Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

BAKALÁŘSKÁ PRÁCE

Plzeň, 2008

Pavel Filipčík

Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Bakalářská práce

Simulace sněžení pro virtuální realitu

Plzeň, 2008 Pavel Filipčík

Abstract

Simulation of snowing for virtual reality

These study is focused on atmospheric effect of snowfall and its simulation in VR applications. In the first part of this study there is described the evolution of snow flakes, and factors, that play the biggest role in process of snow flake forming. Next we describe the most important methods for computer graphics, how we can make good looking snowing. Then we apprise of programming techniques, which are necessary to rendering effect snow. The second part is oriented on implementation effect snow in programming language C# with the help of libraries from DirectX 9.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne *30.4.2008, Pavel Filipčík,*

Obsah

1. Úvod

Aby byl uživatel aplikace - 3D hry - co nejvíce zaujat, snaží se autoři her, hlavně grafici, napodobit co nejpřesvědčivěji reálný svět. Vzpomeňme na časy, kdy 3D hry byly v prvopočátcích. Každý objekt připomínal čtverec. Se stále se zdokonalujícím hardwarovým vybavením bylo možno celkový počet objektů ve scéně zvyšovat a zároveň tyto objekty skládat z většího množství bodů a kvalitněji tak napodobovat skutečné předměty.

Postupem času se stalo opravdovou výzvou simulování atmosférických efektů v real-time aplikacích, protože se jedná o velmi hardwarově náročné efekty. Můžeme sem zařadit sněžení, déšť, nebo mraky. Pokud ve skutečnosti začne sněžit, celkový počet vloček, který vidíme, je těžko spočítatelný, ale je zřejmé, že se jedná o hodnoty přesahující miliony vloček. To je do dnešní doby pro výkon počítačů neřešitelný problém. Proto bylo nutné vymyslet algoritmy, které po stránce vizualizace budou simulovat sníh, a zároveň budou aplikovatelné do real-time aplikací. Je důležité, aby tyto algoritmy byly co nejrychlejší, jelikož výkonu je potřeba i na další obtížné úlohy, jako je ořezávání objektů, detekce kolizí, osvětlení scény a v neposlední řadě aplikace matematických a fyzických zákonů pro lepší simulace vzhledem ke skutečné realitě.

1.1. Cíl práce

Cílem práce je prostudovat reálné chování padajících sněhových vloček a jeho simulaci pro potřeby real-time grafiky. Dále navrhnout a implementovat vhodné metody pro pohyb vloček, pro jejich zobrazení a rychlé vykreslení. Implementovaný efekt otestovat a zdokumentovat použité metody.

1.2. Popis kapitol

Tato práce obsahuje teoretickou a praktickou část. Teoretická část vysvětluje, jak vůbec vzniká sněžení. Dále se zaobírá vykreslováním sněhu v grafice. V praktické části popisuji vlastní implementaci metod potřebných pro vytváření sněhu.

2. Teoretická část

Na následujících stránkách budu popisovat vznik sněžení, druhy sněhových vloček a formy sněhových přeháněk. Podrobně se zaměřím hlavně na vznik sněhové vločky, a faktory ovlivňující její vzhled.

2.1. Sníh

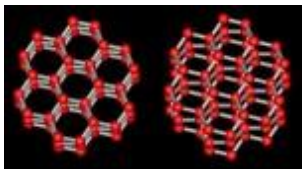
Sníh vzniká v troposféře, což je nejspodnější vrstva zemské atmosféry. Její šířka je kolem 1 km a pravidelně se mění, v letním období se stává širší. Zem obklopuje ve výšce 8-9 kilometrů na pólech, kde je nejníže, směrem k rovníku její výška roste až k 17 kilometrům. V troposféře se vytváří veškeré počasí. Vertikální teplotní gradient troposféry je -0,6. Gradient označuje změnu hodnoty veličiny na jednotku vzdálenosti, v meteorologii se nejčastěji používá stupeň Celsia na 100 metrů vzdálenosti. Teplota tedy v troposféře lineárně klesá s výškou. Troposféra obsahuje veškerou atmosférickou vlhkost. Počasí je ovlivňováno také prvními 30 km další vrstvy zemské atmosféry - stratosféry. Ta má celkovou šířku kolem 50 km, a je zcela bez vlhkosti. Ve stratosféře je vertikální teplotní gradient kladný, tedy teplota zde stoupá, až k -3 stupňům Celsia. Přechod mezi troposférou a stratosférou je označován jako tropopauza, a lze ji považovat za další vrstvu. Dochází zde k ustálení poklesu teploty na výšce, ta se pohybuje kolem -50 stupňů Celsia na pólech a až -85 nad rovníkem.

Sníh je tvořen komplexním systémem sněhových vloček. Sníh je typickým představitelem hydrometeoru, což je atmosférický jev, který je tvořený skupinou zmrzlých ledových krystalků.

2.1.1. Sněhová vločka

Každá sněhová vločka je vlastně zmrzlá voda, tedy tuhé skupenství vody. Správně by se měla nazývat ledový dendrit. Sněhové vločky jsou shluky krystalků, a zároveň si udržují tvar šesticípé hvězdy. Tvar dendritů je dán fyzikální strukturou molekul vody a jejich vazbami v krystalové hexagonální mřížce ledu obrázek 2.1.1. Dendrity vznikají sublimací vodních par na krystalická jádra. Tyto krystalická jádra

mají v každém svém vrcholu 6 molekul vody, které jsou spojeny s dalšími molekulami, které
jsou poblíž. Sněhová vločka vzniká pomocí mikroskopického zrnka prachu, na který se začnou nabalovat krystalky ledu.



Obrázek 2.1.1: Krystalová hexagonální mřížka ledu, obrázek pochází z [7].

Ještě donedávna se vědci přeli, jestli mohou z nebe spadnout dvě totožné vločky. Dodnes se důvěryhodně nepodařilo prokázat výskyt takových vloček, proto se zastává názor, že výskyt dvou stejných vloček je značně nepravděpodobný. Na vznik vločky totiž působí velké množství faktorů, mezi nejdůležitější patří vlhkost vzduchu a teplota, a zároveň předchozí vývoj a stav těchto veličin. Navíc vločka se formuje po celou dráhu letu, protože se teplota neustále mění. Již v roce 1611 se tvarem sněhových vloček zabýval Johanes Kepler. Dalším badatelem jejich tvaru byl René Descartes. Oba vypozorovali, že si každá vločka drží tvar šesticípé hvězdy. Johanes Kepler se domníval, že je to dáno způsobem, jak se v prostoru skládají tělesa. Zástupnými metodami dospěl k názoru, že se šestihranné kvádry dají perfektně skládat v prostoru. To by vysvětlovalo i skladbu včelích pláství s medem.

Způsobů, jak vytvořit lišící se sněhové vločky, je neskutečné množství. Krystalografie je věda, která se zabývá tímto problémem. Uvedu příklad. Pokud máme 20 aut, a chceme je zaparkovat za sebe, na první místo můžeme zaparkovat všechny auta, tedy 20 možností. Na druhé 19 aut a tak dále. Pokud vynásobíme možnosti jen pro prvních pět parkovacích míst, dostaneme se na 28 milionů možností, jak auta zaparkovat. Dodnes bylo popsáno přes 21 tisíc základních tvarů, ze kterých je možné vločky sestavovat, a to se nebere v úvahu, že každý z těchto tvarů může vzniknout s různou délkou nebo tloušťkou. Navíc každá vločka obsahuje průměrně 10^{18} molekul, které se také formují neskutečným počtem kombinací během celého letu vločky. Pokud tedy chceme sestavit sněhovou vločku, máme na výběr ohromné množství kombinací, blížící se nekonečnu. Jen pro zajímavost uvedu, že první člověk, komu se podařilo vytvořit sněhovou vločku v laboratorních podmínkách, byl japonský vědec Ukičiro Nakaja, blíže popsáno [2].

Zde je přehled základních krystalků pro tvorbu sněhové vločky:

- jehlička – vznik za teploty od -4 do -8 stupňů Celsia.
- šestiboká destička: při teplotách od -8 do -12 stupňů Celsia.
- šesticípá hvězdice: při teplotách od -12 do -18 stupňů Celsia.
- prostorové šesticípé hvězdice: při teplotách od -18 do -25 stupňů Celsia.
- šestiboký sloupek: při teplotách od -25 do -40 stupňů Celsia.

Obrázek 2.1.2: Jehlicový, destičkový krystalek, hvězdice a poslední obrázek vzácný výskyt hranolu, který má na koncích destičku. Pochází z [14].

Pokud máme štěstí, můžeme spatřit i vločku, kterou tvoří hranolek, který je na obou koncích zakončený destičkou obrázek 2.1.2.

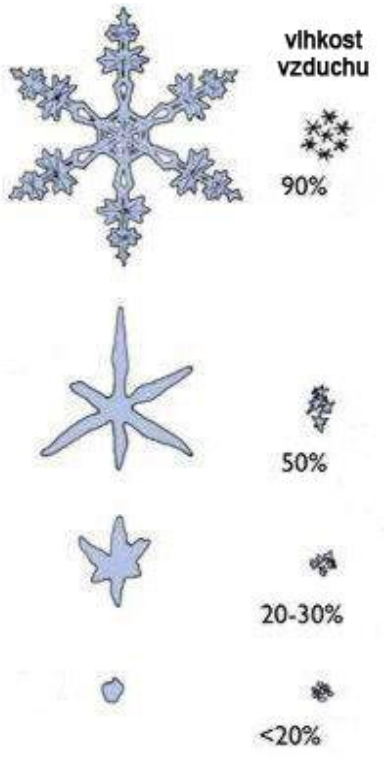
Zbývá otázka, do jakých rozměrů mohou vločky narůst. Průměrná velikost nově vznikajících krystalků se pohybuje mezi 0,005 až několika milimetrů a tloušťka si zachovává 1/10 průměru vločky. Podle teploty dochází k jejich spojování po celou dobu letu vločky. Nejčastěji dorůstají do velikosti 1 až 5 centimetrů. Největší dosud nalezenou vločkou, která je i zapsána v Guinessově knize rekordů je podle [5] vločka, která byla nalezena v Montaně v USA v roce 1887 a měla rozměry 381 milimetrů na šířku a 232 milimetrů na výšku.

2.1.2. Druhy sněhových srážek

Vlhkost ve vzduchu ovlivňuje, jak se budou sněhové vločky formovat. Podle jejich tvaru, obrázek 2.1.2, existuje několik druhů sněhových srážek [1]:

- Sníh – Tvořen ledovými krystalky skládaných do hvězdic. Pokrývá rozsáhlé území, je dlouhotrvající.
- Sněhová přeháňka – Je tvořena z šesticípých prostorových hvězdic. Vyskytuje se náhle a netrvá moc dlouho, pokrývá malé území. Je zde pozorovatelný úbytek viditelnosti v dosahu přeháňky.
- Déšť se sněhem – Pokud sněží a zároveň prší.

- Sněhové krupky – Jejich výskyt je nejčastěji se sněhovou přeháňkou. Krupky jsou ledová zrna, která jsou neprůsvitná, často mají kulovitý, popřípadě kuželovitý tvar o průměru 2 – 5 milimetrů. Při dopadu na zem odskakují nebo se tříští.
- Sněhová zrna – Jako sněhové krupky, jsou to ledová zrna také neprůsvitná, ale jejich tvar je zploštělý nebo protáhlý a bývají tenčí jak 1 mm. Při dopadu na zem neodskakují a ani se netříští.
- Zmrzlý déšť – Vzniká, pokud vločky při letu roztají a znovu zamrznou. Může se taký jednat o zamrzlé dešťové kapky. Výsledkem jsou průhledná ledová zrna, bývají nepravidelných tvarů, vzácně kuželovitá a mají průměr do 5 mm.
- Námrazové krupky – Vyskytují se sněhovou přeháňkou a jedná se o sněhové krupky, které jsou na povrchu pokryté vrstvou ledu. Mají kulovitý tvar a průměr okolo 5 mm.



Obrázek 2.1.2: Závislost tvaru vločky na vlhkosti vzduchu.

Sněhová přeháňka vyvolává dojem nejhustějšího sněžení, a je to způsobeno celkově největšími vločkami, protože k jejich nejčastějšímu řetězení dochází během teplot blízkých nule. S klesající teplotou klesne i vlhkost vzduchu, a sněhové vločky se stávají menšími. Sněhová přeháňka za nízkých teplot se nevyskytuje tak často. Pokud se vyskytne, neobsahuje sněhové vločky formované do hvězdic. Jsou to pouze ve vzduchu zvířené jemné krystaly ledu ve tvaru jehliček o velmi malých rozměrech. Často jsou nazývány diamantovým prachem, protože se ve slunečním světle lesknou jako diamanty.

2.1.3. Síly působící na částice

Pokud se snažíme nasimulovat efekt probíhající na zemském povrchu, neměli bychom zapomínat na globální síly, které budou ovlivňovat jednotlivé částice. Pokud uvažujeme trajektorii jednotlivé sněhové vločky, bude tvořena pomocí gravitační síly, která působí vždy, dále pak silou odporu prostředí, nejčastěji odporem vzduchu, dále tuto trajektorii bude narušovat vliv větru působící v různých směrech, a také další větrné proudy a víry. Nesmíme zapomínat, že i když všechny tyto síly budou působit ve stejném směru a stejnou velikostí, nemusí to ještě nutně znamenat, že se dvě sněhové vločky budou pohybovat po stejných, nebo alespoň podobných drahách. Ptáte se proč? Jak jsem naznačil v kapitole 2.1.1, žádné dvě vločky nejsou stejné. Každá se liší svou strukturou, to znamená, že při aplikaci síly se jedna může začít otáčet, jiná houpat, kolísat, nebo prostě volně padat. Spočítat, jak se každá vločka zachová, je prakticky nemožné i ve skutečném světě, natož to počítat v aplikaci a plýtvat tak drahocenným výkonem.

Počítat pro každou vločku pouze gravitační sílu, znamená složitost výpočtu *O(n),* kde *n* je počet vloček v částicovém systému 2.3.1. Gravitační síla uděluje vločce zrychlení za jednotku času, spočítá se $\mathbf{F_g} = m\,\mathbf{g}$, kde *g* je vektor směřující do středu Země a má nezávisle na výšce vločky stejnou velikost, a to gravitační konstanty, *m* je hmotnost vločky. Gravitační síla ani její směr nezávisí na pozici a poloze sněhové vločky. Pokud bychom se spokojili s tímto jedním vzorcem, rychlost vloček by se neustále zvětšovala. To je ale nemožné, díky odporu vzduchu, který působí také na každou vločku. Proto musíme zavést sílu označovanou jako odpor prostředí, ten bude růst lineárně s rychlostí vločky a bude působit opačným směrem, než je její rychlost. Vztah $\mathbf{F_d} = -k_d\,\mathbf{v}$ vyjadřuje tuto sílu, kde k_d je koeficient odporu prostředí, pro vločku to koeficient odporu vzduchu, $\mathbf{F_d}$ je síla, působící proti směru pohybu vločky a **v** je rychlost vločky.

Pokud budeme brát v úvahu další síly působící na vločku, budeme muset mít pouze omezené množství vloček, pro které se budou všechny tyto síly započítávat. Ve většině dnešních her jsou tyto hodnoty nastaveny implicitně.

2.2. Možnosti vytváření sněhu

Způsobů, jak přenést sníh na obrazovky monitorů, není mnoho. Nejvyužívanější a skoro jedinou dobře použitelnou volbou pro správu pohybu

velkého množství malých částic je použití částicového systému, kapitola 2.3.1. Toto použití se pak liší podle toho, zda se sníh bude generovat jako posloupnost po sobě jdoucích statických obrázků, nebo zda bude generován jako dynamická animace v reálném čase.

2.2.1. Vykreslování do textury

Pro generování posloupnosti obrázků se využívá tzv. vykreslování do textury. To je metoda, pomocí které je možné místo vykreslování objektů přímo do scény, tyto objekty vykreslit jakoby do scény, která ale není zobrazena, ale dojde k jejímu uložení do textury. Tato textura se pak následně vhodně umísťuje do scény, která už se vykresluje. Při vykreslování do textury můžeme přemístit kameru na jinou pozici s jiným úhlem záběru, to se využívá, pokud jsou ve scéně zrcadla. Při vytváření sněhu se kamera přemístí do středu souřadného systému a směřuje ve směru osy Z. Za jednotku času se udělá snímek do textury. Ten je následně uložen do paměti, ale může být uložen i na disk, a je vložen přímo před kameru. Tím vznikne dojem, že sněží. Čím více vloček je ve scéně, tím se stává hardwarově náročnější a tato metoda se stává nepoužitelnou. Proto, aby se ušetřil výkon počítače, je ještě před začátkem vykreslování scény vygenerováno několik textur, každá s pohybem vloček v jiném čase. Ty se uloží do paměti a poté se aplikují přepínáním na scénu. Zároveň se mezi přepínáním textur generují další. Po určitém množství uložených textur už je stačí třeba jen přepínat. Vločky se do textury vykreslují buď v jedné a té samé vzdálenosti od kamery a upravuje se jejich velikost, aby se vyvolal dojem, že jsou dále od kamery, nebo jsou snímány 3D a nemusí se starat o jejich velikost, ale nesmíme zapomenout, že výsledný obrázek je pouze dvojrozměrný. Takto je možné dodělat sníh do fotografií, obrázků, nebo i filmů. Tato metoda má výhodu v tom, že při jejím použití není třeba řešit manipulaci s vločkami při pohybu kamery. Ten se simuluje posouváním již vygenerované textury. Nevýhodou je, že pokud bychom chtěli tuto metodu použít do 3D filmů a her, popřípadě na 3D obrázky, bude jasně patrné, že všechny vločky jsou v té samé vzdálenosti od kamery a že jejich místo dopadu je přímka.

2.2.2. Dynamická animace v reálném čase

Pro pohyb vloček je opět používán částicový systém. Tentokráte ale dochází k okamžitému vykreslování scény na obrazovku, a upravování pozic vloček s každým novým snímkem. Aby výsledný obraz simuloval sníh, je potřeba použít velkého množství sněhových vloček, a to sebou nese hardwarovou náročnost. Kvůli tomu je potřeba vykreslovat jen nejnutnější počet vloček, tedy těch, které jsou opravdu vidět. Největší výhodou při použití této metody je celkový vizuální dojem, který nejpřesvědčivěji napodobuje realitu. Nevýhoda je již zmiňovaná náročnost na výkon počítače.

2.3. Programovací techniky pro zobrazení sněhu

V této kapitole podrobně popíši nejnutnější techniky, které je nutné znát k tomu, aby mohl být simulován reálně vypadající sníh s co možná nejlepším výsledným efektem.

2.3.1. Částicový systém

Co to vlastně částicový systém je? Možná je známější pod anglickým označením Particle system. Částicový systém slouží ke správě velkého počtu malých částic, které mají určitou vlastnost a chování, a pomocí shluku těchto částic je možné vytvořit určitý efekt. Pokud automobil narazí do zdi, můžeme simulovat odlétávající úlomky zdi nebo odlétávající kusy auta právě pomocí částicového systému. V dnešních hrách se často využívají například pro tvorbu ohně, kouře, mlhy, explozí, ohňostrojů, a v neposlední řadě i pro tvorbu deště, sněhu nebo mraků.

Jedna částice v částicovém systému je sama o sobě nevýznamná. Aby byl požadovaný efekt co nejvěrohodnější, je potřeba použít většího množství částic, někdy i počtu dosahujícího několika milionů, třeba v případě deště nebo sněžení. Pro lepší představu, co to jedna částice je, ji můžeme přirovnat k jednomu střepu sklenice, která spadla ze stolu a rozbila se. Každý střep je tedy reprezentován jednou částicí v částicovém systému.

Všechny částice si nejčastěji nesou informaci o své pozici v prostoru, směru, kterým se pohybují a také rychlosti, popřípadě barvě, hmotnosti, nebo sil co na ně působí, třeba gravitace, atd. Další důležitou vlastností je doba života jedné částice. Při ohňostroji pozorujeme, že určitý čas po výbuchu rachejtle už není vidět žádný

efekt. Jednoduše kousky výbušniny vyhoří a vyhasnou. Toto můžeme ve hře docílit nastavením doby života částice na požadovaný čas. Po vypršení tohoto času se částice stává nezajímavou a je vymazána ze systému. Všechny vlastnosti částice musí být udržovány a měněny podle aktuálního stavu ve scéně. Tato obnova by měla probíhat co nejčastěji a v co nejkratší době, hlavně v případě real-time aplikace. To je jeden z důvodů, proč vykreslovat pouze částice, které jsou viditelné z úhlu kamery, a zbytečně neplýtvat výkonem například vykreslováním částic, které jsou za kamerou.

O aktualizaci hodnot vlastností částic se stará částicový systém. Ten postupně načítá všechny částice. Nejprve kontroluje dobu života částice, pokud vypršela, částici zahodí a nezdržuje se obnovováním její pozice. Pokud částici zbývá ještě nějaký čas, je tato doba snížena o danou hodnotu – částice stárne. Poté se kontrolují další vlastnosti, a jsou upraveny podle aktuálního stavu. Následně je zjištěno, jestli počet všech částic neklesl pod určitou hranici, a v případě že se tak stane, je nutné tento počet zvýšit na požadovanou úroveň zavoláním metody pro vytvoření nové částice s přednastavenými hodnotami, popřípadě požadovanými. Tento celý cyklus se může opakovat několikrát, v případě exploze, ale také po celou dobu běhu aplikace, když se jedná o sníh, fontánu, déšť, atd. Z toho všeho je patrné, že pokud chceme dosáhnout co největší realističnosti efektu, je vyžadováno dobré hardwarové vybavení.

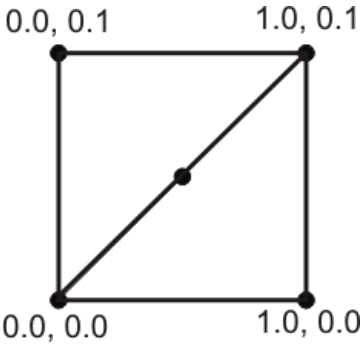
Obrázek 2.3.1: Ukázky použití částicového systému.

Pro úplnost uvedu, že částicový systém je tvořen několika metodami. Obsahuje metodu pro vytvoření částice, dále pro její smazání ze systému. Poté metodu, která zjišťuje zbývající dobu života částice. Další metoda provádí updatování částic a poslední jejich vykreslování. Je doporučeno implementovat metody pro

resetování vlastností všech částic nebo metodu pro resetování vlastnosti jedné částice. Další užitečné metody, nastavují dodatečně vlastnosti, které se použijí na všechny částice stejně, například se často vypne osvětlení, nastaví se alfa míchání, kapitola 2.3.3, poté se nastaví textura, která se aplikuje na částice a další. Ihned po vykreslení všech částic se zavolá metoda, která tyto hodnoty vrátí do původního stavu, tedy opět zapne osvětlení scény, vypne alfa míchání atd. To je například z důvodu, že chceme mít ve scéně osvětlené textury, které nemají používat alfa míchání, a právě u vykreslených tyto hodnoty chceme mít opačně.

2.3.2. Sprite

Částice v částicovém systému mohou mít všechny jednu stejnou texturu, ale častěji ji mají odlišnou, a vhodným překrýváním částic se pomocí průhlednosti formuje požadovaný efekt. K mapování textury se využívá sprite. Je to vlastně plocha, na kterou se namapuje textura - obrázek. Texturu do aplikace musíme namapovat na nějaký objekt. Nejčastěji se bude mapovat na objekt čtverce, protože sama textura je čtverec, i když je možné ji mapovat pouze na trojúhelník. Nejjednodušší způsob, jak lze vytvořit čtverec, je pomocí dvou trojúhelníků, které mají společnou hranu a jsou rovnoramenné. Dále musíme určit, jak se textura nalepí na čtverec. K tomu se využívají texture coordinates, tedy souřadnice textury, nebo je můžeme znát také jako UV souřadnice, obrázek 2.3.2. Tyto mají hodnotu od 0 do 1 a zároveň levý horní roh textury má souřadnici 0,0. Jeden bod textury se v počítačové grafice nazývá texel. Souřadnice UV slouží k tomu, jak se textura srovná napříč povrchu čtverce, laicky jak se nalepí na tento čtverec. Souřadnice U nám určuje výšku textury a souřadnice V její šířku. Každý vrchol čtverce musí mít tyto hodnoty zadané, a podle jejich poměru můžeme například namapovat jen část textury, nebo ji můžeme namapovat vzhůru nohama atd.



Obrázek 2.3.2: Souřadnice textury

Jak je vidět, musí se učinit plno kroků kvůli jednomu obrázku. Toto všechno můžeme obejít použitím spritu, který souží pro tyto účely. Sprite tedy ulehčuje práci s mapováním textur. Je to již vytvořený čtverec, na který se namapuje textura. Ted již stačí zadat pozici, na kterou se má sprite vykreslit ve scéně, a to tak, že na uvedené

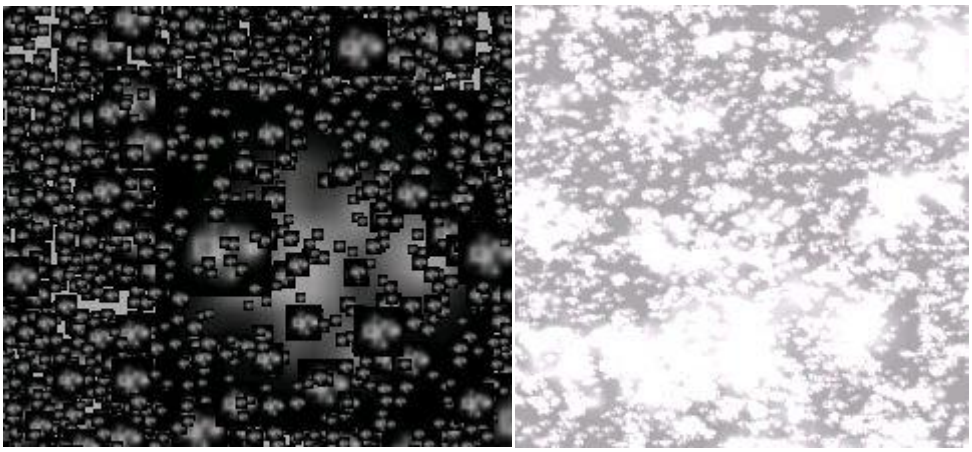
pozici bude střed spritu. Další velkou výhodou spritu je jeho směr ke kameře. Sprite je totiž vždy nasměrován ke kameře celou plochou - jeho normála směřuje ke kameře, takže se nestane, že po otočení kamery o 90 stupňů a přesunutí spritu před kameru, nebude vidět. Proto pokud bychom sprite nepoužívali, museli bychom náš čtverec vhodně rotovat, jinak by nebyl v určitých pozicích vidět, a v jiných z velmi ostrého úhlu, což by deformovalo pohled na jeho texturu. Ovšem tato vlastnost má i jednu velkou nevýhodu, kterou sebou nese právě směřování spritu na kameru. To se projeví v případě, když chceme mít dešťovou kapku jako sprite, a kamera se bude dívat kolmo nahoru nebo dolu. Opět uvidíme celou plochu spritu, což ale v již zmiňovaném případě dešťové kapky přinese dojem, že kapky padají na ležato, jako vysypané špagety. Další užitečnou vlastností spritu je změna jeho velikosti vzhledem ke vzdálenosti od kamery. Můžeme tedy určit, jak moc budou sprity v pozadí menší než ty v popředí, nebo jim nechat stejnou velikost.

2.3.3. Alfa míchání

Tento pojem je rozšířenější v anglickém překladu alpha blending, dále v textu jako AB. AB se využívá k napodobení průsvitnosti materiálů. To si můžeme představit jednoduše. Když máme auto s kouřovými skly, vidíme interiér vozu, ale ten je zatemněný právě kouřovým sklem. AB míchá barvy pixelu, který už byl vykreslen, s barvou nového pixelu, který se má vykreslit na stejné místo. K tomu používá tzv. frame buffer, ve kterém jsou uloženy hodnoty barev každého pixelu, co bude vykreslen na obrazovku. Pokud se má vykreslit pixel, nejdříve se načte z frame bufferu již uložená barva, ta se smíchá s nově příchozí a nově vzniklá se opět uloží do frame bufferu. Pro výsledný obraz je důležité, aby se objekty, co používají AB, vykreslovali seřazené ve správném pořadí - první se vykreslují objekty, které nepoužívají AB. Pak by se měli podle vzdálenosti od kamery seřadit objekty využívající AB a poté je vykreslovat od nejvzdálenějšího. AB je docela náročná operace. Existuje několik způsobů, jak se výsledná barva bude míchat, a lze přesně nastavovat, jestli se použije více barvy z již uloženého pixelu, nebo naopak z nově příchozího, popřípadě může být míchán pouze každý druhý pixel z nově příchozího objektu atd. Více se k této zajímavé problematice lze dozvědět [3]. AB se často používá spolu s průhledností, viz následující kapitola, a lze tak dosáhnout zajímavých efektů pomocí skládání textur za sebe.

2.3.4. Průhlednost

Pokud namapujeme texturu na nějaký objekt, můžeme jí nastavit tzv. alfa kanál, který určuje, jaká barva se nebude vykreslovat. Pokud máme obrázek, kde je na černém pozadí zobrazena kulička a nenastavíme alfa kanál, při vykreslení na jinou než černou plochu uvidíme kolem kuličky černý čtverec. V případě, že hodnota alfa kanálu bude nastavena na černou barvu, bude vykreslena jen samotná kulička, obrázek 2.3.4. Průhlednost textury se často zaměňuje s alfa mícháním, protože alfa míchání vlastně také způsobuje průhlednost, ale úplně v jiném smyslu, kdy dochází k míchání barev. Podporu alfa kanálu mají například obrázky ve formátu tga. Takovýto obrázek si je možné vytvořit v každém lepším grafickém editoru, např. Gimp, Photoshop, Paint Shop Pro a další.

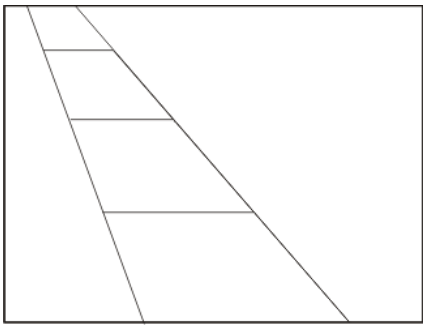


Obrázek 2.3.4: Levá část vykreslení textury bez nastaveného alfa kanálu, pravá s aktivním alfa kanálem.

2.3.5. Filtrování textur

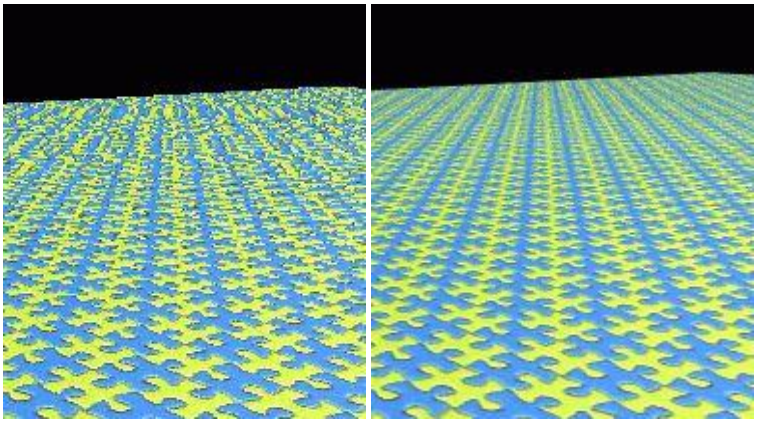
Dnešní hry se bez filtrování textur neobejdou. Jedná se o metodu, která vizuálně vylepšuje celkový dojem při pohledu na scénu.

Nejdříve si vysvětlíme, k čemu je ho potřeba. Máme scénu s cestou, každý metr této cesty je jedna textura s rozlišením 200x200 a se symetrickým vzorem například dlaždic, a proto není poznat, pokud je víc takovýchto textur u sebe, kde která začíná a končí. Pokud tedy cesta vede do dálky, postupně se zmenšuje její šířka, obrázek 2.3.5.1. Proto je zbytečné, aby se stále používala



Obrázek 2.3.5.1: Cesta.

jedna a ta samá textura o stejném rozlišení, ale je lepší použít texturu s nižším rozlišením. Díky tomu šetříme paměť a navíc zrychlujeme aplikaci. A navíc pokud se deformuje velká textura na menší, vznikají v ní artefakty, které kazí vizuální dojem. Když tedy používáme optimalizované textury, nastává problém, protože na sebe přestávají sousední textury opticky navazovat, jak je vidět na obrázku 2.3.5.2.



Obrázek 2.3.5.2: Levá část obrázku - vykreslení textury bez aplikace filtrování textury, pravá po aplikaci. Převzato z [4].

Tento nedostatek řeší právě filtrování textur. Metoda snižování rozlišení textur podle vzdálenosti od kamery se nazývá Mipmapping. Ten tedy způsobuje to, že při načtení textury se vygeneruje sada textur, každá další textura bude mít 2 krát menší rozlišení. Pokud načteme obrázek s rozlišením 16x16, zároveň se vytvoří textury s rozlišením 8x8, 4x4, 2x2 až do 1x1 v paměti na video kartě. Mipmapping se používá i na obrázky, které nemají čtvercový tvar. Mipmapping je tedy aplikován automaticky. My můžeme ovlivnit způsob, jakým na sebe budou textury navazovat, nastavením filtrování textur. Více o mipmappingu [4].

Existuje několik druhů metod pro filtrování textury:

- Bilineární filtrování - nejrychlejší způsob zjemňování textur. Pro výpočet barvy texelu se počítá se čtyřmi nejbližšími texely a výsledná barva je průměrem jejich barev. Toto filtrování je celkově nejjednodušší a nejrychlejší, ale také vizuálně nejhorší, protože dochází ke značnému rozmazání obrazu.
- Tri-lineární filtrování - barva pro jeden pixel je počítána pomocí bilineární interpolace z dvojnásobného počtu bodů než u bilineárního filtrování. Tím

je dosaženo lepšího vzhledu a menšího rozmazání obrazu. Ale stále je znehodnocené tím, že se aplikuje na texturu ve tvaru čtverce, jenže čím dále bude tato textura od kamery, tím více se stává lichoběžníkem a dochází ke ztrátě texelů.

- Anizotropní filtrování - velice pokročilá metoda, při níž je brána v ohledu perspektiva, proto není pro všechny úhly pohledu stejné. Nejdříve jsou vygenerovány lichoběžníky v závislosti na úhlu pohledu kamery a na ty je pak aplikováno tri-lineární filtrování. Při použití anizotropního filtrování se dosahuje nejostřejších výsledků.

3. Realizační část

Vývoj aplikace probíhal pomocí SDK (software development kit) DirectX 9. Ten slouží k vývoji multimediálních aplikací pro operační systémy Windows. DirectX se skládá z mnoha knihoven, které usnadňují vývojářům práci, například DirectSound pro práci se zvukem, DirectInput k odchytávání událostí klávesnice a myši, Direct3D pro práci s 3D grafikou a dalších.

Jako programovací jazyk jsem si zvolil C# a pro něj vývojové prostředí Microsoft Visual Studio 2005 professional edition. Vývoj aplikace probíhal a je optimalizován pro operační systém Windows XP Professional s nainstalovaným service packem 2.

Nyní přejděme k vlastnímu zpracování aplikace.

3.1. Souřadnicový systém v DirectX

Ještě než začnu s popisem vlastních metod pro tvorbu sněhu, objasním, jak se v DirectX tvoří trojrozměrná scéna. Představme si souřadný systém s třemi osami X, Y, Z. Každá osa je kolmá na ostatní dvě. Osy X a Y jsou stejné jako pro dvojrozměrný souřadný systém, tedy X-ová směřuje doprava a Y-ová nahoru. Osa Z směřuje od nás, tedy od kamery, to znamená, že čím dál je objekt vzdálenější, tím je jeho Z-ová souřadnice větší ve smyslu kladné osy. Pokud je kamera umístěna do bodu (0,0,0), její pohled směřuje právě po ose Z v kladném smyslu. Proč to zmiňuji. Pokud místo DirectX používáte OpenGL, tak vězte, že tady je Z-osa v obráceném směru, tedy objekty vzdalující se kameře mají rostoucí zápornou hodnotu.

3.2. Pozice sněhových vloček ve scéně

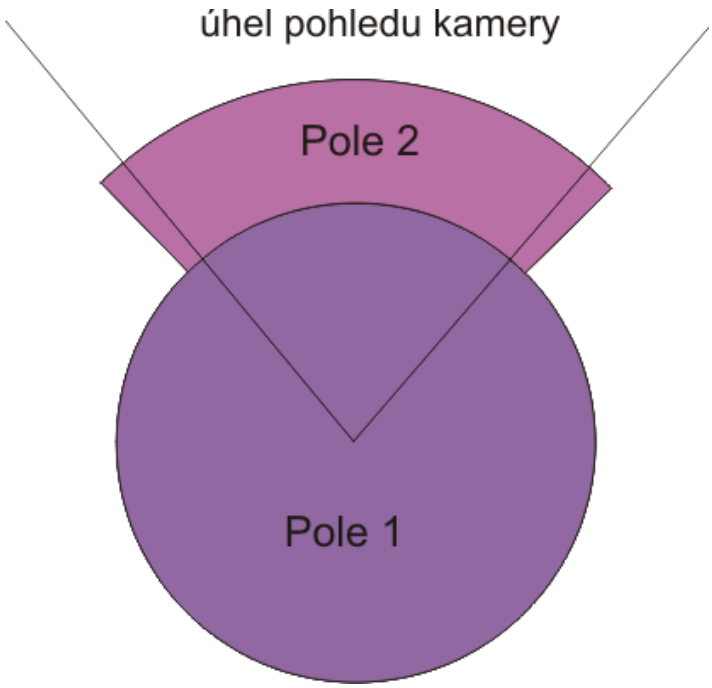
Jak již bylo napsáno v kapitole 2.3.1, pro tvorbu sněhu se využívá téměř výhradně částicového systému. Touto cestou jsem se vydal i já, pouze jsem si ji trošku přizpůsobil.

Než jsem se blíže seznámil s částicovým systémem, ubíraly se moje úvahy mylnými směry. Zamýšlel jsem udělat pole vloček o velikosti několika milionů, a vločky vykreslovat po celé scéně, tak jak to je ve skutečnosti, kam až oko dohlédne i nedohlédne, tedy i na místech, kam kamera nevidí, což je třeba celý prostor za ní. Tyto úvahy jsem také začal aplikovat. Pro generování pole vloček, jak je rozmístit do prostoru, mi stačily pouze 3 údaje, a to maximální výška, ze které bude sníh padat, a poté x-ová a y-ová souřadnice, které určují vzdálenost, kam až bude sněžit. Každé vločce jsem náhodně generoval všechny tři souřadnice podle zadaného omezení, a také jsem nezapomněl na záporné souřadnice, to kvůli počáteční pozici kamery, která je nastavena na (0,1.85,0). Pohyb kamery by se při použití takto nastavených parametrů řešit nemusel, vždyť vločky by byly po celém území. Vzhledem k výkonové náročnosti částicového systému bylo jasné, že tato cesta nebude tou správnou.

Další nápad byl, že tedy nepoužiji tak ohromné množství vloček, ale spokojím se s statisícem a celkovou plochu, na kterou se budou vykreslovat, zmenším. To už bylo výkonově únosnější. Vločky dopadají na čtverec o hraně 15 metrů, přičemž kamera byla umístěna do středu tohoto čtverce. Teď už se ale musela začít řešit interakce kamery a vloček při pohybu. Stačilo ujít oněch 15 metrů a vločky už zde nebyly. První, co jsem vyzkoušel, bylo přičítání pozice kamery ke každé vločce. To zajistilo, že každá vločka si udržovala stále stejnou vzdálenost od kamery, a kamera byla pořád ve středu čtverce vloček. Pokud by se kamera pohybovala jen nepatrně rychle, ani by to nebylo poznat, jenže při větších rychlostech sníh začal vypadat velice nepřirozeně. To přineslo značné zrychlení aplikace oproti předchozímu způsobu, ale pořád to bylo pod hranicí 15 fps. Pokud se fps nepohybují alespoň kolem 25, obraz nebude vypadat plynule, ale bude se trhat.

Pro zvýšení výkonu jsem se snažil toto jedno velké pole vloček rozdělit na desítky menších polí, které byly poskládány kolem kamery jako kostičky. Měl jsem v plánu procházet a vykreslovat jen ty vločky, které byly v polích, co se nacházely před

kamerou ve viditelné oblasti. Realizace se mi ale vůbec nepovedla zprovoznit, protože pohyb vloček mezi poli byl jen těžko uhlídatelný.



Obrázek 3.2: Rozložení spritů do polí, při pohledu na kameru shora.

Teď už mi bylo jasné, že bez ořezávání vloček, a udržování jejich pozic nejlépe pouze před kamerou nebude možné naprogramovat obstojně rychlou aplikaci. Na začátku každé aplikace využívající Direct3D se nastavuje i úhel pohledu kamery. To jsem hodlal využít pro ořezávání vloček při rotaci kamery kolem osy Y, tedy pokud se kamera otáčí do stran. Moje úvaha byla, že bude stačit vykreslovat vločky jen do trošku větší výseče, než je úhel kamery (na obrázku 3.2 značeno jako Pole 2). Následně pokud se kamera začne otáčet, nastavil jsem si dvě booleovské proměnné, jednu pokud se pohybuje kamerou do strany, a druhou, pokud je pohybováno kamerou pouze doleva. První booleovská zajišťuje, že se algoritmus pro ořezávání vloček nebude volat, i když je kamera v klidu. Docházelo by ke zbytečnému zpomalování celé aplikace, a žádnou vločku by to neořezalo. Druhá booleovská slouží k upřesnění pohybu kamery, tedy jestli se otáčí doleva nebo doprava. Pokud se tedy kamera začala otáčet do strany, podle těchto dvou booleovských proměnných jsem přesně věděl jakým, a začal se volat algoritmus, jehož smysl popíši v následující kapitole.

3.3. Ořezávání vloček při pohybu kamery

Moje úvaha byla jednoduchá. Pokud se kamera pootočí, zkontroluji všechny pozice vloček, jestli je jejich pozice stále v úhlu pohledu kamery, a pokud ne, tak tyto vločky přesunu ve směru otáčení o úhel pohledu kamery. Před každým vykreslením vloček si spočítám novou rovnici přímky, podle které pak budu vločky ořezávat. Při ořezávání beru v úvahu pouze X-ovou a Z-ovou souřadnici vločky. Výška vločky je pro mě nezajímavá z důvodu, že by se zas tolik vloček neořezalo a zbytečně by počítání sinus a cosinus pro všechny vločky zdržovalo.

Tato metoda funguje perfektně, pokud se kamera otáčí do určité rychlosti, a pouze do stran. Když dojde k trhnutí myši o větší úhel, algoritmus se tedy nezavolá pro každý stupeň otočení, vločky se ořežou správně, ale přesunou se zase o úhel kamery, což zanechá ve sněhu prostor, kde nesněží. Jedním ze způsobů jak tomuto zamezit, je povolit otáčení kamery v jednom kroku jen pro určitý úhel. To bohužel není nejlepší řešení, protože v dnešních akčních hrách je rychlost otáčení prioritou.

Při používání tohoto typu ořezávání se ukázalo, že má více nedostatků. Při pohledu kamery nahoru je jasně vidět, že vločky jsou ořezané už nad kamerou, takže sněží jen v tomto ořezaném kousku. To jsem vyřešil tak, že pokud se kamera zvedne přes nějakou konstantu, vločky se přestanou ořezávat do stran, ale ořežou se od určité vzdálenosti od kamery. Tím si aplikace zachová svižnost při pohledu nahoru a dolu.

Tímto jsem vyřešil tedy ořezávání vloček při otáčení kamery, ale nevyřešil jsem pohyb kamery a následné otáčení. Rovnice přímek, podle kterých se vločky ořezávají, jsou totiž spočítané jen pro počátek souřadnicového systému. A následná rotace vloček, které zůstanou mimo úhel pohledu, také. Tento problém jsem vyřešil poměrně elegantně. Mám totiž jedno pole, kde udržuji pozice vloček jenom vůči počátku (0,0,0). Před vykreslením každé vločky ji nejdříve rotuji o potřebný úhel, novou pozici uložím do pole, a následně k jejím souřadnicím přičtu souřadnice kamery. Tak je zajištěno, že se vločka bude otáčet kdekoliv v prostoru. Bohužel jsem si neuvědomil, že tímto krokem sníh opět nebude vypadat reálně při chůzi.

Musel jsem se vydat cestou, kdy každá vločka bude nezávislá na pohybu kamery. To jsem hodlal vyřešit konstantou, která značila poloměr kruhu (na obrázku 3.2 značeno jako Pole 1), kde kamera byla středem, a každá vločka, která byla v tomto kruhu, se mohla pohybovat jakkoliv. Pokud se kamera začala pohybovat,

musela se zkontrolovat pro každou vločku vzdálenost od kamery, a tu porovnat s naší konstantou, což byl vlastně poloměr kruhu. Když byla vzdálenost vločky větší, vločka se zahodila, a nová byla vygenerována do směru pohybu kamery. Tak jsem tedy zajistil, že při jakémkoliv pohybu kamery bude neustále sněžit. Teď bylo potřeba vločky zase ořezat.

Přišlo mi jako nesmysl, zahodit všechny části algoritmu na ořezávání, i když nefungoval, jak by měl. Rozhodl jsem se postupovat opačným směrem. Nebudu vločky ořezávat a s ořezanými manipulovat do směru otáčení kamery, ale budu zobrazovat jenom ty vločky, které v tomto úhlu již jsou. Toto mi umožní znovupoužít starý způsob generování vloček v celém prostoru, tedy i za kamerou. Tato zajímavá myšlenka je použita ve finálním algoritmu. Pojďme se tedy podívat, jak to celé funguje.

3.4. Algoritmus pro generování vloček

Na začátku programu, před samotným vykreslováním scény, se vločky musí nejdříve vygenerovat. Pro uchovávání pozic vloček jsem zvolil datový typ pole vektorů. Dále jsem se rozhodl, že bude možné generovat dva typy polí, znázorněno na obrázku 3.2. V prvním poli, které bude kruhového tvaru se středem pozice kamery, se vločky vygenerují do celého kruhu. Druhé pole bude obsahovat vločky, které budou začínat ve vzdálenosti, kde končí vločky z prvního pole, nebo větší, a budou se generovat do zadané vzdálenosti. K tomu slouží třída `GenerujSnih`, které pomocí konstruktoru nastavíme výšku, která bude sloužit pro omezení Y souřadnice vloček, tedy jejich výšku v prostoru. Dále nastavíme výšku kamery, která bude sloužit ke generování výšky vloček pro druhé pole.

Nechtěl jsem mít algoritmus, který bude generovat celé pole vloček, proto jsem vytvořil 2 metody (`vratVlocku`, `vratVlocku2`), které generují pozice vločky jen pro jeden druh pole. První metodě stačí pouze poloměr a výška, a poté náhodně vytvoří vločku, jejíž pozice jsou omezeny tělesem válec se zadanými hodnotami. Stačí vygenerovat pozici Y do maximální hodnoty určené výškou, a X-ová a Z-ová souřadnice se vygenerují tak, že pro nově vzniklé souřadnice vločky musí platit $x^2 + z^2 <= r^2$.

Do druhé metody `vratVlocku2` vstupují důležité parametry `polomer1`, `polomer2`, uhel. `Polomer1` je vzdálenost od kamery, od které se začne generovat pozice vločky. `Polomer2` určuje maximální vzdálenost, kterou mohou vločky mít, a uhel značí úhel pohledu kamery. Vločky se vygenerují do prostoru, který svírá tento úhel. Druhé pole slouží ve výsledku jen pro větší hustotu sněhu, ale jeho údržba je náročnější na výkon, proto ho není nutné používat.

V třídě *GenerujSnih* jsou samozřejmě připraveny také metody (`naplnPrvniCast`, `naplnDruhouCast`), které zajistí naplnění obou polí požadovaným počtem vloček, a také metoda `Snow`, která zavolá obě předchozí metody, a vrátí pouze jedno pole, do kterého uloží všechny vygenerované vločky.

Když jsou pole s vločkami vygenerovány, může být začít vykreslována samotná scéna. Ještě upřesním, proč je lepší pole vloček vygenerovat jako první, a až pak vykreslovat scénu. Obě pole se totiž generují s co největším počtem vloček, a další přidávání již nebude možné, pouze pro nově vytvořené pole. To je z důvodu, že při generování statisíců vloček program na několik setin sekundy zamrzne. Když by se toto stalo při vykreslování scény, obraz se po tuto dobu zasekne, což je nepříjemný jev. Za běhu programu můžeme ovlivňovat výsledný počet zobrazovaných vloček směrem dolu, tedy vločky ubírat, a pak je zase přidávat. Myslím si, že toto je velice elegantní řešení, kdy nedochází k zasekávání obrazu právě při zvětšování počtu vloček.

3.5. Vykreslování vloček

Máme tedy vygenerována pole vloček, a můžeme je začít vykreslovat. O kompletní vykreslování s updatováním pozic se stará třída `Snih`. O kompletní správu sněhu se starají metody `DrawSnowFall` a `DrawSnowFall2`. Vykreslování v obou metodách probíhá až na malé odlišnosti následujícím způsobem. Pro každé pole se tyto postupy liší v drobnostech, které později vysvětlím. Nejdříve jsou spočítány potřebné proměnné, které jsou závislé na pozici kamery, a budou uplatněny pro přesuny vloček, například ze směru pohledu kamery se spočítají rovnice přímek, podle kterých se budou vločky vykreslovat. Pokud se totiž kamera pohne, všechny rovnice musí být změněny. Následně se prochází pole všech vloček. Každá vločka se načte do proměnné `currentsprite`, a pokud došlo k pohybu kamery, kontroluje

se pozice vločky vůči kameře. Jakmile má vločka větší vzdálenost než je definovaná, tak dojde k jejímu přesunu. Následně se kontroluje, zda-li je vločka v úhlu pohledu kamery. Následně se zapne alfa míchání a vločky jsou vykresleny. Teď popíši rozdíly v metodách `DrawSnowFall` a `DrawSnowFall2`.

Pokud vykreslujeme vločky z prvního typu, tvoří kolem kamery jakýsi válec. Tento válec si můžeme představit jako hranici, kde se vločky mohou vyskytovat. Jakmile se kamera začne přesouvat, přesouvá se i tento válec, a to tak, že osa válce prochází stále pozicí kamery, a zároveň je kolmá na osy X a Z. Lze si to jednoduše představit, pokud na sud namontujeme kolečka a budeme s ním pohybovat. Pokud tedy dojde k posunu válce, projde se pozice všech vloček, a když se nějaká dostane mimo válec, je pomocí vztahu pro výpočet středu úsečky, vzorec 3.5.1, přesunuta před válec.

$$S\,[A;B] = [(\frac{A_x + B_x}{2});\ (\frac{A_y + B_y}{2})]$$

Vzorec 3.5.1: Výpočet středu přímky.

My z této rovnice známe bod S, protože to je pozice kamery, a současně bod A, což je pozice vločky, tudíž můžeme dopočítat pozici nové vločky. Velmi jednoduché. Bohužel cesta k této myšlence vedla přes složité výpočty pomocí goniometrických funkcí sinus a cosinus. Ano, přesun vločky můžeme počítat i pomocí rotací se středem v bodě kamery, ale nesmíme zapomínat, že každá matematická operace navíc zdržuje samotný výpočet, a pokud chceme mít program co nejrychlejší, rozhodně je lepší se těmto výpočtům vyhnout. Proto jsem se snažil používat jen nejrychlejší operace jako je sčítání, odečítání, násobení. Samozřejmě jsem se nemohl vyvarovat funkcím sinus a kosinus, a proto když to je možné, počítám tyto hodnoty mimo for cykly. Tím se kód stává drobet nepřehlednější, ale zase je to vyváženo celkovým výkonem aplikace, více kapitola 3.7.

Teď, když jsou po pohybu kamery všechny vločky opět umístěny ve válci, u každé se kontroluje její viditelnost. Pokud leží ve směru úhlu pohledu kamery, bude vykreslena, jinak ne. Zjištění, zda se vločka nachází v tomto úhlu je možné odvodit pomocí rovnice přímky, vzorec 3.5.2.

$$p\colon a * x + b * y + c = 0$$

Vzorec 3.5.2: Obecná rovnice přímky.

[x,y] jsou pozice bodu, který leží na přímce, a (a,b) je normálový vektor přímky. Nejdříve se spočítají dvě rovnice, pro záběr kamery doleva a doprava. Z těchto rovnic získáme konstantu c, pro posunutí středu souřadného systému. Bod c počítáme tak, že známe úhel přímek viditelnosti kamery, z něho dostaneme normálové vektory, a ty pak jen dosadíme do rovnice přímky. Máme tedy konstantu c, a tu použijeme pro kontrolu pozic vloček. Pokud máme pole vloček o velikosti 200 000 vloček, a záběr kamery je devadesát stupňů, je průměrně vykresleno 44 tisíc vloček. Z toho si dokážeme představit, jaký výkon pro zbytečné kreslení vloček, které nejsou vidět, ušetříme.

Nyní máme vločky, které se mají vykreslit, ale ještě než se tak stane, dojde k upravení jejich pozice Y, tedy výšky. Pokud je vločka nad úrovní podlahy, sníží se její pozice o náhodnou hodnotu, a zároveň se pro aplikaci větru změní X-ová a Y-ová souřadnice. Pokud se vločka začne přibližovat podlaze, je automaticky zpomalena, protože to ve výsledku vypadá lépe. Když už dosáhla výšky podlahy, její hodnota se automaticky zvýší podle nastavení aplikace, a celý koloběh se opakuje. Zároveň toto všechno je nezávislé na snímcích za sekundu, takže vločky si udržují pohyb závislý pouze na čase. To nezpůsobuje rozdíly v rychlosti padání vloček se změnou framerate.

3.6. Využití vlastností alfa míchání a filtrování textur

Před vykreslením zbývá už jen zapnout alfa míchání, nastavit texturu, a nechat vločky vykreslit. Pro vločky používám obrázek vytvořený v programu Photoshop. Abych se přiznal, vytvoření obrázku, který by po namapování na sprite a aplikování filtrování textur a alfa míchání splňoval mé požadavky, jak by měl sníh vypadat, byla nejnudnější věc z celého programu. Nejdříve jsem se snažil mít těchto textur alespoň pět, a programově je přepínat. Každá vločka si tedy musela pamatovat, jaká textura je na ní namapována. To vedlo ke zvýšené režii a poklesu fps. S tímto jsem se nechtěl spokojit. Když jsem se dozvěděl, jak funguje alfa míchání, kapitola 2.3.3, tušil jsem, že ho budu moci využít ve svůj prospěch, a to se také nakonec povedlo s kombinací filtrování textur, kapitola 2.3.5. Pro všechny padající vločky mi totiž stačí pouze jedna textura, což je dosti podivuhodné. Ptáte se

jak je to možné? Využil jsem vlastností alfa míchání a filtrování textur. Jak uvádím v kapitole 2.3.3, alfa míchání funguje tak, že pro každý pixel na obrazovce, který bude vykreslen, uchovává buffer, ve kterém jsou uloženy dosavadní barvy. Pro nově příchozí pixel se tyto hodnoty smíchají a znovu uloží. Pokud se již na nějaké místo má vykreslit vločka, a následně ještě jedna, například ve větší vzdálenosti (takže bude automaticky menších rozměrů, vysvětleno v kapitole 2.3.2), jejich barvy se smíchají, a vznikne tím pádem úplně nová vločka, která se zdá mírně odlišná od ostatních. Pokud je za ní více vloček, začíná se od ostatních lišit více. Z toho vyplývá, že nejlepších vizuálních výsledků je dosaženo při použití většího počtu vloček. Ještě uvedu, že jsem schválně použil co nejmenší obrázek, o velikosti 11x11 pixelu, protože jsem chtěl využít možností filtrování textur. Když je filtrování vypnuté, vločky blízko u kamery, které mají největší rozměry, vypadají být kostičkované, což je správně. Pokud filtrování povolíme, tyto kostičky zmizí, a navíc vločky poblíž kamery jsou rozmazány, a v dálce ne. To, že vločky u kamery jsou rozmazané běžná věc, a v aplikaci byla vyžadována. Výběr druhu filtrování textury je prováděn automaticky podle druhu vaší grafické karty je vybráno to nejlepší.

Po vykreslení všech vloček se alfa míchání vypíná.

3.7. Výkonnostní testy

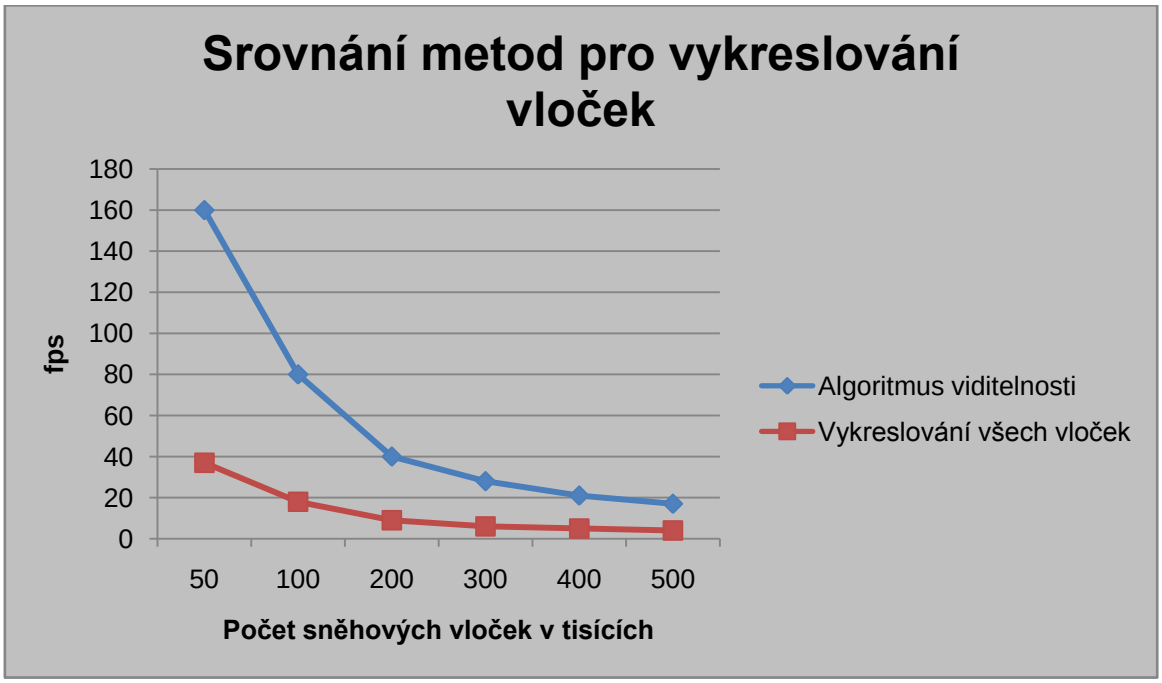
Jak již bylo naznačeno, správa částicového systému je docela výkonově náročná. Při každém vykreslovaném snímku se mění vlastnosti všech částic. To představuje největší zátěž pro výkon procesoru počítače. Dalším faktorem, který může ovlivňovat vykreslovaný počet snímků za jednu sekundu je grafika počítače. Testy jsem provozoval na notebooku:

- Procesor Intel Pentium M 1.7GHz, 533MHz FSB
- 2 x 512 MB DDR II RAM
- Grafická karta Ati X300 64MB s vlastní pamětí, využívající rozhraní PCI-E

Jediným výsledkem, který je zajímavý, je pozorování vývoje křivky závislostí počtu vloček na fps. Je to z toho důvodu, že výsledný efekt sněhu, by při ubrání kvality vykreslování, přestal vypadat jako sníh. Protože jsem značnou část implementace strávil na řešení algoritmu, který se stará o přesouvání vloček tak, aby

při pohybu kamery sníh vypadal jako ve skutečnosti, testy jsem zaměřil právě na tento problém.

Aby testy nebyly zkresleny, každý test jsem pustil vždy od začátku s nastavenými parametry. Testy jsem pouštěl 2 krát, jednou jsem zapnul algoritmus na řešení viditelnosti vloček, a podruhé ho vypnul. To jsem opakoval pro různé počty vloček ve scéně. V grafu 3.7.1 je vidět, že při použití algoritmu se dosahuje vcelku rapidního nárůstu výkonu, oproti případu, kdy není využit. Je to zapříčiněné tím, že při použití tohoto algoritmu se vykreslují pouze vločky, které jsou v zorném poli kamery. Opravdové počty vykreslovaných vloček, při pohledu kamery ve směru osy Z, naleznete v tabulce 3.7.2.



Graf 3.7.1: Výkonnostní test pro různé počty vykreslovaných vloček se zapnutým algoritmem viditelnosti.

Počet všech vloček	Počet fps	Počet vykreslovaných vloček
500 000	17	106 000
400 000	21	85 000
300 000	28	64 000
200 000	40	42 000
100 000	80	21 500
50 000	160	10 000

Tabulka 3.7.2: Opravdový počet vykreslovaných vloček při použití algoritmu viditelnosti.

Povšimněme si přímé úměrnosti, která panuje mezi počtem fps a počtem vykreslovaných vloček. Počet fps se při dvojnásobném zvýšení vykreslovaných vloček dvojnásobně sníží. Domníval jsem se, že to je díky frekvenci procesoru, což se mi potvrdilo, když jsem si testy zopakoval s taktovaným procesorem na frekvenci 600 Mhz. Výsledný počet fps klesl pro každý test přibližně 3x. Z toho je zřejmé, že čím rychlejší procesor budeme vlastnit, tím hustější budeme moci vytvořit sníh, aniž by hodnota fps klesla pod 25 snímků za sekundu. Myslím si, že při hodnotách 42000 skutečně vykreslovaných sněhových vloček už je dosaženo dobře vypadajícího hustého sněžení.

4. Závěr

Cílem práce bylo navrhnout a implementovat vhodné metody pro pohyb vloček, a implementovat metody pro jejich zobrazení a rychlé vykreslení v real time aplikacích. Poté implementovaný efekt otestovat.

Zadání se podařilo splnit ve všech směrech, i když některé body s diskutabilním výsledkem, což je dáno subjektivitou názorů, kdy se každému líbí něco jiného. Po mé stránce jsem spokojen hlavně s dosaženou rychlostí na závislosti hustoty sněhu. Rychlost aplikace byl po dlouhou dobu vývoje kámen úrazu. Další věc, se kterou jsem plně spokojen, je dosažený vzhled výsledného efektu, který si troufám tvrdit, je při správném nastavení aplikace více než dobrý. Toho jsem docílil využitím vlastností alfa míchání a filtrování textur.

K bakalářské práci je přiloženo CD, na kterém je video s natočeným reálným sněžením. Při porovnání s nasimulovaným sněhem v mé aplikaci je patrné, že sněhové vločky vypadají ve skutečnosti menší. Zvolením větších rozměrů vloček, jsem mohl omezit jejich celkový počet na přijatelnou hodnotu, a tak dosahovat lepších výkonnostních testů, které jsou popsány v kapitole 3.7.

Zdrojový kód aplikace jsem se snažil co nejlépe zdokumentovat, pro pozdější znovupoužití a rozšíření, popřípadě zdokonalení, protože si myslím, že základ aplikace je dobře navrhnut. Jádro programu je navrhnuto tak, že jé možné jej použít i na atmosférický efekt déšť, zde by se ale musel vyřešit problém s natáčením spritů na kameru, při pohledu kamery vzhůru, což popisuji v kapitole 2.3.2.

Dalším možným rozšířením by bylo aplikovat atmosférický jev mlhu, nebo mraky,

což by ještě umocnilo výsledný efekt aplikace. Nesmím zapomenout zmínit, že v aplikaci není řešeno usazování sněhu na zemi, které sebou přináší spousty vedlejších efektů, jako je zanechávání stop, sesuvy sněhu, nebo jeho tání a znovu namrzání.

Literatura

[1] Wikipedia. *Sníh* [online]. c2008. [cit. 2008-04-10]. <http://cs.wikipedia.org/wiki/Sníh>

[2] *O sněhové vločce* [online]. <http://druidova.mysteria.cz/UKAZY_VE_VESMIRU/SNEHOVA_VLOCKA.htm>

[3] Luna, D. Frank - Lopez, Rod. *Introduction to 3D game programming with DirectX 9.0.* Wordware Publishing, Inc. 2003. ISBN 1-55622-913-5.

[4] Tišnovský, Pavel. *Grafická knihovna OpenGL (25): mipmapping* [online]. c2003. [cit. 2003-12-23] <http://www.root.cz/clanky/opengl-25-mipmapping/#hrf1>

[5] *Largest snowflake* [online]. <http://www.cnn.com/SPECIALS/2006/winter.weather/interactive/gallery.weather.records/content.1.8.html>

[6] Kopáček, Jaroslav. – Bednář, Jan. *Jak vzniká počasí.* Karolinum, 2005. ISBN 80-246-1002-7.

[7] Libbrecht, G. Kenneth. *A Snowflake Primer* [online]. c1999. <http://www.its.caltech.edu/~atomic/snowcrystals/primer/primer.htm>

[8] *Amatérská prohlídka oblohy* [online]. c2005. [cit. 2005-05-27]. <http://www.astronomie.cz/modules.php?name=News&file=print&sid=698>

[9] Wikipedia. *MSA* [online]. c2007. [cit. 2007-12-17]. <http://cs.wikipedia.org/wiki/MSA>.

[10] Thorn, Alan. *DirectX 9 User Interfaces Design and Implementation*. Wordware Publishing, Inc. 2004. ISBN 1-55622-249-1.

[11] Thorn, Alan. DirectX *9 Graphics, the definitive guide to Direct3D*. Wordware Publishing, Inc. 2005. ISBN-10 1-55622-229-7.

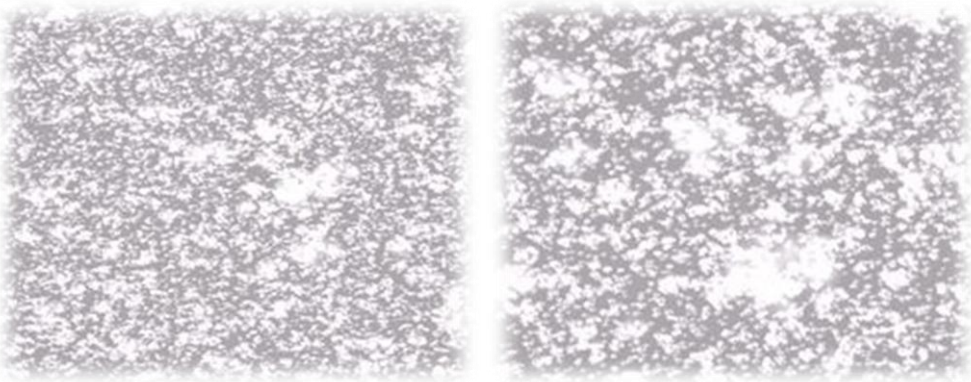
[12] Harrison, T. Lynn. *Introduction to 3D game engine design using directX 9 and C#.* Apress, 2003. ISBN 1-59059-081-3.

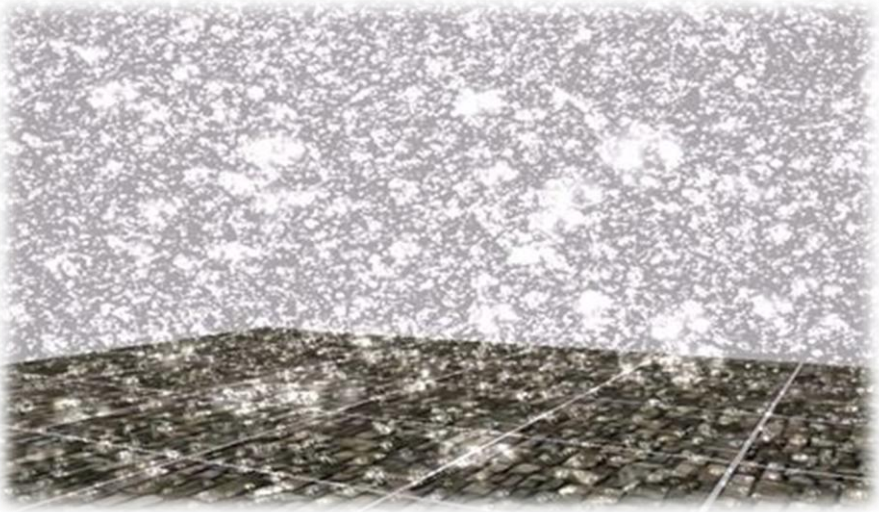
[13] Penton, Ron. *Beginning C# game programming.* Thomson Course Technology PTR, 2005. ISBN 1-59200-517-9.

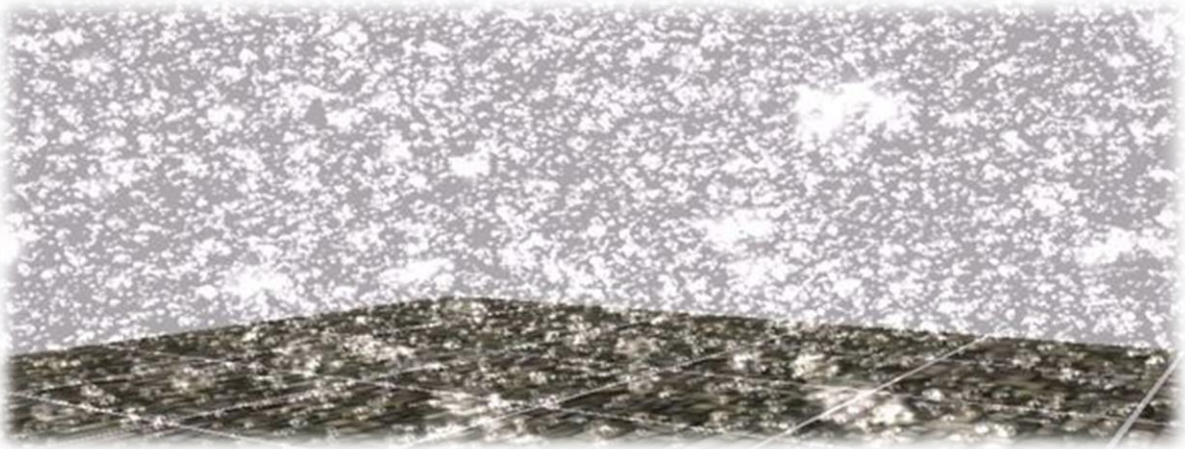
[14] Přeučil, Pavel. *Tajemství sněhových vloček* [online]. C2008. [cit. 2008-02-19]. <http://www.21stoleti.cz/view.php?cisloclanku=2008021818>.

Příloha A

Ukázka simulace sněžení v aplikaci.







Příloha B

Uživatelská příručka

Zdrojové kódy jsou umístěny na přiloženém CD ve složce *Program*. Spustitelná podoba programu je ve složce *Program/Spust/Snow/* se jménem souboru *Snow.exe.* Ke spuštění programu je nutné mít nainstalované DirectX 9 runtime, které je volně dostupné na <*http://www.microsoft.com*>. Program je odladěn pro operační systém Windows XP SP2.

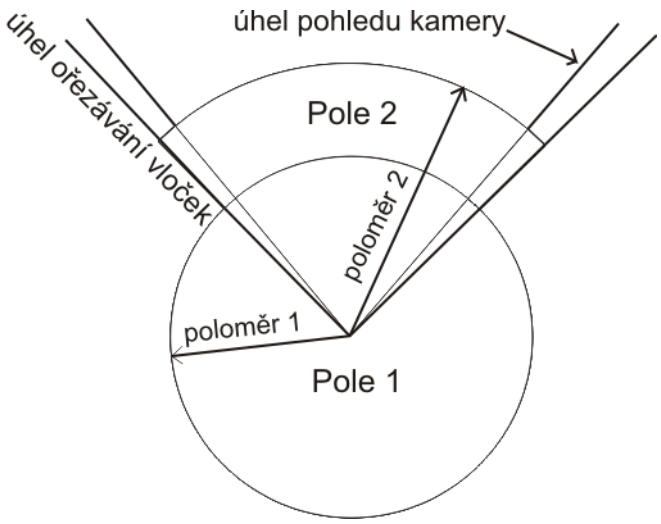
Ovládání programu

Po spuštění programu lze měnit všechny důležité parametry pro simulaci sněhu, v tabulce 5.1 jsou vysvětleny akce k jednotlivým klávesám. Pokud jsou pro jednu akci 2 klávesy, první bude hodnotu zvyšovat, druhá snižovat. Při použití tučně označených kláves je nutné program resetovat – tato akce bude prováděna automaticky. Reset vytvoří nová pole vloček a resetuje pozici kamery.

Klávesa	Akce
pohyb myší	rozhlížení se
R	reset hodnot na defaultní, nebo zadané
K	zapnutí/vypnutí sněžení
Q,E	nastavení výšky, ze které bude sněžit
B,F	nastavení maximálního počtu vloček pole 2
T,G	nastavení počtu vloček pro pole 1
C,V	nastavení poloměru pole 1
Y,X	nastavení poloměru pole 2
I,O	nastavení síly větru
U,J	nastavení směru větru
M,N	nastavení rychlosti vloček
W,A,S,D	pohyb ve scéně

Tabulka 5.1: Ovládání programu - významy kláves.

Pokud jste nečetli dokumentaci, doporučuji pro lepší pochopení, co která hodnota ovlivňuje, prohlédnout obrázek 5.2 na následující stránce.



Obrázek 5.2: Upřesnění nastavovaných hodnot.