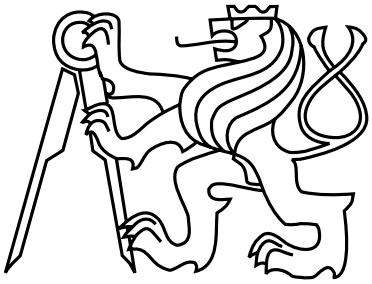
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Realistické zobrazování deště

Markéta Rejdová

Vedoucí práce: Ing. Jaroslav Sloup

Studijní program: Elektrotechnika a informatika, strukturovaný, Bakalářský

Obor: Výpočetní technika

11. června 2009

Poděkování

Na tomto místě bych ráda poděkovala mému vedoucímu bakalářské práce, Ing. Jaroslavu Sloupovi, za poskytnutí tématu práce, za cenné rady a přípomíny a zároveň za poskytnutí materiálů, které byly potřebné k implementaci aplikace a psaní tohoto textu.

Prohlášení

Prohlašuji, že jsem práci vypracovala samostatně a použila jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 11. června 2009 .....

Abstrakt

V současné době se ve většině počítačových her můžeme setkat s atmosférickými jevy, jako je mlha, sníh nebo déšť. Díky těmto jevům mají počítačové hry vizuálně přesvědčivější prostředí.

Tato bakalářská práce se zabývá různými metodami realistického vykreslování deště a následným návrhem a implementací jedné z nich, tj. metody, která využívá částicové systémy a statické textury, a zároveň je založena na fyzikálních zákonech.

Výsledkem je aplikace, která umožňuje simulovat a zobrazovat scény s deštěm v reálném čase. Součástí této aplikace je i jednoduché uživatelské rozhraní, v kterém je možné měnit a nastavovat jednotlivé simulační parametry, jakými jsou např.: velikost a počet kapek.

Abstract

In most current computer games we can see atmospheric phenomena like fog, snow or rain. Thanks to these the games can offer a more visually realistic environment.

This bachelor thesis deals with different methods of realistic rendering of rain and subsequent design and implementation of one of then, namely a method using particle systems and static textures also based on physical laws.

The result is an application which simulates and renders scenes with rain in real time. A part of this application is a simple user interface, enabling the user to change and tweak individual simulation parameters like, for example, the number and size of raindrops.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Ještě do nedávné doby nebyl grafický hardware natolik propracovaný(neměl tolik možností), aby bylo možné přidat do real-timových aplikací libovolné množství efektů. Vzhledem k tomu byla v real-timových aplikacích přisuzována vyšší priorita rychlosti, nikoliv realismu. S postupně narůstajícími možnostmi grafického hardwaru, které jsou v současné době ohromující, se zvyšují nároky uživatelů na vysoký stupeň realismu. Vyžadují vizuálně přesvědčivé prostředí, které se bude, co nejvíce podobat reálnému světu. Roste tak snaha o propojení těchto dvou bodů. Do real-timových aplikací začíná pronikat fotorealismus, který se řídí přírodními zákony. Za tímto účelem začínají vývojáři zavádět do svých aplikací přírodní jevy.

Mnohdy se v počítačové grafice setkáme s přírodními jevy jako např.: real-time osvětlení, stíny, led, mlha, kouř, mraky, vítr, sníh či déšť (viz.obr.1.1, 1.2, 1.3, 1.4). Většina těchto dynamických efektů se vyskytuje v počítačových hrách, aby se hráč mohl více ponořit do reálného prostředí hry.

Obrázek 1.1: Efekt kouře z lokomotivy (ze hry MS Train Simulator).

Obrázek 1.2: Zobrazení stínů a vodopádu (snímek z Far Cry 2).

Vývojáři závodních her musí navíc řešit fyzikální zákony mezi automobily a mokrou vozovkou, musí tak brát v potaz možnost smyků apod. V adventurách již jde čistě o vizuální vzhled, avšak například mlha či změna denní doby brání v pohledu do větší dálky

Obrázek 1.3: Zasněžená krajina (scéna z Hostile waters 4).

Obrázek 1.4: Efekt plujících mraků z akční hry Battlefield.

a tím může i ztížit nalezení určitého předmětu či vyluštění hádanky. V akčních hrách může mlha pro změnu způsobit přehlédnutí nepřítele. Mlha (obr.1.5) dosáhla největšího rozmachu s prvními 3D hrami na PC a to i vzhledem k tomu, že ulehčuje procesoru počítání do skrytých hran. Hororovou atmosféru dodává adventuře Sillent Hill, v které je celé město i jeho okolí zahaleno v mlze (obr.1.6). Do her se snaží vývojáři dostat stále více zajímavějších, realističtějších a dynamických efektů, jako například vítr, který si pohrává s korunami stromů nebo šlápoty ve sněhu, které za sebou zanechává postava hlavní hrdinky v PC hře Syberia 2 (obr.1.7).

Obrázek 1.5: Scéna ze hry Battlefield v mlze.

Obrázek 1.6: Mlha v podání tvůrců hry Sillent Hill.

Nejčastějším jevem počasí v reálném světě je déšť, avšak i přesto doposud postrádá v aplikacích realismus. Metody vykreslování deště v počítačových aplikacích mohou být rozděleny do dvou základních kategorií. První metoda (obr.1.8), ta běžnější, používá pro zobrazování deště částicové systémy a statické textury, které však nevedou k dostatečně reálnému zobrazení tohoto jevu. Druhá metoda, zobrazující dešťové kapky pomalu se pohybující po povrchu objektu, je založena na fyzikálních zákonech, a tudíž je již realističtější, avšak za cenu vysokých výpočetních nároků. Tato bakalářská práce by se měla

Obrázek 1.7: Snímek se stopami ve sněhu z adventury Syberia 2.

zabývat metodou, která se snaží skloubit výhody obou typů metod a eliminovat jejich nevýhody. Tato metoda je schopna realistického vykreslování deště s vysokou snímkovou frekvencí s využitím programovatelného grafického hardwaru. Je založena na fyzikálních vlastnostech dešťových kapek a to jak geometrických, dynamických, tak i optických. Využívá dynamickou texturu s obrazem pozadí, která je namapována na dešťovou kapku dle optických zákonů pomocí fragment shaderu. Tuto metodu se budeme snažit rozšířit tím, že zohledníme retinální persistenci sítnice (zdánlivě kulaté kapky se zobrazí jako pruhy),viz.[1]. Což povede ke zdánlivému rozmazání pohybu kapek. Zároveň představíme další rozšíření, které se bude zabývat osvětlováním dešťových kapek pomocí světelných zdrojů.

Obrázek 1.8: Déšť metodou částicových systémů používaný ve hře Call of Duty

V této bakalářské práci popíšeme fyzikální vlastnosti dešťových kapek, uvedeme předchozí související díla a představíme danou problematiku. Následně se budeme zabývat vybranou metodou realistického real-timového zobrazování dešťových kapek a navrhneme již zmiňovaná rozšíření. Na konec představíme dosažené výsledky, porovnané s fotkami reálného deště, a závěry.

Kapitola 2

Fyzikální vlastnosti dešťových kapek

Déšť je hydrometeor. Řadíme ho mezi srážky vypadávající z oblaků, tj. kapalné vertikální srážky. Déšť tvoří kapky vody o průměru větším než 0,5 mm. Pokud jsou padající dešťové kapky menší než 0,5 mm, mluvíme o mrholení. Ojedinělé vypadávání dešťových kapek se lidově označuje jako krápání. V roce 2004 byly zaznamenány nad Brazílii a Marshallovými ostrovy největší dešťové kapky na Zemi. Některé z nich dosahovaly až 10 milimetrů.

Déšť hraje hlavní roli v hydrologickém cyklu. Vypařená vlhkost oceánů je přenášena nad jeho části a nad pevninu. Zde pak tato vlhkost kondenzuje a vznikají tu oblaka, z nichž vypadávají srážky v podobě deště. Tento cyklus uzavírají řeky. Ty odvádějí dešťovou vodu zpět do moří a oceánů.

2.1 Tvar, velikost a dynamika

Všeobecně rozšířená představa, podle které mají dešťové kapky tvar slzy či pruhu, je nesprávná (viz.[1]). Tento dojem je způsoben vlastností retinální(sítnicové) perzistence, padající dešťové kapky vypadají spíše jako elipsoid. Drobné dešťové kapky jsou téměř kulaté a větší kapky jsou ve spodní části zploštělé. Tento tvar je výsledkem rovnováhy mezi protichůdnými silami. Povrchové napětí usiluje o minimalizaci kontaktní plochy mezi vzduchem a dešťovou kapkou. Vzniká tak kulatý tvar. Aerodynamický tlak usiluje o horizontální roztažení kapky a dává jí tak tvar elipsoidu. Beard a Chuang [7][8] představili komplexnější a přesnější model zakřivení pravidelné koule, založený na váženém součtu kosinů, s využitím následující rovnice:

$$r(\theta) = a\left(1 + \sum_{n=0}^{10} C_n \cos(n\theta)\right), \qquad (1)$$

,

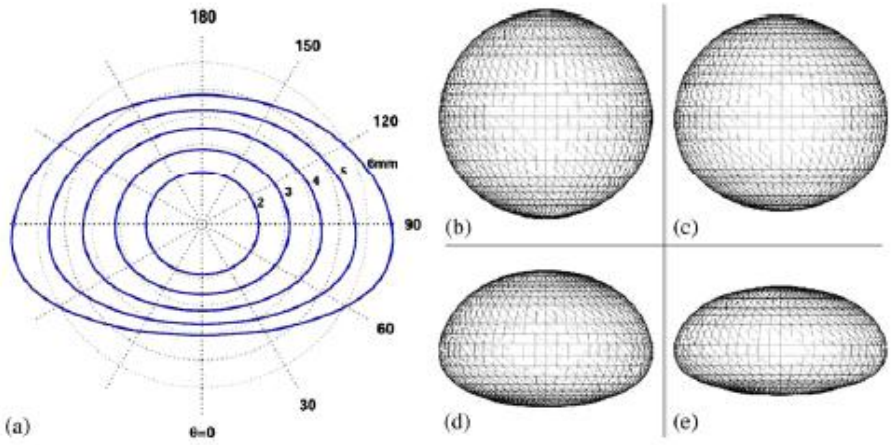
kde a je poloměr nezakřivené koule, umístěné ve středu hmoty kapky. Úhel Θ označuje elevaci, při $\Theta = 0$ směřuje svisle dolů. Několik vzorových tvarových koeficientů Cn je zobrazeno v tab.2.1.

a (mm)	Shape co-efficients ($c_n \times 10^4$) for $n =$										
	0	1	2	3	4	5	6	7	8	9	10
0.5	−28	−30	−83	−22	−3	2	1	0	0	0	0
1.0	−134	−118	−385	−100	−5	17	6	−1	−3	−1	1
3.0	−843	−472	−2040	−240	299	168	−21	−73	−20	25	24
4.5	−1328	−403	−2889	−106	662	153	−146	−111	18	81	31

Tabulka 2.1: Koeficienty tvaru pro různě velké kapky. Převzato z [1]

Na obr.2.1 jsou zobrazeny typické tvary dešťových kapek, vypočítané podle vztahu (1) pro různé poloměry a.

Rychlost pádu dešťové kapky závisí na jejím poloměru. Hodnoty uvedené v tab.2.2 představují rychlosti dešťových kapek, které dosáhly mezní rychlosti při vyrovnání gravitace a třecích sil. Této rychlosti je dosaženo brzy; je to rychlost, kterou má kapka při dopadu na zem.



Obrázek 2.1: Tvary kapek. (a)Porovnání tvarů dešťových kapek s poloměry R = 1, 1.5, 2, 2.5 a 3mm. (b)Tvar nezdeformované kulové kapky o poloměru a=0.5mm. (c)Tvar zdeformované kapky o poloměru a=1mm. (d)Tvar zdeformované kapky o poloměru a=3mm. (e)Tvar zdeformované kapky o poloměru a=4.5mm. Převzato z [1]

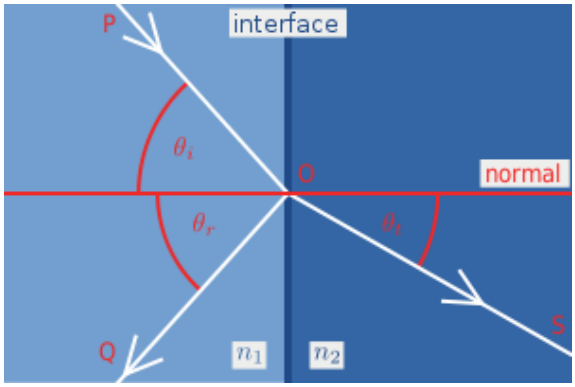
2.2 Optické vlastnosti

V této studii nemáme v úmyslu zabývat se renderováním duhy, z čehož vyplývá, že nemusíme uvažovat vlnový charakter světla. Je fyzikálně správnější zanedbat v případě kapek vlnové vlastnosti světla než vlnovou délku světla, což je náš případ. Můžeme se místo toho zaměřit na vlastnosti definované geometrickou optikou. V této aproximaci je

Spherical drops		Ellipsoidal drops			
Radius (mm)	Speed (m/s)	Radius (mm)	Speed (m/s)	Radius (mm)	Speed (m/s)
0.1	0.72	0.5	4.0	2.5	9.2
0.15	1.17	0.75	5.43	2.75	9.23
0.2	1.62	1.0	6.59	3.0	9.23
0.25	2.06	1.25	7.46	3.25	9.23
0.3	2.47	1.5	8.1	3.5	9.23
0.35	2.87	1.75	8.58	3.75	9.23
0.4	3.27	2.0	8.91	4.0	9.23
0.45	3.67	2.25	9.11		

Tabulka 2.2: Rychlost dešťových kapek v závislosti na jejich poloměrech. Převzato z [1]

světlo považováno za soubor monochromatických paprsků, které se lámou a odrážejí na rozhraní mezi různými médii.



Obrázek 2.2: Snellův zákon odrazu a lomu. Převzato z [12]

Směry odraženého paprsku popisuje na rozhraní zákon odrazu a směr lomeného paprsku zákon Snellův (viz.obr.2.2). Směry odražených a lomených paprsků jsou znázorněny na obrázku 2.3.

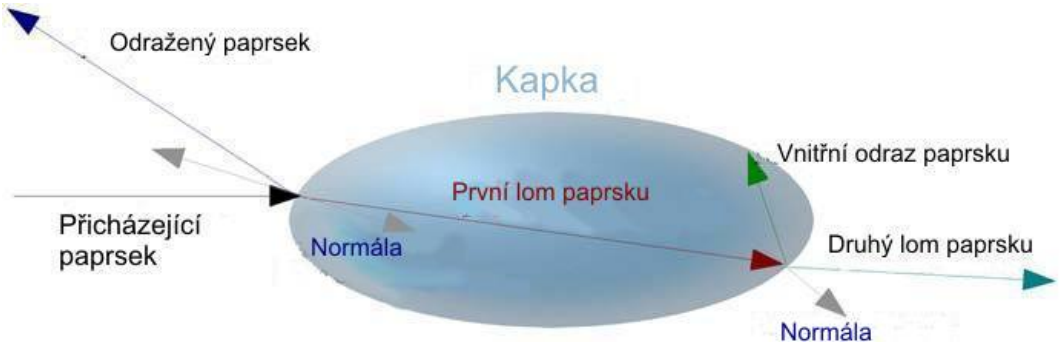
Pro dopadající paprsek, u kterého známe úhel dopadu na rozhraní a polarizaci, je vztah mezi odrazem a lomem dán Fresnelovým faktorem dle vztahu (2) a (3).

$$R_s = \left[\frac{\sin(\theta_t - \theta_i)}{\sin(\theta_t + \theta_i)}\right]^2 = \left[\frac{n_1\cos(\theta_i) - n_2\cos(\theta_t)}{n_1\cos(\theta_i) + n_2\cos(\theta_t)}\right]^2 = \left[\frac{n_1\cos(\theta_i) - n_2\sqrt{1 - \left(\frac{n_1}{n_2}\sin\theta_i\right)^2}}{n_1\cos(\theta_i) + n_2\sqrt{1 - \left(\frac{n_1}{n_2}\sin\theta_i\right)^2}}\right]^2 \quad (2)$$

$$R_p = \left[\frac{\tan(\theta_t - \theta_i)}{\tan(\theta_t + \theta_i)}\right]^2 = \left[\frac{n_1\cos(\theta_t) - n_2\cos(\theta_i)}{n_1\cos(\theta_t) + n_2\cos(\theta_i)}\right]^2 = \left[\frac{n_1\sqrt{1 - \left(\frac{n_1}{n_2}\sin\theta_i\right)^2} - n_2\cos(\theta_i)}{n_1\sqrt{1 - \left(\frac{n_1}{n_2}\sin\theta_i\right)^2} + n_2\cos(\theta_i)}\right]^2 \quad (3)$$

,

Na obr.2.4 můžeme pozorovat příklad lomu na fotografii, pořízené s clonovou rychlostí



Obrázek 2.3: Odraz/lom paprsku v dešťové kapce. Převzato z [1]

1/1000s. Bílé body na fotografované vodní kapce jsou způsobeny bleskem fotoaparátu. Kapka na obrázku právě opustila kohoutek a dosud nedosáhla rovnovážného tvaru.

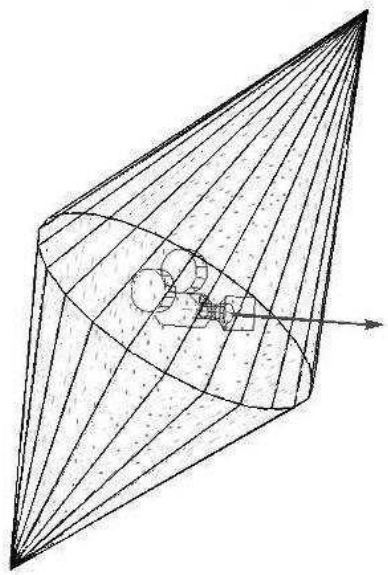
Obrázek 2.4: Fotografie reálné kapky lámající pozadí scény. Převzato z [1]

Kapitola 3

Existující metody simulace deště

V této kapitole si popíšeme některé vybrané metody simulace deště, které s našim tématem úzce souvisí a které jsou v praxi běžně používány. Ve většině počítačových her (např. Unreal Tournament 2004, Need For Speed Underground 2,...) je déšť vykreslován jako jednoduchý částicový systém. Každá částice je zobrazena jako průhledný bílý pruh. Tato metoda zprostředkovává uživateli sice dojem deštivého počasí, ale není příliš realistická. A proto bylo vyvinuto mnoho dalších metod, z nichž některé zde krátce zmíníme.

V počítačové hře Microsoft Flight Simulator 2004 řešili Wang a Wade[2] problém zobrazování deště pomocí dvojitého kužele, který je umístěn kolem pozorovatele (obr.3.1). Na kuželu jsou navinuty textury, které dodávají pocit pohybu. Kužel je nakloněn tak, že odpovídá rychlosti pozorovatele a dodává mu pocit, že kapky padají proti němu(obr.3.2). Tato metoda je rychlejší než částicové systémy, neumožňuje však žádnou interakci mezi deštěm a prostředím. Navíc musí být textura definována pro každý typ srážek zvlášť. Metody navrhované v této studii mají za cíl poskytnout více realismu a flexibility než metody výše zmíněné, aniž by navýšily výpočetní nároky.



Obrázek 3.1: Dvojitý kužel. Převzato z [2].

Obrázek 3.2: Déšť (MS Flight Simulator 2004).

Mnoho studií navrhlo metody, které zobrazují omezené množství pomalu se pohybujících vodních kapek po povrchu s použitím dynamického vytváření cube-map. Tyto metody přinášejí uspokojivé výsledky, ale požadují vysoké výpočetní nároky, částečně kvůli drahému simulačnímu procesu.

Další metody v oblasti počítačového vidění popisovali Starik a Werman[3] a Garg a Nayar[4], [5]. Tyto techniky měly doplnit déšť do videa či ho z něho odstranit(obr.3.3). Pro porozumění problematiky vlivu deště na video byl však zapotřebí velmi přesný teoretický model. Z tohoto důvodu se začala popisovat metoda ray-tracing, která generuje velmi přesné kapky, ovšem za cenu velmi vysokých nároků.

Obrázek 3.3: Snímek s deštěm vytvořeným podle metody Garga a Nayara. Převzato z [4].

Další dílo představené Langrem a kol.[6] navrhlo jednoduchou metodu pro zakřivení imaginárního pozadí, které umožní dojem kapek na předním skle, s použitím algoritmu s velmi nízkými nároky bez jakékoliv souvislosti s fyzikálními vlastnostmi dešťových kapek. Tato metoda tedy postrádá realismus.

3.1 Simulace a zobrazování deště v reálném čase

V této podkapitole si detailněji popíšeme metodu simulace deště, ze které vychází naše řešení a následná implementace. Jak autoři článku [1] uvádějí, výpočet Fresnelova faktoru ukazuje, že odraz se podstatně podílí na barvě povrchu pouze pro některé úhly dopadu. Pro dešťovou kapku to znamená, že je odraz viditelný pouze na okraji kapky.

Pro úhly dopadu menší než 67,5 ° je vliv odrazu světla na vzhled kapky menší než 10%. V aplikaci se typická dešťová kapka s poloměrem 1 mm zobrazuje na obrazovce jako zakřivený kruh s poloměrem 10 pixelů. Pro takovou dešťovou kapku bude odraz podstatný (nad 10%) pouze pro nejkrajnější pixel na těle kapky. Na základě této úvahy se považuje za opodstatněné zanedbat podíl odrazu světla na vzhledu dešťové kapky a zaměřit se na správné stanovení lomu.

Dešťové kapky jsou zobrazovány jako billboardy (malé čtyřúhelníky, které jsou vždy orientovány ke kameře). Obrys dešťové kapky je dán maskou, předem vypočítanou pomocí vztahu (1) pro požadovaný poloměr kapky. Vzhled jednotlivých kapek se vypočítá pomocí fragment shaderu.

Obraz viděný skrz vodní kapku je otočeným a zakřiveným širokoúhlým obrazem pozadí, jak je znázorněno na obr.2.4. Abychom simulovali tento efekt, použijeme funkce

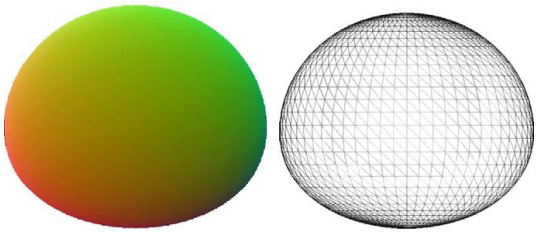
grafického hardwaru pro vykreslení do textury (angl. Render to texture) a získáme tak texturu, která je namapována na každou dešťovou kapku ve fragment shaderu.

Během předvýpočtu vygenerujeme masku pro požadovaný poloměr a uložíme ji do textury. Během běhu programu je scéna pro každý vykreslovaný snímek zachycována na širokoúhlou texturu. Vzhled každého pixelu dešťové kapky je vypočítáván pomocí následujícího postupu:

- s použitím masky určíme, zda je pixel uvnitř nebo vně dešťové kapky;
- je-li uvnitř, použijeme masku k určení směru lomeného vektoru;
- najdeme pozici v zachycené textuře, když zde není žádná změna směru přicházejícího paprsku (viz.obr.3.5);
- v prostoru obrazu přidáme lomený vektor do pozice nalezené v předchozím bodě;
- extrahujeme požadovaný pixel na toto místo.

3.1.1 Předvýpočet masky

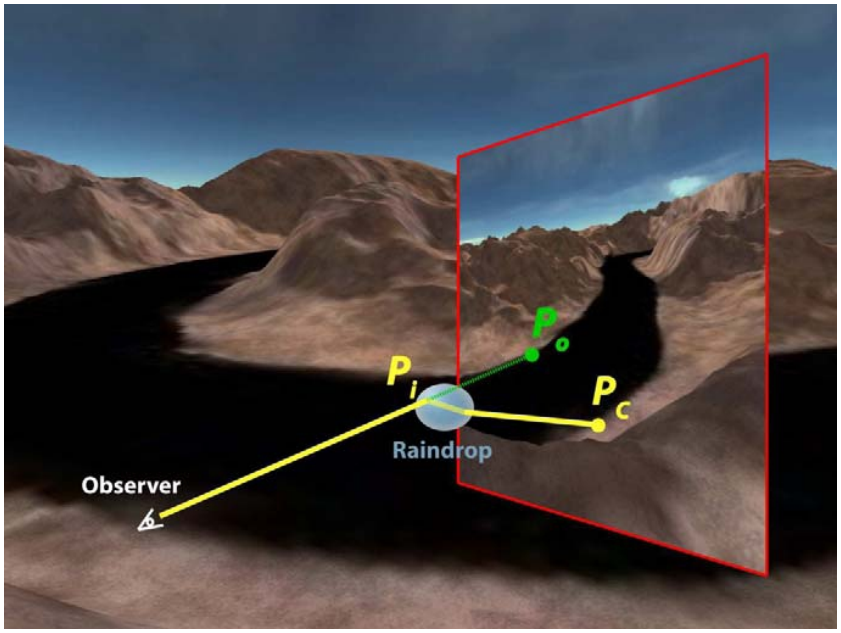
Pomocný program použije vztahu (1) k výpočtu trojrozměrného tvaru dešťové kapky, jejíž poloměr je předán jako parametr. Pro každý pixel tohoto tvaru je vektor lomu předvypočítán a uložen do textury 512 x 512 (obr.3.4). Masku můžeme také získat z libovolného trojrozměrného modelu dešťové kapky. Ve fragment shaderu je maska současně použita k vytvarování kapky a k určení vektoru lomu za nízkých výpočetních nároků – pomocí jednoduchého nahlédnutí do textury. Místo předvýpočtu této masky také můžeme užít funkci lomu CG a vypočítat vektory lomu při běhu programu. Nevýhodou tohoto přístupu je, že zobrazuje dešťové kapky jako dokonalé koule, neboť pro nepravidelné tvary vyžaduje hledání bodu, odkud paprsek vychází z dešťové kapky, z toho plynou velmi vysoké nároky.



Obrázek 3.4: Levý: Maska textury předvypočítaná pro dešťovou kapku o poloměru 1.5mm. Pravý: 3D pohled na dešťovou kapku o poloměru 1.5mm. Převzato z [1].

3.1.2 Zachycení scény na texturu

Kamera je umístěna na stejném místě jako pozorovatel, je stejně orientována a má velmi široký zorný úhel. Tato kamera užívá perspektivní projekci. Textura vytvořená touto kamerou je umístěna na plochu za dešťovými kapkami (jak je znázorněno na obr.3.5). Uvažujeme-li velikost a rychlost billboardů, na které bude tato textura namapována, ukazuje se 512 x 512 jako dostačující rozlišení. Vizuálně dostačující je také filtrování nejbližšího souseda bez použití techniky antialiasingu.



Obrázek 3.5: Získání barvy pixelu dešťové kapky ze zachycené textury. Převzato z [1].

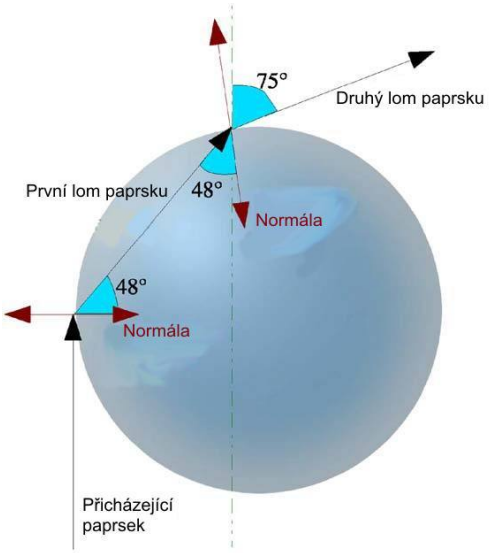
3.1.3 Určení barvy pixelu

Pro každý pixel Pi dešťové kapky extrahuje fragment shader pixel, který je lomený směrem k pozorovateli ze zachycené textury (obr.3.5). Fragment shader nejprve určí, který pixel Po v zachycené textuře je obrazem scény, viděné pozorovatelem ve směru Pi. Po je pixel, který by byl viděn, pokud by kapka měla stejný index lomu jako vzduch. Vektor lomu je pak extrahován z textury masky a spojen do místa Po – obdržíme tak pixel Pc, který udává barvu Pi. Pixelu Po je využito pouze ke stanovení místa nelomeného pixelu v textuře, což znamená, že přímo neovlivňuje konečnou barvu Pi.

3.2 Fyzikální parametry

3.2.1 FOVy kamery

Index lomu pro vzduch je 1, pro vodu 1,33. Na okrajích kapky, kde je odchylka lomu maximální, je úhel mezi paprskem přicházejícím od pozorovatele a kolmicí k povrchu 90 °, jak je znázorněno na obr.3.6. S využitím Snellova zákona dojdeme k závěru, že úhel mezi příchozím paprskem a vnitřně lomeným paprskem je 48 °. Kolmice k bodu, odkud paprsek vychází z kapky, svírá úhel 6 ° s původním příchozím paprskem. Lomený paprsek svírá s touto kolmicí úhel 48 ° a proto se ve vzduchu lomí zpět pod úhlem 81 ° ke kolmici (opět využíváme Snellova zákona) a 75 ° od původního příchozího paprsku. Zorné pole dešťové kapky je tedy široké 150 °. Tuto hodnotu použijeme pro nastavení parametrů kamery FOVy.



Obrázek 3.6: Maximální lom v dešťové kapce. Převzato z [1].

3.2.2 Fyzikální aproximace

Abychom dosáhli fyzikálně přesných výsledků, museli bychom provést ray-tracing se všemi objekty na scéně, což však stěží můžeme provést v reálném čase. Skutečnost, že užíváme pouze jedné textury pro všechny dešťové kapky, představuje malou aproximaci k fyzikálním zákonům, která vede k obrovskému nárůstu renderovací rychlosti. Protože textura není generována v přesné pozici, ve které se nachází kapka, neobsahuje to, co kapka skutečně "vidí". V některých případech může tento fakt vyústit v nezjištěné zastínění či v další deformaci mapované textury. Při simulaci deště jsou kapky velmi malé a pohybují se velmi rychle, tato aproximace tedy nepředstavuje závažnou nevýhodu.

Při našem postupu předpokládáme (na základě výpočtu Fresnelova faktoru), že odraz světla můžeme bezpečně zanedbat vzhledem k jeho minimálnímu vlivu na vzhled dešťových kapek (ovlivněny jsou pouze nejkrajnější pixely na okraji kapky). Vzhled těchto pixelů nebude s využitím naší metody fyzikálně správný; vzhledem k minimálnímu dopadu na vizuální dojem však tento fakt můžeme ignorovat.

3.2.3 Retinální perzistence sítnice

V předchozím textu jsme popsali univerzální model dešťové simulace, který nezohledňuje vnímání ze strany pozorovatele. Kvůli retinální perzistenci vnímá lidské oko nebo kamera dešťové kapky často jako pruhy. Můžeme pozorovat dva mírně odlišné jevy – kamera zachycuje obraz v diskrétním čase, zatímco lidské oko pracuje v čase spojitém.

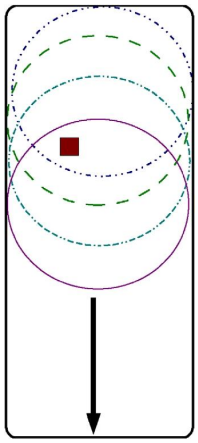
Na fotografii nebo filmu se dešťové kapky zobrazují jako pruhy následkem clonové rychlosti kamery. Zatímco je clona otevřená, spadne kapka o několik centimetrů níž a zapůsobí na film na krátkou vertikální vzdálenost. Tento efekt se většinou označuje jako "motion blur" a nebyl by viditelný pro ideální kameru s nekonečně malou clonovou rychlostí.

Oko pozorující reálný déšť se chová odlišně, ovšem se stejným výsledkem. Oko nemá clonu, ale když se vytvoří obraz na sítnici, trvá 60-90 ms, než se ztratí. Během této časové prodlevy kapka pokračuje v padání a její jednotlivé polohy tvoří na sítnici souvislou sekvenci obrazů a vzniká tak efekt perzistence.

Lidské oko není utvářeno tak, aby vnímalo kapky téměř dokonale kulaté; zdá se tedy, že náš model, ačkoliv je fyzikálně správný, postrádá realismus. Rozšíříme tedy model tak, že zohledníme retinální perzistenci a budeme generovat pruhy založené na přesném modelu kapek.

Abychom mohli simulovat tento efekt, budou dešťové částečky přetvarovány do vertikálních pruhů. Každý pixel v pruhu je ovlivněn postupujícími polohami kapky, jak je znázorněno na obr.3.7. Použitý fragment shader je modifikován následovně – pro každý pixel:

- vypočítáme lom pro několik zvolených poloh kapky;
- z těchto hodnot uděláme průměr;
- snížíme hodnotu α, protože každý pruh je výsledkem pohybu jedné kapky.

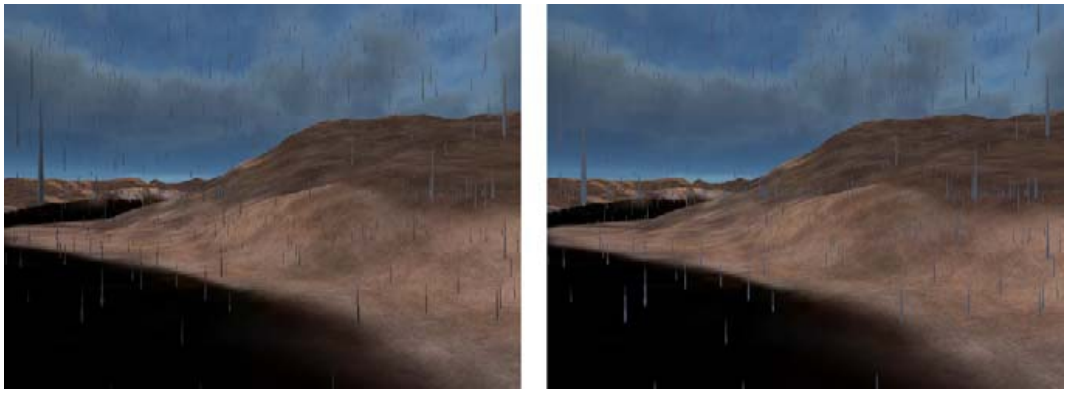


Obrázek 3.7: Ukázka vzorkování jednoho pixelu dešťové kapky. Převzato z [1].

Obr.3.8 dokládá význam tohoto rozšíření ve srovnání se statickými pruhy. Obrázek vlevo využívá metodu zde popsanou, zatímco na obrázku vpravo je na pruhy použita jediná barva. Barva použitá pro statické pruhy je nastavena tak, že vzhled pruhů v pravých horních rozích obou obrázků si co nejpřesněji odpovídá. Pro účely tohoto srovnání byly pruhy na obou obrázcích nastaveny tak, aby se jevily zcela neprůhledné. Můžeme jasně pozorovat značné rozdíly mezi oběma obrázky v levých dolních rozích, což ospravedlňuje mírně vyšší výpočetní nároky metody výše popsané.

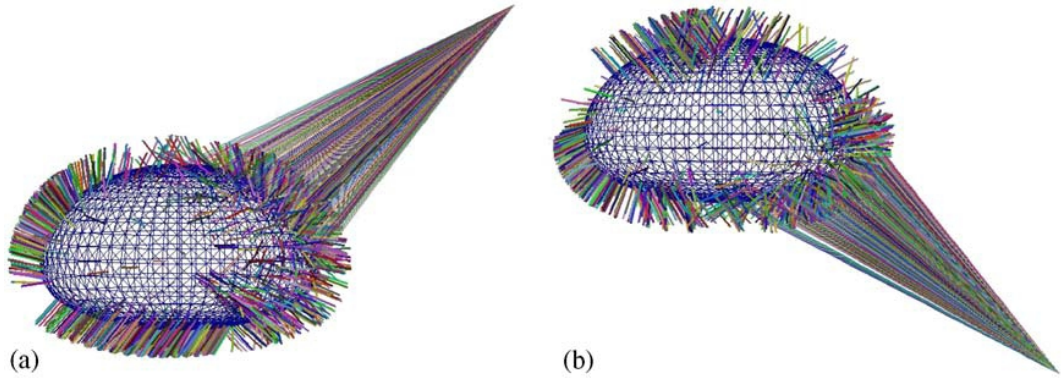
3.2.4 Interakce mezi světlem a dešťovou kapkou

Když déšť padá v blízkosti pouličních světel nebo reflektorů, odrážejí reálné dešťové kapky barvu těchto světelných zdrojů. Optické zákony uvedené v části 2.2 platí, i když je



Obrázek 3.8: Vlevo: S použitím metody persistence popsané v této práci. Vpravo: s použitím statické barvy pruhů. Převzato z [1].

zdroj světla umístěn na scéně. Když je pozorovatel blízko světelného zdroje, mají paprsky vycházející z tohoto zdroje mnohem větší intenzitu než paprsky vycházející z jakéhokoliv jiného místa na scéně. S použitím výše popsané metody by zdroj světla umístěný za pozorovatelem neměl žádný vliv na vytvářené dešťové kapky, protože náš model nepracuje s odrazem světla (který je většinu času zanedbatelný). V případě blízkého světelného zdroje nemůžeme vnější a vnitřní odraz ignorovat, neboť má významný podíl na vzhledu kapky (uvažujeme-li intenzitu paprsků vycházejících ze světelného zdroje).



Obrázek 3.9: Směry paprsků lomených ven z kapky. Převzato z [1].

Obr.3.9 znázorňuje paprsky procházející skrz kapku. Krátké barevné tyčinky znázorňují směr, ve kterém jsou paprsky lámány ven z kapky po 0, 1, 2, nebo 3 vnitřních odrazech (přičemž většina intenzity původního paprsku byla odražena zpět do vzduchu). Můžeme pozorovat, že dokonce po třech vnitřních odrazech můžou být směry vycházejících paprsků klasifikovány do tří skupin:

- zpět ve směru původního paprsku (vnější odraz nebo třetí vnitřní odraz);
- opačná strana kapky, směr nahoru;
- opačná strana kapky, směr dolů.

Zdá se, že jen velmi málo paprsků vychází z kapky do stran.

Nejlepší cestou k vytvoření fyzikálně uspokojivých obrazů by byl výpočet několika vnitřních odrazů, což však můžeme těžko provést v reálném čase a navíc tento postup nevyhovuje našemu billboardovému modelu. Po uvážení výše zmíněných pozorování navrhujeme simulovat interakci světla s dešťovou kapkou pomocí modifikace barvy pixelů dešťových kapek, založené na vzdálenosti mezi dešťovou kapkou a světelným zdrojem.

Světelné paprsky vycházející z dešťové kapky ve směru uživatele by se měly jevit intenzivnějšími než ostatní. Tento jev můžeme vysvětlit pomocí následujícího vztahu:

$$A_1 = C_L * DistFact * \overrightarrow{V_{P->E}} \cdot \overrightarrow{V_N},$$

kde

- C_L je barva zdroje světla,
- DistFact je útlumový faktor založený na vzdálenosti mezi zdrojem světla a uvažovaným pixelem,
- $V_{P \to E}$ je směr paprsku mířícího od pixelu k oku,
- V_N je směr normály kapky u uvažovaného pixelu.

A_1 definuje maximální modifikaci barvy, která může být použita na pixel, přičemž intenzivnější se jeví ty pixely, jejichž normála směřuje k uživateli.

Vycházející paprsky mohou být vytvářeny vnějším odrazem a vracet se zpět ve směru paprsku přicházejícího ze světelného zdroje – platí pak následující vztah:

$$A_2 = max(0, \overrightarrow{V_{P->L}} \cdot \overrightarrow{V_N}),$$

kde

- V_N je normála kapky u uvažovaného pixelu,
- $V_{P \to L}$ je směr paprsku vycházejícího od uvažovaného pixelu ke světelnému zdroji.

A_2 je použito pro vnější odraz; v tomto případě se paprsek vrací zpět v původním směru, skalární součin je kladný. To zabezpečuje fakt, že čím přímější je odraz paprsků, tím více energie získají.

Vycházející paprsky mohou být také výsledkem jednoho nebo dvou vnitřních odrazů a mohou vycházet kolmo k příchozímu paprsku na svislé rovině dopadu. Platí vztah:

$$A_3 = max(0, (-\overrightarrow{V_{P->L_{2D}}}) \cdot \overrightarrow{V_{N_{2D}}} * (1 - (-\overrightarrow{V_{P->L}}) \cdot \overrightarrow{V_N})),$$

kde

- V_N je normála kapky u uvažovaného pixelu,
- $V_{P \to L}$ je směr paprsku vycházejícího od uvažovaného pixelu ke světelnému zdroji,
- $V_{P \to L_{2D}}$ je normalizovaný průmět předchozího paprsku na horizontální rovinu,
- $V_{N_{2D}}$ je normalizovaný průmět normála pixelu na horizontální rovinu.

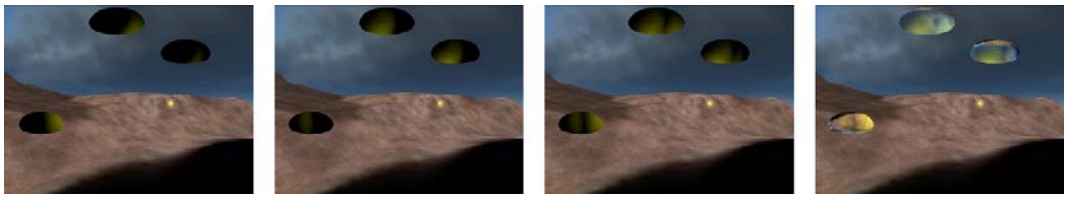
A_3 je použito pro vnitřní lomy. Skalární součin v horizontální rovině zajišťuje, že paprsky nejsou lámány do stran, což by se také podle obr.10 nemělo stávat. Druhý skalární součin podporuje paprsky jdoucí kolmo nahoru a dolů. Nakonec jsou výše uvedené vztahy sjednoceny do vzorce, který používáme pro osvětlení našich dešťových kapek:

$$C_F = (C_O * C_{amb}) + \sum_{lights} (A_1 * (A_2 + A_3)),$$

kde

- C_F je konečná barva pixelu,
- C_O je barva extrahovaná z textury,
- Camb je barva okolního světla na scéně.

Obr.3.10 znázorňuje vliv každého z těchto členů na extrémně velké dešťové kapky. Tento vzorec umožňuje vizuálně uspokojivé výsledky. V této technice můžeme pracovat se dvěma bodovými zdroji světla bez jakékoliv výkonnostní ztráty (tento počet může být velmi jednoduše dále rozšiřován). Toto rozšíření je zcela v souladu s rozšířením o retinální perzistenci, popsaném v části 3.5.3.



Obrázek 3.10: Zleva doprava: Vnější odraz($A_1 * A_2$); vnitřní odraz($A_1 * A_3$); součet obou odrazů; konečný výsledek. Převzato z [1].

Kapitola 4

Návrh řešení

Jak již bylo uvedeno ve 3. kapitole této bakalářské práce, existuje mnoho způsobů vykreslování deště v počítačové grafice. K implementaci jsme vybrali metodu, která je teoreticky popsána v kapitole třetí. Metoda totiž v sobě spojuje možnost vykreslit déšť velmi realisticky a přitom bez navýšení výpočetních nároků. Této metodě se věnuje dílo [1], ze kterého vycházíme při tvorbě této bakalářské práce.

Výsledkem implementace této metody by měla být funkční aplikace, která zobrazí scénu s 3D prostředím. Aplikace se bude moci zobrazit jak v okně, tak ve full screen módu. Prostředí se bude skládat ze skyboxu a jeho textur, který je efektním pozadím pro naši scénu, jelikož úplně změní dojem uživatelů této grafické aplikace, oproti černému pozadí. Dále bude prostředí realizováno pomocí statické krajiny, která bude vygenerována pomocí Terragenu, což je generátor terénu, pomocí něhož je uživatel schopen vytvářet velmi realistické výjevy krajin. V tomto prostředí se bude vykreslovat výsledná simulace deště, na kterou bude v této práci kladen důraz. Simulace déště bude vytvářena pomocí částicového systému. Jednotlivé kapky, pak budou texturovány v závislosti na prostředí a lomu paprsku skrz dešťovou kapku. Toto bude realizováno pomocí shaderu. Lom paprsku bude pro zjednodušení vypočítáván pro kapku tvaru koule.

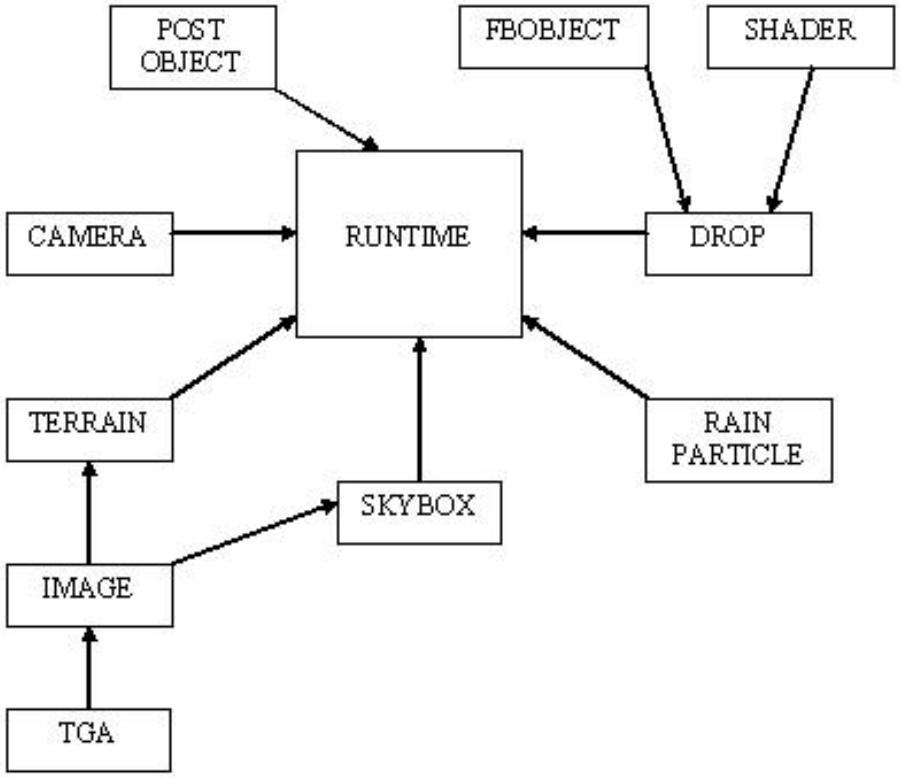
Tvar kapky bude moci uživatel volit mezi čtyřmi tvary, které jsou v tab.2.1. Stejně tak bude možnost měnit velikost a počet kapek v částicovém systému. Uživatel aplikace se bude moci po scéně také volně pohybovat pomocí klávesnice.

4.1 Základní blokové schéma aplikace

V této kapitole představíme základní blokové schéma aplikace a stručně popíšeme návrh jednotlivých funkčních struktur, z kterých se toto blokové schéma skládá.

- RUNTIME

  Třída Runtime bude základním prvkem aplikace. Dojde zde k počáteční inicializaci a nastavení prostředí glut a jeho callback funkcí, propojí veškeré částí aplikace ve funkční celek a bude z ní aplikace spuštěna. Z této třídy bude voláno také vykreslování celé scény. Tedy vykreslování skyboxu, terénu i částicového systému deště. Bude také umožňovat uživateli pohyb po 3D scéně, a to pomocí klávesnice.



Obrázek 4.1: Základní blokové schéma aplikace

Třída bude implementovat i metody pro natočení, pohyb, změnu pozice či nastavení pohledu kamery a jiné.

- TERRAIN

  Název třídy sám napovídá, že zde se načtou textury a model terénu, který jsme již dříve vygenerovali pomocí generátoru krajin, Terragenu.

- SKYBOX

  Třída Skybox načte a vykreslí model a textury skyboxu, který bude tvořit okolní prostředí.

- TGA

  Bude to pomocná třída načítající hlavičku a data obrázku, která budou dále vstupními daty pro třídu Image.

- IMAGE

  Image načte samotný obrázek podle dat, které přijme od třídy Tga.

- RAIN PARTICLE

  Tato třída vytvoří částicový systém. Částicový systém bude generovat N částic, které budou různých velikostí. Velikost jednotlivých částic bude generována náhodně, podle Gaussova rozdělení. Na velikosti těchto částic bude závislá rychlost, kterou budou částice padat dolů. Čím větší bude velikost částice, tím větší bude její rychlost. Částice budou vznikat opět náhodně v horní části uvnitř boxu o určitých rozměrech. Budou se pohybovat směrem dolů a v okamžiku, kdy částice dopadnou pod určitou úroveň, se začnou generovat opět v horní části.

- DROP

  Třída Drop bude reprezentovat lomené paprsky jedné samostatné dešťové kapky. Podle zvoleného tvaru se zde vytvoří model dešťové kapky. Třída bude obstarávat vykreslení modelu do textury, která se použije pro určení refrakčních paprsků pro každý partikl.

- SHADER

  Tato třída bude sloužit jako pomocná třída. Pro práci se shadery je totiž zapotřebí provést množství nutných kroků. Prvním bude vytvoření prázdného objektu, který bude sloužit jako kontejner pro shader. Do něj bude nahrán zdrojový kód shaderu. Shader bude následně zkompilován. Dalším krokem je opět vytvoření prázdného objektu, tentokrát však pro hlavní program. Do tohoto programu se přidají dříve vytvořené objekty. Na závěr se program zkompiluje a nastaví na aktivní.

- FBOBJECT

  Třída FBObject zapouzdřuje vykreslování do textury. Pomocí FBObject se bude vykreslovat 3D scéna, tj. skybox a terén, do 2D textury o rozměrech 512x512, kterou budeme dále využívat při výpočtu barev jednotlivých pixelů kapky.

- POST OBJECT

  Post object slouží k vykreslení výsledné textury třídy FBObject.

Kapitola 5

Implementace

V této kapitole nejdříve představíme zvolené implementační rozhraní, následně popíšeme použité knihovny. Poté přejdeme na popis samotné implementace aplikace. Podkapitoly se budou věnovat těm částem, které považujeme za nejzajímavější, tj. vykreslování do textury, částicovému systému, výpočtu lomu v kapce a perzistenci sítnice.

Při volbě implementačního jazyka jsme respektovali zadání mé bakalářské práce. Zvolili jsem tedy jazyk C/C++, který splňuje v tomto případě nutnost rychlého běhu aplikace a využití grafického hardwaru. Pro tvorbu aplikace byl vybrán standard OpenGL, s kterým jsme se seznámili v předmětu X36ZPG. Bylo by možno použít také DirectX, které využívá téměř 90% počítačových her. Součástí DirectX jsou pomocné funkce umožnující např. načtení shaderů, textur. Bohužel zatím ohledně něj nemáme dostatečné znalosti na psaní bakalářské práce. Aplikace byla vyvíjena v prostředí Microsoft Visual Studio 2008.

5.1 Použité knihovny

Jelikož je aplikace implementován v jazyku C++. Pro tvorbu aplikace byly použity knihovny napsané taktéž v C++, a to OpenGL a jeho nadstavby GLUT, GLU a GLEW. Vzhledem k tomu, že využívám také windows.h, bude aplikace spustitelná pouze na systému Windows. Po drobných úpravách includu by aplikace však měla běžet i na systému Linux. Práce, ale na tomto systému nebyla odzkoušena.

5.1.1 OpenGL

Knihovna OpenGL byla navržena firmou SGI. Byl kladen důraz převážně na její použitelnost jak na různých typech grafických akcelerátorů, tak i tehdy, pokud na určité platformě není žádný grafický akcelerátor nainstalován. OpenGL je používáno jako aplikační programové rozhraní (API) k akcelerovaným grafickým kartám resp. celým grafickým subsystémům. V současnosti lze tuto knihovnu OpenGL použít na různých verzích unixových systémů, OS/2 a na platformách Microsoft Windows.

Knihovna OpenGL je použitelná téměř v libovolném programovacím jazyce. K dispozici je primárně pro jazyky C a C++ hlavičkový soubor, v kterém jsou deklarovány

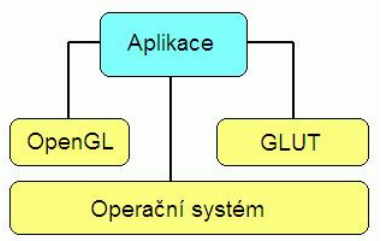
nové datové typy, některé symbolické konstanty a sada kolem 120 funkcí, které tvoří vlastní rozhraní.

Jednou z výhod knihovny OpenGL je její nezávislost na použitém operačním systému, grafických ovladačích a správcích oken. Z tohoto důvodu však neobsahuje funkce pro práci s okny, pro vytváření grafického uživatelského rozhraní(GUI) ani pro zpracování událostí. Tyto funkce lze zajistit buď přímo voláním funkcí příslušného správce oken, nebo lze použít některou z nadstaveb, například knihovnu GLUT (OpenGL Utility Toolkit), o které se více dozvíte v sekci 5.1.2.

Funkce OpenGL poskytují pouze základní rozhraní pro přístup ke grafickým akcelerátorům. Opět však existují rozšiřující knihovny používané společně s OpenGL. Jednou ze základních knihoven je knihovna GLU (sekce 5.1.3).

5.1.2 GLUT

Knihovna GLUT definuje a implementuje aplikační rozhraní pro tvorbu oken a jednoduchého GUI, přičemž je, stejně jako OpenGL, systémově nezávislá. To znamená, že pro práci s okny se na všech systémech používají vždy stejné funkce, které mají stejné parametry. Okenní systém nemá podporu pro komponenty jako např. tlačítka, posuvníky apod. Nabízí pouze jednoduchou práci s okny, klávesnicí a myší. Knihovna GLUT také obsahuje funkce pro vykreslování bitmapového a vektorového písma. Nezávislost na operačním systému i platformě je podložena také faktem, že se ve všech funkcích knihovny GLUT používají pouze základní datové typy jazyka C. Na obrázku 5.1 je nakresleno schéma začlenění knihovny GLUT do grafických aplikací.



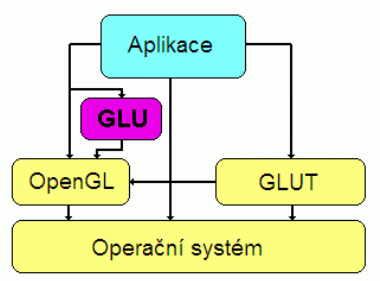
Obrázek 5.1: Začlenění GLUT v aplikaci. Převzato z [10]

5.1.3 GLU

Knihovna GLU umožňuje využívat tesselátory, evaluátory (výpočet souřadnic bodů ležících na parametrických plochách) a vykreslovat kvadriky (koule, válce, kužely a disky). Obsahuje pouze základní sadu jednoduchých kreslících operací. Tato sada operací je sice pro vytváření obrazů dvourozměrných i trojrozměrných scén úplná, ale jedná se převážně o základní funkce. Obsahuje ale také užitečné funkce, které umožňují práci s display-listy.

5.1.4 GLEW

Knihovna GLEW usnadňuje použití extenzí a nových verzí OpenGL na všech platformách. Na Windows umožňuje plně využít možnosti dnešních grafických karet v OpenGL



Obrázek 5.2: Vzájemný vztah mezi knihovnami OpenGL GLU a GLUT. Převzato z [11]

na maximum. GLEW byl testován na různých operačních systémech, např.: Windows, Linux, Mac OS X, FreeBSD, IRIX a Solaris.

5.2 Implementace jednotlivých částí

Nyní si předvedeme, jak již bylo zmíněno v úvodu této kapitoly, části programu, které považujeme za nejzajímavější. Budeme tedy popisovat vykreslení do textury, realizaci částicového systému a předvedeme také shadery, pomocí nichž jsme určili barvy pixelů kapky. Shadery jsme využili také při implementaci rozšíření o jev zvaný perzistence sítnice.

5.2.1 Vykreslování do textury

Vykreslování do textury jsme realizovali pomocí Frame Buffer Object (FBObject). Toto FBObject rozšíření bylo vytvořeno speciálně pro vykreslování do textury. Které pak lze mnohem účinněji a snadněji provádět v porovnání s jinými metodami jako je například pbuffer (Pixel buffer object) či Texture Object.

Musíme nejdříve provést nastavení a stejně jako u ostatních OpenGL objektů vytvořit platný handle.

```
glGenFramebuffersEXT(1, &fbo);
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, _fbo);
```

Poté připojíme Renderbuffer a vytvoříme a nastavíme velikost potřebného úložiště. Následně připojíme Depth render buffer k FBObject.

```
glGenRenderbuffersEXT(1, &_depthBuffer);
glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, _depthBuffer);
glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT, GL_DEPTH_COMPONENT, _width, _height);
glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT,
GL_RENDERBUFFER_EXT, _depthBuffer);
```

Nakonec již dojde k přidání a použití textury pro vykreslení.

```
glGenTextures(1, &_renderTarget);
glBindTexture(GL_TEXTURE_2D, _renderTarget);
glTexImage2D(GL_TEXTURE_2D, 0, format,  width, height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, 0);

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT,
GL_TEXTURE_2D, _renderTarget, 0);
```

Na závěr je potřeba FBObject a Render Buffer vyčistit a odstranit.

Tímto jsme si krátce předvedli práci s FBObject, který jsme využili při implementaci vykreslení do textury.

5.2.2 Realizace částicového systému

Simulace dešťě, jak již bylo mnohokrát řečeno, jsme implementovali pomocí jednoduchého částicového systému. Jehož základní části si nyní popíšeme.

- Pohyb částic

  Částice díky gravitaci padají směrem dolů k záporným hodnotám osy Y. Průběžně je počítána a aktualizována jejich pozice a to na základě spočteného vektoru rychlosti a časového přírůstku mezi snímky. Částice jsou později na této pozici vykreslovány.

- Zánik částic

  Pro každou částici je kontrolována změna souřadnice y. Jakmile hodnota tého souřadnice klesne pod -1, jsou souřadnice této částice přepočítány. Částice se začne generovat na nových souřadnicích.

- Generování nových částic

  Generování částic se děje na základě parametru scény udávajícího množství vygenerovaných částic. Jak je popsáno v pseudokódu, pozice částic jsou generovány náhodně z daných intervalů. Velikost částic je generována podle Gaussova rozložení. Tedy přibližně 66% částic bude mít velikost kolem 1.3mm. Podle velikosti vypočítáváme rychlost, kterou se jednotlivé částice pohybují. Rychlost je závislá na velikosti částic a to podle tabulky 2.2 z druhé kapitoly. Vše jsme jen pro naši potřebu čtyřikrát zpomalili.

```
Generuj_novou_částici(){
  částice.X = Random_z_intervalu [-1,1];
  částice.Y = Random_z_intervalu [1,1.25];
  částice.Z = Random_z_intervalu [-1.5,1.5];
```

```
  částice.Velikost = Gauss (1,1.3,8)
  částice.Rychlost = přepočet_rychlosti_částič_podle_velikosti();
}

Simulace částicového systému() {
  if (počet_vygenerovaných_částic < zadaný_počet_částic) {
     Generuj_ novou_částici();
     Počet_vygenerovaných_částic = počet_vygenerovaných_částic - 1;
  }
  for ( int i = 0;  i < počet_vygenerovaných_částic;  i++) {
   if ( generovaná_částice [ i ].Y < -1.0)
      Generuj_novou_částici( generovaná_částice [i]);
      generovaná_částice [ i ].Y -= (čas  *  rychlost_generované_částice);
  }
}
```

Aby rychlost padajících částic nezávisela na rychlosti hardwaru, na kterém program běží, je do simulace zahrnut i prvek času, tj. všechny animace jsou násobeny časovým přírůstkem mezi předchozím a nynějším snímkem.

5.2.3 Určení barvy pixelu dešťové kapky

Pomocí následujícího fragment shaderu jsme schopni získat barvu jednoho pixelu dešťové kapky a to postupem, který jsme již uvedli v kapitole 3.2.3. Zde následuje výsledná implementace shaderu.

```
uniform sampler2D sampler0;
uniform sampler2D sampler1;

vec4 refractVec(vec2 texCoord) {
  vec4 refract1 = texture2D(sampler0, texCoord.xy);
  return refract1;
}
void main() {
  vec4 texCoord = gl_TexCoord[0];
  texCoord.xy = ((texCoord.xy / texCoord.w) + 1.0) * 0.5;
  vec4 oColor = vec4(0.0, 0.0, 0.0, 0.0);
  vec4 refract1 = refractVec(gl_TexCoord[1].xy );
    if (refract1.a > 0.0 && refract1.z > 0.0) {
      oColor += texture2D(sampler1, texCoord.xy + (refract1.xy / refract1.z));
      count++;
    }
  }
}
```

Nyní si popíšeme jednotlivé kroky implementované ve zdrojového kódu. Na začátku si přiřadíme do vektoru texCoord vstupní texturovací souřadnice z aplikace. Následně texCoord.xy přiřadíme souřadnice vypočteného průsečíku P0, tj. průsečíku mezi texturou

scény a paprskem. Poté přiřádíme vektoru refract1 směrový vektor odraženého paprsku. Podmínkou if rozhodneme zda je paprsek vně či uvnitř kapky (souřadnice a nabývá hodnot 1 nebo 0), a také zda lomený parsek excituje. Souřadnice z nám totiž pro velké úhly nabývá hodnoty 0, a to z toho důvodu, že na krajích kapky nedochází k refrakci. Pokud jsou tedy obě podmínky splněny, dochází k posunu průsečíku P0 o velikost refrakčního paprsku. Zároveň získáme barvu pixelu, průsečíku Pc, do kterého se paprsek posune.

Tento postup, jak již jsme se zmínili v minulé kapitole, je však zcela správný pouze pro dešťové kapky tvaru koule. Pro ostatní zploštělé tvary kapek by bylo zapotřebí složitějšího výpočtu pro oba lomy paprsku v dešťové kapce. Pro zjednodušení však tento výpočet postačuje.

5.2.4 Perzistence sítnice

V této podkapitole popíši změny oproti předchozímu fragment shaderu. Tyto změny realizují simulaci efektu zvaném perzistence sítnice, o kterém jsme se již teoreticky zmiňovali v kapitole 3.2.3. Implementace se tohoto popisu drží. Změny v kódu způsobují přetvarování dešťových částic do vertikálních pruhů. Fragment shader je pro každý pixel kapky modifikován následovně:

```
uniform sampler2D sampler0;
uniform sampler2D sampler1;
uniform float texelOffset;

vec4 refractVec(vec2 texCoord) {
  vec4 refract1 = texture2D(sampler0, texCoord.xy);
  return refract1;
}

void main() {
  vec4 texCoord = gl_TexCoord[0];
  texCoord.xy = ((texCoord.xy / texCoord.w) + 1.0) * 0.5;
  vec4 oColor = vec4(0.0, 0.0, 0.0, 0.0);
  int count = 0;
  vec2 dTc = vec2(0.0, 0.0);
  for (int i = -3; i < 4; i++) {
    dTc.y = i * texelOffset;
    vec4 refract1 = refractVec(gl_TexCoord[1].xy + dTc);
    if (refract1.a > 0.0 && refract1.z > 0.0) {
      oColor += texture2D(sampler1, texCoord.xy + (refract1.xy / refract1.z));
      count++;
    }
  }
}
```

Oproti minulému fragment shaderu zde přibyl hlavně cyklus for, v kterém se budeme postupně posouvat o i pixelů v textuře. Pro tento pixel získáme vždy pomocí funkce

refractVec() směrový vektor. Pak stejně jak v předchozím fragment shaderu posuneme průsečík P0 o velikost refrakčního paprsku a získáme barvu průsečíku Pc. Získáme tím tedy sedm barev, které poté v programu zprůměrujeme a získáme tak výslednou barvu pixelu.

Kapitola 6

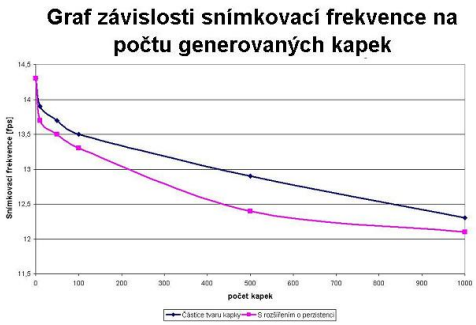
Testování a výsledky

V této kapitole předvedeme vliv jednotlivých parametrů naší simulace na snímkovou frekvenci. Poté představíme dosažené výsledky, demonstrujeme pomocí obrázků vliv jednotlivých parametrů na vzhled simulace a porovnáme výsledné scény z realnými fotografiemi.

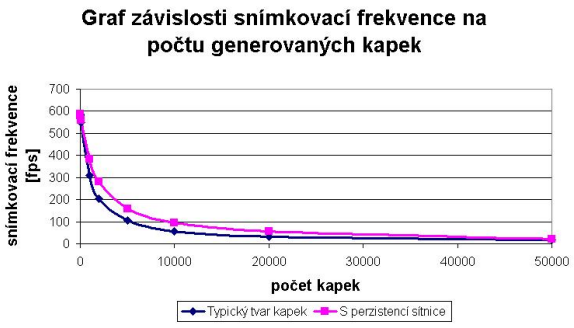
6.1 Měření snímkovací frekvence

Snímkovací frekvence je parametr, který udává plynulost chodu programu na daném hardware. Tohoto ukazatele se využívá při optimalizaci kódu. V následujících tabulkách 6.1, 6.2 a grafech 6.1,6.2 jsou zaneseny hodnoty snímkovací frekvence v jednotkách fps(Frames Pes Second) v závislosti na počtu, velikosti, a typu generovaných dešťových kapek. Měření proběhlo pro defaultní velikost kapky 0.5mm.

Jelikož při měření snímkovací frekvence hrají také velkou roli parametry počítače (převážně procesoru a grafické karty), uvádíme zde hodnoty z dvojího měření. První tabulka hodnot byla získána na notebooku s grafickou kartou nVidia GEFORCE Go 7400 a procesorem DualCore 1.733MHz, druhá na počítači s využitím grafické karty ATi Radeon 3850 a procesoru Core2Duo 2.667MHz.



Obrázek 6.1: Graf závislosti fps na parametrech kapky.



Obrázek 6.2: Druhý graf závislosti fps.

Počet kapek \ Snímkovací frekvence [fps]	Částice tvaru kapky	Částice s rozšířením o perzistenci
0	14,3	14,3
10	13,9	13,7
50	13,7	13,5
100	13,5	13,3
500	12,9	12,4
1000	12,3	12,1

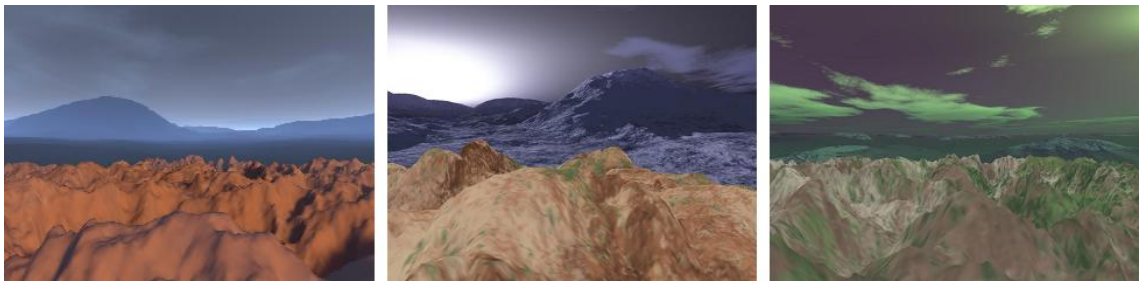
Tabulka 6.1: Hodnoty fps naměřené na notebooku.

Počet kapek \ Snímkovací frekvence [fps]	Částice tvaru kapky	Částice s rozšířením o perzistenci
0	588	588
1	585	585
10	580	581
100	550	558
1000	308	384
2000	205	283
5000	106	160
10000	58	95
20000	32	55
50000	17	22

Tabulka 6.2: Naměřené hodnoty fps na PC.

6.2 Testované scény

Výsledkem práce je aplikace, která umožní simulovat a zobrazovat scény s deštěm v reálném čase. Funkčnost implementace jsme ověřili na mnoha různých scénách. Tři z nich v této kapitole představíme(obr.6.3). Všechny scény(obr.6.4,6.5,6.6,6.7,6.8,6.9) se skládají ze skyboxu a jeho textur a z terénu, který byl vytvořen pomocí generátoru krajin, Terragenu.



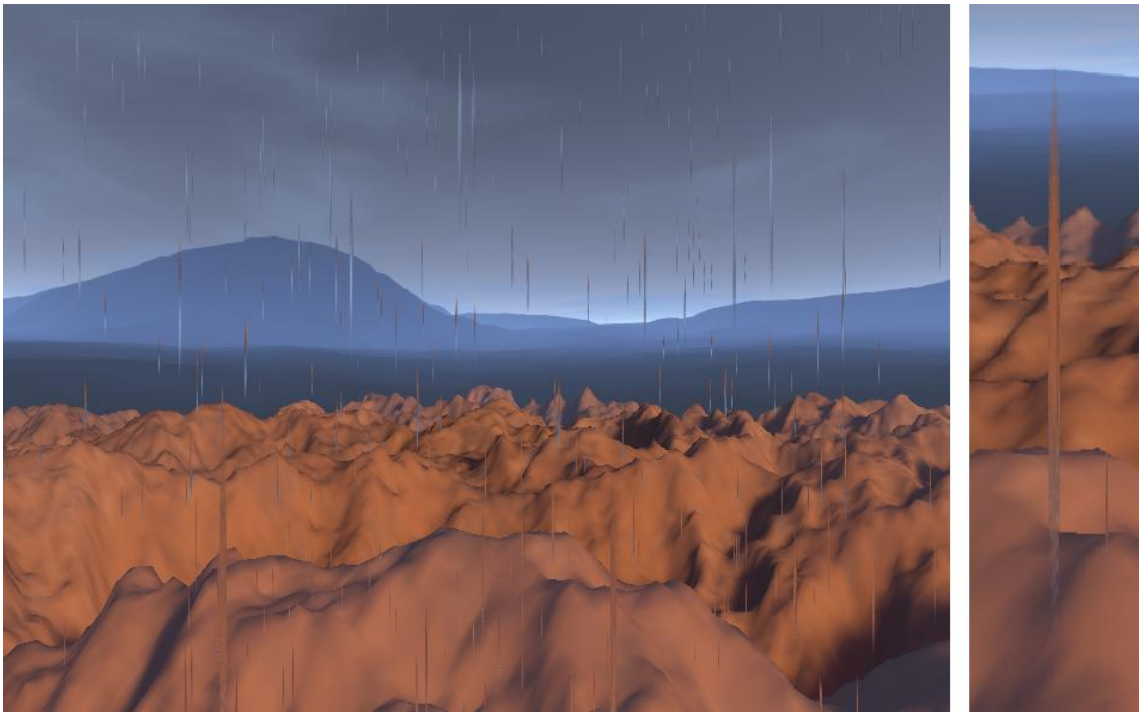
Obrázek 6.3: Testované scény bez simulace deště

6.3 Vliv parametrů na vzhled simulace

Jak jsme již dříve zmínili, součástí implementace je i jednoduché uživatelské rozhraní, pomocí něhož je možno nastavovat jednotlivé parametry dešťových kapek a částicového systému. Proměnlivými parametry je velikost, tvar a počet dešťových kapek. Jaký vliv mají tyto parametry na vzhled výsledné simulace demonstrujeme na obrázcích 6.10, 6.11,6.12.

6.4 Porovnání výsledků se skutečnými fotografiemi

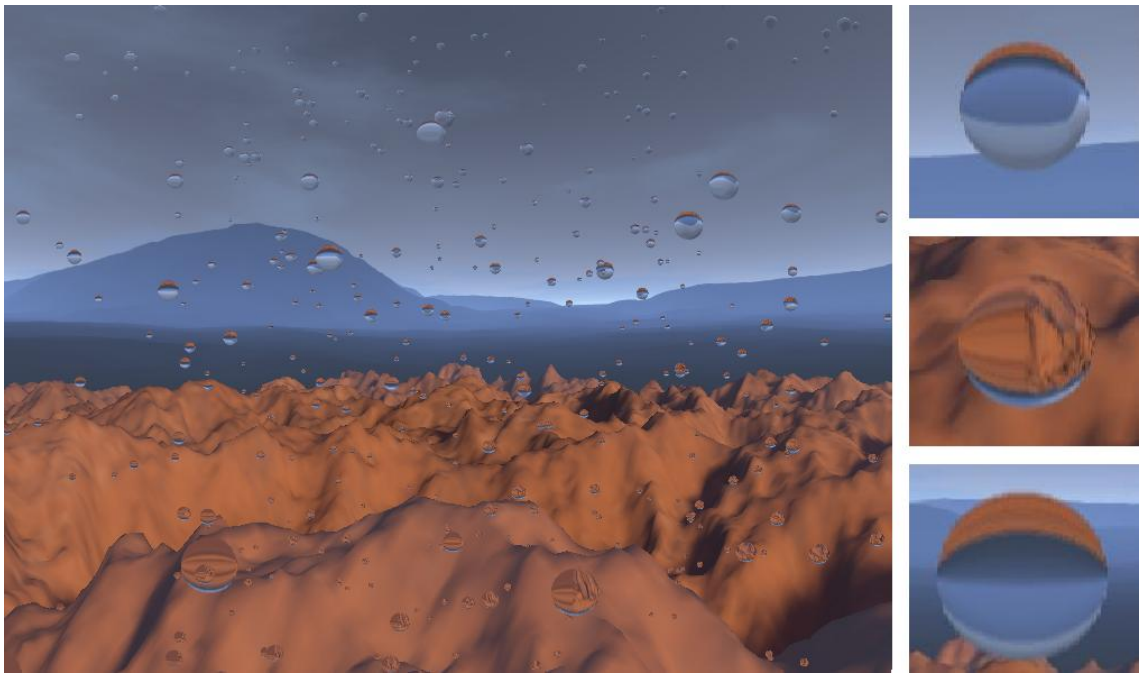
V této části předvedeme závěrečné simulace, které porovnáme s fotografiemi skutečného deště, abychom mohli zhodnotit výsledky naší práce. Na obrázcích 6.13,



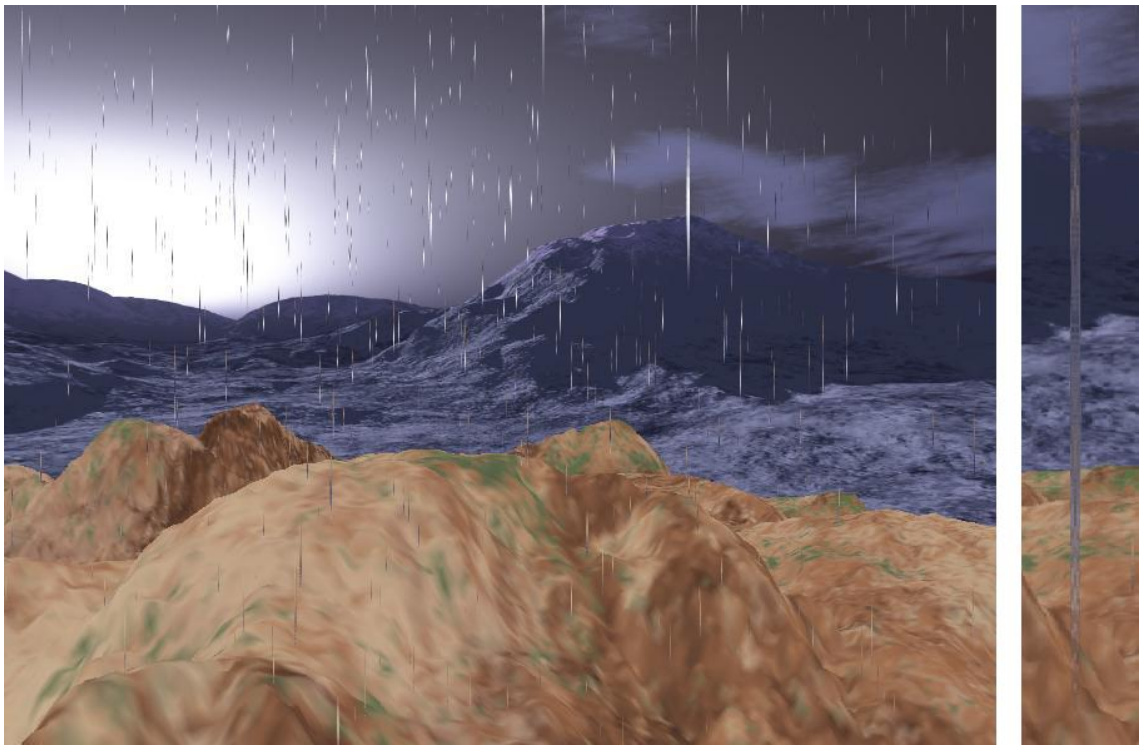
Obrázek 6.4: Testovaná scéna s výslednou simulací deště.

6.14 jsou fotografie reálného deště. Pro porovnání jsou přidány obrázky 6.15,6.16 naší závěrečné simulace.

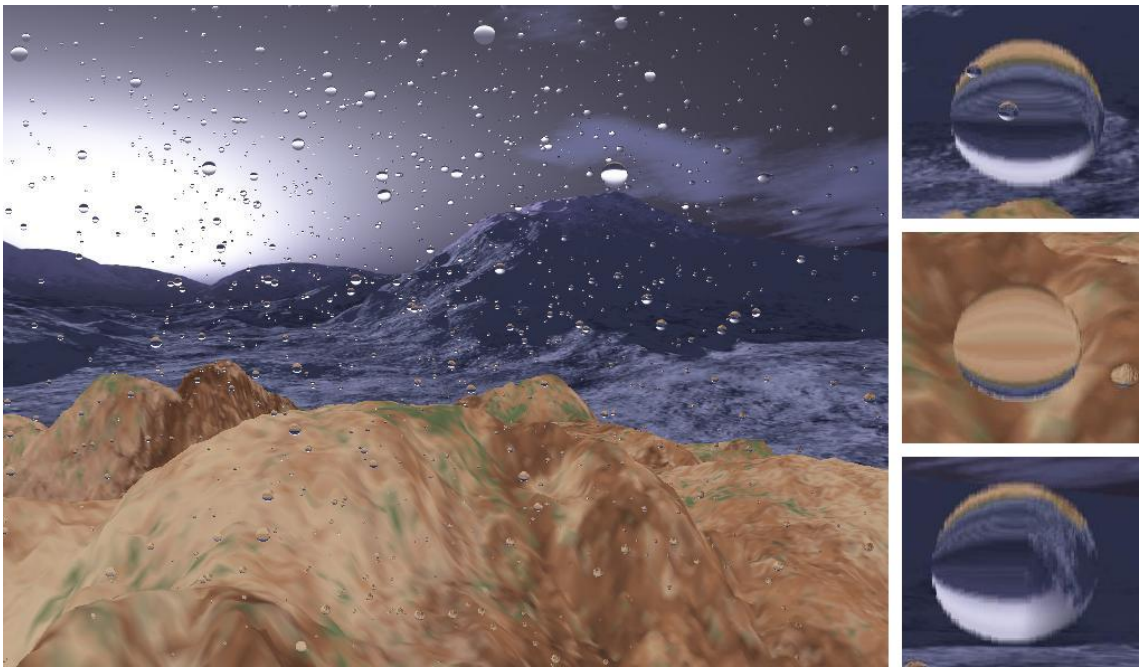
Pomocí změn nastavitelných parametrů jsme se snažili docílit scén, které budou, co nejpodobnější fotografiím skutečného deště. Z přiložených obrázků, se dá považovat simulace deště za zdařilou, avšak je jasné, že stále je co vylepšovat. Jedním z aspektů, kterým se simulace liší od skutečného deště, je například směr paprsků dopadajících na zem. Ve skutečnosti kapky deště nepadají kolmo dolů, ale jsou ovlivňovány řadou faktorů. Jedním z nich je například vítr. Který mění směr padajících kapek.



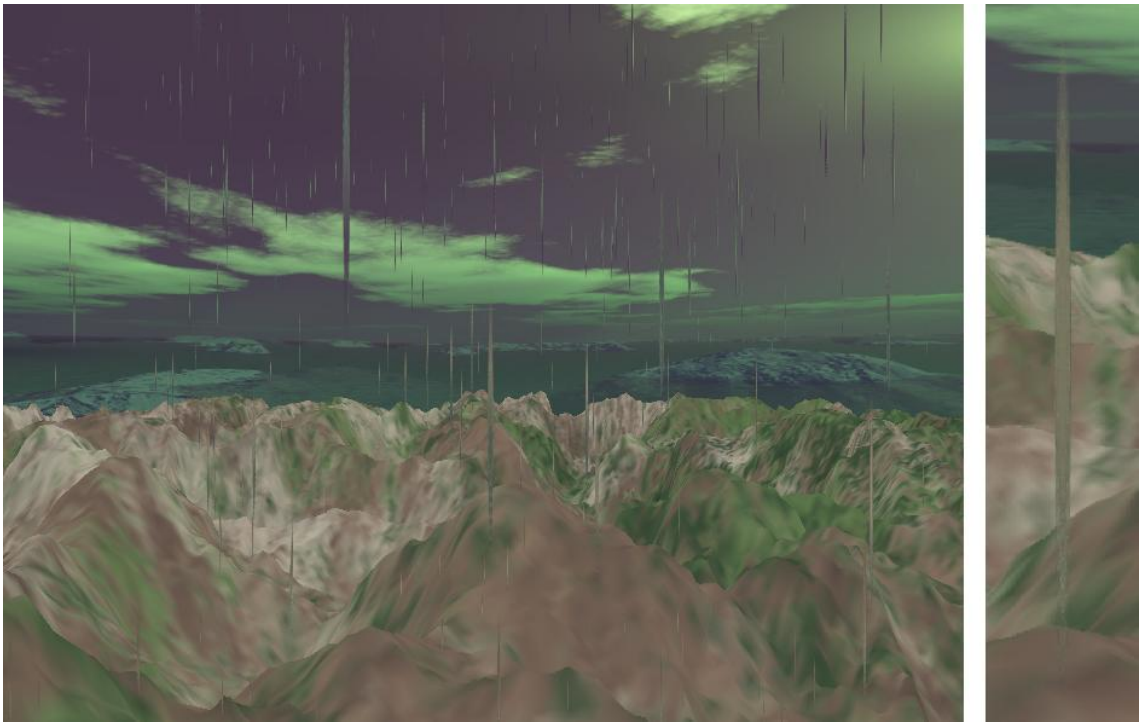
Obrázek 6.5: První testovaná scéna, se simulací částicového systému.



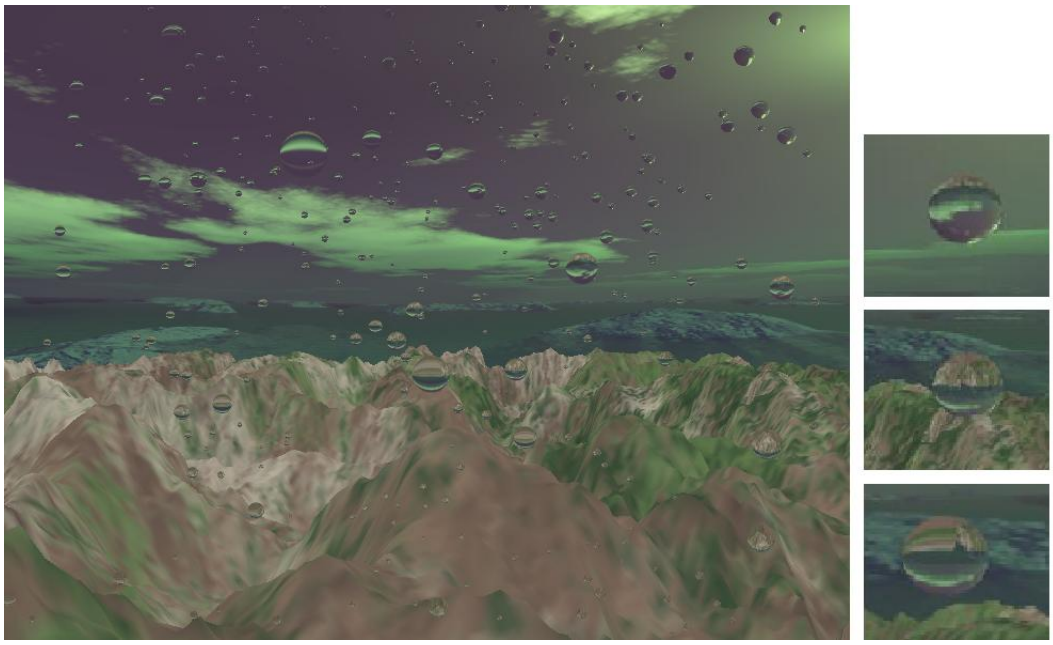
Obrázek 6.6: Testovaná scéna s výslednou simulací deště.



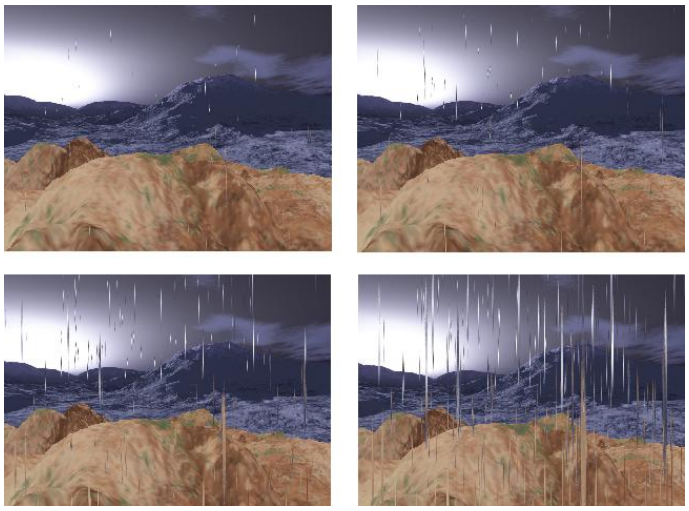
Obrázek 6.7: Druhá testovaná scéna, se simulací částicového systému.



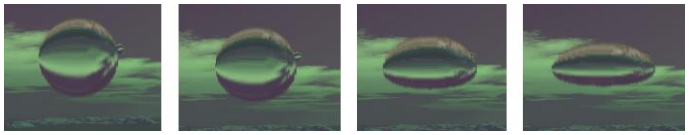
Obrázek 6.8: Testovaná scéna s výslednou simulací deště.



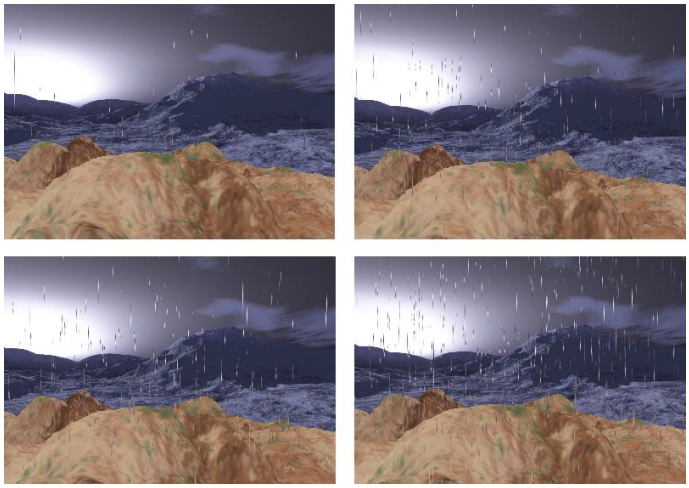
Obrázek 6.9: Třetí testovaná scéna, se simulací částicového systému.



Obrázek 6.10: Ukázka scény při změně velikosti kapek.



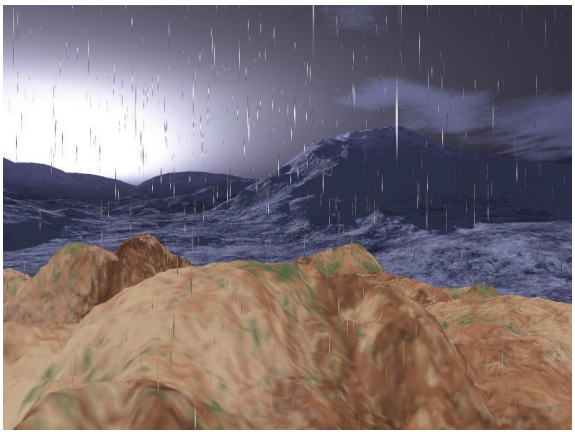
Obrázek 6.11: Zobrazení různých tvarů dešťových kapek.



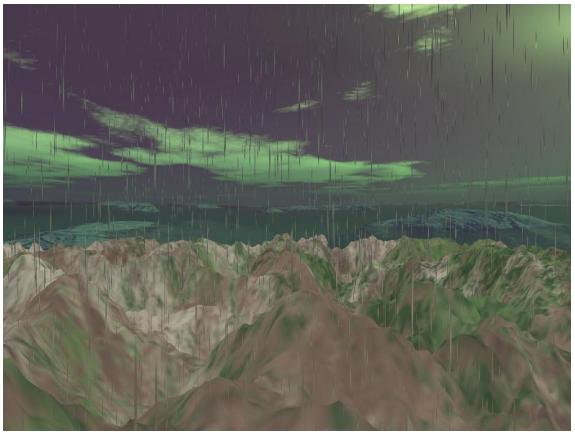
Obrázek 6.12: Demonstrace změny počtu dešťových kapek.

Obrázek 6.13: Fotografie skutečného deště

Obrázek 6.14: Zvěčněná scéna deště



Obrázek 6.15: Výsledná simulace deště umístěná do scény 2



Obrázek 6.16: Třetí scéna s výslednou simulací

Kapitola 7

Závěr

Cílem práce byla implementace metody, která v sobě spojuje realistické vykreslování deště s co možná nejmenším navýšením výpočetních nároků. Tato metoda, která byla popsána v článku [1], se stala základem pro moji bakalářskou práci.

Implementovaný program splňuje všechny body zadání. Výslednou aplikaci by však bylo možno v mnoha ohledech vylepšit či rozšířit. Například možnost vylepšení se přímo nabízí u lomu paprsku kapkou, který byl pro zjednodušení počítán pouze pro kapky tvaru koule. Pro ostatní zploštělé tvary tedy výsledná textura zcela neodpovídá skutečnosti.

Jedním z možných rozšíření by bylo přidaní interakce mezi světlem a deštovou kapkou. V dešťových kapkách by se tedy odrážela barva okolních světelných zdrojů. Toto rozšíření by výsledné simulaci deště ještě přidalo na realističnosti.

Uživatelské rozhraní také není zrovna ideální. Vzhledem k nedostatku času je řešeno trochu provizorně. Realizovat by se jistě dalo vhodnějším způsobem. Nabízí se tu řešení pomocí knihovny GLUI, která byla vytvořena jako nadstavba OpenGL pro tvorbu GUI. Jde o platformově nezávislou knihovnu, která obsahuje funkce pro přidávání mnoha různých ovládacích prvků a může tak uživateli umožnit pohodlné, jednoduché a intuitivní ovládání.

Literatura

[1] ROUSSEAU Pierre, JOLIVET Vincent, GHAZANFARPOUR Djamchid. *Realistic real-time rain rendering.* Computers and Graphics 30, 2006.507–518s.

[2] WANG Niniane, WADE Bretton. *Rendering falling rain and snow.* ACM SIGGRAPH 2004 technical sketches program, 2004.

[3] STARIK S., WERMAN M. *Simulation of rain in videos.* Third international workshop on texture analysis and synthesis, 2003. 95–100s.

[4] GARG K., NAYAR SK. *Photometric model of a rain drop.* Technical Report, Columbia University; 2003.

[5] GARG K., NAYAR SK. *Detection and removal of rain from videos.* 2004 IEEE computer society conference on computer vision and pattern recognition, 2004. 528–35s.

[6] LANGER M.S., ZHANG L., KLEIN A.W., BHATIA A., PEREIRA J., REKHI D. *A spectral-particle hybrid method for rendering falling snow.* Rendering techniques, ACM Press; 2004.

[7] BEARD KV., CHUANG C. *A new model for the equilibrium shape of raindrops.* Journal of Atmospheric Science, 1987. 1509–24s.

[8] BEARD KV., CHUANG C. *A numerical model for the equilibrium shape of electrified raindrops.* Journal of Atmospheric Science, 1990. 1374–89s.

[9] TIŠNOVSKÝ Pavel. *Článek popisující programovou grafickou knihovnu OpenGL.* Http://www.root.cz/clanky/graficka-knihovna-opengl-1/.

[10] TIŠNOVSKÝ Pavel. *Díl seriálu, který se věnuje programování počítačové grafiky a tvorbě přenositelných grafických aplikací s pomocí knihovny GLUT.* http://www.root.cz/clanky/glut-1/.

[11] TIŠNOVSKÝ Pavel. *Úvod seriálu popisující knihovnu GLU.* http://www.root.cz/clanky/opengl-a-nadstavbova-knihovna-glu/.

[12] JANDORA Radek. *Stránky věnující se fyzice.* http://radek.jandora.sweb.cz/f18.htm/.

Příloha A

Uživatelská příručka

A.1 Ovládání

W	krok vpřed
S	krok vzad
A	otočení doleva
D	otočení doprava
Q	posun nahoru
E	posun dolu
Y	pohled nahoru
C	pohled dolu
P	pauza a start simulace
F	přepínání mezi kapkami a pruhy
T	výběr typu kapky
G	snižování počtu kapek
H	zvyšování počtu kapek
B	zmenšování velikosti kapek
N	zvětšování velikosti kapek

A.2 Spuštění aplikace

Spuštění aplikace je jednoduhé. Na přiloženém CD je složka Release, která obsahuje veškeré potřebné knihovny a data pro spuštění aplikace, nachází se zde také spustitelný soubor Rain.exe.

Příloha B

Obsah přiloženého CD

Přiložené CD obsahuje 3 následující složky:

- src

  Složka obsahující soubory se zdrojovými kódy.

- release

  Složka obsahující vše potřebné pro spuštění aplikace a zkompilovanou verzi programu. Naleznete zde potřebné knihovny a data, bez kterých by simulace nenaběhla.

- doc

  Tato složka obsahuje elektronickou podobu textu bakalářské práce bp.pdf.

- lib

  Složka obsahující používané knihovny.

Příloha C

Použité zkratky

API	Application Programming Interface.
GLEW	OpenGL Extension Wrangler Library
GLU	OpenGL Utilities.
GLUT	OpenGL Utility Toolkit. Okenní systém pro zobrazování OpenGL.
GUI	Graphical User Interface. Grafické uživatelské rozhraní.
OpenGL	Open Graphics Library. Grafická knihovna.
SDK	Software Development Kit
SGI	Silicon Graphics Inc.