VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

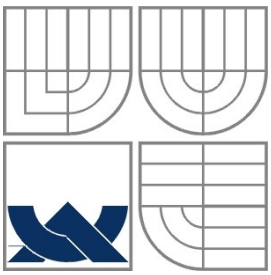
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

GRAFICKÉ INTRO 64KB S POUŽITÍM OPENGL

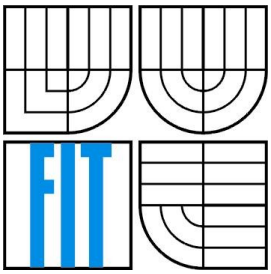
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE TOMÁŠ MILET
AUTHOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

GRAFICKÉ INTRO 64KB S POUŽITÍM OPENGL

GRAPHICS INTRO 64KB USING OPENGL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ MILET

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM VLČEK

BRNO 2010

Abstrakt

Bakalářská práce se zabývá vytvořením grafického intra s velikostí limitovanou 64kB pomocí OpenGL. Popisuje metody pro generování grafických objektů jako jsou textury a terén. Zabývá se částicovým systémem a celulárním automatem.

Abstract

This bachelor's thesis describes creating of graphics intro with limited size using OpenGL. It describes methods for generating graphical objects, such as textures and terrain. It deals with particle system and cellular automata.

Klíčová slova

Grafické intro, OpenGL, 64kB, omezená velikost, generování šumu, vytváření textur, barevné přechody, zobrazení textu v OpenGL, výšková mapa, částicový systém, celulární automat, kalendář událostí

Keywords

Graphics intro, OpenGL, 64kB, limited size, noise generation, texture generation, gradient, text in OpenGL, height map, particle system, cellular automata, calendar of events

Citace

Tomáš Milet: Grafické intro 64kB s použitím OpenGL, bakalářská práce, Brno, FIT VUT v Brně, 2010

Grafické intro 64kB s použitím OpenGL

Prohlášení

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

……
Tomáš Milet
14.5.2010

Poděkování

Rád bych poděkoval Ing. Adamovi Vlčkovi za jeho vedení a rady v průběhu práce a Ing. Adamovi Heroutovi za jeho prvotní nasměrování.

Obsah

1 ÚVOD

1.1 Intro

Cílem této bakalářské práce je grafické intro. Grafické intro [21] je krátké video vytvořené malým programem. Toto video není nikterak interaktivní – posluchač nemá možnost do intra zasahovat a ovlivňovat jej. Obsahem intra může být například předvedení myšlenky, krátký příběh, prezentace schopností programátora, upoutávka na zboží, propagace filmu, charitativní činnost nebo jiné věci. Obsah není nijak svázán a jeho náplň je čistě v rukou tvůrce programu nebo zadavatele projektu. Intra můžou být tvořena jako komerční programy, ale nejčastěji se objevuji jako programy vytvořené pro zábavu. Ve světě existuje několik profesionálních skupin, které se tvorbou grafických inter přímo zabývají. U těchto skupin vznikají i různé pomocné nástroje například pro přehrávání hudby.

Hlavním rysem intra bývá malá velikost programu v kontrastu s velikostí a silou účinku prezentované věci. Cílem intra je zaujmout cílového posluchače. Pobavit jej nebo jinak naladit. Šokovat nebo jakkoliv jinak ovlivnit. Grafická intra se většinou neobejdou bez příběhu, bez pohybu. Grafické možnosti techniky pro zobrazování neustále vzrůstají, a tak, i kdyby intro zobrazovalo sebelepší grafiku, časem zastará, a pokud by neobsahovalo žádný příběh, nebude důvod si jej někdy v budoucnu pustit. Proto je příběh důležitou součástí intra. Mimo vizuální objekty se v intrech objevuje i hudba a text. Grafická intra se stala fenoménem a na internetu jich lze nalézt stovky a tisíce. Bývají různě velikostně omezená a zobrazují nejrůznější témata. Pořádají se soutěže v nejlepších projektech a za vítěznou aplikaci lze získat různá ohodnocení.

1.2 Téma a příběh

Jak již bylo v úvodu řečeno, důležitou složkou intra je příběh. Obsah mého intra je popisován postavou - hlavním hrdinou intra. Postupně, jak promlouvá jsou zobrazovány různé scény. Ze začátku nás postava seznámí s tím, co předcházelo příběhu intra. Postava letěla letadlem, ale kvůli poruše motoru musela letadlo předčasně opustit. Poté se zobrazí scéna průletu horami ukončená pádem. Postava se ocitne v zasněžené pustině hor. Seznámí nás se skutečností, že její jediná šance na přežití je nalezení horské chaty. Poté postava cestuje horami aby nalezla svůj cíl. Cestou se objeví nepříjemnosti. Když konečně dorazí k cíli, zjistí, že tam chata není. Místo ní se zobrazuje objekt nazvaný „chata missing mesh“, což znamená, že objekt chaty nebyl nalezen. Postava tak zjistí, že si s jejím osudem někdo hraje. Rozhodne se proto ukončit svůj život. Vydá se hledat správné místo k tomuto činu. Po chvíli hledání jej nalezne. Místo je označeno „uninitialized memory“ -

neinicializovaná paměť. Postava k tomuto místu přistoupí. Následuje pád černým prostorem s jedničkami a nulami symbolizující přístup k neinicializované paměti. Když postava dopadne na číslici dva, zobrazí se hlášení „segmentation fault“. Vyobrazí se fiktivní pád systému a tím příběh postavy končí. Zobrazí se závěrečné titulky. Konec.

1.3 OpenGL

Pro vykreslení intra je použito OpenGL. OpenGL [12] je zkratka pro „Open Graphics Library“ - otevřená grafická knihovna. Knihovnu vyvinula společnost Silicon Graphics Inc v roce 1992. OpenGL je nízkoúrovňové softwarové rozhraní obsahující několik set funkcí a procedur umožňujících specifikaci a manipulaci grafického objektu v dvojrozměrném nebo třírozměrném prostoru.

Obsahuje funkce pro nastavení vykreslování, transformace, texturování, osvětlení a další. Pomocí OpenGL lze vykreslovat jednoduché grafické prvky jako jsou čáry, body a polygony. Lze jim nastavovat barvu, texturu a způsob, jakým budou vykreslovány. Obsahuje tři transformační matice. Tyto matice se používají pro: projekce trojrozměrného bodu do roviny obrazovky (projekční matice), transformace bodu v prostoru (modelview matice) a pro transformaci texturovacích koordinátů (texturovací matice). Pomocí těchto matic lze změnit vzhled skupin grafických objektů najednou bez zasahování do objektů samotných. OpenGL je multiplatformní knihovna se zpětnou podporou. Obsahuje pouze funkce pro ovládání grafiky.

K OpenGL byly napsány různé nadstavby, které jeho využitelnost zvyšují. Za zmínku stojí knihovna GLU, která poskytuje další užitečné funkce a procedury (například funkce pro práci s NURBS křivkami a pro práci s kvadrikami).

Při implementaci aplikací, které využívají služeb OpenGL se používají i grafická rozšíření. Grafická rozšíření jsou hardwarově závislá rozšíření umožňující další a pokročilejší práci s obrazem. Některá dřívější rozšíření, jako je multitexturing, byla později přidána do novější verze OpenGL (podpora multitexturingu je v OpenGL od verze 1.3). Seznam všech grafických rozšíření je uveden v knihovně „`glext.h`“. Pro zjištění seznamu grafických rozšíření, které grafická karta podporuje se využívá OpenGL funkce `glGetString` s předaným parametrem `GL_EXTENSIONS`.

1.4 Velikost 64kB

Velikost výsledného zkompilovaného programu nesmí překročit 64kB. Tato mez se může z počátku zdát příliš malá nato, aby bylo možné dosáhnou dobrých výsledků. Metod, jak toho dosáhnout, je vícero. Používají se různé programovací techniky, nastavení překladače při překladu a programy (exe packery), které dokáží výslednou velikost velmi zmenšit.

V projektu jsem používal UPX [16] exe packer. Tento exe packer bylo možné používat jak pod operačním systémem Windows, tak pod operačním systémem Linux. Při jeho použití se výsledná velikost programu zmenšila až na jeho třetinu. Na internetu lze nalézt celé řady exe packerů. Bývají různě výkonné. Některý trvá zpracování programu déle – i několik minut. Exe packery dosahují různých poměrů při komprimaci.

Nastavením překladače (gcc) dovolovalo při překladu odstranit nepoužívaná data. Odstranit nepoužívané funkce. Při překladu se používaly, mimo jiné, parametry „-Os“ pro optimalizaci programu na velikost a „-s“ pro odstranění nepotřebných informací.

Protože nelze většinu grafických dat uložit staticky přímo v aplikaci (zabírají příliš mnoho místa) je potřeba najít jiný způsob jejich uložení. Například obrázky ve formátu jpg zabírají i při komprimaci několik kB, a proto je jejich použití jako textur nemožné. Proto se většina grafických objektů nejprve vygeneruje podle dané šablony (algoritmu), která zabírají méně místa. Algoritmu lze předat různé nastavení, a tak může generovat odlišné grafické objekty stejné třídy. Otázkou zůstává: které grafické objekty generovat, a které si uložit pomocí konstantních dat. Některé grafické objekty jsou příliš složité na popis a implementování takovýchto objektů by zabralo spousty času a navíc by pak výsledný kód mohl zabírat i více místa, než kdyby byl objekt uložen v konstantních datech. U tohoto projektu jsem tuto otázku musel řešit u fontů. Původně jsem chtěl jednotlivá písmenka rasterizovat do textury z jejich vektorového popisu, ale usoudil jsem, že by výsledný kód zabíral více místa, než uložení malého obrázku s již předem rasterizovaným fontem.

Další omezení velikosti spočívá v používání funkcí. Pokud je aspoň nějaká část programu podobná jiné a je přiměřeně veliká, je vhodné ji „vytknout“ pomocí funkce. Tímto lze ušetřit i několik kB. Nastává však problém s rychlostí. Kromě času potřebného k vykonání kódu je potřeba i čas pro volání funkce. Příkladem využití funkcí pro zmenšení velikosti aplikace může být napsaná knihovna pro operace s vektory. V programu bylo potřeba s nimi často počítat, a tak vznikla tato knihovna, která většinu potřebných operací obstarává. Dalším příkladem může být knihovna pro obecné abstraktní datové typy.

2 IMPLEMENTACE

Projekt je implementován v jazyce C. Převážně byl implementován pod operačním systémem Linux v textovém editoru Vim [17]. Pod operačním systémem Windows byl používám editor Dev-Cpp. Pro vytvoření a obsluhu okna pod operačním systémem Linux je využíváno služeb knihovny SDL [20]. Pro stejnou práci je pod operačním systémem Windows využito aplikační rozhraní WinAPI [1][18].

Aby bylo možné program zkompilovat na obou systémech, bylo potřeba části kódů, které jsou platformě závislé obalit direktivami preprocesoru jazyka C. Pomocí preprocesoru se poté při kompilaci rozhodne, které implementované funkce překladač použije. Pro snadnější práci vzniklo i malé rozhraní pro vytvoření okna. Pomocí tohoto rozhraní, umístěného v knihovně „`winwindow.h`“, je vytvořeno okno stejnými funkcemi jak pod operačním systémem Linux tak Windows.

V projektu se nevyužívají veškeré implementované funkce a části kódu. Některé byly implementovány navíc z důvodu nejistoty, zda budou používány (například některé operace s vektory). Některé jsou nedokončené části vylepšení, které se již nestihly dodělat (například sčítání barevných přechodů). Některé z těchto části kódů jsou v projektu ponechány, protože otevírají možnosti k budoucímu vylepšování a při kompilaci se odstraní, takže neovlivňují výslednou velikost aplikace.

2.1 Rozdělení

Aplikace je rozdělena do několika knihoven. Hlavní část programu se nachází v souboru „`main.c`“. V knihovně „`winwindows.h`“ se nacházejí funkce pro vytvoření a řízení okna. V knihovnách „`adt.h`“, „`adtfce.h`“, „`list2.h`“, „`btree.h`“ se nacházejí abstraktní datové typy. V knihovnách „`heightmap.h`“, „`grass.h`“, „`bush.h`“, „`snow.h`“, „`sail.h`“, „`flag.h`“ a dalších se nalézají grafické objekty. V knihovně „`fractal.h`“ se nachází funkce pro generování šumu. V dalších knihovnách „`map.h`“, „`colormap.h`“ jsou umístěny funkce a struktury pro práci s barevnými přechody. Knihovny „`mymath.h`“, „`transform.h`“, „`camera.h`“ obstarávají práci s vektory (sčítání, násobení, skalární součin, normalizace a jiné), transformace objektů (posun, měřítko a rotace) a transformaci a nastavení kamery (nastavení bodu, na který se kamera dívá, vypočtení natočení kamery podle úhlu natočení, šířka a poměr stran kamery a další). V knihovnách „`font.h`“, „`model.h`“, „`calendar.h`“, „`mytime.h`“ se nacházejí funkce pro práci s fontem, grafickými modely, kalendářem a časem (vytvoření času, sčítání času, porovnávání času, atd.).

2.2 Abstraktní datové typy

Jelikož je program napsaný v jazyce C, nemůžeme využívat objektově orientované programování. Stejně tak, z hlediska úspory místa, je třeba vytvořit obecné abstraktní datové typy (dvousměrný seznam, binární strom). Lineární seznamy jsou v projektu použity pro různé účely (například pro uložení událostí do kalendáře, pro uložení barevných přechodů, apod.). Vzhledem k tomu, že každý objekt (struktura), který je nutné ukládat do abstraktní datové struktury, má různou velikost a různé metody pro práci a vzhledem k tomu, že jazyk C není objektově orientovaný jazyk (pro využití polymorfismu), musíme navrhnout způsob ukládání těchto prvků.

Pro uložení prvku do abstraktního datového typu použijeme strukturu, která, mimo obvyklých prvků (jako jsou ukazatel na svého následovníka u lineárního seznamu), bude obsahovat ukazatel `void` na data. Tím si zajistíme možnost vložení jakéhokoliv datového objektu do seznamu. Problém však nastane, pokud bychom chtěli takovýto abstraktní datový typ uvolnit z paměti. Program neví, jaká data jsou v prvku uložena, a tak ani neví, jak data uvolnit. Proto uložíme i ukazatel na funkci, která daná data dokáže uvolnit. Stejný problém nastane, pokud bychom chtěli vytvořit kopii. Proto uložíme i ukazatel na funkci pro zkopírování dat. Abstraktní datový typ obsahuje také ukazatel na funkci pro porovnání dat a velikost samotných dat. Tyto obslužné funkce (a velikost dat) jsou uloženy ve struktuře. Při vytváření například seznamu, zadáme tyto přidružené informace. Abychom nemuseli pokaždé, když chceme objekty ukládat do abstraktního datového typu, implementovat znovu základ tohoto typu, vystačíme si jen s jednou, obecnou verzí. Abstraktní datový typ pro dvousměrný seznam je uložen v knihovně „`list2.h`“ a abstraktní datový typ pro binární strom je uložen v knihovně „`btree.h`“. Struktura obsahující ukazatele na obslužné funkce se nachází v knihovně „`adtfce.h`“.

Jelikož se poměrně často musí aplikovat funkce na každý prvek abstraktního datového typu a jelikož je průchod abstraktním datovým typem pokaždé stejný, byly vytvořeny funkce, které toto řeší. Těmto mapovacím funkcím se předá ukazatel na abstraktní datový typ, ukazatel na funkci, která má být aplikovaná na jednotlivé prvky a ukazatel na argumenty funkce. Inspiroval jsem se jazykem Python [15] a funkcí map. Využití je široké. Například, pokud bychom chtěli posunout všechny časy všech událostí v kalendáři (implementované pomocí lineárního seznamu), vytvořili bychom jedinou funkci pro posunutí jedné události a tu předali mapovací funkci společně se seznamem událostí a časem posunu. Nebo pokud bychom ukládali grafické objekty do binárního stromu (abychom vykreslovali jen ty, které jsou vidět), stačilo by při vykreslení zavolat rovněž jen jednu funkci a program by prošel binární strom a na jeho prvky by se aplikovala předaná funkce pro vykreslení.

3 GENEROVÁNÍ ŠUMU

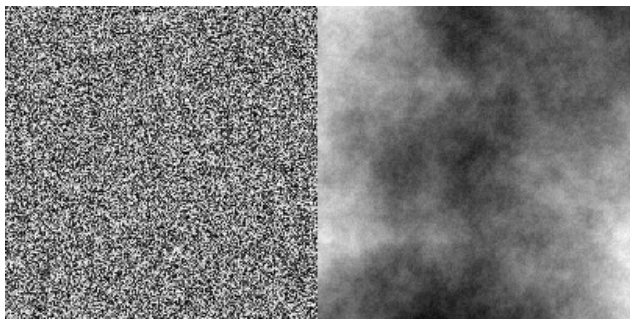
Protože aplikace nemůže obsahovat příliš mnoho statických dat (jako jsou textury, souřadnice, …), neboť by zabíraly příliš mnoho paměti, je potřeba data nějakým způsobem automaticky vygenerovat při spuštění aplikace. Paměťově méně náročné je uložit způsob vytvoření daného objektu, než uložit jeho konkrétní popis. Proto je dána přednost uložení dat pomocí algoritmu k jejich vytvoření než pomocí dat samotných. Generování šumu je použito pro vytváření textur, terénu, trávy, křoví a dalších objektů. Jeho využití je veliké, proto je mu věnována velká pozornost.

3.1 Generování pseudonáhodného čísla

Základem je umět vygenerovat pseudonáhodné číslo. Pro vygenerování pseudonáhodného čísla jsem nejprve používal generátor `rand` z knihovny „`stdlib.h`“. Tento generátor generuje celá čísla s rovnoměrným rozložením pravděpodobnosti na definovaném rozsahu. Generátor se musí inicializovat funkcí `srand`, které se předává celé číslo. Jedná se o takzvaný „seed“. Upustil jsem však od tohoto generátoru. Důvod byl ten, že pod různými operačními systémy se generátor choval jinak. Při nastavení stejného „seed“ pod operačním systémem Windows generoval jiná čísla než pod operačním systémem Linux. Proto jsem převzal [14] generátor ze zdrojových kódu glibc. Číslo pro „seed“ je vzato z funkce `time`, která vrací počet sekund od prvního ledna 1970 – tím se zajistí, že šum bude pokaždé vypadat jinak. Pro lepší práci je číslo vygenerované tímto převzatým generátorem převedeno na číslo s plovoucí řádovou čárkou ve zvoleném rozsahu. Toto zajišťuje funkce `Random`, která se, stejně jako převzatý generátor, nachází v knihovně „`standard.h`“ Dále je už v programu použita jen tato funkce, neboť není potřeba generovat pseudonáhodná celá čísla.

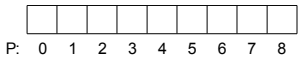
3.2 Metoda 1

Pokud bychom vygenerovali sérii hodnot pomocí funkce `Random`, získali bychom šum (obrázek 3.1). Tento šum však nemá vhodné vlastnosti pro další použití (například pro generování textur), neboť rozdíly v sousedních hodnotách jsou příliš odlišné. Proto pro vygenerování série hodnot negenerujeme jednotlivé hodnoty za sebou, ale z již vygenerovaných hodnot vypočteme další s přidanou náhodou.



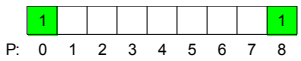
Obrázek 3.1Prosté vygenerování hodnot a vygenerování hodnot s ohledem na ostatní již vygenerované hodnoty

Postup je následující. Budeme chtít vygenerovat pole o devíti prvcích (obrázek 3.2) (prvky v poli budeme značit písmenem P následováno číslem pořadí prvku začínajícím od nuly).



Obrázek 3.2: Pole

Nejprve se vygenerují dvě hodnoty s nastaveným rozsahem. Tyto hodnoty se zapíší do prvního (P0) a posledního (P8) prvku pole. Pole má vygenerované krajní hodnoty (obrázek 3.3).



Obrázek 3.3: Krajní hodnoty

Poté se vypočítá průměrná hodnota z krajních prvků. Tato hodnota se zapíše doprostřed do prvku P4. K hodnotě v prvku P4 se pak připočte další vygenerovaná hodnota tentokráte však s polovičním rozsahem. Pole má vygenerovány prvky P0,P4,P8 (obrázek 3.4).

Obrázek 3.4: Střed

Nyní lze na toto pole pohlédnout jako na dvě oddělená pole P0-P4 a P4-P8, přičemž obě mají již vygenerovány krajní hodnoty (obrázek 3.5).

Obrázek 3.5: Dvě pole A a B

Postup se tedy zopakuje. Pro pole A (P0-P4) se výše popsaným postupem vygeneruje prvek P2. Pro pole B (P4-P8) se vygeneruje prvek P6 (obrázek 3.6).

Obrázek 3.6: Rozdělení A a B

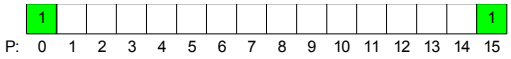
Vzniknou tak čtyři pole P0-P2, P2-P4, P4-P6 a P6-P8. Pro každé se vygeneruje jejich středový prvek. Tento postup, kdy se vygeneruje vždy prostřední prvek se opakuje tak dlouho, dokud mezi krajními prvky pole existují jiné prvky. Jelikož se u jednotlivých menších a menších částí původního pole zmenšuje generovaný rozsah, zmenšuje se i rozdíl mezi sousedními prvky. Teoreticky je u dvou prvků, které jsou u sebe dvakrát blíže, rozdíl hodnot poloviční. Výsledné pole je na obrázku 3.7, kde jednotlivé prvky mají číslo, které značí pořadí generování.

Obrázek 3.7: Výsledek

3.3 Metoda 2

Metoda 1 má několik úskalí. První z nich je možnost generovat sekvenci jen na poli délky 2^n+1. To sebou přináší několik problémů. Proto lze výše popsaný postup mírně upravit tak, aby již délka pole nebyla limitující.

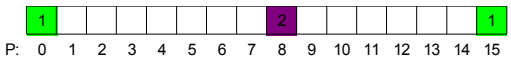
Postup popíšeme na poli délky šestnáct. Nejprve se, stejně jako u postupu výše, vygenerují krajní hodnoty - prvky P0 a P15 (obrázek 3.8).



Obrázek 3.8: Pole

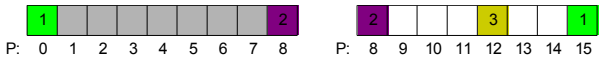
Nyní by výše popsaný postup nemohl být aplikován, protože neexistuje prvek, který je přesným středem pole. Proto se místo středu vybere prvek, který pole rozdělí tak, aby jeho levá část byla co nejdelší a aby na ní mohl být aplikován výše popsaný postup. Index tohoto prvku je číslo 2^n takové, že je menší než délka pole. Pro pole délky 16 je to index 8. Původní pole jsme tedy rozdělili na dvě pole: A (P0-P8) a B (P8-P15). Prvku P8 je nastavena hodnota váhového průměru P0 a P15.

Váha u P0 se vypočítá jako délka pole B děleno délka původního pole (7/16). Váha u P15 se vypočítá jako délka pole A děleno délka původního pole (9/16). Jinými slovy, pokud vybraný střed leží blíže levému okraji, má hodnota v levém okraji větší váhu. Poté se k P8 připočte další vygenerovaná hodnota se zmenšeným rozsahem.



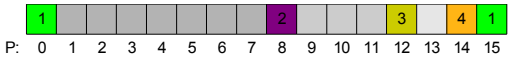
Obrázek 3.9: Vybraný střed

Na pole A (P0-P8) již lze aplikovat první metodu. Pole B má však svoji délku stále různou od 2^n+1. Proto se postup uvedený u této metody znovu aplikuje. Hodnota 2^n menší než 7 (délka pole B) je 4. Prvku P12 (8+4) je nastaven váhový průměr z krajních bodů pole B (P8,P15). Váhy jsou 3/8 pro P8 a 5/8 pro P15. Dále se k P12 připočítá nová náhodná hodnota se znovu zmenšeným rozsahem.



Obrázek 3.10: Vygenerované pole A a pole B

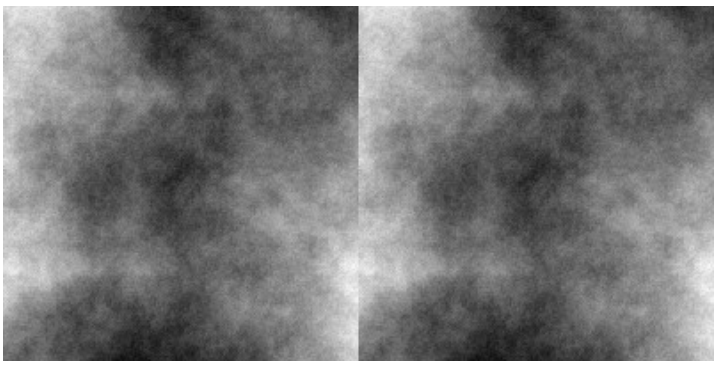
Vzniknou dvě pole: C (P8-P12) a D (P12-15). Na pole C lze opět aplikovat první metodu. Pole D musíme opět stejným způsobem rozdělit. Vzniknou další pole E (P12-P14) a F (P14-P15). Nyní už lze na obě pole aplikovat první metodu. Tím proces generování končí.



Obrázek 3.11: Výsledek

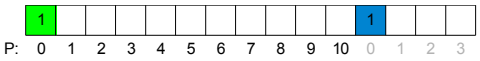
3.4 Metoda 3

Další úskalí, které ani metoda 2 neřeší, je v navazování. Pokud by se pomocí metody 1 nebo 2 vytvořila textura, přechody mezi jednotlivými částmi by byly zřetelné (obrázek 3.12), neboť levá krajní hodnota je jiná, než pravá krajní hodnota .



Obrázek 3.12Viditelná hrana mezi texturami

Tento problém lze částečně odstranit vygenerováním stejné hodnoty do pravého koncového prvku. Vznikne tím ovšem sekvence dvou stejných hodnot hned za sebou. Úplné odstranění tohoto problému je negenerovat pravou koncovou hodnotu. Místo toho se pravá koncová hodnota vybere z levé koncové hodnoty následujícího pole. Jelikož je následující pole stejné, je pravá koncová hodnota nastavena na levou koncovou hodnotu s tím rozdílem, že neleží na posledním indexu pole, ale na indexu o jedničku vyšším. Tímto se zajistí navazování, neboť se bere v potaz i následující pole.

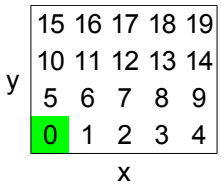


Obrázek 3.13: Pravá krajní hodnota

3.5 Metoda 4

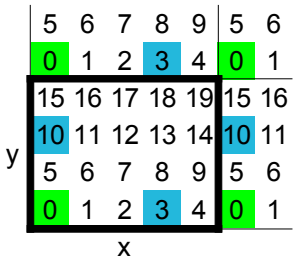
Další úskalí (nebo omezení), které ani metoda 3 neřeší, je nemožnost vygenerovat vícerozměrné pole. Výše popsanými metodami lze kupříkladu vygenerovat texturu o jednom řádku či sloupci, ale pokud bychom hodnoty použili do dvojrozměrného pole, zjistili bychom, že hodnoty, které leží na dalším řádku na sebe nenavazují nebo mají neustále stejnou hodnotu (první případ, při pohledu na dvojrozměrné pole jako na jednorozměrné pole, druhy případ, při pohledu na dvojrozměrné pole jako na pole polí). Navazováním se v dalším textu míní, že rozdíl hodnot v přímo sousedících prvcích je pro každé dva takové prvky v daném rozsahu. Jiným slovy: hodnoty v přímo sousedících prvcích se liší maximálně o danou hodnotu. Kdybychom tedy na dvojrozměrné pole (o velikosti N,M) pohlédli jako na jednorozměrné pole (místo prvků P(0,0) až P(N-1,M-1), prvky P(0) až P(N*M-1)), zjistili bychom, že potřebujeme, aby na sebe prvky navazovaly nejen v řádcích (například mezi prvky P1 a P2) ale i ve sloupcích (například mezi prvky P1 a P(1+N)).

Postup si nejprve popíšeme na dvojrozměrném poli velikosti 5 na ose x a 4 na ose y. Na pole se bude pohlížet jako na jednorozměrné pole s prvky P0 až P19. Stejně jako u předešlých metod, se i tady vygeneruje první hodnota do prvního prvku (P0).



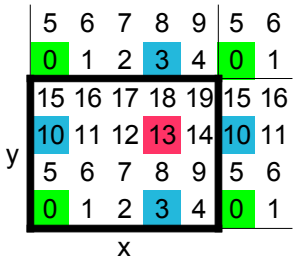
Obrázek 3.14:
2D pole

Poté se v prvním řádku a v prvním sloupci vybere střed (stejně jako u metody 2 je vybráno číslo 2^N menší než délka respektive šířka). Jsou to prvky P3 pro první řádek a P10 pro první sloupec (obrázek 3.15 – pole má zde i sebe sama jako sousedy).



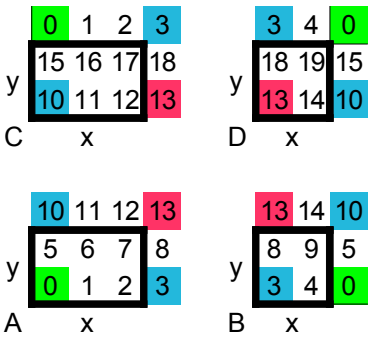
Obrázek 3.15: Středy 1. řádku a sloupce

Nyní, když máme vygenerovány hodnoty uprostřed prvního řádku a sloupce, vygeneruje se hodnota „středu“ celého dvojrozměrného pole. Tato hodnota leží v prvku P13, což je „střed“ řádku daného „středem“ prvního sloupce.



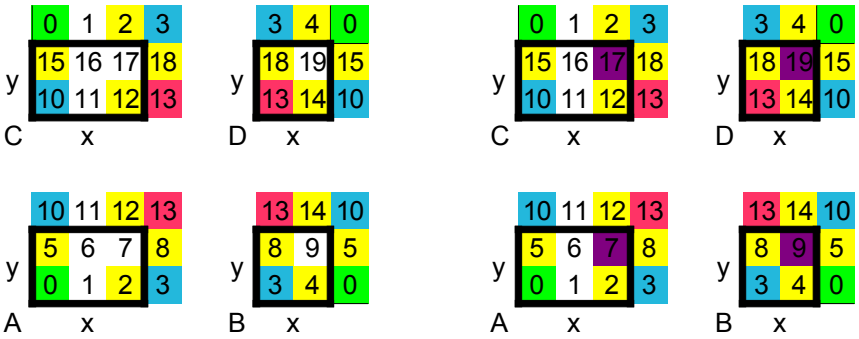
Obrázek 3.16: "Střed" 2D pole

Teď se ovšem hodnota P13 nevypočítává ze dvou krajních bodů, ale ze čtyř (P3,P10,P3,P10) (prvky jsou zde uvedeny dvakrát ty samé. Jedny leží přímo v generovaném poli, druhé leží v sousedních polích (což je opět to samé původní pole)). Výpočet je opět váhový průměr a stejně tak jako výše, i tady je váha dána vzdálenosti prvku od krajních prvků. Poté se k vypočítanému váhovému průměru připočte nově vygenerovaná hodnota. Nyní lze, podobně jako u jednorozměrného pole, toto pole rozdělit na čtyři dvojrozměrná pole (A,B,C,D).



Obrázek 3.17: Rozdělení na 4 2D pole

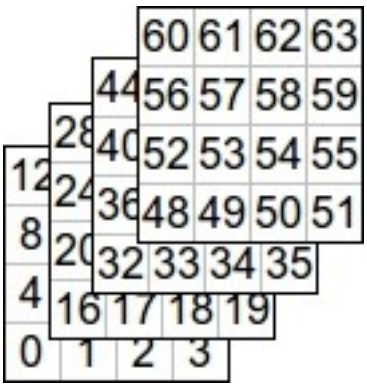
K těmto polím přistupujeme stejně jako k původnímu poli. Vygenerují se jejich první řádky a první sloupce. Poté jejich středy.



Obrázek 3.18: První řádky, první sloupce, středy

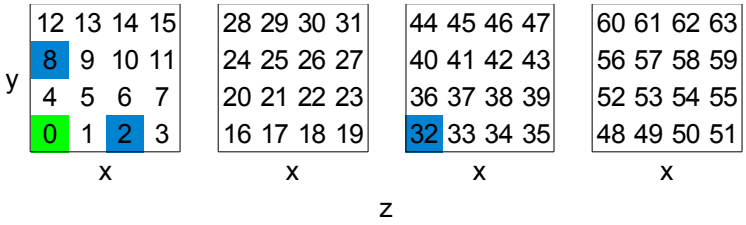
Poté lze opět rozdělit jednotlivá dvojrozměrné pole (A,B,C,D) na další čtyři pole a postup se opakuje, dokud není velikost 1 na 1.

Postup si předvedeme i pro trojrozměrné pole. Pole o rozměrech 4 pro osu x 4 pro osu y a 4 pro osu z Obrázek 3.19 ukazuje jakým způsobem jsou indexovány prvky pole. Na dalších obrázcích jsou jednotlivé vrstvy vykresleny vedle sebe, takže souřadnice na ose z nám vybírá daný obdélník.



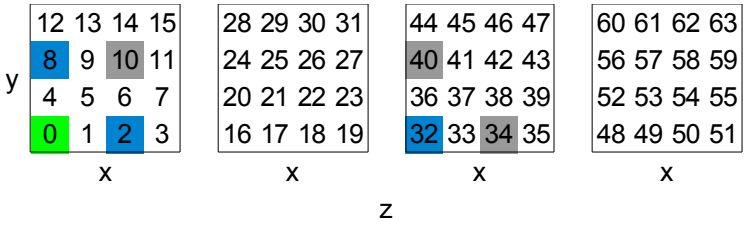
Obrázek 3.193D pole

Vygeneruje se první prvek a poté pro každou osu (x,y,z) se najde střed (prvky P2,P8,P32). Každý z těchto prvků má dva krajní body. Váhově se spočítá hodnota a připočte se náhoda.



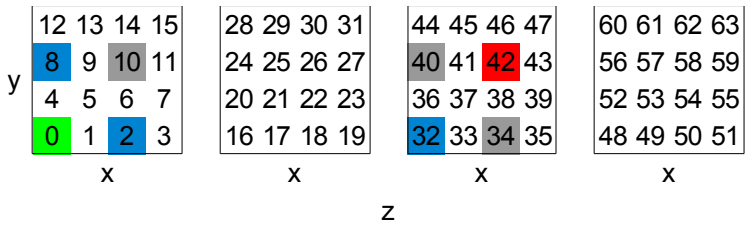
Obrázek 3.20: První prvek a středy os x,y,z

Vygenerují se středy dvourozměrných polí (pole x,y pro z=0, pole x,z pro y=0, pole y,z pro x=0) tedy prvky (P10,P34,P40).



Obrázek 3.21: Středy dvourozměrných polí (xy,xz,yz)

Vygeneruje se střed trojrozměrného pole. Střed trojrozměrného pole je v prvku P42. Tomuto prvku se, obdobně jako u dvojrozměrného pole, spočítá váhový průměr. Nyní již však z šesti krajních prvků (P10,P34,P40,P10,P34,P40). Připočte se náhoda. Nyní lze pole rozdělit na osm trojrozměrných polí a postup se na nich zopakuje.



Obrázek 3.22: Střed 3D pole

Obdobně by se postupovalo pro čtyřrozměrné pole. Postup je stejný jen se přidává další rozměr. V projektu je implementována funkce, která využívá metodu 4 v její obecné podobě – dokáže generovat N dimenzionální pole s určenými rozměry pro danou dimenzi. Tato funkce se jmenuje `fractal_Generate` a nachází se v knihovně „`fractal.h`“. Této funkci je předán počet dimenzí výsledného pole, rozměry v jednotlivých dimenzích, základní rozsah pro funkci `Random` a číslo, které reprezentuje míru vyhlazení. Tímto číslem se dělí rozsah pokaždé, pokud se generuje menší část pole (například u první metody při rozdělení pole na dvě části). Kromě funkce `fractal_Generate` se v knihovně nachází funkce `fractal_Normalize`. Tato funkce slouží k převedení vygenerovaných hodnot do nastaveného rozsahu. Například chceme, aby pole obsahovalo jen hodnoty z rozsahu 0 a 1.

Mimo výše uvedeného postupu jsem zkoušel použít velmi jednoduchou techniku, kterou lze taktéž generovat N rozměrný šum. Pole se nejprve inicializuje na nulu. Poté se rozdělí na dvě části a hodnoty v jedné se zvýší a hodnoty v druhé se sníží. Toto rozdělení se aplikuje několikrát za sebou. Mění se jen množiny prvků, které svoji hodnotu zvyšují a které snižují. Pro dvourozměrné pole by se použila obecná rovnice přímky. Prvkům ležících na levé straně přímky by se hodnota zvýšila a prvkům vpravo by se hodnota snížila. Koeficienty rovnice přímky by se generovaly. U třírozměrného pole by se prvky rozdělily podle roviny. Přestal jsem tento postup využívat z důvodu nenavazování vygenerovaných polí a z důvodu dlouhého generování, protože se každý prvek pole musí projít několikrát.

4 GRAFICKÉ OBJEKTY

Většina grafických objektů, které lze v intru zahlédnout jsou vytvořeny až při spuštění intra. Je to z důvodu snížení paměťových nároků. O vytvoření daného grafického objektu se stará vždy jedna knihovna. Celé intro je z nich sestaveno. Jsou různě umístěné a některé se pohybují. Některé mají více instancí (například tráva). Zobrazují se a zanikají při průběhu intra.

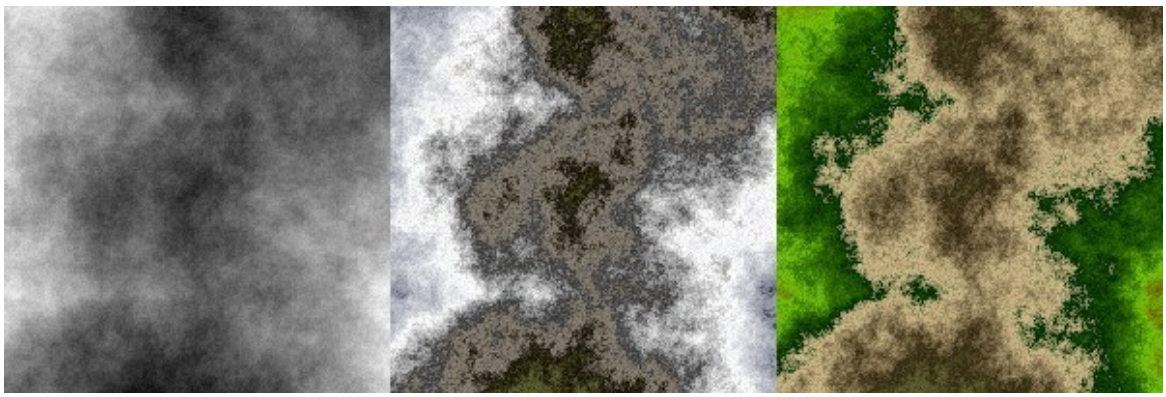
4.1 Barevné přechody

Barevný přechod slouží k převedení jednoho čísla v daném rozsahu na barvu. V programu je to implementováno pomocí tří seznamů. Každý pro jednu barevnou složku (v případě průhlednosti je přidán ještě jeden seznam). Prvek seznamu obsahuje definici úsečky. Tato úsečka se použije pro přemapování hodnoty vstupu (vstup je hodnota x a výstup je hodnota y pro x,y takové, že definují bod leží na úsečce). Pro hodnotu, kterou chceme převést na barvu, tedy nalezneme v seznamech pro každou barevnou složku úsečku, provedeme přemapování a výsledkem jsou hodnoty složek barvy. Barevný přechod se nachází v knihovně „`colormap.h`“. Barevný přechod je uložen v konstantních datech. Pro tvorbu přechodů byl použit grafický editor Gimp, neboť přechody ukládá v textovém formátu. Pro převod tohoto formátu na zdrojový kód v jazyce C je použit jednoduchý skript v jazyce Python.

Obrázek 4.1Barevný přechod

4.2 Textury

Textury [13] slouží pro detailní popis struktury povrchu. Pro vygenerování textury je použit generátor šumu `fractal_Generate`. Tímto je však vytvořena textura pouze v odstínech šedé. Proto, aby nebyla celá scéna černobílá, použijeme na vygenerované pole barevný přechod. Jednotlivé pixely tak obarvíme. Tím získáme plně barevnou texturu. V projektu jsou použity textury dvojrozměrné (vlajka) i textury třírozměrné (terén). Další využití textury je pro vytvoření fontu.



Obrázek 4.2Šum a dvě textury s různými barevnými přechody pro stejný šum.

4.3 Font

V projektu je důležité zobrazení textu. Texty jsou použity jak u samotného příběhu, tak pro vymodelování grafických objektů. Dále se texty zobrazují v závěrečných titulcích. Vzhledem k tomu, že byl projekt z velké části vyvíjen v operačním systému Linux, musely by se implementovat dva způsoby načtení a vytvoření fontu. Proto je font uložený v konstantních datech jako obrázek [6]. Aby se ušetřila paměť má jeden znak fontu velikost šest pixelů na šířku a dvanáct pixelů na výšku . V programu je uložen pouze jeden bit na jeden pixel obrázku. Dále jsou uloženy jen znaky velké abecedy, číslice, otazník, vykřičník, tečka, dvojtečka, uvozovky a čárka. Konstantní pole tedy obsahuje jen 342*12 bitů zarovnaných na osm bitů (342 je šířka 6 pixelů pro 57 znaků).

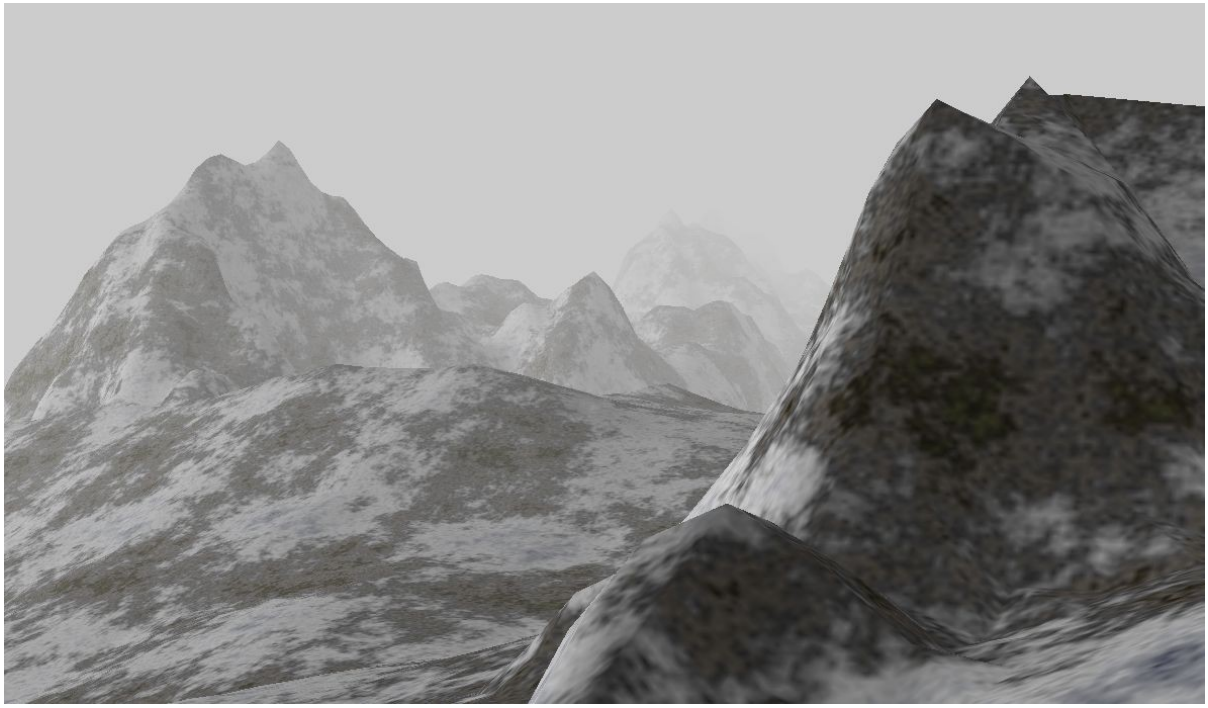
Předtím než je možné zobrazit text, je nutné z konstantních dat vytvořit font tak, aby bylo možné jej zobrazit. K tomu slouží funkce `font_Create`, která se nachází v knihovně „`font.h`“. Tato funkce vytvoří pro každý znak texturu s alfa kanálem. Znaky jsou v textuře uloženy bílou barvou. Poté jsou textury použity pro vykreslení obdélníků. Každý obdélník pro každý znak je uložen do display listu [4]. Tím by se mělo zrychlit vykreslování. Navíc se vykreslování zjednoduší, neboť pro vykreslení jednoho znaku je potřeba jen index display listu získaný z hodnoty znaku. Pro zjednodušení vykreslování je nutné znaky uložit hodnotou začínající od nuly (aby bylo možné indexovat pole display listů). Toto zajišťuje jednoduchý skript v jazyce Python. Veškerý text v programu je uložen v konstantních datech. Pro zjednodušení manipulace s textem jsou implementovány funkce pro vytištění řádku na daných souřadnicích o dané velikosti a barvě. Také jsou implementovány funkce pro vertikální a horizontální centrování textu a zjištění velikosti textu.

Obrázek 4.3Obrázek použitého fontu

4.4 Terén

Terén je v intru použit pro vykreslení hor. Terén je vytvořen pomocí výškové mapy. Výšková mapa [8] je dvojrozměrné pole, kde hodnota prvků pole udává výšku na daných souřadnicích terénu. Výšková mapa je vygenerována jako dvourozměrný šum pomocí funkce `fractal_Generate`. Výšková mapa se nachází v knihovně „`heightmap.h`“. Mimo vygenerování výškové mapy pomocí funkce `heightmap_Generate` se v knihovně nachází i funkce pro zjištění výšky v mapě na daných souřadnicích a pro sčítání dvou výškových map. Terén v intru je složen ze dvou výškových map. V jedné jsou parametry nastaveny tak, aby byla vyhlazenější a nižší. V druhé jsou parametry nastaveny tak, aby měla větší výškové rozdíly. Poté jsou obě mapy sečteny. Vznikne tak terén s rovinami a horami. Aby byl terén pořád stejný při každém spuštění aplikace, je před jeho generováním nastaven vybraný „seed“ pro funkci `Random`.

Další částí terénu je textura. Terén bez textury by byl jen velmi těžce rozlišitelný. U terénu je použita trojrozměrná textura (pro zrychlení byla použita textura o rozměrech 64 na 64 na 64). Stejně jako na vytvoření výškové mapy je i pro vytvoření textury využita funkce `fractal_Generate`. Poté se na vygenerované pole aplikuje barevný přechod vytvořený v grafickém editoru Gimp [11]. Vznikne tak textura, která má základ kamenný se zasněženými místy. Jelikož je textura trojrozměrná je výpočet souřadnic pro texturování jednoduchý. Postačí vydělit souřadnici bodu konstantou, která značí, jak bude textura roztažená, a tuto hodnotu použít jakou texturovací souřadnici. Na celý terén je použita jen jedna textura, která sama na sebe navazuje, ale protože je třírozměrná úplně to stačí - není příliš rozeznat opakující se vzor.



Obrázek 4.4Vygenerovaný terén s trojrozměrnou texturou.

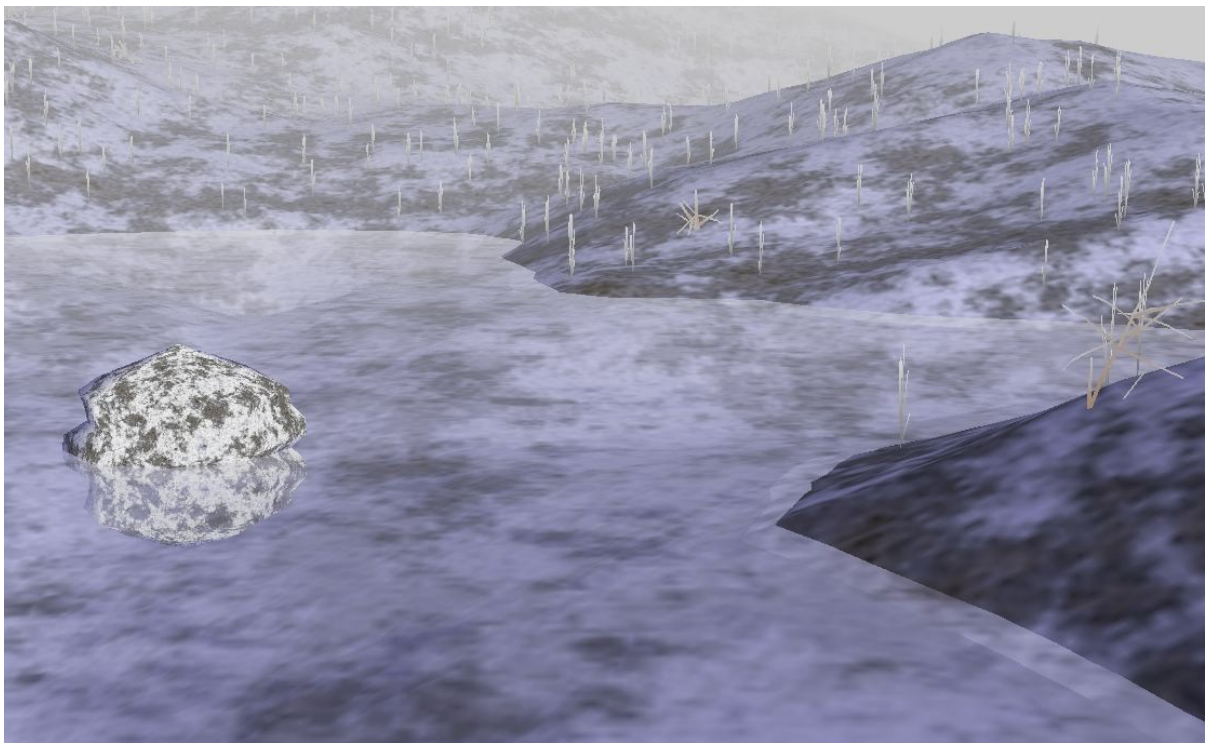
4.5 Křoví

Křoví je další grafický objekt. Křoví v intru je vygenerované pokaždé jinak s ohledem na nastavené parametry. Tyto parametry, mimo jiné, jsou: počet větví a počet kloubů (bodů) na větvi. Křoví je tedy složeno z množiny větví. Větve jsou generovány tak, že každá další generovaná větev začíná na některé z předcházejících větví. Větev je generována mezi dvěma body. První leží na již vygenerované větvi a druhý leží někde v okolí. Při procesu generování větve jsou použity dva jednorozměrné šumy vygenerované funkcí `fractal_Generate`. První z nich je použit jako vzdálenost kloubu (bodu na větvi) od úsečky spojující začátek a konec větve. Druhý je použit jako úhel natočení kolem stejné úsečky. Jinými slovy – pokud bychom kolem úsečky vytvořili několik kružnic (podle počtu bodů na větvi) s různým poloměrem (první šum), na kružnicích vyznačili jeden bod a tyto kružnice vůči sobě pootočili (druhý šum), spojením vyznačených bodů vznikne větev. Na větev je také aplikována gravitace. Čím dále od počátku větve se bod na větvi nachází, tím více je ovlivněn gravitací – je posunut dolů. Pomocí stejného algoritmu by bylo možné generovat například blesky při bouřce.

Obrázek 4.5Vygenerované křoví

4.6 Led

Další ze zimních objektů je led. Led je použit pro vyplnění prohlubní v terénů – v místech, kde by byla v létě voda horských ples. Led je složen ze dvou poloprůhledných vrstev. Horní vrstva má texturu pro náznak nerovností. Je pro ni použita stejná textura jako pro terén. Textura je však několikanásobně roztažená. Dolní vrstva je zde pro druhou stranu ledu a díky ní lze rozeznat tloušťka ledu u břehu. Pro průhlednost je použit takzvaný „blending“. Jedná se o metodu míchání barvy na daném pixelu. Výsledná barva je spočítána z již vykreslené barvy a barvy nově nanášené. Blending se musí explicitně povolit funkcí `glEnable` s parametrem `GL_BLEND`. Pro nastavení míchání se používá funkce `glBlendFunc`, která má dva parametry. Oba lze nastavit na řadu hodnot, a tak lze pomocí blendingu dosáhnout velmi různých výsledků. Od prosté průhlednosti k barevným filtrům. Některé kombinace parametrů mají docela neočekávaný výsledek.



Obrázek 4.6Poloprůhledný led

4.7 Částicový systém

Částicový systém [7] slouží k modelování grafických nebo fyzikálních objektů, které by se pomocí jiných metod jen obtížně modelovaly. Částicový systém je soubor prvků (částic), které se chovají podle určitých pravidel. Každá částice má svoje vlastnosti a svoje počáteční podmínky. Dále pak dobu života, která určuje, kdy daná částice zanikne a kdy se místo ní vytvoří nová. U částicových systémů bývá definován i takzvaný emitor. Emitor je objekt, který produkuje částice. Emitorem může být i samotná částice. Částicový systém je v projektu použit pro několik objektů. Je to pro modelování sněžení, pro pád jedniček a nul a pro vytvoření trávy.

4.7.1 Sněžení

Sněžení je v projektu použito pro zvýšení pocitu zimy. Sněžení je modelováno jako částicový systém, kde částice je jedna vločka. Každá vločka má několik vlastností. Svoji aktuální pozici (třírozměrný vektor) – využívá se též pro vykreslení. Svoji aktuální rychlost (třírozměrný vektor). Svoji velikost. Vykreslení vločky je řízeno funkcí `glBegin` s parametrem `GL_POINTS`. Jelikož je bod vykreslený touto funkcí pořád stejně veliký nezávisle na vzdálenosti je pro vyjádření hloubky každé vločce

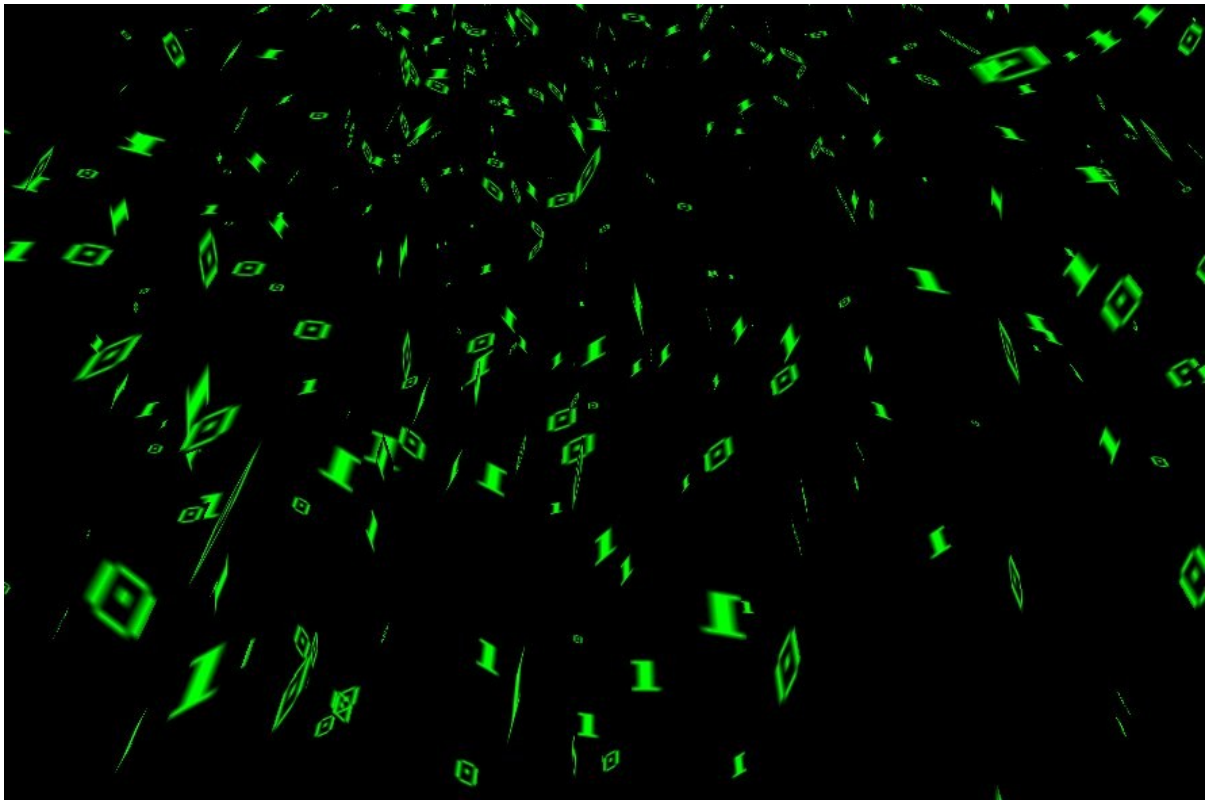
nastavená velikost. V celém systému sněžení jsou definovány hranice. Když tyto hranice vločka překročí, končí život této vločky a místo ní je generována nová. Další pravidlo pro vločku je výpočet dráhy. Na vločku působí gravitace. Nejprve je, při kroku sněžení, připočítána k pozici vločky její rychlost vynásobená časem. Poté je k rychlosti připočítáno časem vynásobené gravitační zrychlení. Následně je vločce připočítána náhodná rychlost. Náhodná rychlost je zde proto, aby vločky nepadaly kolmo dolů. Vločka je ve skutečném světě ovlivňována i sebemenším zavanutím větru. Náhodná rychlost v náhodném směru toto zjednodušuje. Emitor je zde definován jako kruh o poloměru daném hustotou sněžení a počtem částic a nachází se v horní části objektu sněžení. Vločky jsou generovány v bodu ležícím uvnitř kruhu. Objekt sněžení je kvádr. Tento kvádr se pohybuje společně s kamerou, a tak není nutné, aby se generovalo velké množství vloček pro celou scénu.

Obrázek 4.7Částicový systém sněžení

4.7.2 Pád jedniček a nul

Pád jedniček a nul má symbolický význam přístupu k neinicializované paměti. Stejně jako u sněžení i tady se jedná o částicový systém. Částice zde je jeden znak. Obsahuje pozici a rychlost (třírozměrný vektor). Dále pak znak (jedničku nebo nulu) a svoje natočení kolem osy y. Částice končí svůj život pokud její výška překročí nastavenou hranici. Výpočet pohybu částice je stejný jako u výpočtu

pohybu částice vločky u sněžení. Rozdíl je jen v tom, že se nepřipočítává náhodná rychlost a tak znak padá po svislé přímce. Emitor je kvádr. Tento kvádr znaky nemohou opustit. Kvádr je obalený černou schránkou.

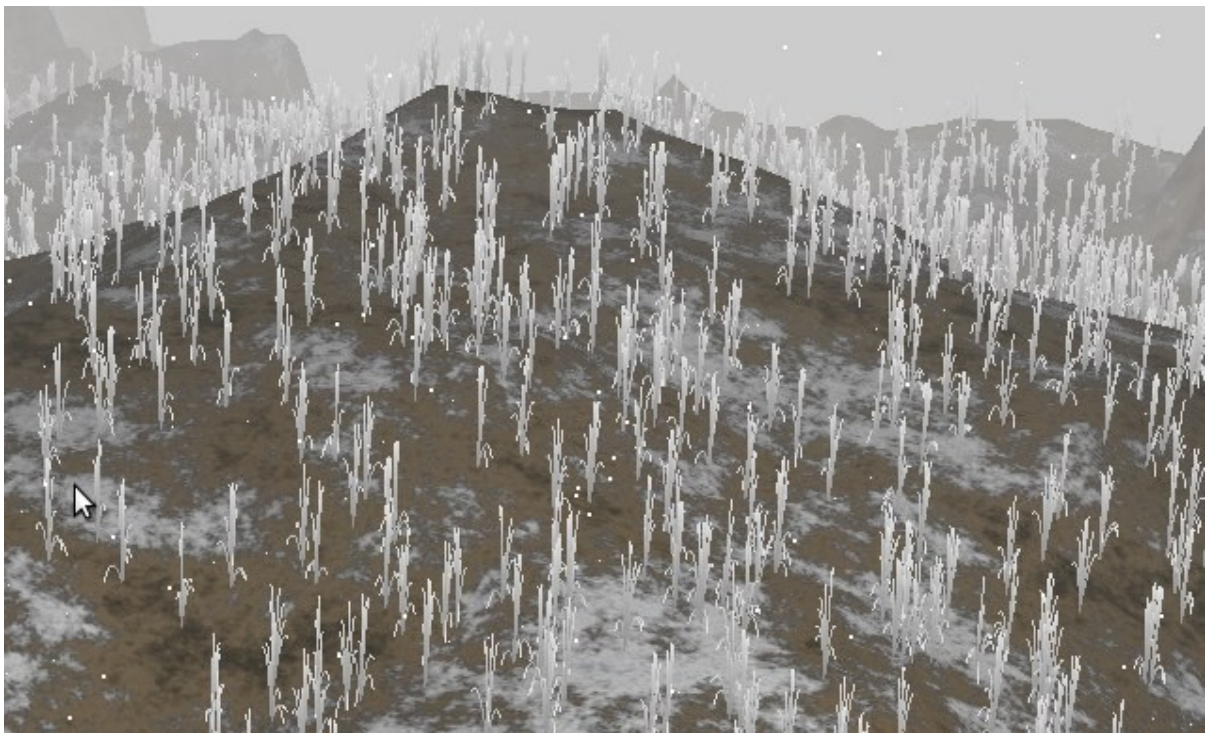


Obrázek 4.8Částicový systém - pád jedniček a nul – symbolika přístupu k neinicializované paměti

4.7.3 Tráva

Poslední použitý částicový systém je využit pro vygenerování trávy. Tráva samotná už s částicovým systémem nemá nic společného. Způsob jejího vytváření lze ovšem také zahrnout do této skupiny. Emitor u generování trávy je bod, kde tráva roste. Z tohoto emitoru vyrůstají výhonky. Výhonek má opět svoji rychlost a pozici. Ze začátku je rychlost růstu výhonku nastavena tak, aby směřovala opačným směrem než gravitace. Podle nastavení počáteční rychlosti se mění výsledný vzhled výhonku. Poté je několikrát spočítána nová pozice konce výhonku. Výhonek je reprezentován jeho dráhou - od jeho vytvoření v emitoru až po jeho konec. Množina výhonků je tráva. U kořene je barva trávy tmavě šedá. Na konci výhonku je barva bílá. Toto barevné nastavení symbolizuje skutečnost, že je tráva zmrzlá. V intru je vygenerováno několik trsů trávy. Jsou rozmístěny po terénu a to tak, aby se objevovaly jen na rovných místech. Proto se použije první vygenerovaná výšková mapa s nastaveným

větším vyhlazováním a tráva se generuje jen na místech, kde se po sečtení výškových map neobjeví skály. Také se tráva negeneruje pod ledem.



Obrázek 4.9Tráva

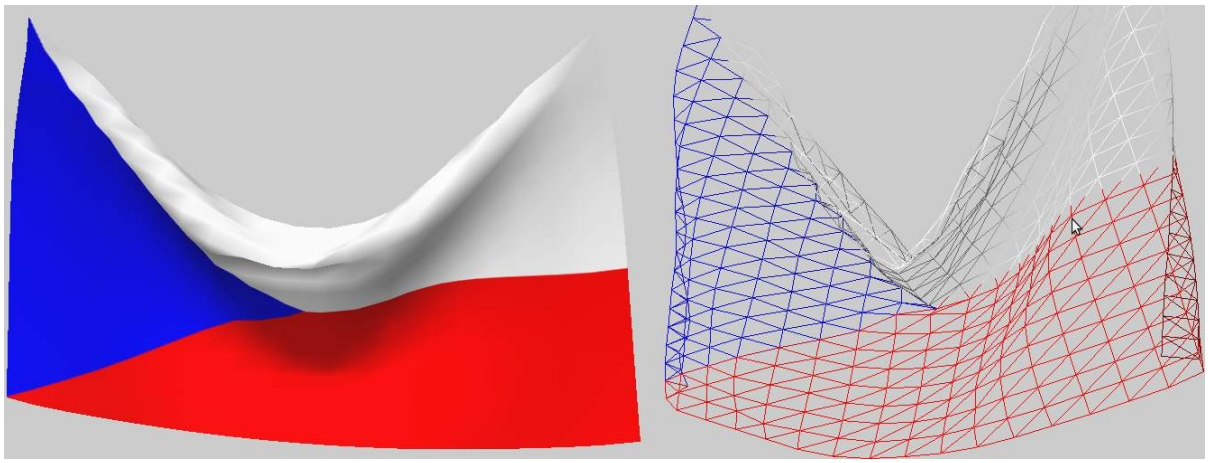
4.8 Celulární automat

Celulární automat[10] je množina buněk s určitým stavem. Chování buňky je určeno jen jejím okolím a stavem buňky. Jednotlivé buňky jsou identické. Celulární automat je použit pro simulaci vlajky.

4.8.1 Vlajka

V intru je možné spatřit i rozpohybovanou vlajku. Tato vlajka se vlní a pohybuje. Nejedná se však o předem nastavený pohyb. Tento pohyb je průběžně vypočítáván. Pro namodelování vlajky byl použit celulární automat. Vlajka je složena z matice bodů. Tyto body reprezentují buňky celulárního automatu. Každá buňka má několik vlastností. Tyto vlastnosti jsou: pozice buňky v prostoru, rychlost buňky v prostoru (třírozměrný vektor), a příznak static, jehož význam spočívá v tom, zda je daná buňka ukotvená (nepohybuje se). Vlajka má definované ještě další vlastnosti a to hmotnost a elastičnost (čím je elastičnost vyšší, tím je vlajka více povislá). Buňky vlajky mají mezi sebou vztah. Tento vztah je síla, která se snaží udržet buňky v definované počáteční vzdálenosti. Pokud jsou dvě

sousedící buňky od sebe dále než je jejich definovaná standardní vzdálenost, působí na ně síla, která je tlačí k sobě. Pokud jsou blíže, působí na ně síla, která je odpuzuje. Tato síla je přímo úměrná rozdílu vzdáleností a nepřímo úměrná elastičnosti vlajky. Postup výpočtu dalšího kroku je následující. Nejprve se k pozicím buněk, které nemají nastavený příznak static připočte jejich rychlost vynásobená časem. Poté se vypočtou nové rychlosti těmto buňkám. Při výpočtu nové rychlosti se nejprve musí spočítat síla, která na danou buňku působí. Tato síla je spočítána jako součet všech sil působících mezi ní a jejími sousedy. Za sousední buňky považujeme osm nejbližších buněk čtvercové sítě. Síla děleno hmotnost nám udává zrychlení. Zrychlení krát čas nám udává rychlost. Aby se pohyb vlajky někdy zastavil, musíme brát v úvahu i ztráty. Proto výslednou rychlost ještě zmenšíme o ztráty.

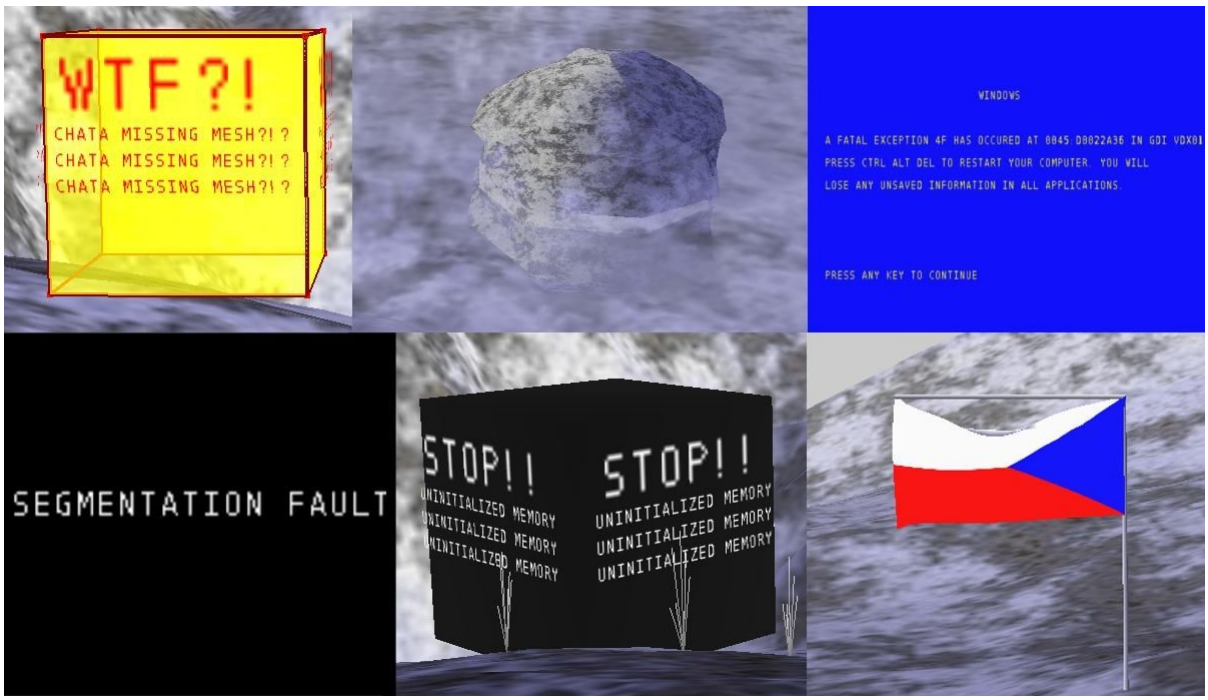


Obrázek 4.10Celulární automat – vlajka a její drátěný model

4.9 Další grafické objekty

V projektu je možné spatřit i další objekty. Jedná se o objekty nutné pro příběhovou linii a objekty doplňkové. Jsou to objekty: model chybějící horské chaty, model takzvané „modré smrti“ často spojované s pádem operačního systému Windows, model neinicializované paměti, model vlajky s tyčí, na které vlaje, model chyby přístupu do paměti a model animovaného kamene.

Textura na animovaném kameni je trojrozměrná. Efekt pohybu povrchu je zařízen transformací texturovací matice. Matice je transformována v čase. Kromě modelu kamene je vykreslen i jeho odraz v ledě. Pomocí jednoduché transformace (nastavení záporného měřítka na ose y) je kámen překlopen a poté vykreslen.



Obrázek 4.11Další grafické objekty

4.10 Převedení grafického objektu na jednotnou strukturu

Aby se k objektům jako je tráva a sněžení přistupovalo jednotně, je třeba je převést na společnou strukturu. Tato struktura se jmenuje „`SModel`“ a nachází se v knihovně „`model.h`“. Model je možné vykreslit, vypočítat jeho nový stav (krokovat jej podle času), nastavit jeho pozici, posunout, změnit měřítko, natočit. Také lze model jednotně uvolnit z paměti. Nastavit příznak pro vykreslování nebo krokování (například pokud už model není třeba vykreslovat, neboť jej již není potřeba). Toto je zajištěno ukazateli na funkce pro vykreslení, krokování, a uvolnění z paměti. Data samotná jsou uložena jako ukazatel na `void`. Model obsahuje strukturu reprezentující jeho ortogonální obálku („`SCover`“ v knihovně „`cover.h`“) a strukturu pro jeho transformace v trojrozměrném prostoru („`STransform`“ v knihovně „`transform.h`“). Všechny grafické objekty ve scéně jsou uloženy prostřednictvím modelu. Funkce pro práci s modelem jsou implementovány tak, aby je bylo možné použít jako parametr funkce pro mapovací funkci. Takže pokud bychom si grafické modely uložili do seznamu, stačilo by pro jejich manipulaci zavolat jen jednu mapovací funkci.

5 ŘÍZENÍ INTRA

Aby intro po spuštění něco provádělo, je potřeba jej rozpohybovat. Nastává ovšem otázka:, kdy, co a jak rozpohybovat. Další problém, který je třeba řešit, je konstantní rychlost animace na rozdílných počítačích.

5.1 Časování

Základem je oddělení výpočtu následujícího stavu objektu od jeho vykreslení. V projektu je to zařízeno pomocí dvou funkcí u grafického modelu – vykreslením (funkce `model_MapDraw`) a krokováním (funkce `model_MapStep`). Vykreslení modelu se děje pokud je to možné. Tedy co nejčastěji. Vypočítání následujícího stavu modelu se však odehrává jen někdy. A to v časovačem definovaném čase.

Časovač vyvolává svoji obslužnou funkci periodicky. V obslužné funkci se vypočítávají nové stavy modelů – volá se pro ně funkce `model_MapStep`, které je, kromě odkazu na model, předána i konstantní doba mezi jednotlivými obsluhami. V projektu je obsluha vyvolána vždy po deseti milisekundách. Mimo obsluhu probíhá vykreslování. Při vykonávání obsluhy se také zvýší aktuální čas. Tento čas je poté použit pro kalendář událostí.

5.2 Kalendář událostí

Kalendář událostí [10] je podle času události vzestupně seřazený seznam událostí. Kalendář se v projektu používá pro přepnutí vykreslování grafických objektů, pro změnu pohybu kamery a pro změnu scény.

Je implementován pomocí lineárního seznamu v knihovně „`calendar.h`“. Prvek lineárního seznamu obsahuje čas, kdy se má daná akce pro událost spustit a ukazatel na funkci, která akci reprezentuje. Do kalendáře je možné události vkládat pomocí funkce `calendar_Insert`. Funkci, která vykonává událost, je předán ukazatel na samotný seznam. Tím se zajistí možnost, aby při vykonávání akce bylo možné vygenerovat další události a uložit je do kalendáře. Kalendář obsahuje funkci `calendar_Action` pro vykonání akcí. Funkci je předáván aktuální čas. Při jejím volání se pak postupně vykonají všechny akce u událostí, které mají aktivační čas menší nebo roven aktuálnímu času. Vzhledem k tom, že je seznam událostí seřazen postačí projít jen ty události, u kterých již aktivační čas nastal. Poté, co se vykoná akce, je událost ze seznamu odstraněna. Před

spuštěním intra se kalendář inicializuje a vloží se do něj potřebné události. Funkce `calendar_Action` je volána vždy při obsluze časovače.

Čas události v kalendáři událostí je struktura obsahující počet milisekund. Struktura se nachází v knihovně „`mytime.h`“. V knihovně jsou implementovány funkce pro sčítání odčítání času. Pro jeho násobení. Pro jeho vytvoření (ze zadaného počtu hodin, minut, vteřin a milisekund) a pro jeho porovnávání.

Aby se grafické objekty zobrazovaly postupně, jsou nastavovány jejich příznaky pro vykreslení pomocí událostí v kalendáři. Stejně tak jsou nastavovány příznaky pro krokování, aby se program urychlil, pokud není potřeba počítat další krok daného modelu.

6 ZÁVĚR

Při implementaci projektu jsem získal mnoho zkušeností v oblasti počítačové grafiky. Také jsem se lépe naučil organizovat velký projekt. Naučil jsem se jak pracovat s OpenGL. Naučil jsem se, jak procedurálně generovat grafické objekty. Získal jsem znalosti pro tvorbu velikostně limitovaných programů, implementaci jednoduchých částicových systémů a celulárních automatů. Snažil jsem se implementovat algoritmy tak, aby byly znovupoužitelné a aby byly co nejobecnější. Při tvorbě programu jsem dbal na zásady programování (jako jsou komentáře, strukturování projektu,...). Zjistil jsem, že projekt nemusí nutně znamenat jen práci, ale i zábavu. Program zkomprimovaný pomocí exe packeru UPX nepřesáhl 40kB a tak zbylo místo pro další rozšíření,

Největší problémy jsem měl s přeprogramováním částí pro zobrazení okna pod operačním systémem Windows a podporou trojrozměrných textury pod tímto systémem. Pokud se použije pro překlad intra starší knihovna OpenGL (verze starší než 1.2) je podpora trojrozměrných textur řešena pomocí grafického rozšíření. Další problémy se vyskytly při časování. Nemalý problém nastal při simulování pohybu vlajky. Vlajka se při špatně nastavených parametrech rozkmitala a rozpadla se. Velkou část času jsem strávil nad implementováním algoritmu pro generování N rozměrného šumu. Nicméně se snaha vyplatila a algoritmus je dostatečně obecný a rychlý, a proto by měl být znovu využitelný.

Možných rozšíření pro budoucí pokračování v projektu je mnoho: vytvoření různých grafických objektů, přidání fyziky a kolizí objektů, vylepšení osvětlení a texturování, přidání statických a dynamických stínů, přidání optimalizace pro zobrazení, vylepšení zobrazení textů a další. Jiné rozšíření by spočívalo v přidání zvuků a hudby. U tohoto projektu jsem si sice stačil napsat vlastní hudbu, ale nestihl jsem implementovat funkce pro přehrávání a především synchronizaci obrazu s rytmem skladby.

Literatura

[1] Stránky WWW - NeHe OpenGL Tutoriály - Lekce 1 - Vytvoření OpenGL okna ve Windows
http://nehe.ceske-hry.cz/tut_01.php
[dostupné 7.5.2010]

[2] Stránky WWW - NeHe OpenGL Tutoriály - Lekce 6 - Textury
http://nehe.ceske-hry.cz/tut_06.php
[dostupné 7.5.2010]

[3] Stránky WWW - NeHe OpenGL Tutoriály - Lekce 8 - Blending
http://nehe.ceske-hry.cz/tut_08.php
[dostupné 7.5.2010]

[4] Stránky WWW - NeHe OpenGL Tutoriály - Lekce 12 - Display list
http://nehe.ceske-hry.cz/tut_12.php
[dostupné 7.5.2010]

[5] Stránky WWW - NeHe OpenGL Tutoriály - Lekce 16 - Mlha
http://nehe.ceske-hry.cz/tut_16.php
[dostupné 7.5.2010]

[6] Stránky WWW - NeHe OpenGL Tutoriály - Lekce 17 - 2D fonty z textur
http://nehe.ceske-hry.cz/tut_17.php
[dostupné 7.5.2010]

[7] Stránky WWW - NeHe OpenGL Tutoriály - Lekce 19 - Částicové systémy
http://nehe.ceske-hry.cz/tut_19.php
[dostupné 7.5.2010]

[8] Stránky WWW - NeHe OpenGL Tutoriály - Lekce 34 - Generování terénů a krajin za použití výškového mapování textur
http://nehe.ceske-hry.cz/tut_34.php
[dostupné 7.5.2010]

[9] Stránky WWW - NeHe OpenGL Tutoriály - Lekce 19 - Fyzikální simulace lana
http://nehe.ceske-hry.cz/tut_40.php
[dostupné 7.5.2010]

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

[11] Stránky WWW – Gimp
http://www.gimp.org/
[dostupné 7.5.2010]

[12] Stránky WWW – OpenGL
http://www.opengl.org/
[dostupné 7.5.2010]

[13] Ing. Přemysl Kršek, Ph.D.: Základy počítačové grafiky – IZG – Studijní opora, Brno, FIT VUT verze 0.9.

[14] WWW odkaz – zdrojový kód glibc pro ranr_r v stdlib/rand_r.c.
http://ftp.gnu.org/gnu/glibc/glibc-2.9.tar.gz
[dostupné 7.5.2010]

[15] Stránky WWW – Jazyk Python
http://www.python.org/
[dostupné 7.5.2010]

[16] Stránky WWW – UPX
http://upx.sourceforge.net/
[dostupné 7.5.2010]

[17] Stránky WWW – editor Vim
http://www.vim.org/
[dostupné 7.5.2010]

[18] Stránky WWW – msdn
http://msdn.microsoft.com/cs-cz/default.aspx
[dostupné 7.5.2010]

[19] Stránky WWW – makefile pro velké projekty
http://kengine.sourceforge.net/tutorial/g/makefile.htm
[dostupné 7.5.2010]

[20] Stránky WWW – SDL
http://www.libsdl.org/
[dostupné 7.5.2010]

[21] Stránky WWW – česká demoscéna
http://www.scene.cz/
[dostupné 7.5.2010]

Seznam příloh

Příloha 1. CD