Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Implementácia algoritmu na simuláciu plynu na atomárnej úrovni

(bakalárska práca)

Autor: Marek Jančuška

Školiteľ: Mgr. Jana Katreniaková

Bratislava, 2007

Čestne prehlasujem, že som bakalársku prácu vypracoval samostatne s použitím uvedenej literatúry.

Ďakujem mojej školiteľke Mgr. Jane Katreniakovej za jej rady a pripomienky pri písaní tejto práce.

Abstrakt: Cieľom tejto práce je implementácia efektívneho algoritmu na simuláciu plynu na úrovni atómov. Obsahuje popis použitého modelu, detaily implementácie a porovnanie s existujúcimi implementáciami.

Kľúčové slová: simulácia, ideálny plyn

Obsah

Kapitola 1

Úvod

Počítačové simulácie predstavujú zaujímavú alternatívu k reálnym experimentom. Použitie simulácie je často lacnejšie a pohodlnejšie ako reálny experiment. Navyše niekedy nemáme študovaný objekt k dispozícii. Vtedy je simulácia jedinou možnosťou, ako študovať daný jav, napríklad vývoj vesmíru tesne po veľkom tresku alebo klimatické zmeny.

V tejto práci sa venujem simulácii plynu v uzavretej nádobe. Práca obsahuje popis implementácie aj s dôkazom správnosti, ako aj samotnú implementáciu. Na dosiahnutie realistických výsledkov je potrebné simulovať veľké množstvá častíc, takže je vhodné použiť čo najrýchlejší algoritmus. Ďalším cieľom je flexibilita – aby sa program dal použiť na simuláciu čo najširšieho spektra rôznych experimentov.

Táto implementácia sa zrejme nedá použiť na zisťovanie kvantitatívnych údajov, ktoré by sa zhodovali s reálnymi meraniami. Dôvodom sú viaceré zjednodušenia (dvojrozmerný priestor, jednoatómové častice, zanedbanie rotácie). To však nie je prekážka pri skúmaní javov z kvalitatívneho hľadiska. Preto je program vhodný najmä na vzdelávacie účely, na predvedenie vlastností plynu. Aj z autorovovych skúseností vyplýva, že je ľahšie uveriť experimentu (hoci aj virtuálnemu) ako vzorcom.

V kapitole 2 sa budeme zaoberať problémami súvisiacimi s fyzikálnym modelom.

Kapitole 3 sa venuje algoritmickým a implementačným aspektom simulácie.

V kapitole 4 predstavíme skriptovanie ako nástroj na ovládanie simulácie a jej prispôsobenie požiadavkám užívateľa.

V kapitole 5 porovnáme našu implementáciu s niektorými existujúcimi implementáciami.

Kapitola 2

Základné pojmy

2.1 Fyzikálny model

Program simuluje plyn na úrovni elementárnych častíc(atómov) v dvojrozmernom priestore. Simulácia obsahuje tri druhy objektov: častice, steny a zložené objekty. Základnými vlastnosťami každého objektu sú hmotnosť a rýchlosť. Použitý model je malá modifikácia ideálneho plynu; rozdiel je v tom, že nezanedbávame objem častíc. Tento model je v literatúre označovaný ako "hard sphere gas" ([10]) alebo Van der Waalsov model ([9]).

Častice sú modelované pevnými, dokonale tvrdými guľami. Mimo zrážky sa pohybujú voľným pádom (tj. rovnomerne zrýchlený pohyb so zrýchlením $\vec{g}$). Častice sa navzájom ovplyvňujú iba zrážkami. Pri zrážke platia zákony zachovania hybnosti a energie.

Majme častice p, q s polomermi r_p, r_q. Zrážka (p, q) nastane, ked $|\vec{x}_p - \vec{x}_q| = r_p + r_q$.

Steny sú modelované úsečkami, polpriamkami alebo priamkami. Každá stena je rovnobežná s osou x alebo y.

Zložené objekty sa skladajú z viacerých jednoduchých objektov. Hlavnou vlasnosťou zložených objektov je to, že všetky členské objekty musia mať rovnaký vektor rýchlosti.

Fyzikálny model neuvažuje rotáciu objektov. Ak modelujeme atómy po-

mocou gulí, tak to nie je podstatné. Situácia by bola iná pri modelovaní napr. 2-atómových molekúl pomocou dvoch zlepených gulí. Model bez rotácie je však značne jednoduchší.

Ďalšie zjednodušenie spočíva v tom, že účastníkom každej zrážky musí byť častica. Vďaka tomu sa netreba zaoberať $9 = 3^2$ dvojicami typov objektov, ale stačia 3 dvojice. Okrem toho nie je jasné, ako spracovať napr. zrážku dvoch stien.

2.2 Detekcia zrážok

Dôležitou časťou programu je detekcia zrážok. Existujú dve metódy:

aposteriórna Simulujeme po malých časových krokoch. Pri simulácii jedného kroku neberieme kolízie do úvahy, ale po odsimulovaní kroku testujeme, či sa nejaké objekty nezrazili. Teda zrážku zistíme až po tom, ako nastala.

apriórna Čas zrážky vypočítame pred zrážkou samotnou.

Výhodou aposteriórnej metódy je jej jednoduchosť. Nevýhodou je nepresnosť a možnosť vynechania zrážky. Táto metóda je navyše pomalá – pre 1 krok vyžaduje až $O(N^2)$ testov (kde N je počet objektov v príslušnej oblasti). Apriórna metóda je menej všeobecná, vyžaduje viac znalostí o simulovaných objektoch. Na druhej strane je presnejšia, nevynecháva zrážky a umožňuje spracovať 1 zrážku v čase $O(N)$ (avšak za cenu zvýšenia počtu zrážok, lebo musíme spracúvať aj neplatné zrážky).

Pri implementácii som zvolil apriórnu metódu detekcie zrážok, a to hlavne kvôli rýchlosti a presnosti. Výpočet času zrážky vedie ku kvadratickým rovniciam, ktoré sa ľahko riešia. Táto voľba však priniesla aj obmedzenie v podobe ignorovania rotácie. Rotácia objektov by totiž viedla k rovniciam, ktoré nemajú riešenie v uzavretom tvare. Tie by bolo potrebné riešiť iteračnými metódami, čo môže byť pomalé a okrem toho nie je zaručené nájdenie všetkých koreňov.

2.3 Výpočet času zrážky

Majme dve častice p,q. Nech $\vec{x}_p$ je poloha častice p, $\vec{v}_p$ je jej rýchlosť, r_p je jej polomer (analogicky pre q). $\vec{g}$ bude označovať vektor gravitačného zrýchlenia. Polohu môžeme zapísať ako funkciu času:

$$\vec{x}_p(t) = \vec{x}_p(0) + \vec{v}_p(0)t + \vec{g}t^2$$

Zrážka nastane, keď

$$|\vec{x}_p(t) - \vec{x}_q(t)| = r_p + r_q$$

teda

$$|\vec{x}_p(0) + \vec{v}_p(0)t - \vec{x}_q(0) - \vec{v}_q(0)t| = r_p + r_q$$

Ak označíme $\vec{x}_p(0) - \vec{x}_q(0) = (x, y)$, $\vec{v}_p(0) - \vec{v}_q(0) = (v_x, v_y)$, tak máme

$$|(x, y) + (v_x, v_y)t| = r_p + r_q$$

$$|(x + v_x t, y + v_y t)| = r_p + r_q$$

$$\sqrt{(x + v_x t)^2 + (y + v_y t)^2} = r_p + r_q$$

Ľavá strana je zrejme kladná, môžeme obe strany umocniť na druhú.

$$(x + v_x t)^2 + (y + v_y t)^2 = (r_p + r_q)^2$$

$$(v_x^2 + v_y^2)t^2 + 2(xv_x + yv_y)t + (x^2 + y^2 - (r_p + r_q)^2) = 0$$

Dostali sme kvadratickú rovnicu s neznámou t. Ak rovnica nemá reálne korene, častice sa nezrazia. Inak sa zrazia v čase, ktorý zodpovedá menšiemu z koreňov.

Čas zrážky častice p s horizontálnou stenou w vypočítame z rovnice:

$$|y_p(t) - y_w| = r_p$$

2.4 Spracovanie zrážky

Majme objekty p, o, ktoré sa práve zrazili. Cieľom je vypočítať ich rýchlosti po zrážke. Aby sme mohli použiť ten istý postup na zrážky rôznych objektov, každý objekt bude vedieť vypočítať kolmicu dopadu pre danú časticu. Pri zrážke sa mení len zložka rýchlosti rovnobežná s kolmicou dopadu. Pre jednoduchosť nech druhá zložka (kolmá na kolmicu dopadu) oboch objektov je nulová. Označme veľkosti(teda nie vektory) rýchlostí oboch objektov pred zrážkou v_p, v_o, po zrážke v'_p, v'_o, hmotnosti m_p, m_o. Potom platí

$$v'_p = \frac{v_p(m_p - m_o) + 2v_o m_o}{m_p + m_o}$$

$$v'_o = \frac{v_o(m_o - m_p) + 2v_p m_p}{m_p + m_o}$$

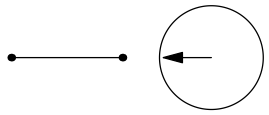
Tieto vzorce sa dajú odvodiť zo zákonov zachovania energie a hybnosti.

Ako určíme kolmicu dopadu:

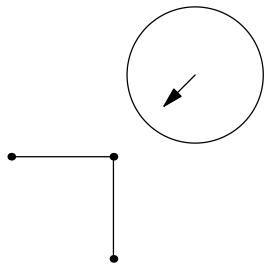
- pre časticu: je to spojnica stredov oboch častíc
- pre stenu: kolmica na stenu
- pre zložený objekt: kolmicu určí ten členský objekt, ktorý sa nachádza v mieste dopadu

2.5 Zložené objekty

Zložené objekty slúžia najmä ako korektná implementácia konečných stien (tj. stien modelovaných úsečkami alebo lomenými čiarami). Na začiatku a na konci lomenej čiary, ako aj v bodoch zlomu (tam, kde lomená čiara nemá dotyčnicu), sú umiestnené častice s nulovým polomerom. Bez týchto častíc by na obr. 2.1 a 2.2 nenastala zrážka, takže častice by mohli v určitých miestach prechádzať cez steny, čo nie je žiaduce.



Obr. 2.1: Zrážka častice s úsečkou



Obr. 2.2: Zrážka častice s lomenou čiarou

Kapitola 3

Základný algoritmus

Keďže simulácia objektov v čase medzi zrážkami je triviálna, dôležitou časťou programu je algoritmus na nájdenie ďalšej zrážy. V tejto kapitole taký algoritmus popíšeme a dokážeme jeho správnosť. Cieľom algoritmu je nájsť pre daný stav simulácie ďalšiu zrážku.

Algoritmus je podobný algoritmu (HAD) v [10], líši sa len v niektorých detailoch.

3.1 Myšlienka algoritmu

Trojicu (p, q, t), kde p a q sú objekty a $t \in \mathbb{R}$, budeme nazývať zrážka. (p, q, t) vyjadruje predpoklad, že p a q sa zrazia v čase t. Každý objekt p si bude pamätať zrážku c, p nazveme vlastíkom zrážky c. Ak p je vlastníkom zrážky (p, q, t), q nazveme partner objektu p (collision parter of p). Môže sa stať, že zrážka častice p bola označená za neplatnú. V takom prípade hovoríme, že p nemá partnera. Aj v prípade neplatnej zrážky má zmysel hovoriť o čase zrážky.

Na začiatku simulácie pre každý objekt vyskúšame všetky ostatné objekty a vyberieme ten, ktorý má čas zrážky najmenší. Nasledujúcu zrážku zistíme tak, že vyberieme časticu s najmenším časom zrážky. Nech je to častica p. Ak p nemá partnera, tak iba nájdeme nového partnera pre p. Inak nech

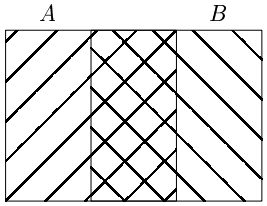
objekt o je partner objektu p. Zmeníme rýchlosti p a o tak, ako to zodpovedá zrážke. Následne označíme všetky zrážky, ktorých účastníkom je p alebo o, za neplatné. (Teda tie častice, ktoré doteraz mali za partnera p alebo o, odteraz nebudú mať partnera.) Potom hľadáme novú zrážku pre p aj pre o. Ak je však aktuálna zrážka neplatná, tak iba hľadáme nového partnera pre p.

3.2 Rozdelenie na oblasti

Uvedený algoritmus je možné zrýchliť. Treba si všimnúť, že zrážka dvoch vzdialených častíc je málo pravdepodobná. Preto simulovaný objem rozdelíme na oblasti(cells) tvaru obdĺžnika. Pri hľadaní partnera pre objekt o berieme do úvahy iba objekty z tej istej oblasti a zo susedných oblastí. Pre každú oblasť si budeme pamätať spájaný zoznam objektov, ktoré sa v danej oblasti nachádzajú. Ak sa objekt o nachádza v zozname prislúchajúcom k oblasti A, budeme hovoriť, že o je registrovaný v A. Ked objekt vstupuje do zóny, resp. ju opúšťa, treba aktualizovať príslušné zoznamy. Aby sme zistili čas vstupu do resp. výstupu zo zóny, zavedieme nové objekty – hranice oblasti. Pri výpočte času zrážky s objektom sa hranica oblasti správa ako stena, ale pri zrážke s objektom upraví príslušné zoznamy.

Ostáva určiť, do ktorej oblasti patrí objekt, ktorý práve prechádza cez hranicu oblasti. Nech objekt o prechádza cez hranicu medzi oblasťami A a B, pričom oblasť A opúšťa (tj. v budúcnosti tam nebude patiť) a do oblasti B vstupuje (tj. v minulosti tam nepatril). Bolo by prirodzené, keby o bol registrovaný v oboch oblastiach A aj B. Takéto riešenie by však vyžadovalo pre každý prechod spracovať dve udalosti – vstup do B a opustenie A. Bolo by zložitejšie a pravdepodobne aj pomalšie. Na druhej strane je toto riešenie všeobecnejšie, a implementácia zložených objektov by si nevyžadovala špeciálne úpravy, popísané v časti 3.5.

Inou možnosťou sú prekrývajúce sa oblasti. Ak bude prienik susedných oblastí dostatočne veľký, aby sa doň zmestil celý objekt o, potom v každom okamihu existuje taká oblasť, že celý objekt o do nej patrí. Takže o je regis-



Obr. 3.1: Prekrývajúce sa oblasti

trovaný iba v oblasti A, až kým nenarazí na hranicu oblasti A. Po tom je registrovaný iba v oblasti B. Pri tomto prechode teda ignorujeme (vstupnú) hranicu oblasti B. Teraz je zrejmé, prečo treba pri hľadaní partnera pre o brať do úvahy aj susedné oblasti: je totiž možné, že objekt sa nachádza v oblasti A a pritom je registrovaný v susednej oblasti B (samozrejme sa musí nachádzať aj v B, teda v prieniku A a B). Treba dodať, že šírka prieniku $A \cap B$ nesmie byť väčšia ako polovica šírky oblasti, inak by mala A prienik aj s oblasťou napravo od B.

Aby sme vedeli rýchlo nájsť nasledujúcu zrážku, budeme mať všetky zrážky uložené v halde, utriedené podľa ich času zrážky. Je však potrebné, aby sme z haldy vedeli aj rýchlo odstraňovať prvky. Pretože keď sa zrazí p s o, spracuje sa len zrážka patriaca objektu p. Zrážku patriacu o treba vtedy z haldy odstrániť. Ak by sme to neurobili, v halde by sa hromadili nepotrebné prvky. Ak nepotrebné prvky priebežne odstraňujeme, máme v halde pre každý objekt jeden prvok. Teda jedna operácia s haldou trvá $O(\log N)$, kde N je počet objektov. Odstaňovanie vieme uskutočniť rýchlo ($O(\log N)$), ak si bude každý objekt pamätať, kde v halde má svoju zrážku. Keď sa zrážku patriaca o bude v halde presúvať, informujeme o tom objekt o. Odstraňovanie prvku z ľubovoľnej pozície je podobné odstraňovaniu najmenšieho prvku.

3.3 Dôkaz správnosti algoritmu

Pre každý objekt p budeme uvažovať postupnosť všetkých ostatných objektov s_p. Objekty v s_p budú utriedené vzostupne podľa času zrážky s p. Pre

tie objekty, ktoré sa s p v budúcnosti nezrazia, považujeme čas zrážky za nekonečný (tj. v s_p budú na konci). i-ty prvok s_p budeme značiť $s_p(i)$, prvý prvok s_p bude $s_p(1)$. Časom zrážky objektu o budeme rozumieť čas, kedy o predpokladá zrážku so svojim partnerom. Označíme ho $t_c(o)$. Tento čas si o pamätá a sa mení iba pri zrážke, ktorej účastníkom je o. Časom zrážky dvojice (p, o) budeme rozumieť čas, kedy by sa p a q zrazili, keby neexistovali iné objekty, označíme ho $t_c(p, o)$. Nech p je partner objektu o. Čas zrážky objektu o a nazveme správny, ak čas zrážky o sa rovná času zrážky (p, o). (Čas zrážky o môže byť nesprávny, ak partner o pred zrážkou s o zmení rýchlosť). Partnera p objektu o nazveme správnym, ak $s_o(1) = p$.

Budeme skúmať, ako sa zmení stav simulácie po vykonaní týchto 3 operácii postupne v tomto poradí:

- všetkým objektom, ktorých partnerom je p, označíme zrážku za neplatnú
- zmeníme rýchlosť p
- nájdeme nového partnera pre p, tj. partera p nastavíme na $s_p(1)$

Túto usporiadanú trojicu operácii budeme nazývať zmena rýchlosti objektu p. Treba si všimnúť, že toto je postupnosť operácii, ktorú robíme pri spracúvaní zrážky (p, o) pre p aj pre o. Dá sa povedať, že túto postupnosť operácii vykonáme aj pri spracovaní neplatnej zrážky, aj keď prvé dve operácie nič nezmenia.

Najprv dokážeme platnosť tohto jednoduchého invariantu:

(I1) Ak má objekt p partnera, tak čas zrážky p je správny.

Bezprostredne po nájdení nového partnera pre p (I1) zrejme platí – teda aj hneď po inicializácii simulácie aj hneď po zmene rýchlosti p. Ak sa zmení rýchlosť partnera objektu p, p stratí partnera, teda (I1) stále platí. Ak sa nezmení ani rýchlosť p, ani rýchlosť jeho partnera, čas zrážky p ostáva správny. Pri spracovaní neplatnej zrážky nájdeme príslušnému objektu nového partnera, teda aj tu sa (I1) zachováva.

Teraz dokážeme ďalší invariant:

(I2) Pre každú dvojicu objektov (p, r) platí $t_c(r) \leq t_c(p, r)$ alebo $t_c(p) \leq t_c(p, r)$

(neformálne: o zrážke (p, r) buď p (ak $t_c(p) = t_c(p, r)$) alebo r vie, alebo sa o nej včas(tj. predtým, ako sa zrážka uskutoční) p (ak $t_c(p) < t_c(p, r)$) alebo r dozvie.)

Po inicializácii zrejme invariant platí (v inicializácii vlastne pre p vytvoríme s_p a $s_p(1)$ sa stane partnerom p a z usporiadania s_p vyplýva, že $t_c(p) \leq t_c(p, r)$. Ideme dokázať, že (I2) ostane v platnosti, keď pre objekt p urobíme zmenu rýchlosti.

Majme dvojicu (p, o). Pre p sme práve našli nový čas zrážky, teda $t_c(p) \leq t_c(o)$, lebo $t_c(p)$ hľadáme ako minimum z časov $\{t_c(p, r) | r$ je objekt $\}$.

Majme dvojicu (o, r), pričom $p \neq o$, $p \neq r$. Z invariantu vyplýva, že pred zmenou platilo $t_c(o) \leq t_c(o, r)$ alebo $t_c(r) \leq t_c(o, r)$. Ale pri zmene rýchlosti p sa hodnoty $t_c(o)$, $t_c(r)$ ani $t_c(r, o)$ nezmenili, teda invariant platí aj potom. (Mohli sa zmeniť partneri r resp. o, o nich však invariant nehovorí a $t_c(o)$ sa môže zmeniť len pri zmene rýchlosti o).

Aj po spracovaní neplatnej zrážky ostáva (I2) v platnosti, lebo príslušnému objektu nájdeme nového partnera.

Z invariantu odvodíme, že opísaný algoritmus nevynechá žiadnu zrážku. Nech nasledujúca (skutočná) zrážka je (p, o). Najprv dokážeme, že algoritmus ako ďalšiu zrážku spracuje buď (p, o) alebo neplatnú zrážku. Nech r je objekt s najmenším časom zrážky $t_c(r)$. Algoritmus spracuje zrážku objektu r ako nasledujúcu. Ak $t_c(r) < t_c(p, o)$, nemôže byť zrážka objektu r platná, lebo potom by bola aj správna (podľa (I1)). To by bol spor s predpokladom, že nasledujúca zrážka je (p, o). Ak $t_c(r) = t_c(p, o)$, algoritmus spracuje buď neplatnú alebo správnu zrážku v čase $t_c(p, o)$. (Ak je správna, nemusí to byť nutne zrážka (p, o), ale nejaká iná s rovnakým časom. Keďže medzi týmito zrážkami nevieme rozlišovať, nemôžeme požadovať silnejšiu vlastnosť.) Prípad $t_c(r) > t_c(p, o)$ nemôže nastať, lebo z (I2) vyplýva $t_c(p) \leq t_c(p, o)$ alebo $t_c(o) \leq t_c(p, o)$. A to je spor s minimalitou $t_c(r)$.

Tým sme dokázali, že algoritmus nespracuje zrážky, ktoré nemá spracovať. Ešte treba ukázať, že sa nezacyklí. Zacyklenie by mohlo nastať, keby sme stále spracúvali neplatné zrážky. Ale po spracovaní neplatnej zrážky počet objektov s neplatnou zrážkou určite klesne o 1. Preto nemôže po sebe nasledovať viac ako N neplatných zrážok (kde N je počet objektov).

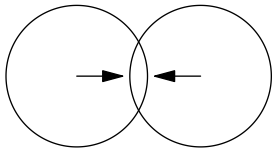
Zatiaľ sme sa venovali len správnosti jednoduchšieho algoritmu, ktorý nedelí priestor na oblasti. Rozdelenie na oblasti zachováva jednoduchý invariant, že ak sa častica nachádza v oblasti A (hoci aj čiastočne, teda aj ak práve opúšťa A), tak je registrovaná v oblasti A alebo v nejakej susednej oblasti. Teda ak má nasledovať zrážka (p, o), tak buď p a o ležia v susedných oblastiach, alebo ešte pred zrážkou (p, o) nastane zrážka p alebo o s hranicou oblasti.

3.4 Rozlišovanie minulých a budúcich udalostí

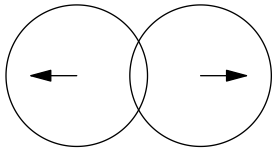
Je dôležité, aby sme vedeli správne rozlišovať minulé a budúce udalosti. V opačnom prípade by sa totiž mohlo stať, že nejakú udalosť preskočíme (ak ju mylne označíme za minulú), alebo spracujeme viackrát (ak ju mylne označíme za budúcu); v druhom prípade sa simulácia pravdepodobne zacyklí. Treba podoknúť, že rozlišovanie sa týka len nových udalostí – ak už udalosť bola naplánovaná, určite sa spracuje. Majme udalosť E. Označme aktuálny čas t_c, čas udalosti E označme t_E. Ako zistíme, či E už nastala (takže ju môžeme ignorovať), alebo nie (a treba ju spracovať)? Máme tieto možnosti:

1. Testovať $t_E < t_c$. Toto riešenie však nie je vhodné kvôli nepresnosti počítačovej aritmetiky. Mohlo by sa stať, že pre malé $\varepsilon > 0$ bude platiť $t_E = t_c - \varepsilon$ napriek tomu, že udalosť E sme ešte nezaregistrovali.

2. Testovať, či $t_E > t_c + \varepsilon$ pre pevne zvolené ε. Tu je problémom, ako správne zvoliť ε.



Obr. 3.2: Prekrývajúce sa častice, ktoré by sa mali zraziť



Obr. 3.3: Prekrývajúce sa častice – nemali by sa už zraziť

3. Využijeme, že väčšina udalostí sú zrážky. Použijeme metódu č. 1, ale pridáme dodatočnú podmienku. Nepovolíme, aby sa častica dvakrát po sebe zrazila s tým istým objektom. Teda ak sa častica p zrazila s objektom o, tak zrážku (p, o) považujeme automaticky za minulú. Toto riešenie bráni zacykleniu, ale možnosť vynechania zrážky zrejme ostáva. Okrem toho je príliš obmedzujúce. Pri nenulovej gravitácii sa totiž môže stať, že častica spadne, odrazí sa od steny a znovu spadne a odrazí sa od tej istej steny. Ďalšie obmedzenie sa týka zložených objektov – častica sa môže odraziť od jednej časti a následne od inej časti toho istého objektu.

4. Uvažujme objekty p, q. Označíme t_1 čas vstupu, tj. čas, kedy sa objekty prvýkrát dotýkajú. Teda v čase t_1 by mala nastať zrážka. t_2 bude čas výstupu, tj. čas, kedy sa objekty dotýkajú, ale vzďaľujú sa od seba. V časovom intervale (t_1, t_2) sa objekty p, q prekrývajú. Zrážku v čase t_1 budeme považovať za minulú, ak $\frac{t_1+t_2}{2} < t_c$. Inými slovami, ak je čas vstupu bližšie k súčasnosti, zrážku považujeme za budúcu, inak za minulú. Tento prístup teoreticky môže zlyhať, ale iba pri obrovských rýchlostiach a malých polomeroch častíc.

Všimnime si obr. 3.2. Najvhodnejšie je zrejme posúdiť túto zrážku ako budúcu, lebo takáto situácia mohla nastať vplyvom malej zaokrúhľovacej chyby. Metóda č. 1 bude takúto zrážku považovať za minulú. Pri metóde č. 2 to závisí od zvoleného ε – ak je príliš veľké, zrážku označí za minulú. Metóda č. 3 zrejme zrážku chybne označí za minulú, tak ako metóda č. 1. Metóda č. 4 označí zrážku za budúcu, lebo čas vstupu je evidentne bližšie k súčasnosti ako čas výstupu.

Analogicky môžeme analyzovať aj situáciu na obr. 3.3 – túto zrážku treba označiť za minulú. Správny výsledok určite vrátia metódy č. 3 a č. 4.

Implementovaná je metóda č. 4. Ideálna by asi bola kombinácia metód č. 3 a č. 4, pričom metódu č. 3 by sme nepoužili v prípade stien a zložených objektov. To je však komplikovanejšie a prínos tohto riešenia je malý – pri testovaní sa nikdy nestalo, že by sa metóda č. 4 zacyklila.

3.5 Implementačné detaily

Steny a zložené objekty sa nemusia zmestiť do jednej oblasti, preto k nim treba pristupovať osobitne. Nebudú registrované v žiadnej oblasti, ale pri hľadaní nového partnera ich budeme vždy brať do úvahy (ako keby boli registrované vo všetkých oblastiach).

Čo sa týka stien, ktoré ohraničujú simulovaný priestor, tieto nemenia svoju rýchlosť. Preto nikdy nie je nutné označovať zrážky častíc s týmito stenami za neplatné.

Kapitola 4

Skriptovanie

Pre zvýšenie flexibility simulácie sa dajú použiť skripty. Skript je program, ktorý komunikuje so simuláciou a umožňuje zisťovať a meniť jej parametre. Implementácia podporuje skripty v jazyku Python ([8]). Podpora skriptov je implementovaná pomocou knižnice Boost.Python ([4]).

Najdôležitejšou úlohou skriptu je inicializovať simuláciu, teda určiť počiatočné polohy, rýchlosti a hmotnosti objektov, gravitáciu, počet oblastí a veľkosť simulovaného priestoru. Skript ďalej môže naplánovať udalosť v simulácii. Udalosť je dvojica (t, f) s týmto významom: keď simulácia dosiahne čas t, zavolá sa funkcia f skriptu. Keď príslušná udalosť nastane, môže skript meniť rýchlosti objektov a zisťovať polohy a rýchlosti objektov.

Kapitola 5

Existujúce implementácie

Zaujímavým projektom z oblasti molekulárnej dynamiky je systém GROMACS ([6]). Jeho zameranie je však mierne odlišné od tejto práce, lebo uvažuje aj vzájomné pôsobenie častíc mimo zrážok

Rovnaký model ako v tejto práci je použitý v programe [7]. Nevýhodou je však absencia gravitácie a skriptov, teda je možné realizovať len zabudované experimenty.

Ďalej existuje niekoľko Java apletov, ktoré sú podobne obmedzené([1, 2, 3])..

Kapitola 6

Záver

Program implementovaný v tejto práci umožňuje predviesť rôzne vlastnosti plynu, napríklad Archimedov zákon, Maxwellovo rozdelenie podľa rýchlosti, prenos tepla a zákony platné v ideálnom plyne. Pomocou skriptov sa dajú naprogramovať aj tepelné stroje, napr. motor.

Program by sa ešte dal vylepšiť. Veľmi zaujímavé by bolo, keby sa dali pridávať a odoberať častice počas behu simulácie, nielen pri inicializácii. V súčasnej implementácii je problém s pridávaním, lebo častice sa nesmú prekrývať. Bolo by potrebné hľadať voľné miesto pre novú časticu, na čo je zrejme najvhodnejšie použiť Voronoiove diagramy. Ak budeme napr. vľavo pridávať častice a vpravo ich budeme odoberať, dostaneme prúdenie zľava doprava. Tomuto prúdeniu môžeme postaviť do cesty prekážky rôzneho tvaru a merať ich odpor prostredia. To je vlastne ekvivalent aerodynamického tunela.

Iné zlepšenie sa týka rozšírenia do 3 rozmerov. To by vyžadovalo modifikácie kódu, ktorý počíta čas zrážky a spracúva zrážky, ako aj zmenu pri hľadaní susedných oblastí. Najväčším problémom by však bola korektná implementácia konečných stien. Pri dvoch rozmeroch sme na hranice a do bodov zlomu umiestnili častice s nulovým polomerom, pri troch rozmeroch by to museli byť úsečky a častice.

Literatúra

[1] intro.chem.okstate.edu/1314F00/Laboratory/GLP.htm .

[2] http://www.chm.davidson.edu/ronutt/che115/Ideal/Ideal.htm .

[3] http://www.bluegrass.kctcs.edu/LCC/RCP/gas.html .

[4] Boost.python – knižnica na spojenie c++ a pythonu.

[5] Collision detection.
http://en.wikipedia.org/wiki/Collision_detection .

[6] Gromacs.
http://www.gromacs.org .

[7] Ideal gas in 3d.
http://www.physics-software.com/software.html .

[8] Programovací jazyk python.
http://www.python.org .

[9] Van der waals equation.
http://en.wikipedia.org/wiki/Van_der_Waals_equation .

[10] Alan T. Krantz. Analysis of an efficient algorithm for the hard-sphere problem. *ACM Transactions on Modeling and Computer Simulation*, 6(3):185–209, July 1996.