Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Deni Bačić

Simulacija molekulske dinamike na platformi iOS

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Matija Marolt
Somentor: doc. dr. Janez Mavri

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00569 / 2013
Datum: 6.11.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **DENI BAČIĆ**

Naslov: **SIMULACIJA MOLEKULSKE DINAMIKE NA PLATFORMI IOS**
SIMULATION OF MOLECULAR DYNAMICS ON IOS PLATFORM

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomski nalogi preučite področje simulacije molekulske dinamike. Za platformo iOS razvijte aplikacijo, ki po fizikalnih zakonih simulira molekulsko dinamiko Lennard-Jonesovih delcev in rezultate vizualizira. Analizirajte tudi učinkovitost različnih načinov za implementacijo numerične simulacije.

Mentor:
doc. dr. Matija Marolt

Somentor:
doc. dr. Janez Mavri

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Deni Bačić, z vpisno številko **63070312**, sem avtor diplomskega dela z naslovom:

Simulacija molekulske dinamike na platformi iOS

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matije Marolta in somentorstvom doc. dr. Janeza Mavrija,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 6. novembra 2013 Podpis avtorja:

Zahvaljujem se vsem, ki so mi v času študija stali ob strani, posebej staršem za ohranitev upanja in Debori, ki je v tem času skrbela za mojo zdravo prehrano. Sodelavcem iz Laboratorija za računalniške bioznanosti in bioinformatiko na Kemijskem inštitutu – na čelu s somentorjem doc. dr. Janezom Mavrijem, delo z vami je neusahljiv vir računalniških izzivov. Nenazadnje pa tudi mentorju doc. dr. Matiji Maroltu za pomoč in podporo pri izdelavi diplomske naloge.

Staršem.

Kazalo

Povzetek

Diplomsko delo obravnava razvoj aplikacije za simulacijo molekulske dinamike na platformi iOS z uporabo knjižnice Sprite Kit.

Uporaba mobilnih naprav za tovrstne izzive je smiselna s stališča dostopnosti in varčevanja energije. V sodobni teoretični kemiji se problemi rešujejo na velikih superračunalnikih, velika večina pa jih temelji na osnovah molekulske dinamike, ki je tema tega diplomskega dela.

Poleg teorije molekulske dinamike se v diplomskem delu ukvarjam s težavami in izzivi razvoja za iOS platformo, uporabljenimi orodji in novo knjižnico za izdelavo grafičnih aplikacij, imenovano Sprite Kit.

Rezultat diplomske naloge je prototip aplikacije, ki po fizikalnih zakonih simulira molekulsko dinamiko Lennard-Jonesovih delcev, vsi izračuni in prikaz pa se izvajajo na mobilni napravi. Poleg simulacije so v aplikaciji predstavljeni vsi pristopi, kar uporabniku omogoča neposreden vpogled v delovanje dinamike in s tem razumevanje problema.

Ključne besede: molekulska dinamika, iOS, razvoj, aplikacija, Sprite Kit.

Abstract

This thesis deals with the development of applications for simulation of molecular dynamics on the iOS platform with use of Sprite Kit library.

The use of mobile devices for such challenges makes sense from the standpoint of accessibility and energy saving. In modern theoretical chemistry, problems are solved on large supercomputers. The vast majority of todays problems is based on basic molecular dynamics, which is the subject of this thesis.

In addition to the theory of molecular dynamics, the thesis deals with the problems and challenges of development for the iOS platform, used tools and the new library for graphical applications called Sprite Kit.

The result of this work is a prototype application, which simulates 2d molecular dynamics of Lennard-Jones particles according to the laws of physics, where all of the calculations and display are carried out on a mobile device. In addition to the simulation, all of the approaches are explained inside of the application. That gives user direct insight into the dynamics and better understanding of the problem.

Keywords: molecular dynamics, iOS, development, application, Sprite Kit.

Poglavje 1

Uvod

Dandanes nas obkrožajo vse bolj zmogljive naprave. Naj bodo to osebni računalniki, tablice ali telefoni. Prepogosto ne izkoriščamo moči, ki nam jo ponujajo. Nekaj, kar je pred leti spadalo v kategorijo superračunalnikov, ki so si jih lahko privoščile le redke velike organizacije, imamo danes takorekoč na dlani.

Če so v prejšnjem stoletju resni računalniki zasedali več nadstropij, danes lahko enako računsko moč dobimo iz naprav velikosti škatlice vžigalic. Znanost je skoraj na vseh področjih še vedno prepletena s pojmom superračunalništva. Napoved vremena in ostalih meteorološko zanimivih pojavov, verjetnost in statistika, simulacije tekočin ter ostala fizikalna vprašanja so le delček mozaika raznovrstnih uporab superračunalnikov. Težava, ki se pojavlja, pa je zahtevna uporaba takšnih sistemov. Raziskovalci potrebujejo širok interdisciplinaren spekter znanj in dobro poznavanje porazdeljenih sistemov, da bi lahko kvalitetno izkoriščali strojno opremo.

V smislu približanja simulacij, ki jih danes izvajamo na zmogljivih sistemih, začetnikom in znanstvenikom iz eksperimentalnih panog biokemije, smo si za cilj zastavili interaktivno, didaktično mobilno aplikacijo, ki simulira osnovno molekulsko dinamiko. S takšnim pristopom jih skušamo motivirati za uporabo računske podpore. S tovrstnim sodelovanjem lahko natančneje določimo obetavne poskuse, prihranimo čas ter zmanjšamo stroške raziskav.

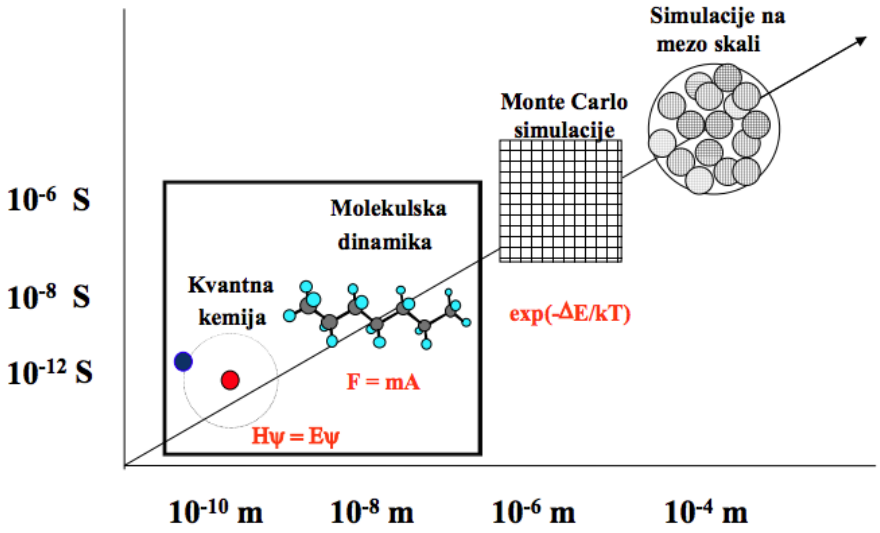
1.1 Organizacija diplomske naloge

V začetku diplomskega dela je opisana molekulska dinamika, kar bralcu brez izkušenj na področju biokemije nudi vpogled in potrebne osnove za razumevanje problema. V nadaljevanju diplomskega dela so natančno predstavljena razvojna platforma (iOS) in orodja (Xcode), ki so bila uporabljena pri izdelavi aplikacije. Jedro naloge predstavlja opis implementiranih lastnosti molekulske dinamike in težave s katerimi smo se srečevali pri delu. Opisana so tudi primerjave različnih programskih zank in razlike v hitrosti njihovega izvajanja.

Poglavje 2

Molekulska dinamika

2.1 Kaj je molekulska dinamika?



Slika 2.1: Molekulske (atomistične) simulacije [1].

Molekulska dinamika (MD) je metoda za simulacijo kinetičnih in termodinamičnih lastnosti molekularnih sistemov z uporabo Newtonovih enačb gibanja. Ponazarja interakcije med atomi in molekulami v določenem časovnem

intervalu s pomočjo numerične integracije enačb gibanja, ponavadi se v ta namen uporablja Verletove algoritme.

Gibanje atomov vedno lahko aproksimalno opišemo kot točkaste delce, ki se premikajo v prostoru, se medsebojno zaletavajo, tavajo naokrog in oscilirajo v skladu s sosedi.

Zaradi velikega števila delcev, iz katerih so ponavadi sestavljeni realistični sistemi, njihovih lastnosti ni mogoče analitično določiti. Z uporabo numeričnih metod molekularna dinamika uspešno zaobide ta problem. Raziskovalcu nudi povezavo med laboratorijskimi eksperimenti in teorijo, torej lahko molekulsko dinamiko definiramo tudi kot nekakšen navidezni poskus.

Pri molekulski dinamiki upoštevamo enake zakone kakor v klasični mehaniki, najbolj pomemben pa je drugi Newtonov zakon:

$$F_i = m_i \cdot a_i \tag{2.1}$$

za vsak atom i v sistemu, sestavljenem iz N atomov. V enačbi predstavlja m_i maso atoma, a_i njegov pospešek, F_i pa silo, ki na delec deluje zaradi interakcije z ostalimi atomi, njuna vrednost pa je določena z negativnim odvodom potencialne energije na koordinato.

$$a_i = \frac{d^2 r_i}{dt^2} \tag{2.2}$$

V nasprotju s pristopom Monte Carlo, je molekularna dinamika deterministična tehnika: če podamo začetno stanje pozicij in hitrosti, dobimo v teoriji natančno določen razvoj sistema v času. V praksi pa zaradi končnosti časovnega koraka integracije in aritmetičnih napak pri zaokroževanju izračunane krivulje gibanja atomov odstopajo od dejanske poti [2].

2.2 Zgodovina

Začetki molekulske dinamike, kot jo poznamo danes, segajo v pozna petdeseta leta prejšnjega stoletja. V tem času sta ameriška fizika Berni Alder in Thomas Everett Wainwright izdala serijo člankov na temo raziskav interakcij med togimi kroglami [3] [4]. V takšnem sistemu delci vplivajo med seboj

preko neposrednega trka, med posameznimi trki pa se prosto in neodvisno gibljejo po prostoru. Izračune sta izvajala na računalniku Universal Automatic Computer (UNIVAC) ter IBM 704 [2]. Njune študije so postavile temelje za večje število ugotovitev na področju preprostih tekočin in koloidov.

UNIVAC je bil šele drugi komercialni računalnik v Združenih državah Amerike. V uporabo je bil predan 14. junija 1951. Med drugim je poznan po tem, da je na temelju majhnega vzorca, velikega zgolj en odstotek, televizijska hiša CBS z njegovo pomočjo uspešno napovedala rezultate ameriških volitev leta 1952 in zmago pripisala Dwightu D. Eisenhowerju. Sestavljen je bil iz 5200 elektronk, tehtal je nekaj več kot 13 ton, med delovanjem pa je povprečno uporabljal 125 kW električne energije. Zmožen je bil opraviti približno 1905 operacij na sekundo, s hitrostjo 2.25 MHz. Zgolj procesna in pomnilniška enota sta zasedli prostor velikosti 27 kubičnih metrov, celoten računalnik pa je zasedel približno 35,5 kvadratnih metrov [5].

IBM 704 je bil prvi množično proizvajan računalnik v Združenih državah Amerike, ki je podpiral operacije s plavajočo vejico. V sekundi je bil sposoben opraviti do 4000 operacij. IBM je med leti 1955 in 1960 izdelal 123 primerkov računalnika 704 [6].

Leta 1964 je Aneesur Rahman izvedel prvo simulacijo z uporabo realnih potencialov za tekoči argon na računalniku CDC 3600 [7]. 24-bitni računalnik je bil sposoben opraviti milijon operacij na sekundo (1 MIP) in je v tistem času užival status superračunalnika. Simulacije so bile izvedene s pomočjo Lennard-Jonesovega potenciala s sistemom 864 atomov. [2]. Uporabljeni algoritmi so še danes osnova za različne programske rešitve na tem področju.

Desetletje pozneje sta Aneesur Rahman in Frank H. Stillinger izdelala prvo simulacijo realnega sistema v tekoči vodi [8].

Prve simulacije proteinov so se pojavile leta 1977 s simulacijo govejega inhibitorja pankreatičnega tripsina (ang. bovine pancreatic trypsin inhibitor (BPTI)) [9].

2.3 Molekulska dinamika danes

Izjemno pomembno vlogo imajo danes biomolekularne simulacije pri raziskovanju tekočin. Z njihovo pomočjo nenehno odkrivajo nova dejstva o viskoznosti in pretoku toplote. Velik doprinos imajo tudi na področju odkrivanja napak v kristalih ter pri analizi prelomov materialov.

Molekulska dinamika je odprla vrata študijam o makromolekulah, vključno z biološkimi sistemi, kot so proteini in nukleinske kisline. V farmacevtski industriji se pogosto uporablja za načrtovanje novih učinkovin, z njihovim testiranjem v simulacijah se lahko raziskovalci izognejo praviloma dražji dejanski sintezi učinkovin in jo izvedejo zgolj za obetavne spojine.

S kombinacijo kvantne kemije in molekulske dinamike lahko napovemo hitrost kemijske reakcije v encimu ali vodni raztopini. S tem se odpira računalniško zelo zanimivo področje biokatalize.

2.3.1 Programska oprema

Med najbolj razširjene programske pakete na področju biomolekularnih simulacij danes štejemo GROMACS, GROMOS, CHARMM, NAMD in AMBER. Vsak izmed njih podpira uporabo več različnih polj sil (ang. force field), ki imajo običajno enako ime kakor sam program (GROMOS, CHARMM, OPLS, AMBER). Fragmentiranost se je pojavila pri razvoju paketov MD, ker so običajno nastali tako, da si je vsaka skupina raziskovalcev oziroma razvijalcev zamislila neko svoje polje sil, ki so ga implementirali v svoj program. Kasneje so v pakete postopoma vključili tudi druga polja in tako lahko danes v enem paketu uporabljamo različne pristope, razlike med jedri MD pa so ostale.

2.3.2 Programska oprema na mobilnih platformah

Na platformi iOS trenutno še ne obstaja aplikacija, ki bi izvajala molekulsko dinamiko, najbolj se temu približa Mobile HyperChem. Omogoča nam osnovno gradnjo simulacijskega modela, ki ga lahko s povezavo z namiznim

Paket	3d[1]	Model[2]	Min[3]	MC[4]	GPU	Licenca
AMBER	Ne	Da	Da	Ne	Da	Komercialna
CHARMM	Ne	Da	Da	Da	Ne	Komercialna
GROMACS	Ne	Ne	Ne	Ne	Da	GNU GPLv2
GROMOS	Ne	Da	Da	Da	Da	Komercialna
VMD + NAMD	Da	Da	Da	Ne	Da	Akademska

Tabela 2.1: Primerjava lastnosti najbolj razširjenih programskih paketov.

računalnikom dejansko simuliramo. Okleščena različica mobilne aplikacije je na voljo zastonj, ponujajo pa tudi plačljivo verzijo, ki vsebuje še nekaj dodatnih zmogljivosti.

Android nam ponuja dve zanimivi aplikaciji, in sicer Atomdroid ter YASARA. Obe sta dostopni brezplačno na spletu, podpirata pa enostavne simulacije molekulske dinamike in izgradnjo modelov. Razvijalci YASARE obljubljajo tudi izdajo aplikacije na platformi iOS, problem pa vidijo v tem, da je njihova koda prilagojena za arhitekturo Intel/AMD in ne za ARM.

2.3.3 Nobelova nagrada za kemijo 2013

Prav osnovna molekulska dinamika je temelj dela letos podeljene Nobelove nagrade za kemijo. Prejeli so jo Martin Karplus (glavni razvijalec programskega paketa CHARMM), Michael Levitt ter Arieh Warshel (programski paket Molaris) za doprinos k razvoju programske opreme na področju teoretične kemije.

[1] 3d prikaz.
[2] Orodje za izgradnjo modelov.
[3] Optimizacija.
[4] Pristop Monte Carlo.

Poglavje 3

Razvojna platforma in orodja

Mobilne naprave dandanes doživljajo pravi razcvet in Applov operacijski sistem iOS igra eno izmed ključnih vlog. Homogenost naprav, majhno število resolucij, odlična orodja in veliko število uporabnikov so njene največje prednosti s stališča razvijalcev.

3.1 Apple iPhone

Apple iPhone je serija pametnih mobilnih telefonov, prva različica je na tržišče prispela januarja leta 2007. Vsebovala je 32-bitni procesor Samsung RISC ARM, delovne frekvence 612 MHz, 128MB delovnega spomina in največ 16 GB vgrajenega pomnilnika. Podjetje skrbi za vsakoletne nove inačice in do danes so se v prodaji nahajali modeli iPhone (2007), iPhone 3G (2008), iPhone 3GS (2009), iPhone 4 (2010), iPhone 4S (2011), iPhone 5 (2012), iPhone 5c in iPhone 5s (2013).

Zadnja različica razpolaga s 64-bitnim dvojedrnim procesorjem Apple A7 z delovno frekvenco 1,3 GHz. Vsebuje 1 GB DDR3 delovnega spomina in največ 64 GB vgrajenega pomnilnika. Zanimivost v zadnji različici je koprocesor, ki skrbi za obdelavo podatkov vseh senzorjev naprave. Čip, poimenovan M7 (okrajšava za ang. motion, premikanje) je izdelan v sodelovanju

[1] http://www.sparkletechnews.com

Slika 3.1: Modeli iPhone od originalnega iPhone do iPhone 5.[1]

s podjetjem NXP semiconductors in deluje pri nazivni frekvenci 150 MHz. Koprocesor nenehno obdeluje podatke in jih po potrebi dostavlja aplikacijam, s takšnim pristopom ima uporabnik nemudoma ob namestitvi aplikacije dostop do zgodovine premikanja naprave.

3.2 Operacijski sistem iOS

Operacijski sistem iOS, je bil prvič predstavljen leta 2007 pod imenom iPhone OS. Skrajšano ime, ki ga nosi še danes, se je prvič pojavilo v različici 4, leta 2010. V tistem času je poganjal napravi iPhone ter iPod Touch, kasneje pa še iPad ter Apple TV.

Popularnost lahko pripišemo predvsem velikemu številu aplikacij, objavljenih je več kot milijon, opravljenih pa je bilo približno 60 milijard prenosov le-teh [10]. Poudariti je potrebno, da je bil App Store predstavljen šele v iOS 2. V tabeli 3.1 je prikazano povečevanje števila objavljenih aplikacij in število posameznih prenosov na naprave.

Nove različice izhajajo praviloma letno, do danes so izšle iPhone OS 1 (2007), iPhone OS 2 (2008), iPhone OS 3 (2009), iOS 4 (2010), iOS 5 (2011), iOS 6 (2012) in iOS 7 (2013).

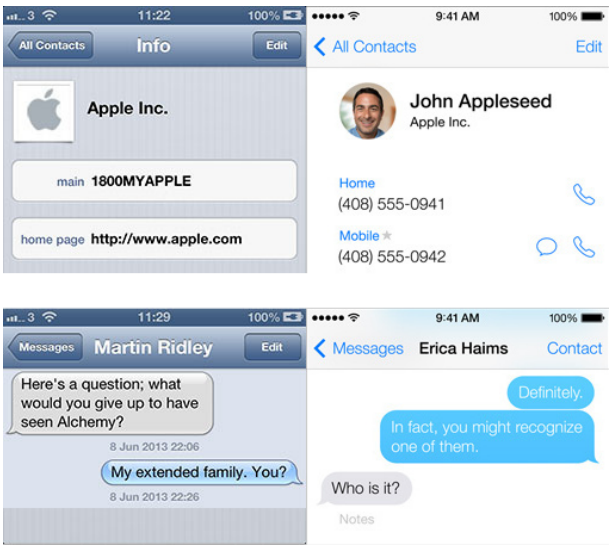
iOS temelji na jedru Darwin, ki poganja namizne računalnike Apple z

Leto	Število aplikacij	Število prenosov
2008	500	0
2009	65.000	1.500.000.000
2010	225.000+	5.000.000.000+
2011	425.000+	15.000.000.000+
2012	650.000+	30.000.000.000+
2013	1.000.000+	60.000.000.000+

Tabela 3.1: Naraščanje obsega App Store s časom.

operacijskim sistemom Mac OS X. Razvoj je zasnovan na dejstvu, da bo interakcijo z uporabnikom opravljal zaslon, občutljiv na dotik. Velikost sistema je s prvotnih 87MB leta 2007 v zadnji iteraciji letos narasla na več kot 1,4GB.

3.3 iOS 7

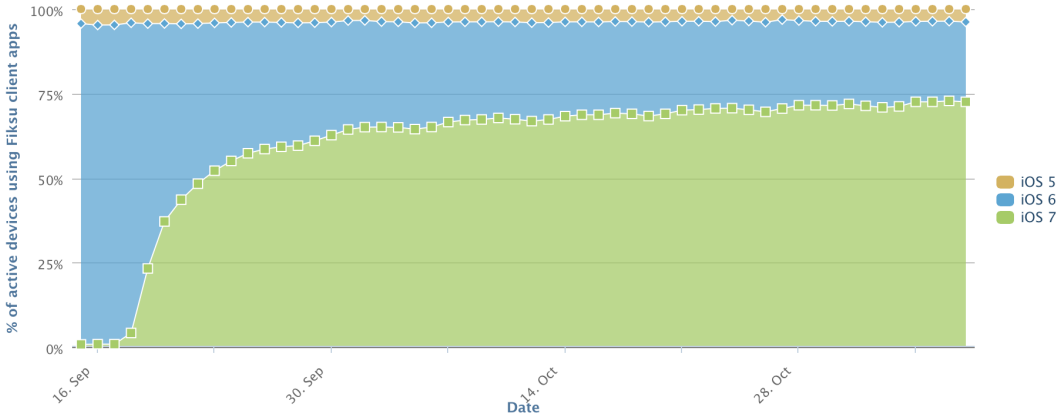


Slika 3.2: Primerjava uporabniškega vmesnika, iOS 6 proti iOS 7.[1]

Predstavitev iOS 7 je predstavljala svojevrsten mejnik v razvoju. Prvič v zgodovini operacijskega sistema so korenito spremenili grafični vmesnik, bistvene razlike so prikazane na sliki 3.2. Prvi posnetek zaslona prikazuje spremembe v imeniku (aplikacija Contacts), drugi pa spremembe pri aplikaciji za sporočila (Messages).

Uporabniški vmesnik je poenostavljen, razvijalci pa so skoraj popolnoma opustili uporabo prispodob iz realnega življenja (ang. skeuomorphism). Najbolj prepoznavni primeri iz preteklosti so bili šivano usnje pri koledarju, izgled beležke in podobno. Takšen pristop je ljudem v začetku olajšal uporabo naprav, saj so z izkušnjami lahko sami ugotovili, kaj je potrebno storiti. V današnjem času pa so uporabniki že dovolj izkušeni in tako izdatna uporaba prispodob ni več nujno potrebna.

Celoten izgled je sedaj bolj odprt in svetel, aplikacije imajo na razpolago več prostora za prikaz svojega vmesnika in preglednost je veliko večja.

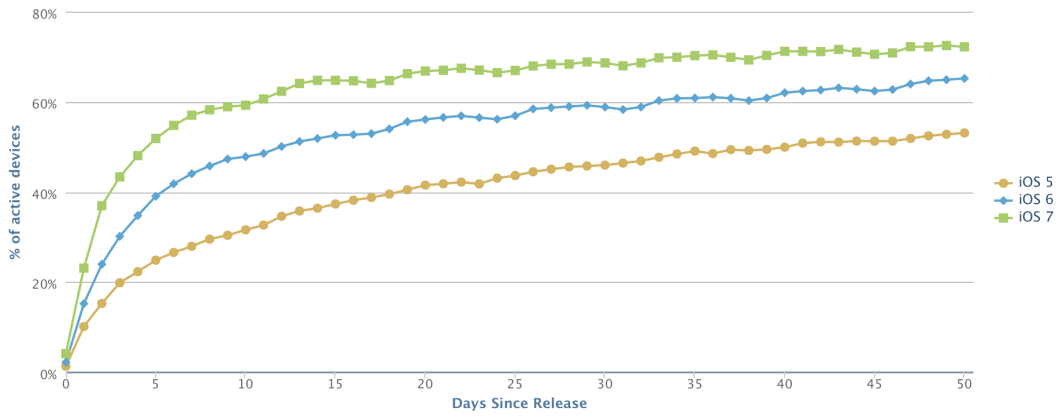


Slika 3.3: Odstotek uporabnikov z nameščenim iOS 7.[2]

Operacijski sistem je analitike presenetil z rastjo namestitev. Je najhitreje sprejet iOS do sedaj. V prvih desetih dneh je že presegel 50-odstotni tržni delež, rast pa se še vedno nadaljuje, trenutno je iOS7 nameščen na približno 75 odstotkov naprav.

[1] http://www.apple.com

[2] http://www.fiksu.com



Slika 3.4: Relativen odstotek uporabnikov z nameščenim iOS 7 glede na predhodne različice.[1]

Odstotek namestitev glede na dneve od izdaje v primerjavi s predhodniki je viden na grafu 3.4, podatki pokrivajo prvih 50 dni od izdaje. Analiza zajema tudi naprave, ki novega operacijskega sistema zaradi strojne podhranjenosti ne podpirajo, tako da je realna vrednost verjetno še višja.

3.3.1 Abstraktne plasti iOS

Za uporabo strojne opreme iOS vsebuje več različnih abstraktnih plasti, ki nam omogočajo preprost dostop in povezavo s samo kodo, prikazane so v tabeli 3.2.

Core OS
Core Services
Media
Cocoa Touch

Tabela 3.2: Abstraktne plasti iOS.

[1] http://www.fiksu.com

Core OS

Plast Core OS vsebuje nizkonivojske lastnosti, na katerih temelji večina preostalih tehnologij. Tukaj najdemo ogrodja za pospeševanje procesorsko zahtevnih funkcij (Digital Signal Processing (DSP), linearna algebra, izračuni pri procesiranju slik,...), ki so prilagojena strojni opremi, na kateri teče iOS. Vključeno je tudi ogrodje za delo z Bluetooth napravami, ki porabijo minimalno količino energije. Vsebuje podporo za delo z zunanjimi napravami, upravljanje s certifikati, ključi in gesli, ne smemo pa pozabiti na osnovo vsega, jedro Darwin.

Core Services

Core Services vsebuje osnovne sistemske servise, ki jih uporabljajo vse aplikacije. Med visoko-nivojske lastnosti spadajo med drugim podpore za:

- iCloud (deljenje dokumentov oziroma manjših podatkovnih datotek med napravami)
- Automatic Reference Counting (ARC) (na nivoju prevajalnika skrbi za samodejen nadzor nad življenjskim ciklom posameznih objektov)
- SQLite (znotraj aplikacije ustvari majhno bazo brez uporabe oddaljenega strežnika)
- Extensible Markup Language (XML) (vključena je podpora za razčlenjevanje XML datotek)
- blokovne objekte (ang. block objects)
- zaščito občutljivih podatkov
- deljenje datotek med aplikacijami
- Grand Central Dispatch (GCD) (tehnologija za upravljanje izvajanja aplikacije povzeta iz Berkeley Software Distribution (BSD))
- In-App nakupe

Media

Plast Media vsebuje ogrodja za delo z grafiko, zvokom in videom. Programerju omogoča predvajanje zvočnih in video posnetkov, risanje 2d vektorjev in uporabo vgrajene kamere.

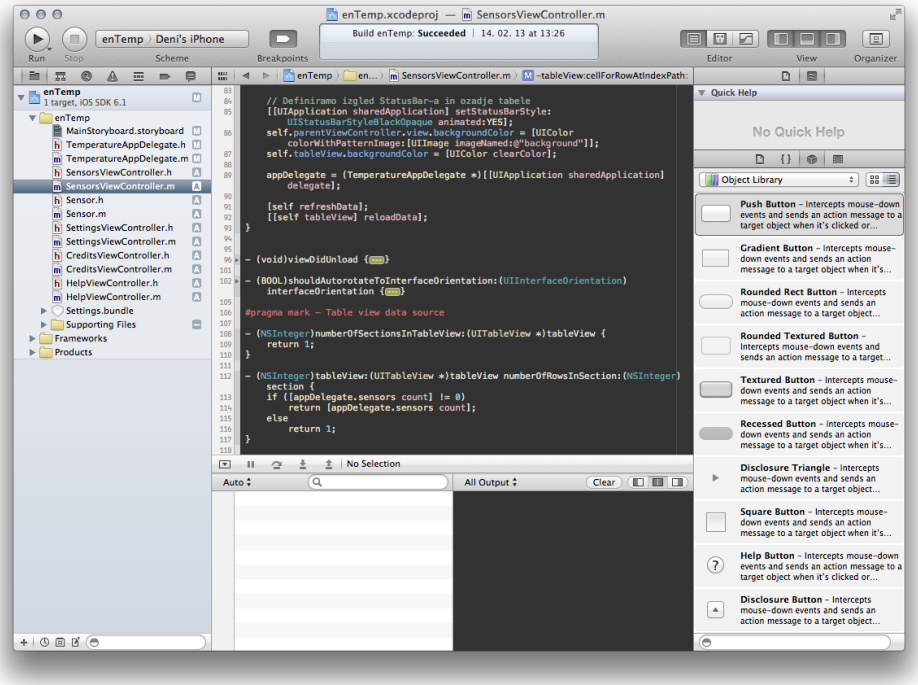
Cocoa Touch

Cocoa Touch plast vsebuje osnovna ogrodja za izdelavo iOS aplikacij. Določa infrastrukturo aplikacije in podporo ključnim tehnologijam, kot so push opomniki (strežniški ali lokalni), večopravilnost, dotikovni vmesnik in več visoko nivojskih sistemskih servisov.

3.4 Razvojno okolje Apple Xcode

Namensko integrirano razvojno okolje (ang. Integrated Development Environment (IDE)) Xcode, predstavlja osnovo za razvoj aplikacij za OS X in iOS (prvič v različici 3.1). Xcode se trenutno nahaja v različici 5.0 in je brezplačno na voljo preko Mac App Store. Orodje ponuja vse potrebno za izdelavo in upravljanje iOS projektov, oblikovanje uporabniških vmesnikov, prevajanje, poganjanje in razhroščevanje v simulatorju ali na napravi. Primer uporabe je razviden s slike 3.5.

Osnovni programski jezik je Objective-C, večnamenski, visokonivojski, objektno-orientirani jezik, katerega sintaksa temelji na Smalltalku. Poleg Objective-C lahko uporabljamo kodo, napisano v C-ju. Xcode omogoča zagon in testiranje aplikacij v priloženem simulatorju iOS, za preizkus na fizični napravi pa se je potrebno registrirati kot Apple iOS Developer. Med posebnosti programskega jezika lahko štejemo kategorije (ang. categories, uporabljamo jih za dodajanje funkcionalnosti v že definirane razrede, tudi v tiste, za katere nimamo izvorne kode), protokole (ang. protocols, z njimi definiramo metode, ki niso direktno vezane na razred) in bloke (ang. blocks, zaključeni skupki kode, ki nam pomagajo pri vzporednem in asinhronem izvajanju funkcij ter poenostavijo kodo za pogosto uporabljanje operacije, kot



Slika 3.5: Namensko integrirano razvojno okolje Xcode, namenjeno razvoju za platformi iOS in OS X.

so sortiranje, testiranje in preštevanje).

Izvorna koda 3.1: Primer kode napisane v programskem jeziku Objective-C

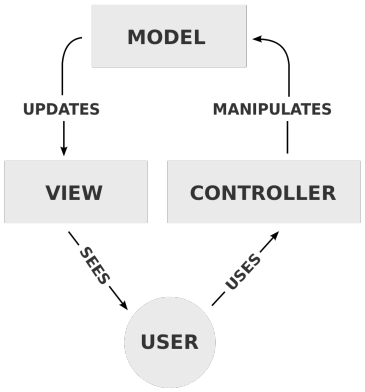
```
#import <stdio.h>

int main( int argc, const char *argv[] ) {
    printf( "Hello world!\n" );
    return 0;
}
```

3.5 Aplikacije za iOS in njih omejitve

Za razvoj aplikacij iOS potrebujemo računalnik z operacijskim sistemom Macintosh (OS X) ter nameščena orodja Xcode. V zadnjem času pa se pojavljajo tudi alternative, kot so Xamarin, ki nam omogočajo razvoj tudi na platformi Microsoft Windows v okolju Microsoft Visual studio.

3.5.1 MVC

Xcode sam nas uvede in deloma prisili v uporabo pristopa Model-View-Controller (MVC). MVC je oblika načrtovanja programske opreme, ki ločuje prikaz informacij in uporabnikovo interakcijo. Poleg razdelitve aplikacije na tri različne komponente MVC definira tudi določene interakcije med njimi, prikaz definicije je na sliki 3.6.



Slika 3.6: Diagram sestavnih delov pristopa MVC.[1]

Podobno kot vrsta drugih konceptov je bil MVC izumljen s strani Trygvea Reenskauga, norveškega profesorja in raziskovalca, v okviru programskega jezika Smalltalk. Reenskaug v članku iz leta 1979 tako opisujejo definicije posameznih elementov MVC [11]:

[1] http://www.regisfrey.com

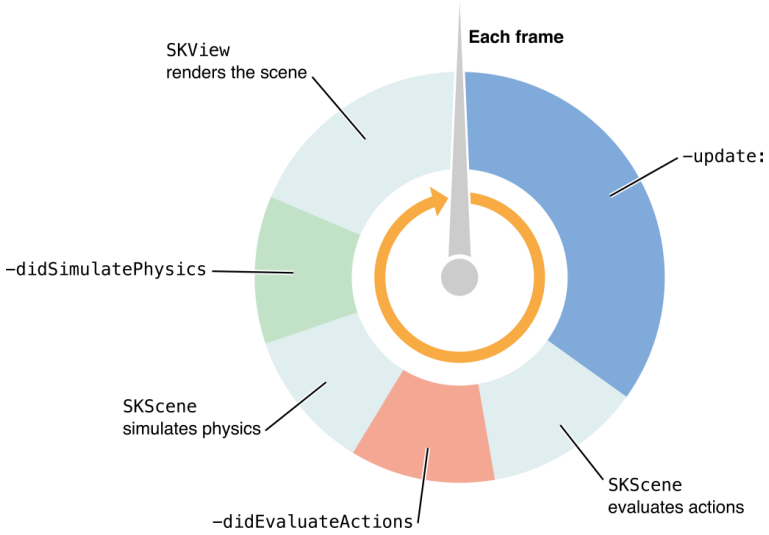
- Model predstavlja znanje, lahko je en sam objekt, večinoma pa gre za strukturo objektov. Primer modela je HyperText Markup Language (HTML).

- Pogled (ang. view) je vizualna predstavitev modela. Primer pogleda je Cascading Style Sheets (CSS).

- Upravitelj (ang. controller) je povezava med uporabnikom in sistemom. Uporabniku omogoči vnos in pregled informacij preko prikazanih pogledov. Primer upravitelja je spletni brskalnik.

3.6 Knjižnica Sprite Kit

V sklopu iOS 7 in pripadajočega SDK (ang. Software Development Kit) je Apple izdal knjižnico za izdelavo 2d iger, imenovano Sprite Kit. Poleg iOS bo knjižnica delovala tudi na zadnji verziji Mac OS X 10.9, komercialno poimenovani Mavericks. Posledično knjižnica omogoča razvijalcem uporabo enake kode za obe Applovi platformi.



Slika 3.7: Neskončna zanka, implementirana v knjižnici Sprite Kit [12].

Sprite Kit ponuja grafični prikaz in ogrodje za animacijo, ki ga lahko razvijalci uporabijo za prikaz in animiranje teksturiranih slik (ang. sprite). Kakor sorodne knjižnice, ki služijo izdelavi iger, tudi Sprite Kit uporablja tradicionalno neskončno zanko, posamezni segmenti so prikazani na sliki 3.7. Medtem ko igra skrbi za vsebino scene, Sprite Kit poskrbi za učinkovito uporabo strojne opreme za prikaz posameznih sličic, združenih v animacijo.

V knjižnici pa lahko najdemo tudi nekaj dodatnih funkcionalnosti, ki so ponavadi uporabljene v igrah, med te spadata podpora za predvajanje zvoka in simulacija osnovne fizike. Prednost uporabe knjižnice je tudi v globoki integraciji v razvojno okolje Xcode, ki pripomore k enostavnejši implementaciji. Sprite Kit podpira več različnih vsebin:

- Teksturirani in neteksturirani pravokotniki (ang. sprite)
- Besedilo
- Poljubne oblike z uporabo CGPath
- Video

Velika pomanjkljivost knjižnice, ki bo verjetno odvrnila največje število razvijalcev je ta, da z uporabo Sprite Kit razvijalci izgubijo možnost neposrednega prevajanja kode za druge platforme.

3.6.1 Zakaj Sprite Kit?

Odločitev o uporabi knjižnice Sprite Kit sem sprejel, ker sem poleg Cocos2d, Microsoft XNA ter XNI želel spoznati še kakšno alternativo. Zanimalo me je, kako so implementirane posamezne funkcije, kako težko je v primerjavi z ostalimi samo delo z njo ter kakšen je vpliv knjižnice, izdane s strani Appla, na učinkovitost in hitrost izvajanja kode. Knjižnica je popolnoma nova in pomanjkanje dokumentacije je predstavljalo svojevrsten izziv. Primerjava lastnosti knjižnic Sprite Kit ter Cocos2d je prikazana v tabeli 3.3.

Lastnost	Sprite Kit	Cocos2d
Podpora za Objective-C	Ne	Da
Grafični pogon	Da	Da
Animacije	Da	Da
Fizikalni pogon	Da	Da (Box2d ali Chipmunk)
Efekt delcev	Da	Da
Integracija v Xcode	Da	Ne
Samodejna izdelava atlasov	Da	Ne
Vgrajen urejevalnik delcev	Da	Ne
Senčilniki	Ne	Da
Kamera	Ne	Da

Tabela 3.3: Primerjava lastnosti knjižnic Sprite Kit in Cocos2d.[1]

Dolgoročna kompatibilnost

Izdelava aplikacij s pomočjo zunanjih knjižnic je dvorezen meč. Nikoli ne vemo, če bodo orodja združljiva s posodobitvami v prihodnosti oziroma če bo aplikacija po posodobitvi sploh delovala brez težav. V primerih, ko se težave pojavijo, ni jasno, koliko časa bo skupnost potrebovala za izdajo popravkov. Cocos2d je šolski primer odprtokodnega projekta, koda se nenehno razvija in pri vsaki izdaji so potrebni varnostni koraki, da zagotovijo izvajanje kode na najnovejši strojni opremi in različici iOS. S knjižnico Sprite Kit Apple ponuja orodja, s katerimi zagotavlja, da se bo koda brez težav izvajala na vseh združljivih napravah.

Razvijalcu prijazne rešitve

Preprosta uporaba je eden temeljnih razlogov za razširjenost igralnih pogonov kot je Cocos2d. Vsi nizko-nivojski programski klici (ang. API, Application programming interface) so transformirani v preproste metode. Sprite Kit

[1] http://mobile.tutsplus.com/tutorials/iphone/spritekit-vs-cocos2d/

sledi pristopu, ki ponuja na stotine metod, s ciljem poenostavitve procesa izdelovanja iger. Razvijalcu ponuja dobro oblikovan Apple API s priloženo popolno strukturirano dokumentacijo. Največja prednost pa je ta, da je v paket vključeno vse, kar potrebujemo za razvoj, fizikalni pogon, zvočne efekte, efekte delcev, teksture, upravljanje s sceno, itd.

3.7 Objava aplikacije

Smisel vsake aplikacije je doseči čim večji krog uporabnikov. Na platformi iOS nam to omogoča objava aplikacije na App Store, kar je tudi edini uraden način za izdajo aplikacije za širši krog uporabnikov.

Apple omogoča še distribucijo Ad-Hoc, pri kateri lahko aplikacijo ročno namestimo na do sto registriranih naprav ter izdajo za zaključene skupine, kar ponavadi pride v poštev v podjetjih.

3.7.1 Registracija

Za objavo aplikacije, se moramo najprej registrirati v Apple iOS Developer Program. Letna registracija stane $99 in nam omogoča objavo neomejenega števila aplikacij. V enem letu lahko registriramo do sto različnih naprav, katere lahko uporabljamo za testiranje.

Z vstopom v program dobimo tudi dostop do razvijalskega foruma in razvojnih (ti. beta) različic iOS ter Software Development Kit (SDK). Letos je Apple razvijalcem prvič zastonj ponudil tudi OS X Server, ki ga je mogoče uporabljati za samodejno testiranje aplikacij na različnih fizično priključenih napravah.

Poglavje 4

Implementacija

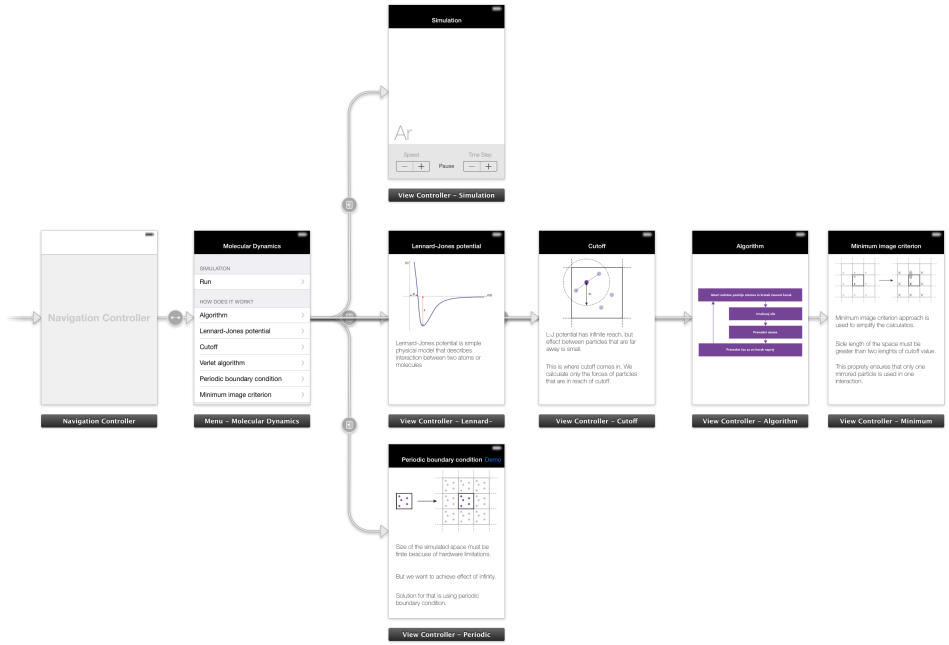
4.1 Grafični vmesnik

Za izdelavo osnovnega grafičnega vmesnika aplikacije je bilo uporabljen orodje Interface Builder, ki je del razvojnega okolja Xcode. Omogoča nam enostavno postavitev elementov, kot so tekstovne oznake, polja, gumbi, drsniki in slike. Vsakemu izmed njih lahko določimo različne lastnosti (pozicija, velikost, barva), poleg tega pa lahko na gradnike vežemo še akcije (dogodki ob interakciji, npr. klik na gumb) ter tarče (komu poslati informacije o interakciji).

Aplikacija je v grobem sestavljena iz izbirnega menija, simulacije, kjer se izvajajo interakcije ter didaktičnega dela, v katerem so predstavljeni pomembnejši deli delovanja algoritma, primeri so prikazani na sliki 4.2. Na sliki 4.1 je prikazana uporaba orodja Interface Builder, s katerim smo natančno opredelili posamezne zaslone v aplikaciji.

4.2 Algoritem molekulske dinamike

Pri vsaki simulaciji je potrebno začeti z načrtom, v primeru molekulske dinamike je bil začetek sestavljanje algoritma za izvajanje, prikazanega na sliki 4.3.

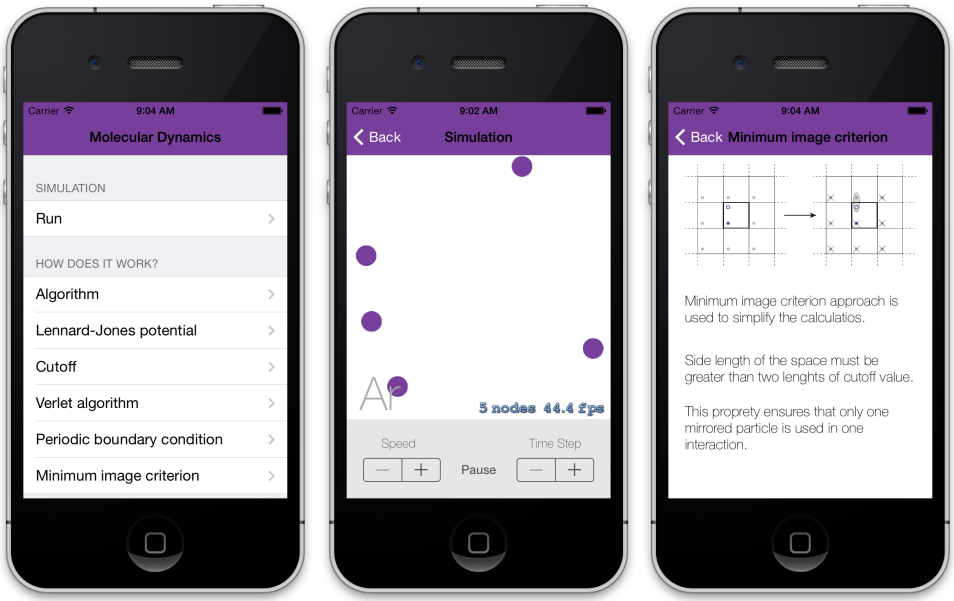


Slika 4.1: Shema posameznih zaslonov v grafičnem vmesniku.

Prve simulacije so bile dvodimenzionalne, predvsem zaradi njihove strojne zahtevnosti. Dvodimenzionalne simulacije kljub izključitvi ene dimenzije zajamejo večino fizikalne kompleksnosti. Namenoma smo se odločili za dvodimenzionalni sistem in sicer zaradi poenostavljene vizualizacije in manjše porabe procesorske moči. Dvodimenzionalni sistem je fizikalno relevanten, saj ustreza gibanju delcev, absorbiranih na površini. Difuzijsko gibanje membranskih proteinov po membrani prav tako predstavlja dvodimenzionalen problem v biomolekularnih simulacijah.

Izvajanje se začne z izbiro začetnih pozicij atomov in določitvijo časovnega koraka. Položaji atomov so določeni naključno na dvodimenzionalni ploskvi, za časovni korak pa smo izbrali $1 fs$. Na tem mestu določimo še mejni polmer in seveda velikost simulacijskega prostora.

V drugem koraku algoritma s pomočjo Lennard-Jonesovega potenciala ter časovne integracije z Verletovim algoritmom napovemo, kam se bodo delci



Slika 4.2: Začetni zaslon aplikacije z izbirnikom, prikaz simulacije in prikaz razlage simulacijskih pristopov.

premaknili v naslednjem časovnem koraku.

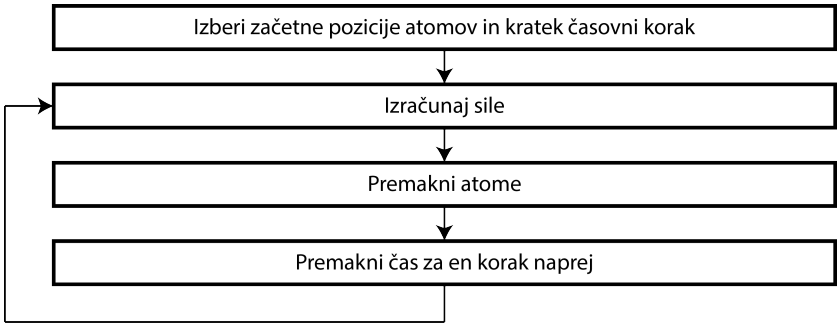
Preostaneta nam le še postavitev delcev na izračunane koordinate in prestop v naslednji cikel s premikom časa za en korak naprej.

4.3 Mikrokanonična porazdelitev

Če so vse sile, ki se pojavljajo v Newtonovih enačbah gibanja, v relaciji s potencialno energijo sistema, potem se skupna energija sistema ohranja:

$$E = E_{kin} + E_{pot} \tag{4.1}$$

Če sta tudi število atomov N in prostornina simuliranega prostora V konstantna, pravimo da je MD izvedena v mikrokanonični porazdelitvi.



Slika 4.3: Algoritem za izvajanje simulacije molekulske dinamike.

4.4 Lennard-Jonesov potencial

Lennard-Jonesov potencial 12-6 je razmeroma preprost fizikalni model, ki opisuje približno interakcijo med pari nevtralnih atomov oziroma molekul, določen je z enačbo:

$$\phi_{LJ}(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{4.2}$$

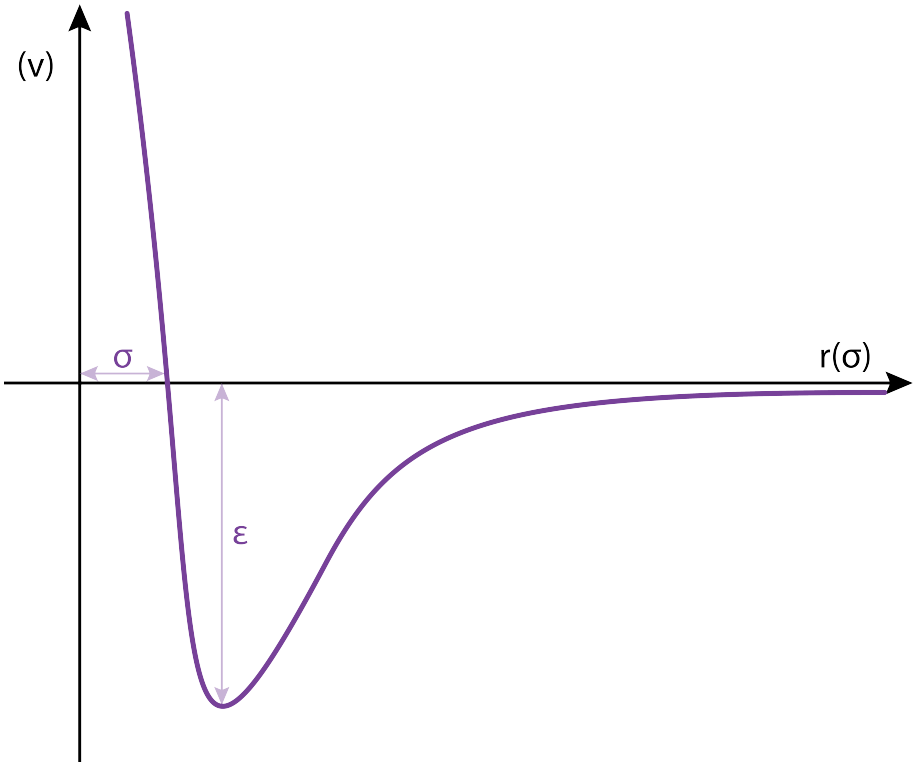
kjer ε označuje globino potenciala, σ končno razdaljo, na kateri je potencial med delci enak nič, r pa predstavlja razdaljo med delcema. Nižja vrednost ε pomeni močnejšo interakcijo med dvema delcema. Vizualno predstavitev prikazuje slika 4.4.
Potencial sestavljata vsoti odbojnega in privlačnega člena [1]:

1. Člen r^{-12} opisuje kratkosežne (Paulijeve) odbojne sile, ki nastanejo zaradi odboja pri kontaktu elektronskih orbital, na sliki 4.5 predstavljen z modro barvo,

2. r^{-6} člen pa opisuje privlačne daljnosežne interakcije (van der Waalsove disperzijske sile), na sliki 4.5 predstavljen z rdečo barvo.

Izvorna koda 4.1: Lennard-Jonesov potencial

```
EPS = 125.7 * 1.3806488 * pow(10, -5);
SIG = 0.3345; // nm
```



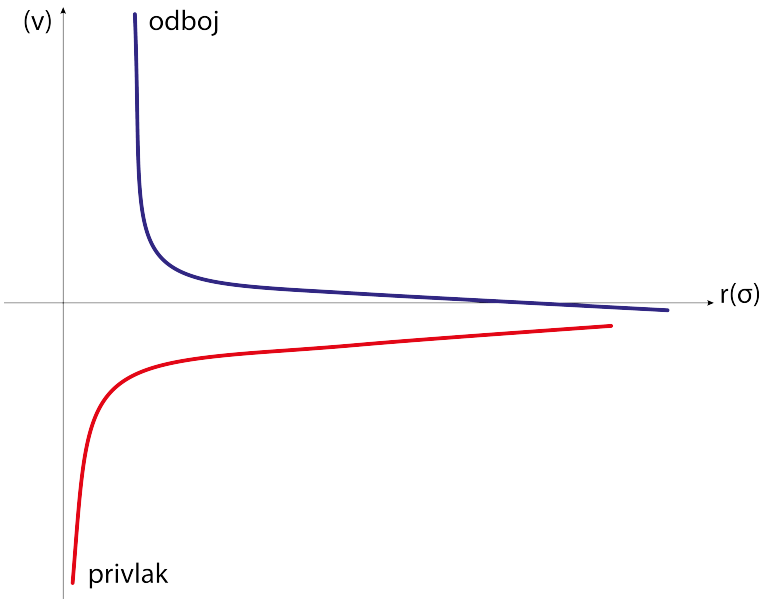
Slika 4.4: Graf odvisnosti energije od razdalje za Lennard-Jonesov potencial.

```
LJ_A = 4 * EPS * pow(SIG, 12);
LJ_B = 4 * EPS * pow(SIG, 6);

float b = -12 * LJ_A/pow(r, 14) + 6 * LJ_B/pow(r, 8);
```

4.4.1 Mejni polmer

Slovenska terminologija biomolekularnih simulacij še ni dorečena. Tako je med drugim izraz "mejni polmer" zgolj dobronameren predlog prevoda iz angleškega jezika.

Običajno zaradi neskončnega dosega potenciala iz enačbe 4.2 uvedemo mejni polmer R_c (ang. cutoff) in ignoriramo interakcije med atomi, ki so

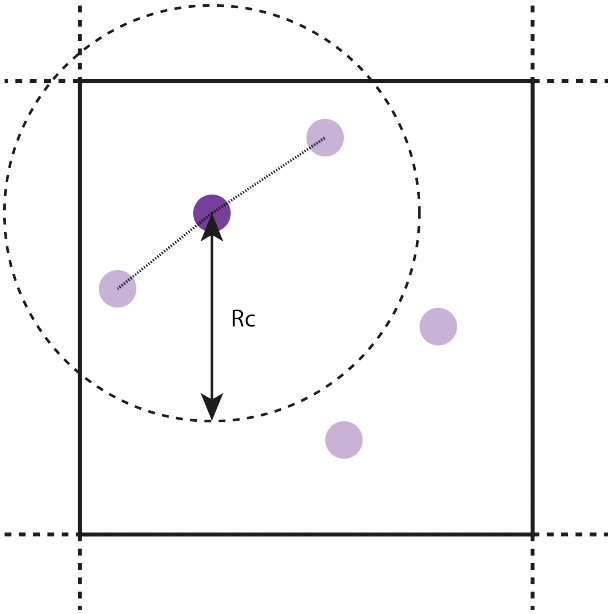


Slika 4.5: Ločeni komponenti za odboj in privlak v Lennard-Jonesovemu potencialu.

med seboj oddaljeni za več kot R_c. Z uvedbo mejnega polmera računamo samo interakcije med delci, za katere sumimo, da imajo nezanemarljiv medsebojni vpliv. Računanje interakcij z bolj oddaljenimi delci preskočimo, saj je vpliv na opazovan delec zanemarljiv. S pomočjo takšne rešitve prihranimo veliko količino računalniške moči. Na sliki 4.6 vidimo, kako z uporabo mejnega polmera določimo, katere atome iz okolice bomo upoštevali pri izračunu interakcije.

Uporaba mejnega polmera prinese tudi nezaželen efektČ ko delec preskoči mejno razdaljo, tudi energija rahlo naraste. V situacijah ko imamo več takšnih primerov hkrati, je velika verjetnost, da nam sistem ne ohrani celokupne energije v simulaciji. V izogib temu problemu je potencial ponavadi zamaknjen, da pri mejni razdalji izgine:

$$V(r) = \begin{cases} \phi_{LJ}(r) - \phi_{LJ}(R_c) & \text{pri } r \leq R_c \\ 0 & \text{pri } r \leq R_c \end{cases} \tag{4.3}$$



Slika 4.6: Prikaz uporabe mejnega polmera pri izbiri delcev za interakcijo.

4.5 Fizične meje sistema

V računalniški simulaciji ne moremo simulirati neskončno velikega sistema, moramo mu določiti meje. Najbolj preprosta rešitev je, da na določeno mesto meje enostavno postavimo in zapremo atome znotraj tako narejenega prostora. Težava pri takšni rešitvi je ta, da ni najbolj realistična. Tudi če vzamemo zelo velik sistem, bo število atomov N znotraj le-tega zanemarljivo v primerjavi s številom vseh atomov v delcu snovi. Razmerje med njimi bo veliko večje kot je v realnem svetu in efekti površine bodo imeli prevelik vpliv.

4.5.1 Periodični robni pogoj

Nastalo težavo ponavadi rešimo s pomočjo periodičnega robnega pogoja (ang. periodic boundary condition). Vse delce zapremo v škatlo, to škatlo pa v vseh kartezičnih smereh podvajamo v neskončnost in s tem prostor oziroma ravnino popolnoma napolnimo kot je prikazano na sliki 4.7. Lahko si pred-

Fizikalna količina	Enota	Vrednost za argon (Ar)
dolžina	σ	3.4×10^{-10} m
energija	ε	1.65×10^{-21} J
masa	m	6.69×10^{-26} kg
čas	$\sigma(m/\varepsilon)^{1/2}$	2.17×10^{-12} s
hitrost	$(\varepsilon/m)^{1/2}$	1.57×10^{2} m/s
sila	ε/σ	4.85×10^{-12} N
pritisk	ε/σ^3	4.20×10^{7} N/m^2
temperatura	ε/k_B	120 K

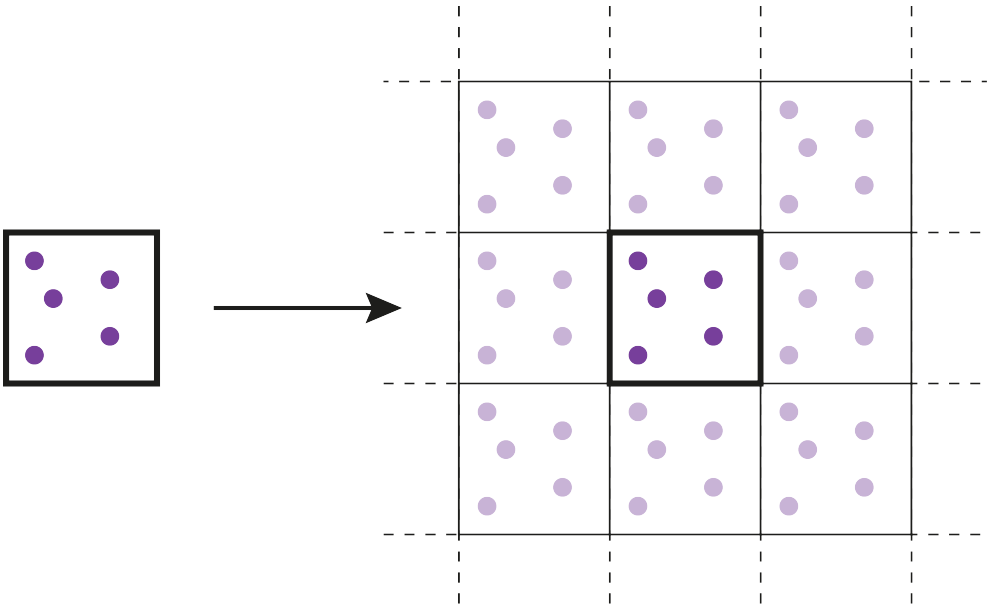
Tabela 4.1: Sistem enot, uporabljen v simulaciji MD interakcije delcev z uporabo Lennard-Jonesovega potenciala za tekoči argon (Ar) [13].

Fizikalna količina	Enota	Ekvivalent SI
dolžina	1 Ångström	10^{-10} m
energija	1 kcal/mol	4184 kJ/mol
naboj	1 elektron	$1.6021892 \times 10^{-19}$ C

Tabela 4.2: Enote, ki niso del sistema SI, vendar so zelo primerne za uporabo pri simulacijah molekulske dinamike [1].

stavljamo, da simuliramo kristal.

Vsi delci, ki jih dobimo s takšno implementacijo, se premikajo skupaj, v simulaciji pa je predstavljen in prikazan samo en prostor. S ponavljanjem sistemov v neskončnost posamezen delec ne vpliva samo na delce, ki se nahajajo v istem prostoru, ampak tudi na vse delce, ki se nahajajo v prostorih poleg osnovnega. Predstavljamo si lahko, da lahko interakcija poteka skozi meje prostora. S tem smo odpravili vpliv mejnih površin na naš sistem. Ob uporabi periodičnega robnega pogoja pozicija simuliranega prostora ne vpliva na sile delcev. Lahko bi sklepali, da s takšnim pristopom zelo povečamo število delcev, ki medsebojno vplivajo drug na drugega. V resnici pa zaradi precej



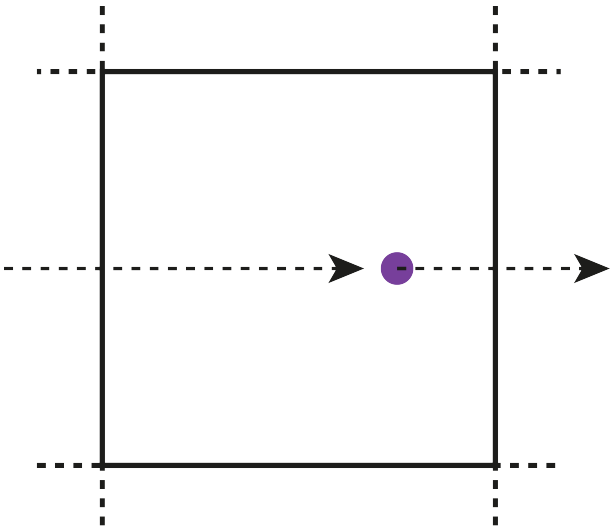
Slika 4.7: Ponavljanje sistemov v vse smeri z uporabo periodičnega robnega pogoja s formalno vpeljanim kristalnim poljem.

kratkega mejnega polmera R_c temu ni tako.

Atom, ki zadane rob simulacijskega prostora, moramo premakniti na ustrezno mesto na drugi strani (slika 4.8), implementacija je prikazana v izvorni kodi 4.2. Pri vsakem koraku preverimo, če se je atom s svojo sredino dotaknil stene in ga po potrebi premaknemo na drugo stran.

Izvorna koda 4.2: Periodični robni pogoj

```
// Periodic boundary condition
if (atom.position.y >= self.frame.size.height)
    atom.position = (CGPointMake(atom.position.x, 0));
else if (atom.position.x >= self.frame.size.width)
    atom.position = (CGPointMake(0, atom.position.y));
else if (atom.position.y <= 0)
    atom.position = (CGPointMake(atom.position.x, self.frame.size.height));
else if (atom.position.x <= 0)
    atom.position = (CGPointMake(self.frame.size.width, atom.position.y));
```



Slika 4.8: Prehod delca na drugo stran pri stiku z robom simuliranega prostora.

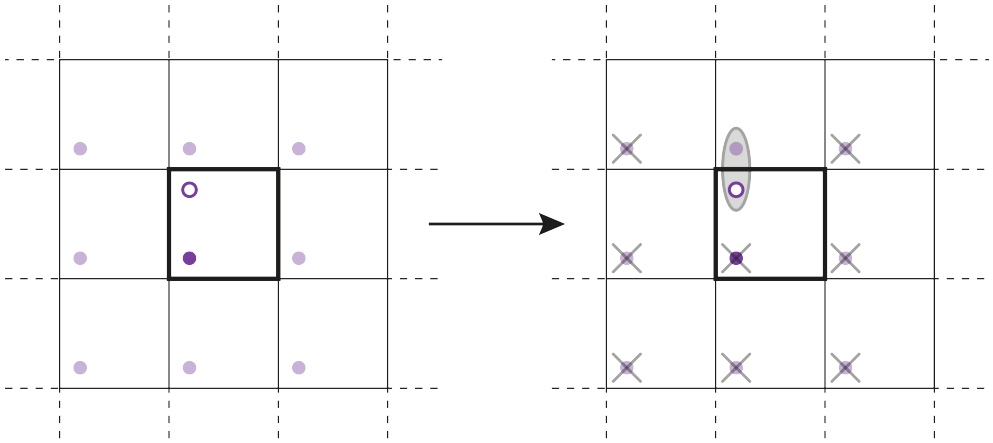
Razdaljo med dvema delcema izračunamo s pomočjo splošne formule za izračun razdalje med dvema točkama v koordinatnem sistemu, popravimo pa jo tako, da upoštevamo tudi delce, ki so čez mejo prostora, implementacija je razvidna iz izvorna kode 4.3.

Izvorna koda 4.3: Razdalja med delcema

```
double sideh = self.frame.size.width/2;
double side = self.frame.size.width;
double xx = atom2.position.x-atom.position.x;
double yy = atom2.position.y-atom.position.y;

if (xx < sideh*-1)
    xx = xx + side;
else if (xx > sideh)
    xx = xx - side;

if (yy < sideh*-1)
    yy = yy + side;
```

```
else if (yy > sideh)
    yy = yy - side;

double distance = sqrt(xx*xx+yy*yy);
```

4.5.2 Minimalni kriterij za preslikavo



Slika 4.9: Uporaba minimalnega kriterija za preslikavo, s katerim zagotovimo izbiro le enega delca v preslikanih soležnih prostorih.

Program MD lahko še bolj poenostavimo z uporabo tako imenovanega minimalnega kriterija za preslikavo (ang. minimum image criterion). Zagotoviti moramo, da je dolžina stranice simuliranega prostora daljša od dveh dolžin mejnega polmera ($2R_c$). S slike 4.9 je razvidno, da ob upoštevanju tega pogoja delec i (modre barve), vpliva samo na en delec j (rdeče barve), izmed vseh, v katere se j preslika. Minimalni kriterij za preslikavo nam narekuje, da izmed vseh preslikav delca j izberemo samo najbližjo, ostale pa zanemarimo. Na ta način smo se v pretežni meri znebili nezaželenih efektov kristalnega polja.

4.6 Verletov algoritem

Pomemben element simulacije molekulske dinamike je njen algoritem za časovno integracijo. Z njim integriramo enačbe gibanja posameznih delcev, ki imajo medsebojen vpliv in določimo trajektorijo za naslednji časovni korak.

Algoritem za časovno interakcijo je diferenčna metoda (ang. finite difference method), pri kateri je čas končen in diskreten, časovni korak Δt pa je razdalja med zaporednima točkama na časovni premici. Če poznamo pozicije in nekatere odvode ob času t, nam integracija izračuna količine v prihodnosti $t + \Delta t$. Z iteracijo postopka lahko sledimo časovni evoluciji sistema poljubno dolgo.

Približna natančnost postopka pa vodi tudi v nekatere napake:

- Napake zaradi krajšanja
- Napake zaradi zaokroževanja

Diferenčne enačbe rešujemo s pomočjo Taylorjevih vrst, pri katerih uporabljamo krajšanje. Napake te vrste niso odvisne od implementacije algoritma, temveč so del njega. Druga vrsta napak, tiste zaradi zaokroževanja, so posledica natančnosti implementacije. Računalnik lahko operira s končnim številom mest za decimalno vejico in ob zaokroževanju se natančnost manjša. Vpliv takšnih napak lahko seveda manjšamo z večanjem natančnosti pri implementaciji, a jih popolnoma ni mogoče odpraviti.

Najbolj enostavno zmanjšamo vpliv napak z manjšanjem časovnega koraka Δt. Pri velikih vrednostih Δt so velike tudi napake pri krajšanju, z zmanjševanjem Δt pa se hitro manjšajo. Napake zaradi zaokroževanja niso toliko odvisne od Δt, postanejo pa skoraj zanemarljive pri uporabi 64-bitne natančnosti.

Največkrat uporabljen algoritem za časovno integracijo v molekulski dinamiki je t. i. Verletov algoritem. V osnovi želimo zapisati dve Taylorjevi

vrsti tretjega reda, eno za nazaj in eno za naprej v času:

$$\begin{aligned} r(t+\Delta t) &= r(t) + v(t)\Delta t + \left(\tfrac{1}{2}\right) a(t)\Delta t^2 + \left(\tfrac{1}{6}\right) b(t)\Delta t^3 + O(\Delta t^4) \\ r(t-\Delta t) &= r(t) - v(t)\Delta t + \left(\tfrac{1}{2}\right) a(t)\Delta t^2 - \left(\tfrac{1}{6}\right) b(t)\Delta t^3 + O(\Delta t^4) \end{aligned} \tag{4.4}$$

kjer v označuje hitrosti, a pospeške in b tretji odvod od r v odvisnosti od t. Z združevanjem omenjenih enačb dobimo:

$$r(t+\Delta t) = 2r(t) - r(t-\Delta t) + a(t)\Delta t^2 + O(\Delta t^4) \tag{4.5}$$

Neposreden izračun hitrosti je pri takšni obliki Verletovega algoritma nemogoč. Čeprav hitrosti niso potrebne za evolucijo sistema skozi čas, so zelo pomembne pri preverjanju pravilnosti simulacije MD, ki mora ustrezati zakonu o ohranitvi energije. Potrebujemo jih za računanje kinetične energije K, s pomočjo katere testiramo ohranjanje energije z enačbo $E = K+V$. Hitrosti sicer lahko izračunamo s ponovitvijo postopka, a temu se lahko izognemo z uporabo hitrostnega Verletovega algoritma:

$$\begin{aligned} a(t) &= -\left(\tfrac{1}{m}\right) \nabla V(r(t)) \\ r\left(t+\Delta t\right) &= r(t) + v(t)\Delta t + \left(\tfrac{1}{2}\right) a(t)\Delta t^2 \\ v\left(t+\tfrac{\Delta t}{2}\right) &= v(t) + \left(\tfrac{1}{2}\right) a(t)\Delta t \\ a\left(t+\Delta t\right) &= -\left(\tfrac{1}{m}\right) \nabla V(r(t+\Delta t)) \\ v\left(t+\Delta t\right) &= v\left(t+\tfrac{\Delta t}{2}\right) + \left(\tfrac{1}{2}\right) a(t+\Delta t)\Delta t \end{aligned} \tag{4.6}$$

Hitrostni Verletov algoritem nam omogoča nam izračun pozicije, hitrosti in pospeška delcev za $t+\Delta t$ iz istih količin hkrati ob času t. Na ta način se izognemo potrebi za dvakratnim shranjevanjem podatkov za vsako izmed teh količin, uporabimo pa trikrat več pomnilnika.

Dejansko nas pri simulacijah MD ne zanima natančna trajektorija, kot naprimer pri balističnih izračunih, temveč enačbe gibanja rešujemo bolj zaradi vzorčenja faznega prostora.

4.7 Primerjava zank FOR

Velik del algoritmov, uporabljenih v aplikaciji, temelji na zanki FOR. V primeru molekulske dinamike gre za nenehno sprehajanje skozi seznam delcev in optimizacija zanke lahko bistveno pripomore k hitrosti izvajanja simulacije.

Za primerjavo hitrosti izvajanja različnih zank sem spisal testni primer. V poljubno velik seznam vnesemo naključna števila, v zanki pa jih seštejemo. Meritve so opravljene z uporabo razlike v času začetka in zaključka izvajanja. Objective-C pozna več različnih zank FOR:

- objectAtIndex naštevanje uporablja zanko FOR, ki povečuje celoštevilski števec in do podatkov dostopa s pomočjo metode objectAtIndex, je najbolj enostaven način naštevanja.

Izvorna koda 4.4: objectAtIndex

```
for (int i=0; i<[nsArray count]; i++)
    total += [[nsArray objectAtIndex:i] integerValue];
```

- NSEnumerator

Izvorna koda 4.5: NSEnumerator

```
NSEnumerator *enumerator = [nsArray objectEnumerator];
   id object;
   while (object = [enumerator nextObject]) {
      total += [object integerValue];
   }
```

- NSFastEnumerator uporablja hiter seznam C za optimizacijo iteracije. Tak način je hitrejši od navadne zanke NSEnumerator, poleg tega pa

ima v Objective-C tudi prilagojeno sintakso.

Izvorna koda 4.6: NSFastEnumerator

```
for (NSNumber *i in nsArray)
    total += [i integerValue];
```

- Block enumerator, podpira zaporedno in vzporedno izvajanje

Izvorna koda 4.7: Block enumerator

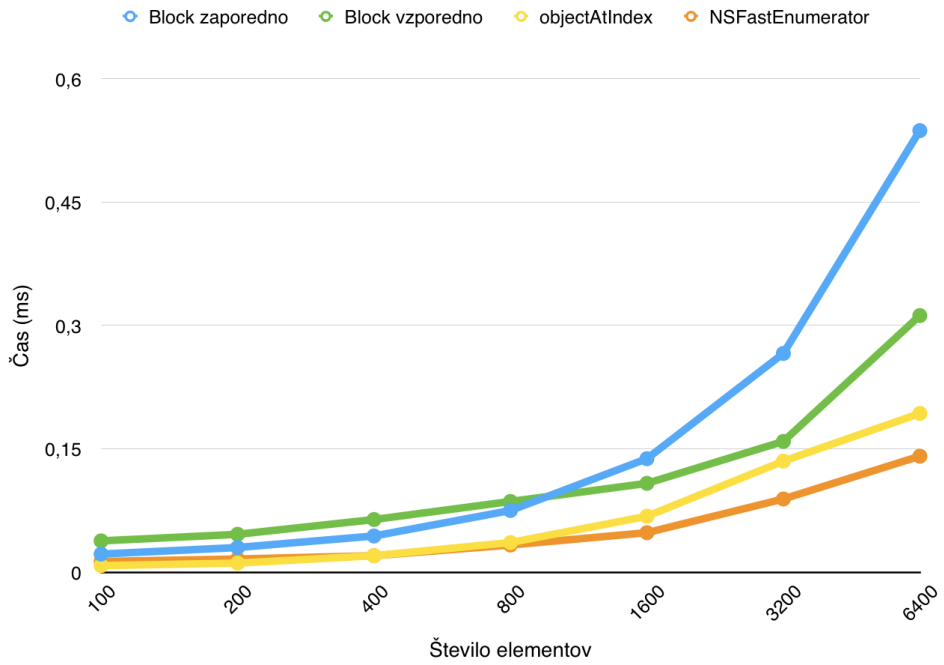
```
[nsArray enumerateObjectsUsingBlock:^(id object, NSUInteger index, BOOL *stop)
        {
    [object integerValue];
}];

[nsArray enumerateObjectsWithOptions:NSEnumerationConcurrent usingBlock:^(id
        object, NSUInteger idx, BOOL *stop) {
    [object integerValue];
}];
```

- Uporaba metode makeObjectsPerformSelector, ki vsem elementom pošlje enak ukaz

Namesto standardnega seznama NSArray, vgrajenega v Objective-C, je mogoča uporaba tudi seznama iz programskega jezika C. Hitrost slednjega je nekoliko višja, a seznamu naknadno ni mogoče spreminjati velikosti, z uporabo seznama C pa zgubimo tudi vse prednosti in napredne zmogljivosti seznama NSArray. V analizi sem se osredotočil zgolj na prvi dve zanki, ki sta se izvajali nad navadnim seznamom tipa NSArray.

Izvorna koda 4.8: Merjenje časa izvajanje

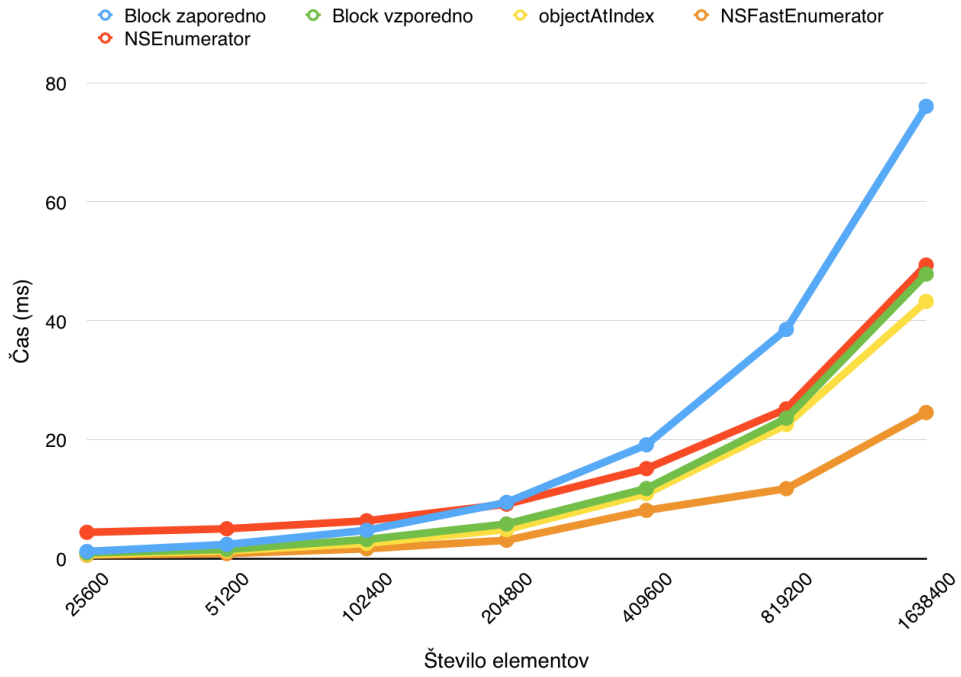
```
methodStart = [NSDate date];
// izvajanje zanke
executionTime = [[NSDate date] timeIntervalSinceDate:methodStart];
NSLog(@"t = %f", executionTime*1000);
```



Slika 4.10: Primerjava hitrosti izvajanja zank za majhno število elementov.

Rezultati so prikazani na grafu 4.10 za majhno število elementov ter 4.11 za veliko število elementov. Na ordinatni osi je prikazan čas izvajanja zanke, na abscisni osi pa število elementov, uporabljenih v seznamu. Modra črta prikazuje hitrost izvajanja zanke pri uporabi iteratorja (izvorna koda 4.6) v milisekundah, rdeča črta pa uporabo indeksiranja (izvorna koda 4.4). Rezultati so pokazali, da se prednost uporabe iteratorja za manjše število elementov niti ne opazi, natančen pregled celo pokaže, da je iterator za nabor manjši od 800 elementov, tudi do 2-krat počasnejši. Z naraščanjem števila elementov pa se hitrost občutno poveča in faktor hitrosti govori v prid uporabi iteratorja.

Preizkusi so pokazali, da pri simulaciji, ki je tema te diplomske naloge, ne moremo pričakovati več kot 50 delcev v sistemu, saj hitrost delovanja z večanjem števila le-teh strmo pada. Prav zaradi tega sem pri izdelavi aplikacije uporabil izključno zanke, ki elemente indeksirajo. Zamenjava za iterator



Slika 4.11: Primerjava hitrosti izvajanja zank za veliko število elementov.

bi bila smiselna v primeru uporabe enake programske kode na neprimerljivo zmogljivejši strojni opremi, ki jo poganja OS X.

Poglavje 5

Sklepne ugotovitve

Tema diplomske naloge se je izkazala kot zelo zanimiva z več perspektiv. Računalniško znanje smo nadgradili in spoznali razvoj aplikacij za eno izmed najbolj profitabilnih mobilnih platform. Znanje, potrebno za razvoj, pa samo po sebi ne prinaša nobene dodane vrednosti, če ne najdemo realnega problema, pri katerem ga lahko izkoristimo. Prav zaradi tega smo morali podrobno spoznati biomolekularne simulacije in različne pristope za njihovo izvajanje.

Smisel aplikacije je prikaz molekulske dinamike, podprt z razlago delovanja in metod za simulacijo. Potencialni uporabniki so vsi, ki jih osnove biomolekularnih simulacij zanimajo, bodisi izhajajo iz računalniškega sveta ali sveta kemije. Aplikacija se je izkazala kot zelo uporabna za raziskovalce, ki se ukvarjajo z eksperimentalno kemijo oziroma farmacijo, saj se v trenutnih univerzitetnih programih zelo malo dotaknejo teoretične plati. Za je aplikacija lahko hiter pregled osnov, ki jih lahko uporabijo pri teoretični računski podpori za svoje eksperimente.

V prihodnosti je mogoča nadgradnja aplikacije z lokalizacijo in prevajanjem v več jezikov. Smiselna je tudi optimizacija aplikacije in pospešitev delovanja, s strani nekoga, ki ima boljši vpogled v molekulsko dinamiko. Na novejših napravah lahko simuliramo dodatno dimenzijo in območje simulacije preselimo iz dveh dimenzij v tridimenzionalni prostor. Področje didaktičnih

vsebin in razlag bi bilo smiselno nadgraditi z interaktivnimi vsebinami, ki bi se odzivale na uporabnikove ukaze in s tem skrajšale čas, potreben za razumevanje in izboljšale uporabniško izkušnjo. Za boljši izkoristek vsebin je v prihodnosti predvidena tudi širitev na tablične naprave iPad.

Aplikacija, izdelana v sklopu diplomskega dela, ob zaključku pisanja čaka na pregled in objavo v AppStore.

Seznam kratic

API Application Programming Interface 20

ARC Automatic Reference Counting 14

BPTI bovine pancreatic trypsin inhibitor 5

BSD Berkeley Software Distribution 14

CSS Cascading Style Sheets 18

DSP Digital Signal Processing 14

GCD Grand Central Dispatch 14

HTML HyperText Markup Language 18

IDE Integrated Development Environment 15

MD molekulska dinamika 3, 25, 30, 33, 35, 36, 47

MVC Model-View-Controller 17

SDK Software Development Kit 18, 21

UNIVAC Universal Automatic Computer 5

XML Extensible Markup Language 14

XNI XNA for iOS 19

Slike

Tabele

Literatura

[1] A. Perdih, "Seminar iz molekulskega modeliranja pri Farmacevtski kemiji III," 2012.

[2] F. Ercolessi, "A molecular dynamics primer," *Spring College in Computational Physics, ICTP, Trieste*, 1997.

[3] B. Alder and T. Wainwright, "Phase transition for a hard sphere system," *The Journal of Chemical Physics*, vol. 27, no. 5, pp. 1208–1209, 1957.

[4] B. Alder and T. Wainwright, "Studies in molecular dynamics. I. General method," *The Journal of Chemical Physics*, vol. 31, p. 459, 1959.

[5] J. P. Eckert Jr, J. R. Weiner, H. F. Welsh, and H. F. Mitchell, "The UNIVAC system," in *Papers and discussions presented at the Dec. 10-12, 1951, joint AIEE-IRE computer conference: Review of electronic digital computers*, ACM, 1951.

[6] IBM, "704 data processing system," 2013. http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP704.html.

[7] A. Rahman, "Correlations in the motion of atoms in liquid argon," *phys. Rev*, vol. 136, no. 2A, pp. 405–411, 1964.

[8] F. H. Stillinger and A. Rahman, "Improved simulation of liquid water by molecular dynamics," *The Journal of Chemical Physics*, vol. 60, p. 1545, 1974.

[9] J. A. McCammon, "Dynamics of folded proteins," *Nature*, vol. 267, p. 16, 1977.

[10] "Apple WWDC 2012 Keynote," 2012.

[11] T. M. H. Reenskaug, "The original MVC reports," 1979.

[12] *Sprite Kit Programming Guide*. Apple inc., 2013.

[13] A. T. Beu, *Molecular Dynamics Simulations*. University "Babes-Bolyai", Faculty of Physics, Cluj-Napoca, Romania, 2011.