmi, což tvoří hlavní omezení při tvorbě těchto inter. Intra s omezenou velikostí jsou většinou určena pro nejmodernější hardware a největší výzvou při jejich tvorbě je generování obsahu. Nejběžnější omezení velikosti spustitelného souboru jsou 4 kB a 64 kB. Na obrázku 2.1 jsou ukázky z některých slavných inter, která obsahovala města, stejně jako to řešené v této práci.

Obrázek 2.1: Vlevo Debris od skupiny Farbrausch, uprostřed The Prophecy – Project Nemesis od Conspiracy a vpravo The Timeless od Mercury.

Kapitola 3

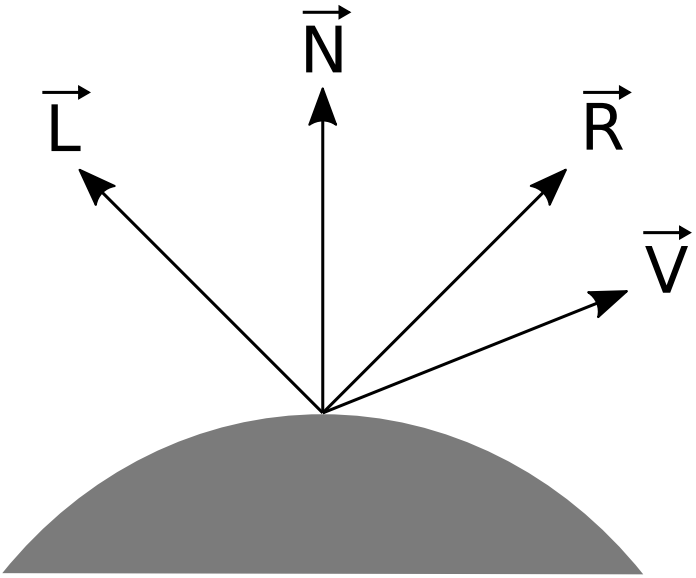
Grafické techniky a techniky generování obsahu

Práce s grafikou je v počítačových systémech velmi žádaná, ale také velmi náročná oblast. Obzvláště náročné je vykreslování scén připomínajících realitu. Vykreslování pomocí simulace fyzikálních zákonů je i na dnešních nejvýkonnějších počítačích velice časově náročné, a proto pro vykreslování v reálném čase nevhodné. Byly však vyvinuty různé techniky, jak napodobit určité aspekty reality při relativně nízkých výpočetních nárocích a při zachování přijatelného vzhledu. Tato kapitola se věnuje těmto technikám, ale také technikám generování obsahu, který je pomocí nich zobrazován, a jejich souvislostem.

3.1 OpenGL

OpenGL (Open Graphics Library) je API (Application Programming Interface) pro komunikaci s grafickým hardware [17]. Je multiplatformní a použitelné v mnoha programovacích jazycích. Sestává z několika stovek procedur a funkcí. Vývoj začala firma Silicon Graphics Inc. v roce 1991. Od roku 1992 byla specifikace spravována konsorciem OpenGL Architecture Review Board a v roce 2006 ji převzalo konsorcium The Khronos Group. Díky takovému způsobu správy a otevřenosti API není možné, aby kvůli krachu nebo rozhodnutí jedné firmy byl vývoj zastaven. Zároveň rozhodnutí o budoucích úpravách nemohou být ve prospěch pouze jedné z firem a tím se mezi nimi drží zdravá konkurence.

Ve starších verzích OpenGL byl proces vykreslování fixní, grafický hardware nebyl moc výkoný a v procesu kreslení byl hodně zapojen procesor. Od OpenGL verze 2.0 je možné některé fáze kreslení programovat a to přináší mnoho výhod a možností. Hlavními fázemi, bez programování kterých se žádný program využívající moderní OpenGL neobejde, jsou vertex a fragment shader. Program vertex shaderu je prováděn pro každý vrchol kresleného objektu a obvykle se v něm provádějí minimálně transformace mezi prostory. Fragment shader je prováděn pro každý fragment a často obsahuje texturování a osvětlovací model. Programování probíhá v jazyce GLSL (OpenGL Shading Language), jehož syntax je založena na jazyku C [1]. Obsahuje však spoustu vestavěných funkcí a několik typů, které plní speciální účel při kreslení a jeho použití usnadňují.



Obrázek 3.1: Vektory použité pro výpočet Phongova osvětlovacího modelu. $\vec{L}$ je vektor ke světlu, $\vec{N}$ značí normálový vektor povrchu, $\vec{R}$ je přímý směr odrazu světla a $\vec{V}$ vektor ke kameře.

3.2 Osvětlovací model

Osvětlovací model určuje barvu objektu vzhledem k jeho natočení a pozici světla ve scéně. V nějaké formě je využíván při téměř jakémkoliv 3D vykreslování, protože jinak nelze rozlišit hrany objektů. Osvětlovací modely se dělí na realistické a empirické. Realistické využívají fyzikální podstaty světla a dosahují velmi dobrých výsledků. Jejich použití je však vysoce hardwarově náročné. Empirické jsou zjednodušené a rychlé, a proto jsou vhodné pro vykreslování v reálném čase. Základním empirickým osvětlovacím modelem je Lambertův. Phongův model ho poté rozšiřuje o další vlastnosti, které ho přibližují k chování reálného světla [23].

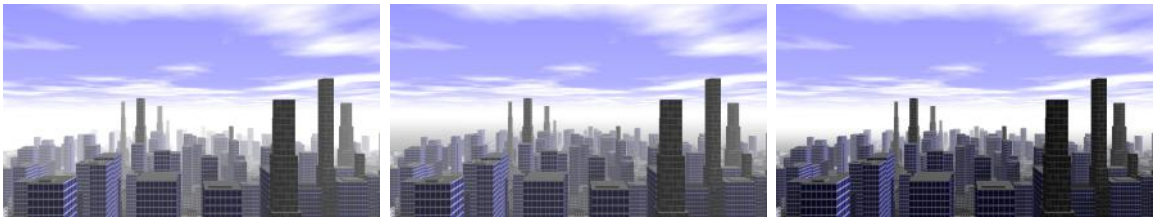
Lambertův model pro určení osvětlení bodu využívá vektor od světla k bodu a normálu povrchu. Pokud jsou tyto dva vektory rovnoběžné, osvětlení má maximální intenzitu. Čím více je objekt nakloněn, tím intenzita klesá a když jsou tyto vektory kolmé, je nulová. Výslednému světlu se říká difuzní, protože napodubuje chování reálného světla, které se odráží od mnoha nerovností povrchu a tím je po něm rozptýleno. Vztah pro výpočet Lambertova modelu je následující:

$$I_D = I_L \cdot r_D \cdot (\vec{N} \cdot \vec{L}) \tag{3.1}$$

kde I_D je intenzita odraženého světla, I_L je intenzita přicházejícího světla a r_D je koeficient difuze.

Phongův osvětlovací model přidává k difuzní složce světla ještě spekulární a ambientní. Spekulární složka se projevuje u lesklých povrchů a závisí i na úhlu pozorovatele. Ambientní složka malou měrou nasvětluje vše a tím napodobuje rozptýlené světlo, šířící se z okolních objektů. Výsledné světlo Phongova osvětlovacího modelu se vypočítá jako součet těchto částí:

$$I_P = I_A + I_D + I_S \tag{3.2}$$



Obrázek 3.2: Tři způsoby implementace mlhy. Vlevo lineární, uprostřed exponenciální, vpravo exponenciální s exponentem umocněným na druhou.

Spekulární složka je dána výpočtem:

$$I_S = I_L \cdot r_S(\vec{V} \cdot \vec{R})^{n_S} \tag{3.3}$$

kde I_L je intenzita přicházejícího světla, r_S je koeficient reflexe a n_S je koeficient ostrosti. Vektor odrazu světla se spočítá jako:

$$R = 2 \cdot (\vec{N} \cdot \vec{L}) \cdot \vec{N} - \vec{L} \tag{3.4}$$

Na obrázku 3.1 je vidět pozice vektorů použitých pro výpočet osvětlení. Ambientní složka je určena:

$$I_A = I_a \cdot r_D \tag{3.5}$$

kde I_a je množství ambientního světla.

3.3 Mlha

Použití efektu mlhy velikou mírou přispívá k realizmu výsledné scény. Přesto se jedná o hardwarově i programátorsky nenáročnou techniku.

Princip závisí na funkci určující zamlženost obrazového bodu. Parametrem funkce je vzdálenost od kamery a výsledkem hodnota poměru mlhy od 0 do 1. Výsledný poměr se použije pro interpolaci mezi původní barvou a barvou mlhy. Běžně se používají tři varianty funkce: lineární, exponenciální a exponenciální s exponentem umocněným na druhou.

Lineární varianta je nejjednodušší. Se vzdáleností lineárně stoupá poměr mlhy:

$$y = depth \cdot k \tag{3.6}$$

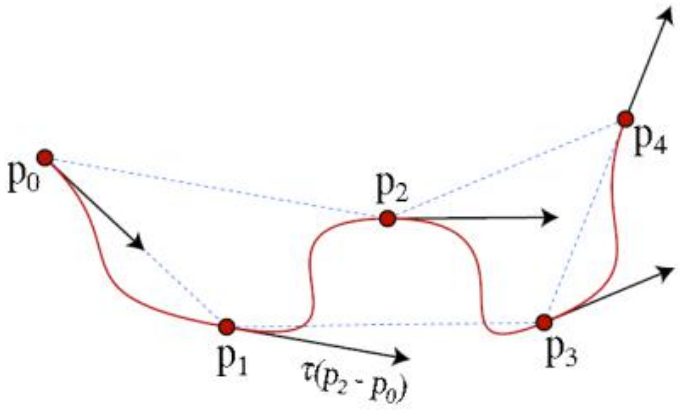
kde *depth* je hloubková informace bodu a k je koeficient určující hustotu mlhy. Exponenciální mlha začíná maximální viditelností v blízkosti kamery a klesá exponenciální křivkou:

$$y = 1 - e^{-depth \cdot k} \tag{3.7}$$

Exponenciální s exponentem umocněným na druhou je podobná exponenciální, ale má prudší nárůst zamlženosti:

$$y = 1 - e^{-(depth \cdot k)^2} \tag{3.8}$$

Na obrázku 3.2 je vidět porovnání těchto základních variant mlh.



Obrázek 3.3: Jak je vidět, tečna křivky v bodě je rovnoběžná s přímkou procházející předchozím a následujícím bodem. Na tomto obrázku jsou počáteční a koncový bod zdvojeny, proto přímka vychází již z nich (převzato z [22]).

3.4 Catmull-Rom spline

Většina objektů reálného světa se neobejde pouze se základními ideálními tvary jako je přímka či kružnice. Pokud bychom však měli popisovat konkrétní body těchto objektů, bylo by to velice paměťově náročné. Proto se využívají křivky, které je možné definovat pouze několika body, případně dalšími parametry. Tyto křivky jsou schopné dobře aproximovat reálné objekty či trasy pohybu objektů s možností neomezeně dopočítávat body na nich ležící. Křivky se dělí na interpolační a aproximační. Interpolační přímo prochází body, kterými jsou definovány, aproximační jimi neprochází.

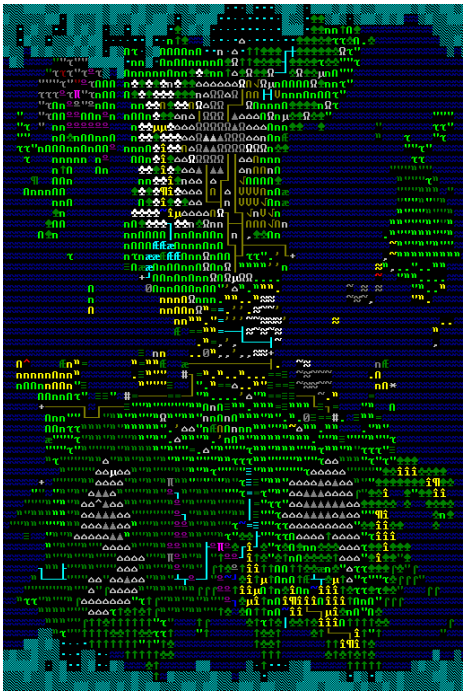
Catmull-Rom spline křivka je interpolační a je určená N body. Začíná ve druhém zadaném bodě a končí v předposledním. Pro pevné určení počátku a konce křivky je možné zdvojit počáteční a koncový bod. Výsledná křivka nemusí celá ležet v konvexní obálce kolem bodů, je proto třeba používat ji opatrně. Další vlastnosti křivky lze vidět na obrázku 3.3. Pomocí maticového zápisu je možné Catmull-Rom spline křikvu definovat následovně [3]:

$$Q(t) = \frac{1}{2}\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 3 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix} \tag{3.9}$$

3.5 Procedurální generování

Každá smysluplná aplikace pracuje s nějakou formou obsahu. Ať už se jedná o text, video či formátovaný dokument. Tento obsah může být vytvořen manuálně předem (nakreslení obrázku, napsání textu) nebo procedurálně za běhu aplikace.

Pro použití procedurálního generování je potřeba algoritmus, podle kterého je obsah generován. Tento algoritmus, ve formě programu, může, ale nemusí využívat prvku náhody. Často se také využívá pseudonáhody, která vytváří iluzi náhody, ale její výsledek



Obrázek 3.4: Pomocí procedurálního generování je ve hře Dwarf Fortress generován celý svět [16].

je deterministický. Tato vlastnost se výhodně využívá při tvorbě grafických inter, protože vygenerovaný obsah vypadá vždy stejně.

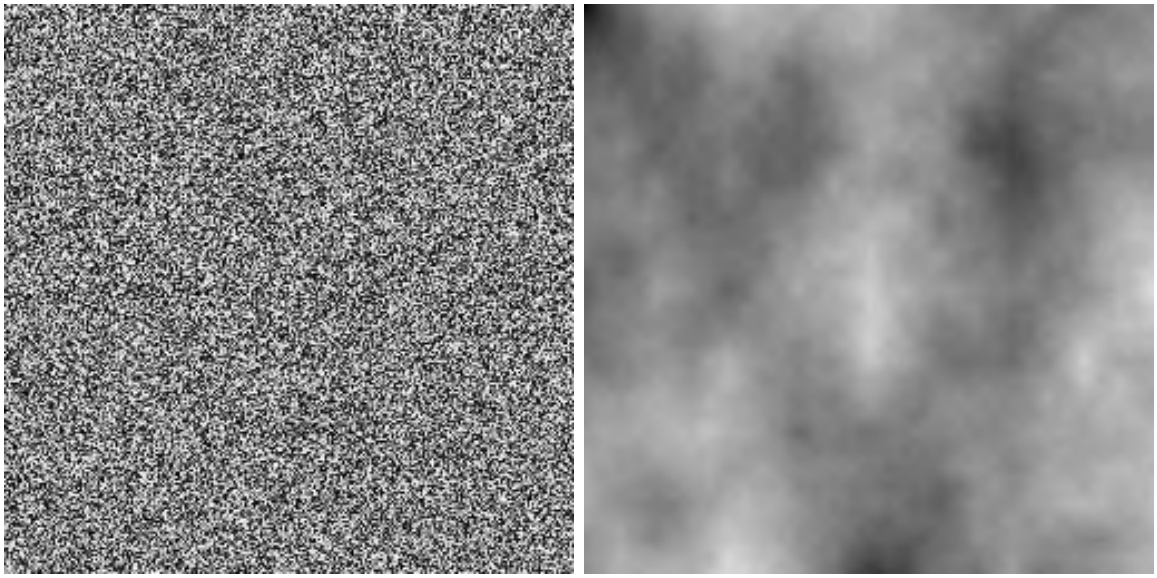
Hlavní výhoda procedurálního generování pro grafická intra spočívá ve velikosti algoritmu oproti velikosti dat, které je možné vygenerovat (a které by jinak museli být někde uloženy). Některé funkce mohou zabírat desítky bajtů a jimi generovaný obsah gigabajty dat. Další výhodou je usnadnění práce. Stačí vytvořit algoritmus a je možné generovat libovolné množství dat. Nevýhodná je naopak menší kontrola nad detaily a náročnost vytváření algoritmů. Při nevhodném použití mohou být patrná použitá pravidla a výsledky generování se mohou jevit mechanicky.

Hra Dwarf Fortress [2] ukazuje kam až se dá s procedurálním generováním zajít. Generuje pomocí něho celý svět. Pravidla zajišťují, aby umístění objektů dávalo smysl a výsledkem je svět obydlený všemožnými rasami a s bohatou historií. Na obrázku 3.4 je ukázka vygenerovaného světa z této hry.

3.6 Šum

Šum je možné generovat náhodnými (pseudonáhodnými) hodnotami v jedné či ve více dimenzích. V počítačové grafice ho lze využít jako fraktální šum jeho sčítáním o různých frekvencích. Tento šum má výhodné vlastnosti pro generování textur.

Tvorba fraktálního šumu začíná vygenerováním dvourozměrného pole náhodných hodnot (obyčejného šumu). Pokud se toto pole přiblíží, na výsledném šumu jsou vidět hrany původních pixelů. Proto se pro zjemnění využívá bilineární interpolace. Takto zjemněný šum stále nevypadá příliš přirozeně. Při opakovaném přibližování a sčítání těchto šumů



Obrázek 3.5: Vlevo šum a vpravo z něho vytvořený fraktální šum. Vzniká opakovaným přibližováním a sčítáním šumu původního.

však vznikne fraktální šum, kterému se také říká turbulence. Na obrázku 3.5 je vidět šum a z něho vygenerovaný fraktální šum. Z něho poté lze jednoduše vytvářet textury připomínající mraky [19].

Perlinův šum

Perlinův šum je technika pro procedurální generování textur. Vyvinul ho Ken Perlin a v roce 1985 prezentoval na konferenci SIGGRAPH [15]. Používá se při tvorbě textur a efektů jako je oheň, kouř nebo mraky. Nejčastěji je používán jako dvou nebo trojrozměrný, je ale možné použít i více rozměrů. Generuje se pomocí součtu šumových funkcí o různých frekvencích a amplitudách. Jeho výhoda oproti obyčejnému šumu náhodných hodnot je v rozložení frekvencí.

3.7 Skybox

Skybox slouží pro zobrazení vzdálených oblastí ve scéně. Obvykle ho tvoří krychle, na které jsou namapovány textury okolí. Tato krychle se pohybuje současně s kamerou, která je umístěna v jejím středu. Vzdálené objekty jsou umístěny v textuře skyboxu, a tím snižují výpočetní nároky na vykreslování. Jak už název napovídá, primárně se používá pro kreslení oblohy. Při použití vhodných textur, které na sebe dokonale navazují, vzniká věrná iluze okolí. Příklad textur je na obrázku 4.7.

Pro správné zobrazení skyboxu je třeba použít směrový vektor kamery k navzorkování jeho textur. Protože se bere pouze vektor, pozice kamery nemá vliv a okolí je statické (podobně jako se jeví vzdálené objekty v realitě). Aby byl skybox vykreslen vždy dál, než jsou ostatní objekty scény, kreslí se jako první s vypnutým hloubkovým testem a zbytek scény ho překryje.

3.8 Shadow Mapping

Použití stínů zlepšuje představu o hloubce a velikou měrou přispívá k realističnosti scény. Existuje několik technik pro tvorbu stínů od těch nejjednodušších, využívajících aproximací a nedokonalosti lidského vnímání, až po opravdu složité a hardwarově náročné.

Shadow mapping je hojně používaná hardwarově příliš nenáročná technika, která nevyžaduje od objektů žádné speciální vlastnosti. Jeho koncept představil v roce 1978 Lance Williams [21]. Dříve byla kvůli nedostatku výkonu technika využívána v předkreslených scénách, ale dnes již není problém použití v reálném čase.

Myšlenka shadow mappingu je vykreslit scénu z pohledu světla. Vzdálenost ke zdroji světla lze poté získat z hloubkového bufferu. Obsah tohoto bufferu se nazývá stínová mapa (shadow map). Poté je scéna kreslena normálním způsobem z pozice kamery. Pro každý pixel je možné spočítat pozici v prostoru. Pokud je vzdálenost mezi tímto objektem a světlem větší než vzdálenost uložená ve stínové mapě, objekt je ve stínu, jinak je osvětlen [12].

Jediné co je potřeba při kreslení z pohledu světla je hloubková informace. Proto je vhodné nepoužívat při něm osvětlovaní, texturování, a zápis barev. Díky tomu může dojít k několikanásobnému urychlení kreslení. Pokud se pozice světla či objektů ve scéně nemění, je možné opakovaně používat stejnou stínovou mapu, protože její použití není závislé na pozici kamery.

Při použití výše popsané metody shadow mappingu je problém s aliasingem. Kvůli nízké přesnosti hloubkového bufferu jsou osvětlené oblasti, neležící přímo kolmo ke světlu, postihnuté nepěknými artefakty. Pro jejich řešení se používá určitá hodnota tolerance (anglicky bias), ve které pokud vzdálenost od světla leží, tak je plocha stále ještě osvětlena. Na obrázku 3.6 jsou vidět neblahé účinky jejího nepoužití. Když použijeme pevně danou toleranci je problém s plochami svírajícími ostrý úhel ke světlu. Zvýšení tolerance pomůže zabránit aliasingu na těchto plochách, ale nastává problém zvaný Peter Panning, kvůli kterému jsou všechny stíny posunuty a nemusí vůbec ležet pod objekty, které je vrhají. Proto je vhodné použít adaptivní toleranci v závislosti na úhlu vektoru světla a normály povrchu [20]:

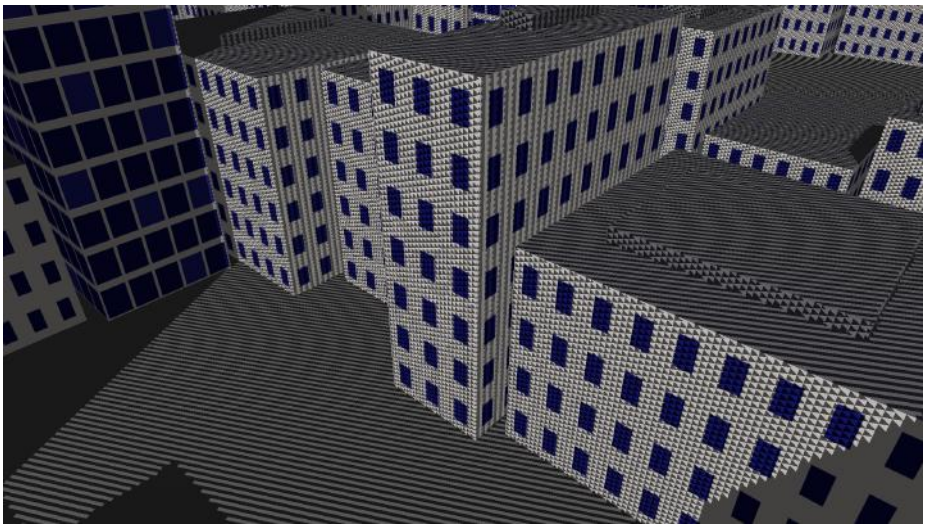
$$bias = max(k_1 \cdot (1 - \vec{N} \cdot \vec{L}), k_2) \tag{3.10}$$

kde $\vec{N}$ je normála povrchu, $\vec{L}$ je normalizovaný vektor ke světlu, k_1 maximální tolerance a k_2 je minimální tolerance.

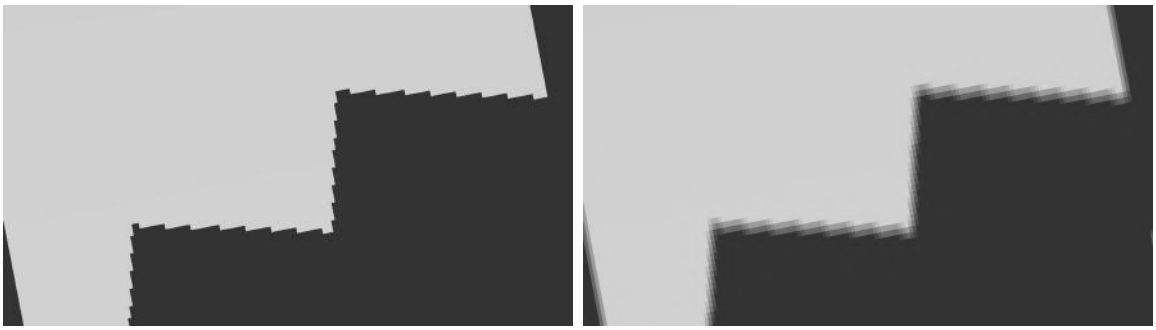
Další problém s aliasingem plyne z rozlišení stínové mapy. Pro jeho zmírnění se používá několik metod. Díky zmenšení zorného pole při zápisu do stínové mapy připadá její větší část na objekty se stíny, a tím je efektivní rozlišení větší. Další zlepšení přinese zvýšení rozlišení stínové mapy, jeho nevýhodou jsou však vysoké nároky na paměť. Pro zjemnění hran stínů je dále možné použít PCF (Percentage Closer Filtering). Tato technika pomocí bilineární interpolace zohledňuje i okolní body ve stínové mapě. Na obrázku 3.7 je vidět její účinek.

3.9 Hudba

Hudba je velice důležitou částí intra. Udává jeho tempo a při vhodné provázanosti s dějem umocňuje celý zážitek. Pro intro s omezenou velikostí však nelze použít většina z běžných formátů, protože nejsou dostatečně úsporné. Jeden z možných použitelných formátů by byl MIDI. V něm používané nástroje ale nemusí znít stejně na všech zařízeních, proto se častěji používají syntetizéry. Ty bývají dostatečně malé a dokáží generovat přenositelné a některé i velice kvalitní nástroje a zvuky.



Obrázek 3.6: Artefakty vznikající při nepoužití tolerance.



Obrázek 3.7: Zjemnění hran stínů. Vlevo obyčejný stín, vpravo s použitím techniky PCF.

Softwarový syntetizér je počítačový program, který generuje zvuky a hudbu. Nejprve vytvoří nějaký základní signál jako je sinusoida, trojúhelníkový nebo pilový průběh. Tento signál je poté filtrován, zesilován, ořezáván a modulován, až vznikne zvuk použitelný jako nástroj a hodící se do hudby. Nástrojů může být více a používají se i další efekty měnící jejich vlastnosti.

Tvoření hudby pro syntetizéry není jednoduché. Pro konkrétní zvuk nástroje je třeba definovat nastavení všech součástí syntetizéru. Tímto způsobem je možné generovat unikátní a skvěle znějící hudbu. Nalezení dobře znějících nástrojů je však obtížné a vyžaduje hodně trpělivosti a zkušeností. Poté již lze tvořit skladby nastavením délky, výšky a času startu jednotlivých not. Takto vytvořené skladby jsou obvykle velmi malé a netrpí žádnými kompresními vadami, protože výsledný zvuk je generován až při přehrávání.

Pro plynulý průběh hudby je třeba zajistit vždy dostatek dat v bufferu, ze kterého se přehrává. Plnění bufferu lze zajistit dvěma způsoby. Vytvořit samostatné vlákno, ve kterém běží syntetizér a kontinuálně ho doplňuje, nebo tuto činnost zahrnout do hlavní smyčky. Při použití vlákna je třeba zvýšit jeho prioritu, jinak by mohlo docházet k nedostatku jeho běhu kvůli vykreslování. S vyšší prioritou jemu přidělenou však může docházet k jeho spuštění v nevhodnou dobu, kdy je potřeba vykreslit snímek. Plnění v hlavní smyčce může být problémové, pokud je buffer vyčerpán, než se k němu dostane běh programu [9].

Jako příklad syntetizéru s přehrávačem lze uvést knihovnu `libv2` [8]. Její přehrávač používá kombinaci způsobů zmíněných v předešlém odstavci. Běží v samostatném vláknu a provádí smyčku činností: zjistí aktuální přehrávanou pozici bufferu, vyplní ho od poslední známé pozice do té aktuální následujícími daty hudby, chvíli čeká. Problém přerušení kreslení je řešen tak, že běh vlákna je spuštěn synchronizační událostí až po vykreslení snímku. Pokud by mělo však dojít k vypršení bufferu s hudbou, je vlákno přehrávače spuštěno vypršením časovače [9].

Kapitola 4

Návrh generování města

Generování města lze využít mnoha způsoby. Generované město lze uplatnit v různých počítačových hrách, ale také při simulaci např.: pro trénování záchranných složek a simulaci přírodních i jiných katastrof.

Podobným problémem se zabývá práce [6]. Ta ovšem necílí na malou velikost a rozsah a složitost v ní generovaného města je nesrovnatelně větší než v této.

Město se skládá z bloků a každý blok je složen z budov. Mezi bloky vedou silnice a hlavní silnice se rozbíhá od středu města do čtyř směrů. Tato kapitola popisuje tvorbu prvků města, a jak jsou propojeny. Na obrázku 4.1 jsou tyto prvky vidět pohromadě.

4.1 Budovy

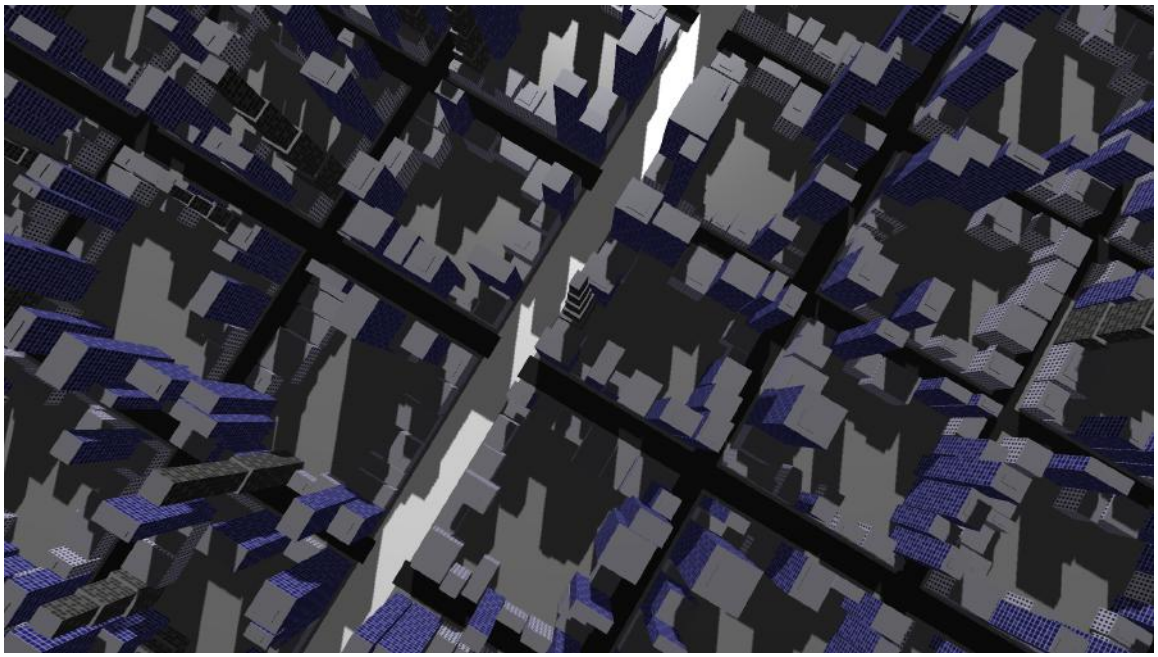
V intru se nachází tři typy budov: domy, mrakodrapy a stupňující se mrakodrapy. Základním tělesem ze kterého se skládají je kvádr. Funkce generující budovu přijímá parametry jako její rozměry, typ a pozici, případně počet stupňů. Tyto parametry jsou generovány jinou funkcí, která přijímá rozsah hodnot a využívá generátor pseudonáhodných čísel. Výsledné budovy vypadají díky tomu náhodně a pomocí nastavení rozsahu hodnot je lze generovat na míru. Aby střechy budov nebyly tak jednotvárné, mohou na nich mít domy a obyčejné mrakodrapy stavbu. Stavba je reprezentovaná malým kvádrem, a její umístění je také generované pseudonáhodně.

Domy a mrakodrapy se liší pouze jejich možnými rozměry a texturou. Stupňující se mrakodrapy mají 2 až 5 úrovní, kde každý další stupeň je zúžený oproti stupni předchozímu. Tento typ budovy má také vlastní texturu. Umisťování budov určuje blok, do kterého patří. Řídí jejich rozměry a je v něm obsažena pravděpodobnost, s jakou se budou vyskytovat. Na obrázku 4.2 jsou vidět ukázky vygenerovaných budov.

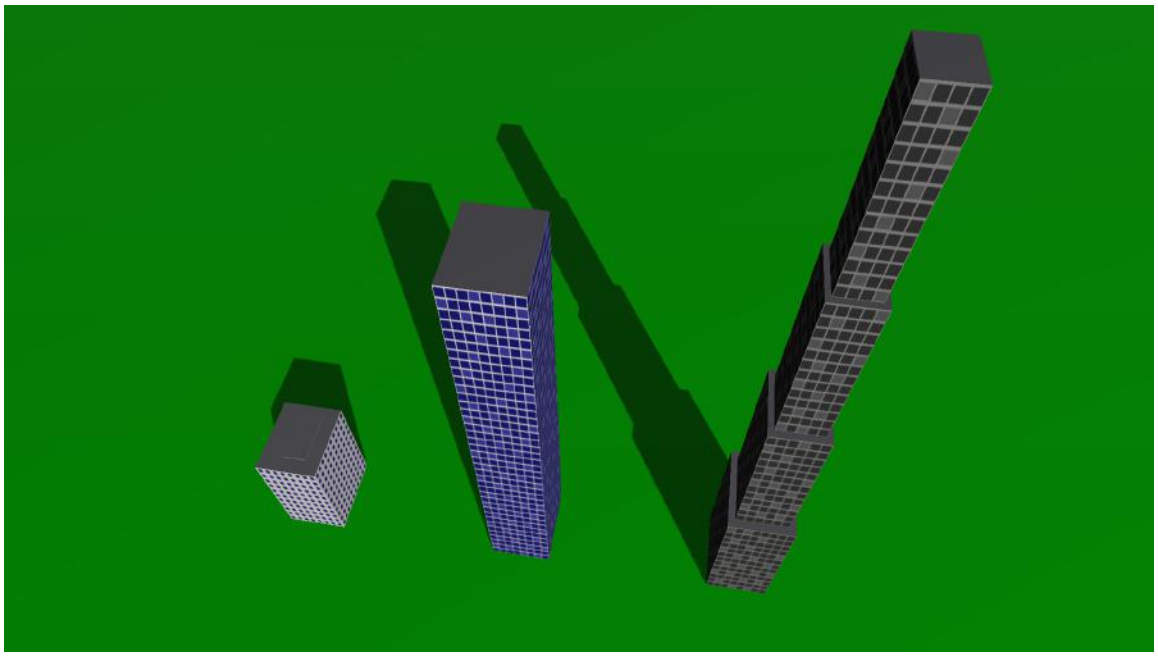
4.2 Bloky budov

Blok je umístěn na ploše představující zem, má tvar obdélníku (ve speciálním případě čtverce) a po jeho obvodu jsou umístěny budovy. Základní parametry bloku jsou pozice a rozměry. Dalším jeho parametrem je výška, na které je založena maximální výška budov náležících bloku. Poslední vlastností bloku je pravděpodobnost výskytu mrakodrapu.

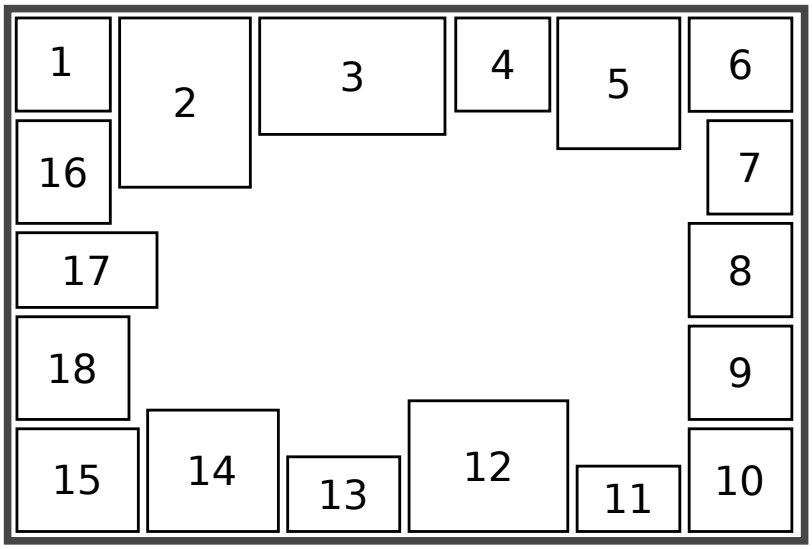
Budovy jsou postupně generovány s pseudonáhodnými rozměry (v jistých mezích) a ty krajní mají rozměry takové, aby vyplnily zbývající místo v bloku. Tuto činnost zaobaluje funkce `genBlock()`, která využívá všech výše popsaných parametrů a s její pomocí se poté



Obrázek 4.1: Shora lze rozeznat způsob jakým bylo město generováno.



Obrázek 4.2: Typy budov nacházející se v intru. Vlevo dům, uprostřed mrakodrap, vpravo stupňující se mrakodrap.



Obrázek 4.3: Pořadí generování budov v bloku. U budov 7, 16 a 18 je třeba zajistit, aby jejich šířka nezasáhla do těch ležících za nimi. Toho je dosaženo uložením pozic všech riskantních budov. Při následném generování těch, které by do nich mohly zasahovat, je jejich šířka omezena.

po blocích generuje celé město. Na obrázku 4.3 je vidět pořadí v jakém se budovy v bloku generují.

4.3 Město

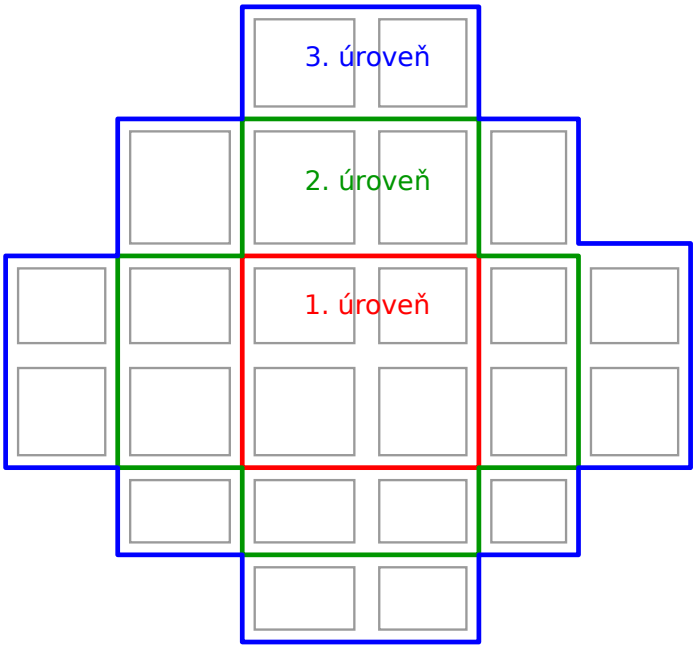
Město tvoří mřížka bloků budov mezi kterými jsou silnice. Jako první jsou vygenerovány šířky řad bloků v obou směrech. Generování poté probíhá po úrovních od středu s tím, že čtveřice bloků uprostřed tvoří první úroveň. Každá další řada ohraničujících bloků je další úroveň.

V závislosti na úrovni je každému bloku přidělena maximální výška a pravděpodobnost výskytu mrakodrapu. Díky těmto parametrům je vyšší pravděpodobnost, že ve středu města budou vyšší budovy a zároveň větší počet mrakodrapů.

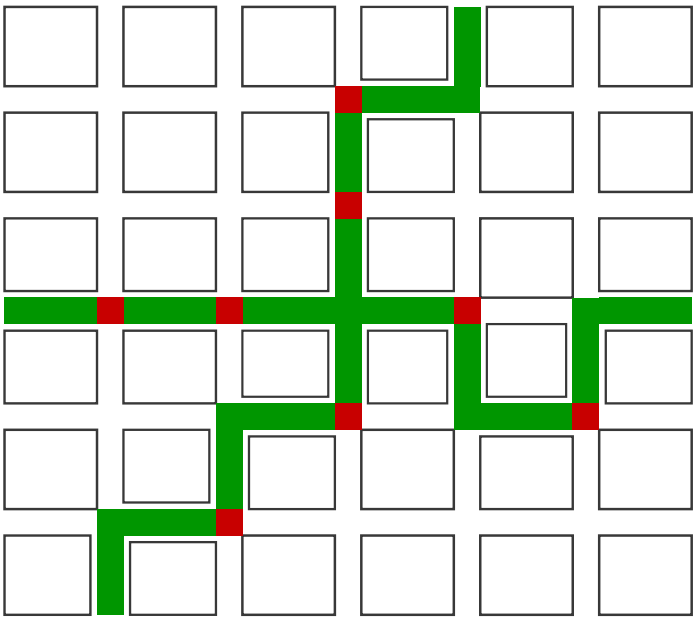
Se stoupajícím počtem bloků a budov, které se vykreslují, vzrůstají také výpočetní nároky. Pro vytvoření velkého města s nižšími nároky lze bloky od zadané úrovně vykreslovat do textury. Tato textura je poté umístěna pod město a jeho okolí a dotváří vzdálené budovy, na kterých tolik v intru nezáleží.

4.4 Silnice

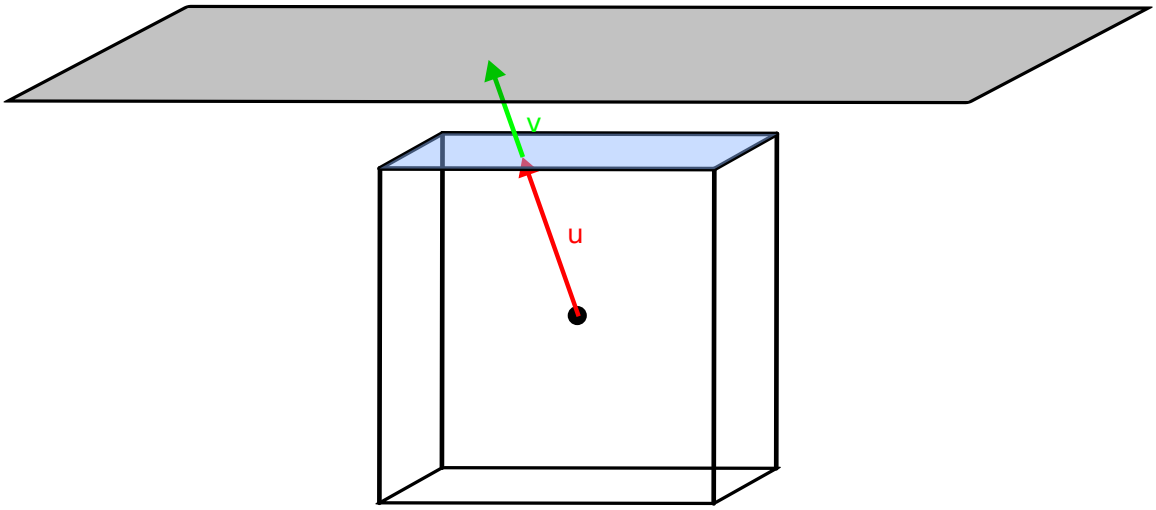
Silnice jsou umístěny mezi bloky budov. Hlavní silnice se ze středu města rozbíhá do čtyř směrů. Od normálních silnic jsou odlišeny bílou barvou a jsou širší stejně jako celá ulice, kterou vedou. Za každým blokem, který hlavní silnice od středu města překoná, je šance, že odbočí vpravo či vlevo. Pravděpodobnost že odbočí je nižší než pravděpodobnost pokračování rovně. To zlepšuje stabilitu její pozice a přidává na uvěřitelnosti. Po opuštění města



Obrázek 4.4: Bloky s budovami jsou generovány od středu po úrovních. Podle velikosti města jsou určeny parametry pravděpodobnosti výskytu mrakodrapu a maximální výšky budov. S rostoucí úrovní hodnoty těchto parametrů klesají. Je možné zvolit úroveň, od které budovy v blocích již nebudou trojrozměrné, ale budou kresleny do textury.



Obrázek 4.5: Znázornění možné trasy hlavní silnice. Zelená reprezentuje hlavní silnici, červená její místa na kterých probíhá rozhodnutí o dalším směru.



Obrázek 4.6: Ze středu skyboxu je veden vektor $\vec{u}$, k němu je přičten vektor $\vec{v}$, a tím je dosažena výška textury s oblohou. Souřadnice šířky a hloubky nyní obsahují pozici v textuře. Hodnota textury v tomto bodě je přepočítána a barva je uložena do textury skyboxu. Obdobný způsob je využit pro výpočet všech bodů textury a ostatních textur skyboxu.

již nezahýbá a ztrácí se v dáli. Na obrázku 4.5 je vidět možný způsob jejího vygenerování se znázorněním míst, na kterých se rozhoduje o odbočkách.

4.5 Textury

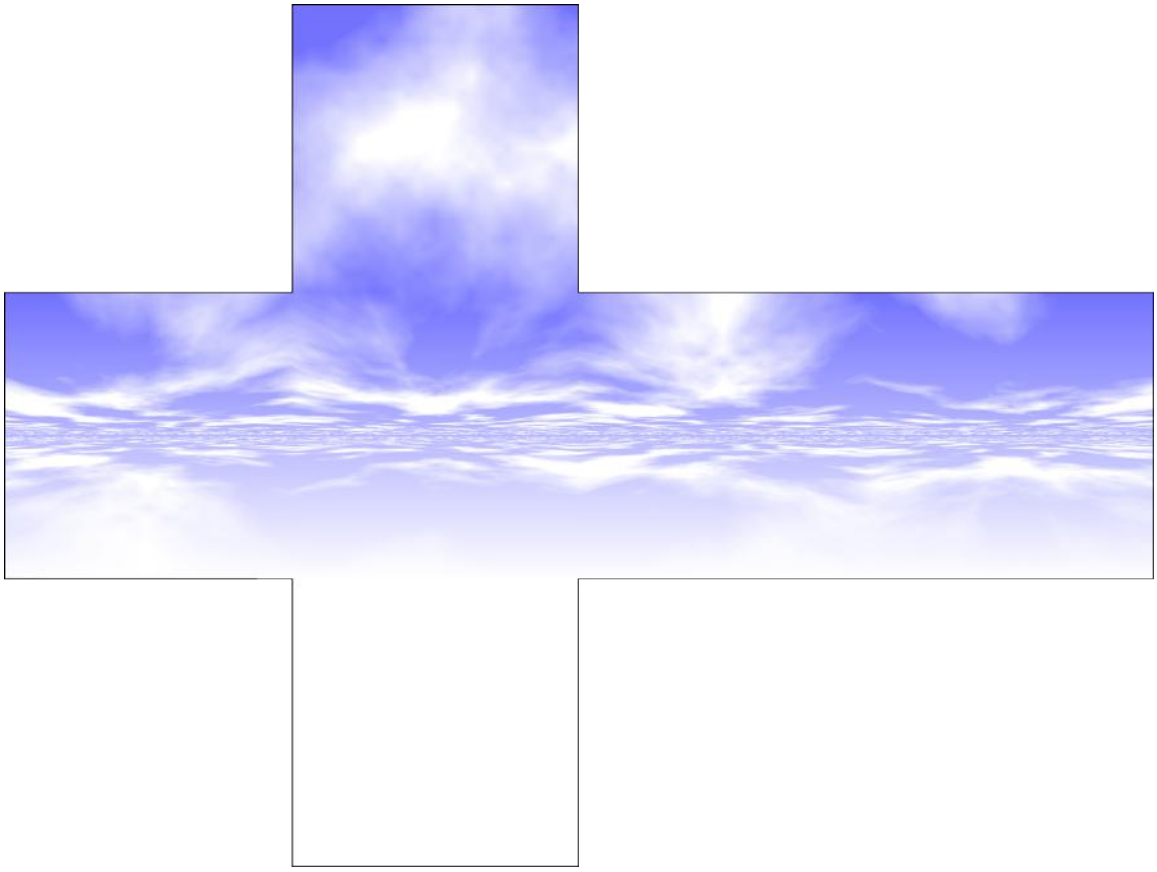
Textury slouží pro odlišení a dokreslení povrchu objektů. Mohou obsahovat informace o objektu, které se používají pro různé efekty, jako např.: spekulární mapy, mapy normál, bump mapy. V této práci jsou využity pouze základní textury určující barvu objektu na daném místě.

Textury budov jsou tvořeny okny a zdmi. Generování probíhá jednoduše v cyklu for, kde je procházen každý bod textury a podle jeho pozice je určena barva. Výsledná textura obsahuje jedno okno a okolo něho zeď. Poté je pomocí opakování kreslena po celé budově.

Aby okna na budovu horizontálně i vertikálně pasovala, je pro určení texturovací souřadnice šířka resp. výška budovy podělena šířkou resp. výškou okna a zaokrouhlena. Pro zvýšení variability textur budov jsou pomocí shaderu některá okna zesvětlena. Je k tomu využit generátor pseudonáhodných čísel, který jako vstup přijímá přibližnou pozici okna a ID vrcholu kreslené plochy v rámci bloku budov. Tím se jeví světlá okna dostatečně náhodně a zároveň je možné toto řešení použít v shaderu, který pracuje paralelně a pro každý bod nezávisle.

4.6 Skybox

Obloha intra je umístěna ve skyboxu. Jako první se vygeneruje dvourozměrná plocha nebe s mraky pomocí fraktálního šumu. Tato plocha má tvar čtverce a rozměr osminásobně větší než strana skyboxu, aby byly mraky vidět i v dálce. Následně je namapována podle 4.6 na stěny skyboxu, a tím vzniká iluze rozprostírajícího se nebe. Při mapování fraktálního šumu na textury skyboxu je hodnota nižší než prahová považována za nebe a mraky se objevují



Obrázek 4.7: Textury použitého skyboxu. Ve tvaru krychle obalují scénu. Spodní strana je vždy zakryta zemí, proto pro ní není potřeba vytvářet texturu.

až při jejím překonání. To dělá výslednou oblohu reálnější. Do oblohy za mraky je zároveň s klesající výškou přimíchávána bílá barva pro plynulejší přechod do mlhy na horizontu. Pro zaručení dokonalého přechodu nebe a země je v shaderu do textury přimíchávána další bílá barva. To zároveň skryje aliasing vznikající v dálce. Na obrázku 4.7 je vidět použitý skybox v rozbalené podobě.

4.7 Kamera

Kamera je pro intro velice důležitá, určuje co přesně divák vidí a její pohyb dotváří atmosféru výsledného díla. Pohyb kamery může být interaktivně ovládán uživatelem nebo pevně uložen v programu. V intru jsou implementovány oba způsoby. Interaktivní slouží pouze pro usnadnění plánování pohybu kamery a pevně daný v programu představuje pohyb v samotném intru.

Nejběžnější forma pohybu v intru jsou různé průlety a záběry jeho prostředí. Pro ně je potřeba znát pozici a zároveň směr pohledu kamery pro každý použitý bod. Pozice je uložena jako trojrozměrná souřadnice v prostoru intra a směr jako normalizovaný vektor. Dále je uložena doba trvání přechodu mezi body pro možnost změny rychlosti. Tyto body na sebe navazují a vytváří spolu záběr. Záběry tvoří výsledný pohyb celým intrem.

Mezi body záběru je proložena Catmull-Rom křivka, která vyžaduje 4 body. Počáteční bod je zdvojený, aby na něm kamera skutečně začínala svůj běh. Další body jsou postupně brány ze záběru a jeho poslední bod je opět zdvojen pro dojezd kamery. O určení výsledné pozice a natočení kamery se stará funkce, která přijímá pouze seznam záběrů a používá globální časovou proměnnou. Časování, přepínání a zdvojování bodů je řešeno uvnitř ní, takže nedochází k redundanci žádných bodů.

Kapitola 5

Implementace

Výsledné grafické intro využívá téměř všechny techniky popsané dříve v kapitole 3 a generování města probíhá podle návrhu v kapitole 4.

Program byl implementován za pomoci vývojového nástroje Microsoft Visual Studio v programovacím jazyku C++, a z větší části je použito programovacího jazyka C (jehož je C++ téměř nadmnožinou), protože si myslím, že pro takovýto typ projektu je tento jazyk vhodnější. Použití objektů a dalších možností jazyka C++ je totiž vhodné spíše pro obsáhlejší projekty a přináší s sebou zbytečnou režii. Proto jsou prvky tohoto jazyka použity hlavně při práci s knihovnou GLM, která ho vyžaduje.

5.1 OpenGL

OpenGL je využito ve verzi 4.4, novější verzi nezvládá integrovaná grafická karta, takže by nebylo možné otestovat na ní program. Jsou použity buffery pro uložení vrcholů objektů a textur. Na textury je aplikován mip mapping. Pro implementaci shadow mappingu je využit zápis hloubkové mapy do textury.

Program obsahuje 3 kombinace vertex–fragment shader. Všechny shadery jsou uloženy jako řetězcové literály, protože u intra není povoleno načítat data ze souborů nevyskytujících se v běžném systému a načítání by navíc přidalo zbytečnou režii. Jedna z nich se stará o kreslení skyboxu, kreslí jeho textury a přimíchává bílou barvu v závislosti na výšce pro plynulejší přechod mezi nebem a zemí. Další obstarává ukládání stínové mapy. Ve vertex shaderu probíhají transformace a fragment shader je prázdný, protože informace o hloubce objektů je ukládána automaticky. Nejsložitější jsou shadery pro kreslení scény. Vertex shader provádí transformace a předává informace určené k interpolaci do fragment shaderu. Ten se stará o výpočet stínů a PCF, pseudonáhodné zesvětlování oken budov, osvětlovací model a mlhu.

5.2 Načítání OpenGL funkcí

Ze souboru `opengl32.dll` v operačním systému Microsoft Windows jsou dostupné OpenGL funkce pouze do verze 1.1. Pro použití funkcí z vyšších verzí OpenGL je potřeba načíst je za běhu.

Nejjednodušší způsob jak toho docílit je pomocí knihovny GLEW [10]. Ta se postará o načtení a ověření všech dostupných OpenGL funkcí. Díky jednoduchosti jejího použití je vhodná pro většinu běžných OpenGL aplikací a v této práci byla zprvu také použita. Její

nevýhody jsou delší doba kompilace a její velikost, která je při tvorbě intra kritická. V intru navíc nejsou využity všechny OpenGL funkce, a tak bylo od jejího použití upuštěno.

Pro manuální načítání funkcí je potřeba dříve zmíněná knihovna `opengl32.dll`. Z ní je načtena speciální funkce `wglGetProcAddress()`, která slouží pro načítání dalších funkcí OpenGL. Každá z nich je poté načtena pomocí jejího jména a je třeba znát její parametry. Konstanty OpenGL, bez kterých se jeho používání neobejde, lze nalézt na oficiálních stránkách API [11].

5.3 Použité knihovny

Při vývoji jakéhokoliv software je dobrá praktika neimplementovat vše od začátku a raději využít již vytvořené a dobře odzkoušené knihovny. Studium použití (hlavně rozsáhlých) knihoven může být sice časově náročné, ale časová náročnost vlastní implementace a ladění je obvykle větší, a navíc použití známých knihoven vede k lepšímu porozumění při práci v týmu. Tato kapitola popisuje knihovny použité v této práci.

Windows API

Windows API [13] je v práci použito pro tvorbu okna a jeho propojení s OpenGL. Nejprve jsou pomocí funkce `MonitorFromPoint()` zjištěny informace o monitoru, mezi kterými je také jeho rozlišení. Toto rozlišení je poté předáno funkci `CreateWindow()`, která vytvoří okno. Parametr pro styl okna je nastaven na `WS_POPUP`, což ho zbavuje dekorací a díky tomu obraz intra zaplní celou obrazovku.

Dále jsou z API využita vyskakovací okna pro hlášení chyb, které by divák v ideálním případě neměl vidět. Windows API zprávy jsou použity pro ukončení klávesou `Esc` a další klávesy jsou využity pro ladící účely při nastavování kamery. Poslední využití funkcí API je pro měření času uplynulého mezi snímky. Doba mezi snímky je přepočítána na sekundy a uložena do globální časové proměnné.

GLM

GLM neboli OpenGL Mathematics je matematická knihovna pro C++ založená na specifikaci GLSL (OpenGL Shading Language) [4]. Zahrnuje třídy a funkce, které se drží stejných názvových konvencí a funkcionality jako tento jazyk. Zároveň obsahuje velké množství dalších užitečných funkcí jako například transformace matic a práci s kvaterniony.

V intru je tato knihovna použita pro výpočet: matice kamery, projekční matice s perspektivním zkreslením pro scénu a projekční matice světla. Dále také výpočet bodu na Catmull Rom křivce a interpolace mezi těmito body, po kterých se pak pohybuje kamera.

libv2

Knihovna `libv2` obsahuje syntetizér a slouží pro přehrávání `v2m` souborů [8]. Formát souborů `v2m`, stejně jako knihovna, byl vyvinut skupinou Farbrausch pro potřeby inter s omezenou velikostí. Jeho myšlenka je založena na formátu MIDI. Zatímco v MIDI jsou příkazy pro jednotlivé kanály uloženy chronologicky, `v2m` má příkazy pro každý kanál uloženy zvlášť. To umožňuje kvalitnější komprimaci, jelikož opakující se hudební vzory jednotlivých kanálů leží v posloupnosti.

Knihovna přistupuje k hudbě z paměti, do které je v době překladu načten soubor `v2m`. K načtení souboru je využito direktivy `incbin` jazyka symbolických adres. Ta načte

bajty souboru přímo do paměti. Pro překlad tohoto zdrojového souboru je použit překladač `nasm` [18]. Výsledný `obj` soubor je slinkován se zbytkem programu, a tím je zaručen přístup k datům hudby. Poté knihovna nabízí funkce pro přehrávání, pozastavení a ukončení přehrávání. Ke zvukové kartě přistupuje pomocí DirectSound, jež je popsáno níže.

DirectSound

DirectSound je zastaralá, ale stále podporovaná součást DirectX, což je sbírka API od firmy Microsoft, zaměřující se především na multimediální využití. Slouží pro přístup ke zvukové kartě počítače a navíc nabízí možnosti nahrávání a míchání zvuku. V projektu je využito pouze nepřímo přes knihovnu `libv2`, je však nutné zahrnout linkování souboru `dsound.lib`.

5.4 Hudba

Pro hudbu je využit V2 synthesizer system od skupiny Farbrausch. Tento syntetizér lze snadno přidat do intra jako statickou knihovnu, která je popsána v předchozí podkapitole. Je použita skladba The Secrets Of Cubic Tower od tvůrce se jménem Leopart, která byla také využita ve stejnojmenném intru od skupiny deMarche. Intro i skladba jsou dostupné ke stažení na [5].

Kapitola 6

Metody snížení velikosti aplikace

Malá velikost aplikace může být výhodná z mnoha různých důvodů. Jedním z důvodů je menší potřebné místo na pevném disku, či jiném médiu a také rychlejší přenosy mezi nimi. Další výhodou je menší místo, které zabírá program v paměti, a tím nechává více volné paměti pro jiná použití. Menší program je obvykle rychleji překládán, a tak urychluje vývoj aplikace.

Následující metody nemusí způsobovat výhody ve všech výše zmíněných směrech. Některé zlepšují pouze jeden aspekt, některé jejich kombinaci.

6.1 Nastavení překladače

Pro překlad programu je třeba provést spoustu různých operací a způsob jejich provedení může velkým způsobem ovlivnit výsledný soubor. V této práci je využito Microsoft Visual Studio 2015, proto následující popis nastavení platí hlavně pro tuto verzi nástroje. Je však pravděpodobné, že podobné nastavení jsou dostupné i v jeho jiných verzích.

Nejprve je potřeba zvolit architekturu, pro kterou se bude překládat. V dnešní době již architekturu `x86-64` využívá velké množství programů. Oproti `x86` nabízí nepoměrně větší množství virtuální paměti a také využití 64bitových registrů a další vylepšení. Pro intro s omezenou velikostí je však výhodnější program zaměřený na architekturu `x86`, protože dosahuje mnohem menších velikostí souboru.

Základní nastavení optimalizace určuje, na co se optimalizátor zaměřuje: maximální rychlost programu, minimální velikost souboru a nebo neoptimalizovat (vhodné pro ladění). Pro intro mohou být vhodné oba způsoby optimalizace. V této práci byla zvolena optimalizace pro maximální rychlost programu, protože o snížení velikosti se starají hlavně programy pro zabalení spustitelného souboru.

Dále byla zakázána volba bezpečnostních kontrol, která hlídá přetečení zásobníku. U intra jde především o rychlost a velikost, neobsahuje žádná citlivá data. Model čísel s plovoucí řádovou čárkou byl nastaven na rychlé (fast), protože přesnost není důležitá a ve výsledku toto nastavení není patrné. Také byly pro urychlení vypnuty kontroly za běhu.

Linkovací fáze překladu zajišťuje spojení objektových souborů a statických knihoven ve spustitelný soubor. Aby byl výsledný soubor co nejmenší, je vhodné přidat pouze knihovny a soubory, které jsou skutečně potřeba. Je zakázáno generování souboru manifestu, který obsahuje informace o linkování a bývá přibalen ke spustitelnému souboru. Dále je z bezpečnostních důvodů obvykle nastavena náhodná bázová adresa. S touto možností aktivní není možné použít program `kkrunchy`, a proto byla také vypnuta.

6.2 Programy pro zabalení spustitelného souboru

Použití programů pro zabalení spustitelného souboru (anglicky exe packery) je pro intra s omezenou velikostí velice výhodné. Dokaží výrazně snížit velikost souboru programu. Jejich nevýhodou je mírný nárůst trvání spouštění programu, protože ten je potřeba nejprve rozbalit do paměti. V paměti se poté nachází rozbalený program a pokud packer nepodporuje rozbalování na místě, tak i jeho program pro dekomprimaci.

UPX

The Ultimate Packer for eXecutables neboli UPX je volně dostupný přenosný packer [14]. Spadá pod licenci GPL s vyjímkou, že jím zkomprimované programy lze komerčně využít. Dosahuje slušného komprimačního poměru a dobré rychlosti.

kkrunchy

Kkrunchy je malý exe packer vyvinutý skupinou Farbrausch primárně pro 64 kB intra [7]. Nabízí jeden z nejlepších komprimačních poměrů a je výhodné ho používat s knihovnou `libv2` od stejných autorů, protože s ní dosahuje výrazně lepších výsledků než ostatní programy. Protože tento program přesouvá bázovou adresu programu, je potřeba nastavením překladače zakázat její náhodné určení.

Kapitola 7

Vyhodnocení

Intro bylo vyvinuto a ozkoušeno na na notebooku s grafickou kartou GeForce GTX 760M a procesorem Intel® Core™ i7-4700HQ. Lze ho spustit jak na dedikované tak i integrované grafické kartě, avšak při generování velkého města a použití vysokého rozlišení stínové mapy běží na integrované kartě s nízkým počtem snímků za sekundu.

Obsazení RAM po spuštění několik sekund kolísá. Během této doby probíhá generování obsahu a jeho odesílání do paměti grafické karty. Po zhruba čtyřech sekundách se velikost obsazené paměti ustálí na necelých 180 MB a tak zůstává po zbytek běhu aplikace.

V tabulce 7.1 je vidět, že `kkrunchy` snižuje velikost nejlépe. Hlavním důvodem tohoto výsledku je přítomnost hudby za použití knihovny `libv2`. Formát souborů, které používá, je vyvinut pro co nejlepší komprimační výsledky programu `kkrunchy`.

	velikost [kB]
původní	133
UPX	77
kkrunchy	34

Tabulka 7.1: Velikost aplikace a její velikost po zabalení.

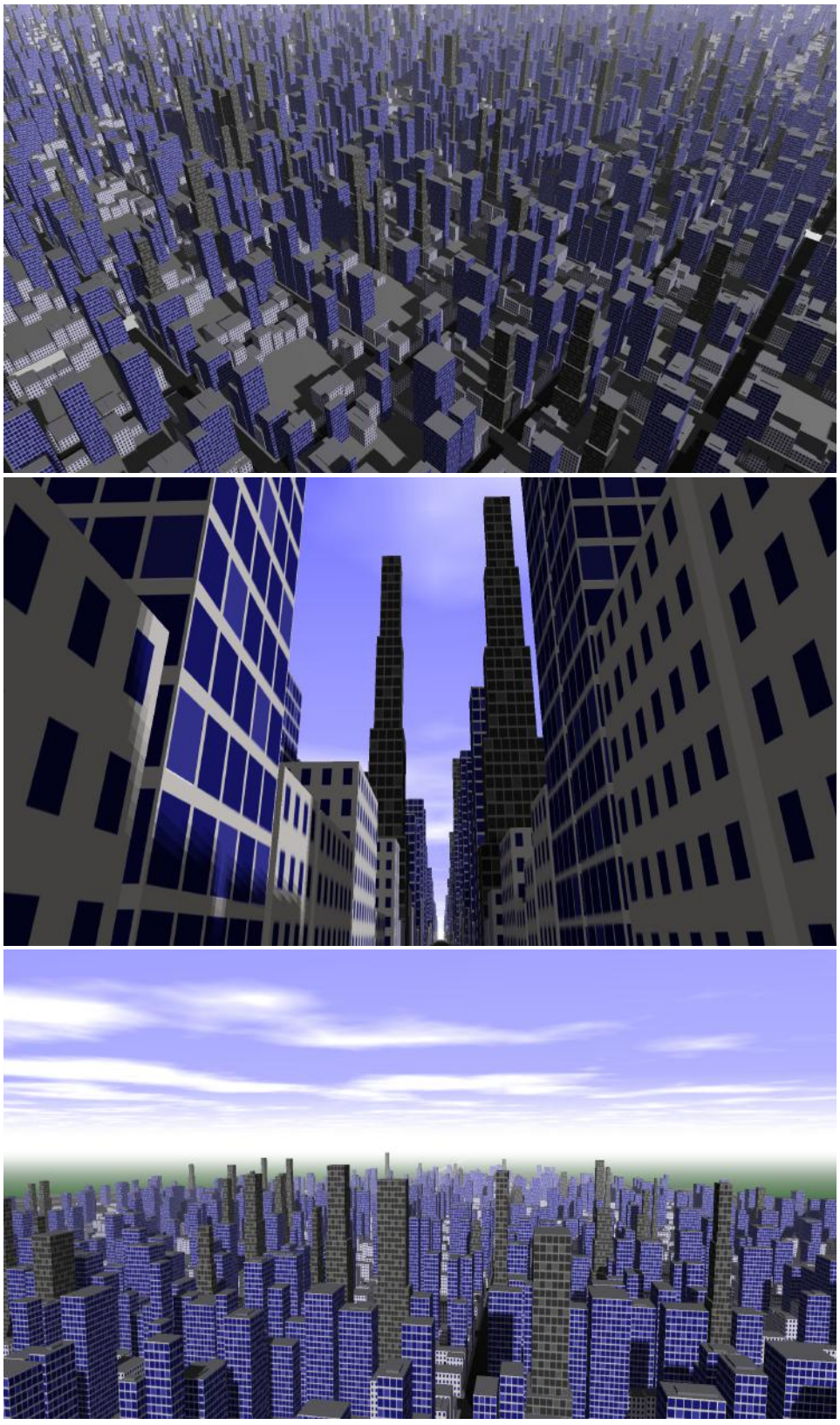
V grafu 7.1 jsou vidět naměřené hodnoty snímků za sekundu během celého intra. Jak je vidět, nevyskytují se v něm žádné velké propady a hodnoty se drží kolem 60 snímků za sekundu, což je počet snímků, který maximálně dokáže zobrazit většina dnešních monitorů. V čase 80 až 100 sekund jsou vidět nejvyšší hodnoty zapříčiněné pomalými pohledy zevnitř města k obloze. Následující pád počtu snímků za sekundu je způsoben záběrem z výšky obsahujícím téměř celé město.

Ať už je grafické intro jakkoliv technicky udělané, nejvíce záleží na tom, jak se líbí lidem a zda je zaujme. Proto jsem výsledné intro zaslal několika známým a položil každému dvě otázky: co je na intru nejvíce zaujalo a co se jim nelíbilo, nebo nejvíce kazilo dojem z grafického hlediska. Celkově byli ohlasy pozitivní. Kladně nejvíce diváci zmiňovali pohyb kamery a stíny. Také se jim líbilo, že je město tak velké a záběry v jeho uličkách. Za nedostatky považovali provedení hranice města, jeho malou barevnou různorodost, a to že některé budovy jsou příliš úzké. Někomu také vadily nenadálé pohyby kamerou narušující plynulost.

Tato zpětná vazba by mohla sloužit k dalšímu zlepšování práce. Pro zlepšení hranic by bylo možné přidat další vrstvu s předměstskou částí, kde by se nacházely menší typy domů a jiné uspořádání silnic. Tato vrstva v odevzdaném intru není, protože jeho hlavní

Obrázek 7.1: Průběh snímků za sekundu při běhu na grafické kartě GeForce GTX 760M a procesoru Intel® Core™ i7-4700HQ s rozlišením 1920x1080.

zaměření bylo na střed města. Problém s úzkými budovami je diskutabilní, podle mě jich intro obsahuje vhodné množství a nijak nepřekáží, bylo by však možné upravit některé parametry a jejich výskyt redukovat. Nenadále pohyby kamerou jsou převážně záměrné, bylo by však možné pokusit se o jejich lepší zpracování, aby se zamlouvaly většímu množství diváků.



Obrázek 7.2: Několik snímků výsledného města.

Kapitola 8

Závěr

Cílem této práce bylo nastudovat techniky pro tvorbu grafického intra s omezenou velikostí a následně takové intro vytvořit. Součástí zadání byl požadavek na použití OpenGL, se kterým jsem neměl žádné předcházející praktické zkušenosti, a proto bylo nezbytné důkladně ho nastudovat. Dále byly pro tvorbu intra nastudovány a použity techniky osvětlení scény, mlha, principy procedurálního generování, tvorba textur pomocí šumu, technika skyboxu a shadow mapping. Byly vyvinuty techniky pro generování jednoduchého města a kamerový systém pro pohyb v intru. Výsledný spustitelný program má velikost 34 kB, tím je splněn požadavek, že jeho velikost nesmí přesáhnout 64 kB.

V projektu lze pokračovat mnoha způsoby. Bylo by možné přidat další prvky města jako jsou složitější budovy nebo nové typy bloků. Další možností by bylo přidat dynamické objekty jako dopravu. O té jsem původně uvažoval, ale nakonec jsem čas raději věnoval zdokonalování prvků již obsažených pro ucelenější zážitek. Dále lze do intra zapracovat složitější techniky jako dynamický skybox, generování terénu okolí nebo textury doplňující detaily objektů, jako bump mapy a spekulární mapy.

Literatura

[1] The OpenGL Shading Language. [Online; navštíveno 11.05.2017].
URL https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.4.50.pdf

[2] Adams, T.; Adams, Z.: Dwarf Fortress. [Online; navštíveno 06.05.2017].
URL http://www.bay12games.com/dwarves/

[3] Barnhill, R. E.; Riesenfeld, R. F.: *Computer aided geometric design*. Academic Press, 1974, ISBN 978-0-12-079050-0.

[4] Creation, G.-T.: OpenGL Mathematics. [Online; navštíveno 02.05.2017].
URL http://glm.g-truc.net/0.9.8/index.html

[5] deMarche: The Secrets Of Cubic Tower. [Online; navštíveno 05.05.2017].
URL http://www.pouet.net/prod.php?which=55348

[6] Demir, I.; Aliaga, D. G.; Benes, B.: Proceduralization of Buildings at City Scale. In *2014 2nd International Conference on 3D Vision*, ročník 1, Dec 2014, ISSN 1550-6185, s. 456–463.

[7] Giesen, F.: the Ultimate Packer for eXecutables. [Online; navštíveno 23.04.2017].
URL https://upx.github.io/

[8] Hinrichs, T.: libv2 1.5. [Online; navštíveno 02.05.2017].
URL https://github.com/farbrausch/fr_public/tree/master/altona_wz4/wz4/wz4player/libv2

[9] Hinrichs, T.: The Workings of FR-08's Sound System. [Online; navštíveno 12.05.2017].
URL http://in4k.untergrund.net/various%20web%20articles/fr08snd3.htm

[10] Ikits, M.; Magallon, M.; Stewart, N.: The OpenGL Extension Wrangler Library. [Online; navštíveno 23.04.2017].
URL http://glew.sourceforge.net/

[11] Inc., T. K. G.: Hlavičkový soubor s konstantami OpenGL. [Online; navštíveno 02.05.2017].
URL https://www.khronos.org/registry/OpenGL/api/GL/glext.h

[12] Liu, N.; Pang, M. Y.: Shadow Mapping Algorithms: A Complete Survey. In *2009 International Symposium on Computer Network and Multimedia Technology*, Jan 2009, s. 1–5.

[13] Microsoft: Windows API Index. [Online; navštíveno 15.05.2017].
URL https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516(v=vs.85).aspx

[14] Oberhumer, M. F.; Molnár, L.; Reiser, J. F.: the Ultimate Packer for eXecutables. [Online; navštíveno 23.04.2017].
URL https://upx.github.io/

[15] Perlin, K.: An Image Synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, New York, NY, USA: ACM, 1985, ISBN 0-89791-166-0, s. 287–296.

[16] Sears, K.: How Two Brothers From Silverdale Made The World's Most Complex Video Game. [Online; navštíveno 11.05.2017].
URL http://www.seattleweekly.com/arts/enormous-dwarf/

[17] Segal, M.; Akeley, K.: The OpenGL Graphics System: A Specification. [Online; navštíveno 11.05.2017].
URL https://www.khronos.org/registry/OpenGL/specs/gl/glspec45.core.pdf

[18] Tatham, S.; Hall, J.: Nasm. [Online; navštíveno 15.05.2017].
URL http://www.nasm.us/

[19] Vandevenne, L.: Texture Generation using Random Noise. [Online; navštíveno 15.05.2017].
URL http://lodev.org/cgtutor/randomnoise.html

[20] de Vries, J.: Shadow Mapping. [Online; navštíveno 13.05.2017].
URL https://learnopengl.com/#!Advanced-Lighting/Shadows/Shadow-Mapping

[21] Williams, L.: Casting Curved Shadows on Curved Surfaces. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '78, New York, NY, USA: ACM, 1978, s. 270–274.

[22] Yilmaz, S.: A Tutorial on Cumputer Animation – II. [Online; navštíveno 16.05.2017].
URL http://www.keremcaliskan.com/a-tutorial-on-computer-animation-ii-2/

[23] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika*. Computer Press, 2004, ISBN 80-251-0454-0.