AGH
Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Fizyki i Informatyki Stosowanej

Praca inżynierska

Olaf Schab

kierunek studiów: **informatyka stosowana**

Symulacja komputerowa zaniku sygnału luminescencyjnego w skaleniach

Opiekun: **dr inż. Grzegorz Gach**

Kraków, styczeń 2017

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

.....
(czytelny podpis)

Merytoryczna ocena pracy przez opiekuna

Merytoryczna ocena pracy przez recenzenta

Składam serdeczne podziękowanie dr inż. Grzegorzowi Gachowi za cenne wskazówki i pomoc przy pisaniu niniejszej pracy.

Spis treści

Rozdział 1

Wstęp

W archeologii i geologii wiek próbek można określić wykorzystując skalenie. Jest to możliwe dzięki zjawisku stymulowanej luminescencji. Już w latach 60- tych XX wieku powstał pomysł wykorzystania termoluminescencji, która znalazła wiele zastosowań w archeologii i naukach o Ziemi. Jednym z głównych tematów badań jest datowanie z wykorzystaniem skaleni przy użyciu ich termoluminescencji oraz atermicznym zanikiem luminescencji powodującym zaniżanie daty próbek, który utrudnia datowanie. Atermiczny zanik związany jest z faktem zachodzenia rekombinacji zlokalizowanej głównie poprzez zachodzące zjawisko tunelowania elektronów z pułapek elektronowych oraz centrów rekombinacyjnych.

1.1 Założenia projektu

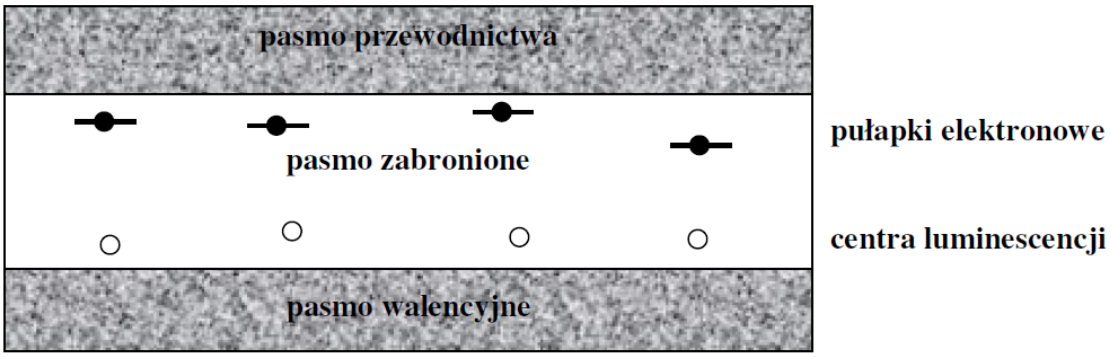
Celem projektu było stworzenie programu symulującego atermiczny zanik sygnału luminescencyjnego w skaleniach. Głównym założeniem było otrzymanie wykresu obrazującego zmianę ilości elektronów znajdujących się w stanie wzbudzonym (znajdujących się w pułapkach) wraz z upływem czasu.

1.2 Zjawisko luminescencji w skaleniach

Jednym ze składników środowiska naturalnego jest promieniowanie jonizujące. Najważniejszym jego źródłem są izotopy promieniotwórcze zawarte w skorupie ziemskiej, atmosferze i biosferze, emitujące promieniowanie α, β i γ. W ostatnim okresie, bardzo krótkim w sensie geologicznym oraz historycznym, pojawiły się nowe jego źródła, związane z działalnością człowieka w zakresie zbrojeń atomowych i energetyki jądrowej. W dalszym jednak ciągu najważniejszym źródłem promieniowania jonizującego w środowisku są izotopy promieniotwórcze, które weszły w skład Ziemi w okresie formowania się Układu Słonecznego. Są to przede wszystkim długożyciowe izotopy uranu U^{238}, U^{235} i toru Th^{232}. Promieniowanie to niesie energię, którą pochłaniają wszystkie substancje występujące w środowisku - w tym skalenie. Oczywistym jest, że im dłużej

substancja jest poddana promieniowaniu, tym pochłonie jego większą dawkę.

Rzeczywisty kryształ skalenia nie jest idealny. Zawiera on zawsze nieregularności i defekty struktury sieci krystalicznej. Część z nich, znajdująca się w paśmie wzbronionym, może mieć charakter pułapek, które są zdolne do wychwytywania elektronów z pasma przewodnictwa i przetrzymywania ich przez długi czas. Stan kryształu, w którym część lub wszystkie pułapki są zapełnione schwytanymi elektronami, charakteryzuje się nadwyżką energii w porównaniu ze stanem podstawowym. Nadwyżka ta może zostać wyzwolona i wyemitowana np. w postaci światła. Właśnie to zjawisko wykorzystywane jest jako jedna z metod datowania obiektów archeologicznych i osadów geologicznych.

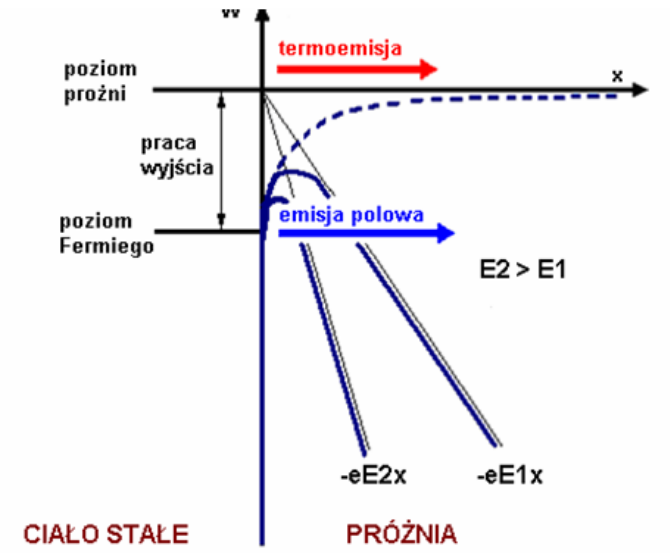


Rysunek 1.1: Struktura pasmowa skalenia. [1]

Zdarza się jednak, że mimo iż uwięziony elektron nie otrzymał dodatkowej energii, będzie w stanie zrekombinować z dziurą. Jest to możliwe dzięki **zjawisku tunelowania**.

1.3 Zjawisko tunelowania

Tunelowanie to proces kwantowo-mechaniczny, w którym cząstka ma niezerowe prawdopodobieństwo przejścia przez barierę potencjalną nawet wtedy, gdy energia cząstki jest mniejsza od wysokości bariery potencjału.



Rysunek 1.2: Ilustracja mechanizmu emisji polowej. [2]

W przypadku skaleni prawdopodobieństwo, że elektron nie przetuneluje do centrum rekombinacji wyrażone jest wzorem: [3]

$$P = e^{\frac{-t}{\tau}} \tag{1.1}$$

$$\tau = S^{-1} e^{\alpha r} \tag{1.2}$$

gdzie:

- t - czas
- S - stała częstotliwość prób ucieczki
- α - stała zależna od różnicy energii pułapki i dziury
- r - odległość do przetunelowania

Wzór 1.1 wynika z rozwiązania 1D równania Schrödingera z barierą potencjału.

Rozdział 2

Wybór stosowanych technologii

2.1 Język C++11 oraz wybór kompilatora

Aplikacja została napisana z użyciem języka C++11 [4]. Język ten wybrano z kilku powodów. Skorzystano tu z jego natury jako języka zorientowanego obiektowo. Dzięki temu podczas wykonania programu stworzono klasy odpowiadające obiektom w świecie realnym - elektrony, pułapki elektronowe itp. Kolejnym powodem, dla którego wykorzystano język C++ jest jego wydajność. Kod napisany w języku C++ jest kompilowany bezpośrednio i w pełni do kodu maszynowego wykonywanego przez procesor komputera. Dzięki temu otrzymujemy maksymalną możliwą wydajność, inaczej niż kiedy używamy innych obiektowych języków programowania wysokiego poziomu, które są kompilowane do kodu zarządzanego (tzw. kodu bajtowego na przykład w przypadku Javy), gdzie dodatkowy narzut generowany jest przez warstwę pośrednią w postaci maszyny wirtualnej.

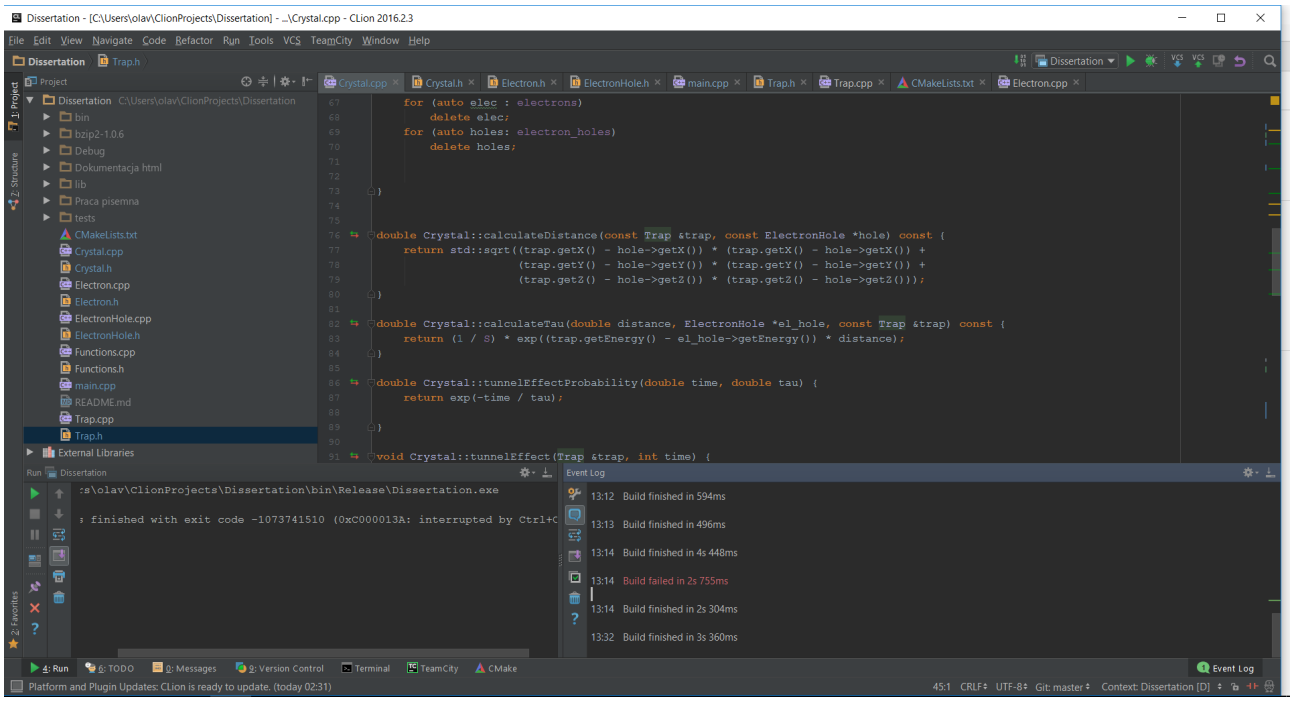
C++11 jest standardem języka C++ zastępującym standard C++03. Wprowadza on kilka dodatków do rdzenia języka oraz znacznie rozszerza bibliotekę standardową C++. Dla przykładu wprowadza on słowo kluczowe *auto* jako zastępczy typ zmiennej, który zostanie wydedukowany na podstawie wartości, za pomocą której zmienna zostanie zainicjalizowana. Nowa biblioteka wprowadza również kilka silników do generacji liczb pseudolosowych. W projekcie wykorzystano generator oparty na jednostajnie ciągłym rozkładzie prawdopodobieństwa.

Do kompilacji użyto darmowego kompilatora GCC (Gnu Compiler Collecion), konkretnie jego implementacji dla platformy Windows - MinGW w wersji 3.2.2 (Minimalist GNU for Windows).

2.2 Środowisko programistyczne CLion

Program był tworzony z użyciem wieloplatformowego zintegrowanego środowiska programistycznego języków C/C++ - **CLion** (wersja 2016.2.3) produkcji spółki JetBrains. Było to podyktowane głównie znajomością produktów firmy JetBrains. CLion jest produktem komercyj-

nym, jednak skorzystano tu z darmowej licencji studenckiej. IDE produkcji JetBrains obsługuje również system zarządzania kompilacją **CMake**, którego główną cechą jest niezależność od używanego kompilatora oraz platformy sprzętowej - CMake nie kompiluje programu samodzielnie lecz tworzy pliki z regułami kompilacji dla konkretnego środowiska.



Rysunek 2.1: Wygląd okna programu CLion [1]

Dużą zaletą podczas pracy z IDE produkcji JetBrains jest jego intuicyjna i prosta obsługa całości projektu. Dodatkowo CLion posiada możliwość bezpośredniego połączenia projektu z wybranym serwerem kontroli wersji co uprościło pracę nad projektem. Z racji tego, że w chwili obecnej program do kompilacji używa tylko systemu CMake, niezbędne było nauczenie się jego obsługi oraz odpowiedniej składni. Ostatecznie kod pliku *CMakeLists.txt* wygląda następująco:

```
cmake_minimum_required(VERSION 3.6)
project(Dissertation)
set(CMAKE_EXE_LINKER_FLAGS "-static-libgcc -static-libstdc++ -static")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=gnu++11 -O2 -g -Wall")
SET(CMAKE_FIND_LIBRARY_SUFFIXES ".a")
SET(BUILD_SHARED_LIBRARIES OFF)
set(SOURCE_FILES main.cpp Electron.h Electron.cpp Trap.cpp Trap.h Functions.cpp Function
add_executable(Dissertation ${SOURCE_FILES})
target_link_libraries(Dissertation)
```

2.3 System kontroli wersji Git

System kontroli wersji śledzi wszystkie zmiany dokonywane w plikach i umożliwia przywołanie dowolnej wcześniejszej wersji. Jest to przydatne narzędzie w łączeniu zmian dokonanych przez wiele osób w różnym czasie.

W procesie tworzenia aplikacji wykorzystano system kontroli wersji Git oraz darmowy hostingowy serwis internetowy GitHub. Głównym powodem wykorzystania tego systemu była jego popularność oraz prostota użytkowania. System kontroli wersji okazał się być niezwykle użytecznym narzędziem pozwalającym na śledzenie zmian w kodzie oraz wprowadzanie testowych rozwiązań bez ryzyka zniszczenia kodu. Cały projekt można pobrać ze strony:

`https://github.com/Sharkuu/Dissertation`

2.4 Wizualizacja wyników - Gnuplot

Po wygenerowaniu danych,by zwizualizować wykres obrazujący zmianę ilości elektronów znajdujących się w stanie wzbudzonym (znajdujących się w pułapkach) wraz z upływem czasu, skorzystano z programu **Gnuplot**.

Gnuplot jest narzędziem do kreślenia wykresów 2D i 3D, sterowanym z wiersza poleceń. Ta jego cecha oraz możliwość pobierania zewnętrznych danych umożliwia jego wykorzystanie do wizualizacji wyników z innych programów. Możliwe są dwa tryby pracy:

- Jeżeli uruchomimy program bez podania żadnych parametrów (polecenie gnuplot), będzie on pracował w trybie interaktywnym (będzie oczekiwał na komendy użytkownika i kolejno je wykonywał). Aby zakończyć pracę, należy wpisać komendę exit, quit lub q.

- Jeżeli podamy parametr (polecenie gnuplot plik), program wykona polecenia zawarte w pliku i zakończy pracę (tryb wsadowy)

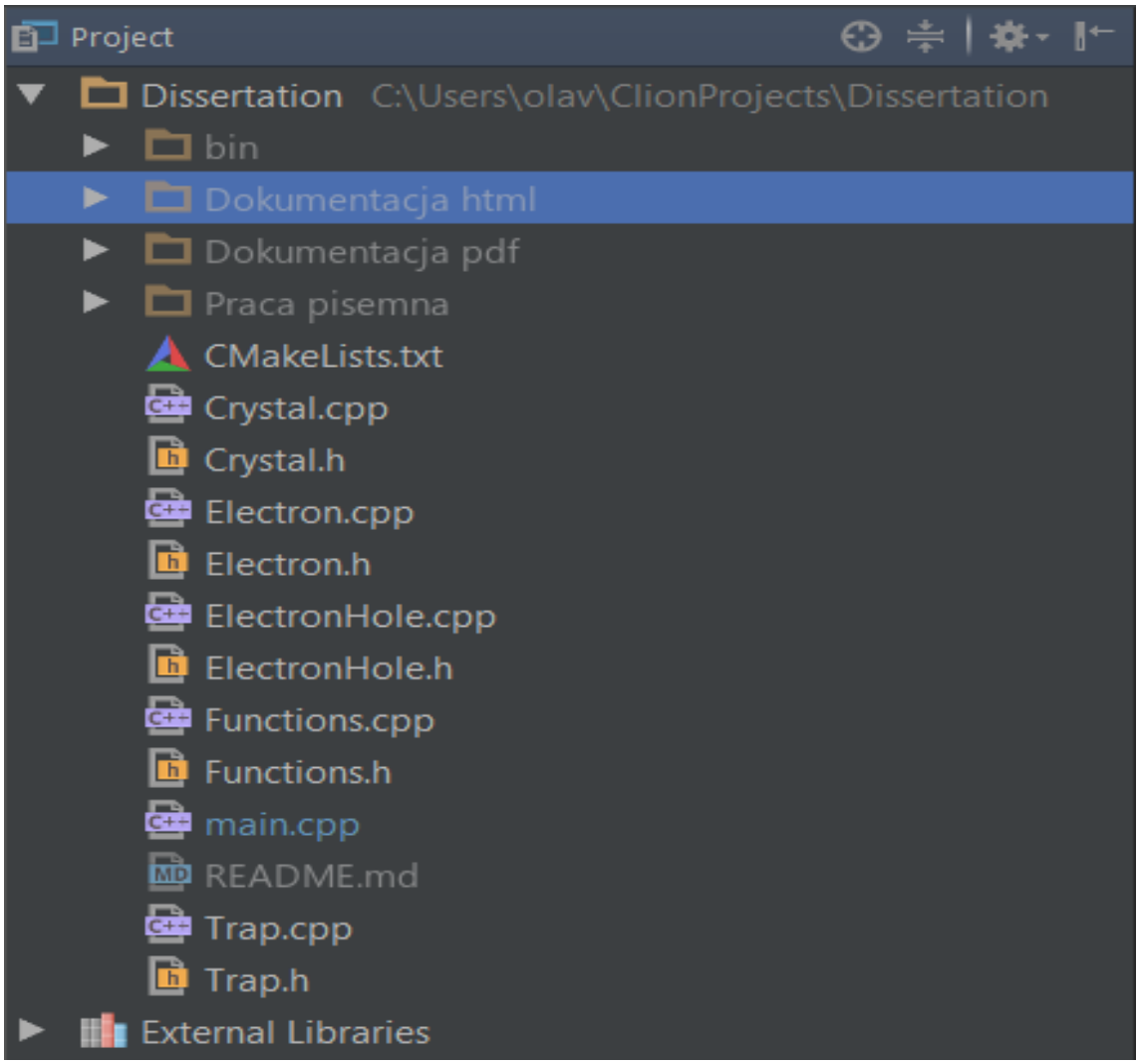
2.5 Valgrind

W czasie tworzenia aplikacji użyto narzędzia **Valgrind**. Służy on do debugowania, wykrywania wycieków pamięci oraz profilowania aplikacji. Dzięki wykorzystaniu tego narzędzia upewniono się, że nie ma żadnych wycieków pamięci, zwiększono szybkość działania programu oraz zmniejszono ilość błędów występujących na początku tworzenia programu.

Rozdział 3

Implementacja

Głównym założeniem projektu było stworzenie go w taki sposób, aby kod jak najtrafniej odzwierciedlał rzeczywistość. W tym celu stworzono klasy obiektów reprezentujące rzeczywiste byty w świecie realnym.



Rysunek 3.1: Struktura projektu.

3.1 Kod

3.1.1 Konwencja nazw

W czasie tworzenia projektu kierowano się standardowym, spójnym stylem nazywania klas, metod i zmiennych.

Nazwy klas zaczynają się z wielkiej litery. Jeśli nazwa zawiera więcej niż jedno słowo to każde z nich zaczyna się wielką literą np. *ElectronHole*.

Nazwy zmiennych oraz metod zaczynają się z małej litery. Jeśli nazwy złożone są z wielu słów, każde kolejne słowo - poza pierwszym zaczyna się z wielkiej litery np. *tunnelEffectProbability(double time, double tau)*.

Argumenty funkcji zaczynają się z małej litery. W przypadku użycia argumentu składającego się z wielu wyrazów oddzielane są one znakiem '_' np. *Crystal(long long int n_el, long long n_holes, double min, double max)*.

3.1.2 Formatowanie i dokumentacja

W projekcie użyto spójnego stylu "wcinania" kodu za pomocą funkcji **Reformat Code** programu CLion (skrót Ctrl + Alt + L). Dzięki temu kod stał się dużo bardziej przejrzysty dla programisty.

Każda z klas została umieszczona w osobnych plikach .cpp oraz .h . W plikach .h znajduje się deklaracja klasy wraz z jej metodami i zmiennymi. W plikach .cpp zostały umieszczone implementacje metod dla odpowiednich klas.

Aby kod był łatwiejszy do zrozumienia oraz dalszego rozwijania wygenerowana została dokumentacja projektu wykonana za pomocą programu **doxygen**. Zawiera ona informacje na temat metod i składowych wszystkich wykorzystywanych klas. Stworzono dwie wersje dokumentacji: *PDF* oraz *HTML* (umieszczona na darmowym hostingu cba.pl).

3.2 Klasy

Każda z klas posiada własne metody w zależności od swojego przeznaczenia.

Klasa *Electron* odzwierciedla cząstkę elementarną - elektron. Zmiana ilości tych cząstek w stanie wzbudzonym tj. znajdujących się w pułapce jest głównym celem wykonania tej symulacji.

ElectronHole jest reprezentacją dziury elektronowej, z którą to elektron po wykorzystaniu zjawiska tunelowego zrekombinuje. Głównymi składowymi tej klasy są: wektor pozycji dziury, wskaźnik na obiekt typu *Trap* oraz energia dziury wyrażona w elektronowolotach.

Klasa *Trap* odpowiada defektom w sieci krystalicznej czyli pułapkom, które mogą przechwycić elektron lub dziurę elektronową. Posiada metodę *isOccupied()*, która jest warunkiem sprawdzanym za każdym razem, gdy symulowany jest efekt tunelowy. Na potrzeby wykonania

tej symulacji założono, że na samym jej początku wszystkie obiekty reprezentujące elektrony oraz dziury elektronowe znajdują się już w pułapkach. Oznacza to, że konstruktor tej klasy ustawia wskaźnik *electron* na podany obiekt typu *Electron*. Dodatkowo w symulacji ustalono, że dany elektron jeśli spełni warunek wystąpienia zjawiska tunelowego - po jego zajściu i rekombinacji z dziurą - nie może przetunelować ponownie. Zostało to podyktowane zmniejszeniem oczekiwanego czasu działania programu.

Klasa *Crystal* reprezentuje rzeczywisty kryształ, który będzie poddany symulacji zaniku sygnału luminescencyjnego. Zawiera on w sobie inne obiekty takie jak:

- Elektrony
- Dziury elektronowe
- Obiekty odpowiadające defektom sieci krystalicznej - tzw. pułapki

Jest to główna klasa zarządzająca całą symulacją. Posiada metody wyliczające niezbędne parametry równania 1.1, a także zapisujące wynik działania programu do pliku w odpowiednich jednostkach.

3.3 Opis działania programu

W celu otrzymania danych niezbędnych do wygenerowania wykresu zależności między ilością elektronów w pułapkach a upływem czasu, uruchomiono stworzony program. Obiekt *Crystal* w swoim konstruktorze generuje podaną ilość dziur elektronowych, pułapek oraz elektronów, a następnie przechowuje je w oddzielnych kontenerach. Dla każdej z tych cząstek wywoływany jest konstruktor (ustawiający współrzędne położenia) z argumentami, które są losowane z podanego wcześniej przedziału (wartości są podane w Angstremach tj. 1 Å = 10^{-10}m. Jednostka ta nie jest jednostką układu SI, lecz jest stosowana w fizyce przy opisywaniu obiektów i zjawisk zachodzących w skali atomowej). Jak podkreślono wcześniej, symulacja zakłada, że na jej starcie każdy elektron znajduje się w pułapce. Powoduje to, że para elektron - pułapka ma identyczne współrzędne położenia, a także obiekt klasy *Trap* przyjmuje wskaźnik na uwięziony w nim elektron.

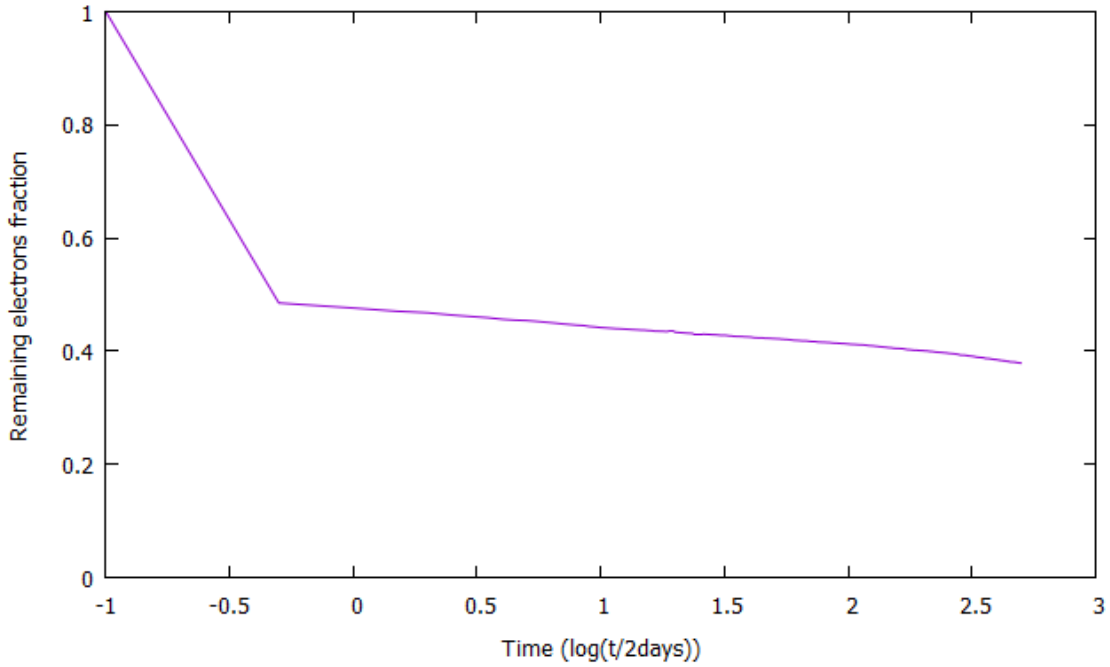
Następnie za pomocą metody *startSimulation(int time)* obiektu klasy *Crystal* rozpoczyna symulację. Symulowany upływ czasu zależy od wartości argumentu *time*, który jest wyrażony w dniach tj. wywołanie *startSimulation(365)* oznacza rozpoczęcie działania symulacji symulującej efekt zaniku sygnału luminescencyjnego w czasie 1 roku.

Podczas wykonywania symulacji każda pułapka elektronowa jest sprawdzana, czy przechowuje w sobie uwięziony elektron. Jeśli warunek jest spełniony, to dla elektronu oraz dziur elektronowych znajdujących się w pułapkach, za pomocą wzoru 1.1 wyliczane jest prawdopodobieństwo zajścia efektu tunelowego. Jeśli prawdopodobieństwo jest mniejsze (ponieważ wzór

1.1 oblicza prawdopodobieństwo, że tunelowanie nie zajdzie) od losowej liczny z zakresu [0,1] to elektron opuszcza swoją pułapkę (wskaźnik obiektu klasy *Trap* na elektron ustawiany jest na wartość *NULL*), a następnie zmienia swoje dotychczasowe położenie na współrzędne, do których przetunelował (współrzędne centrum rekombinacyjnego/dziury). Powtarzane jest to tak długo, aż program zasymuluje podany mu wcześniej czas trwania symulacji. Jak wynika z założeń projektu wspomnianych wcześniej - jeśli elektron przetunelował do centrum rekombinacji przynajmniej raz, nie będzie on brał więcej udziału w obliczaniu prawdopodobieństwa zajścia efektu tunelowego.

Po skończeniu symulacji, program zapisuje do pliku tekstowego otrzymane wyniki w formacie *czas;ilość elektronów w stanie wzbudzenia*, gdzie czas wyrażony jest w $\log(\frac{t}{2dni})$, a ilość elektronów oznacza jaka część z początkowych elektronów nadal znajduje się w pułapkach. Za pomocą skryptu *wykres.plt* w folderze *bin/Release* generowany jest odpowiedni wykres ilustrujący otrzymane dane.

Przykładowo wygenerowany wykres (analiza otrzymanych wykresów znajduje się w dalszej części pracy):



Rysunek 3.2: Przykładowa wizualizacja otrzymanych danych

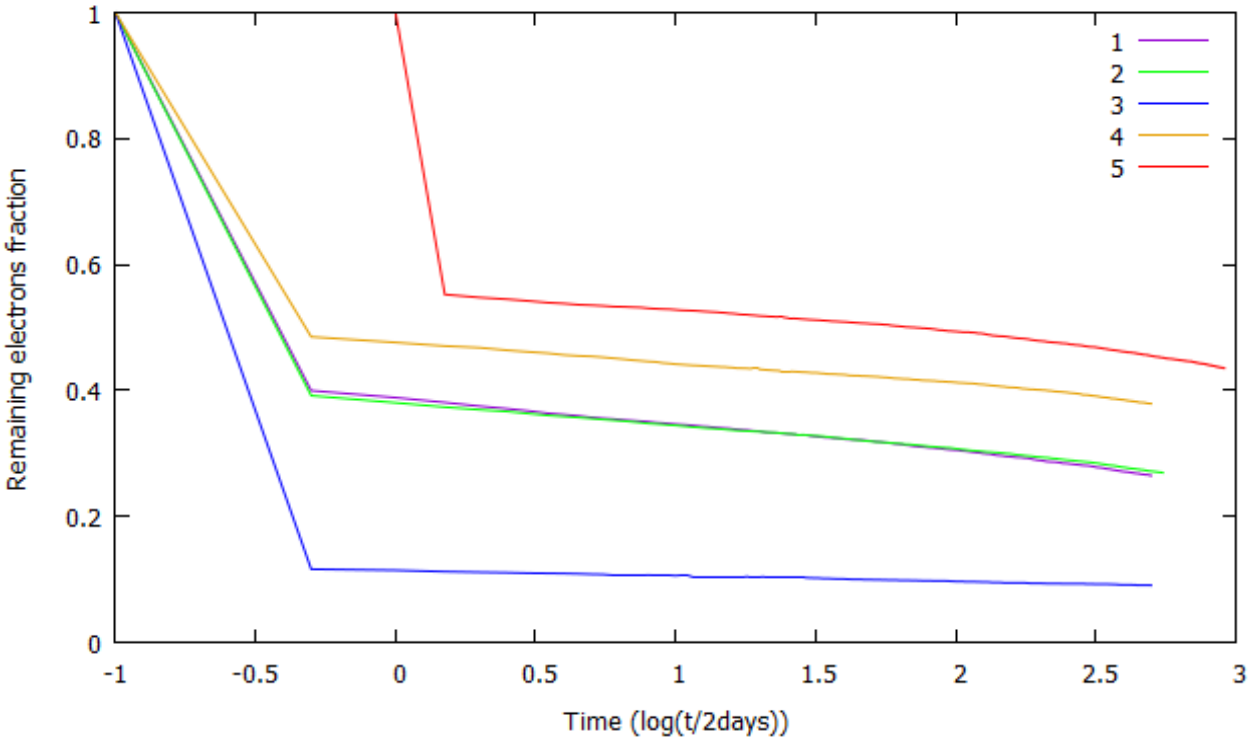
Rozdział 4

Analiza wyników

Program był tworzony, testowany oraz wykonywany na komputerze posiadającym 4 rdzeniowy procesor Intel Core i7-6700HQ oraz pamięć 8 GB (SO-DIMM DDR4, 2133MHz).

4.1 Wygenerowane wykresy

W celu analizy otrzymanych danych, program został uruchomiony pewną ilość razy, za każdym razem z innymi danymi wejściowymi. Otrzymane wykresy przedstawiają zmianę ilości elektronów znajdujących się w pułapkach w czasie trwania symulacji. Umieszczono je na jednym wspólnym wykresie w celu prostszego porównania.



Rysunek 4.1: Porównanie otrzymanych wartości z różnymi parametrami wejściowymi

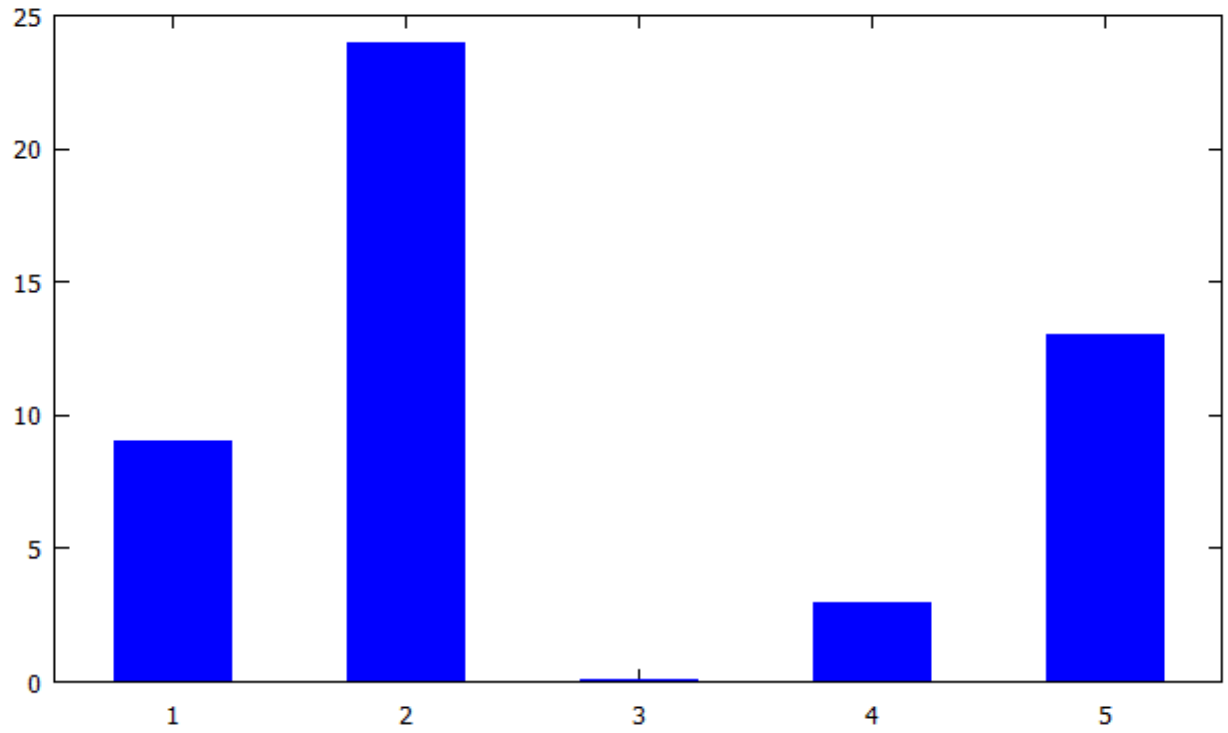
- 1 - Ilość elektronów: 10^4, ilość dziur elektronowych: $2 * 10^4$, zakres, w którym losowane są położenia cząstek: [-1000,1000], symulowany czas: 1000 dni. Czas wykonania: 9 h.
- 2 - Ilość elektronów: $2 * 10^4$, ilość dziur elektronowych: $3 * 10^4$, zakres, w którym losowane są położenia cząstek: [-1100,1100], symulowany czas: 1100 dni. Czas wykonania: 24 h.
- 3 - Ilość elektronów: 10^4, ilość dziur elektronowych: 10^4, zakres, w którym losowane są położenia cząstek: [-400,400], symulowany czas: 1000 dni. Czas wykonania: 5 min.
- 4 - Ilość elektronów: 10^4, ilość dziur elektronowych: 10^4, zakres, w którym losowane są położenia cząstek: [-800,800], symulowany czas: 1000 dni. Czas wykonania: 3 h.
- 5 - Ilość elektronów: 10^4, ilość dziur elektronowych: 10^4, zakres, w którym losowane są położenia cząstek: [-900,900], symulowany czas: 5 lat. Czas wykonania: 13 h. [1]

4.2 Analiza wykresów

Analizując otrzymane wykresy na rysunku 4.1 można zauważyć znaczny spadek ilości elektronów w pułapkach na samym początku wykonywania programu. Jest to spowodowane faktem, że wszystkie współrzędne cząstek są losowane z podanego zakresu. Oznacza to, że na początku symulacji wiele pułapek elektronowych zawierających elektrony znajduje się bardzo blisko centrów rekombinacji. Powołując się na równania 1.1 oraz 1.2 obserwujemy, że odległość między pułapką a centrum rekombinacji znacząco wpływa na prawdopodobieństwo zajścia efektu tunelowego, co skutkuje zaobserwowanym początkowym spadkiem ilości wzbudzonych elektronów. Po wystąpieniu znaczącego spadku można zaobserwować, że ilość elektronów znajdujących się w pułapkach zaczyna maleć z dużo mniejszą częstotliwością. Wykres zaczyna wtedy przypominać funkcję liniową.

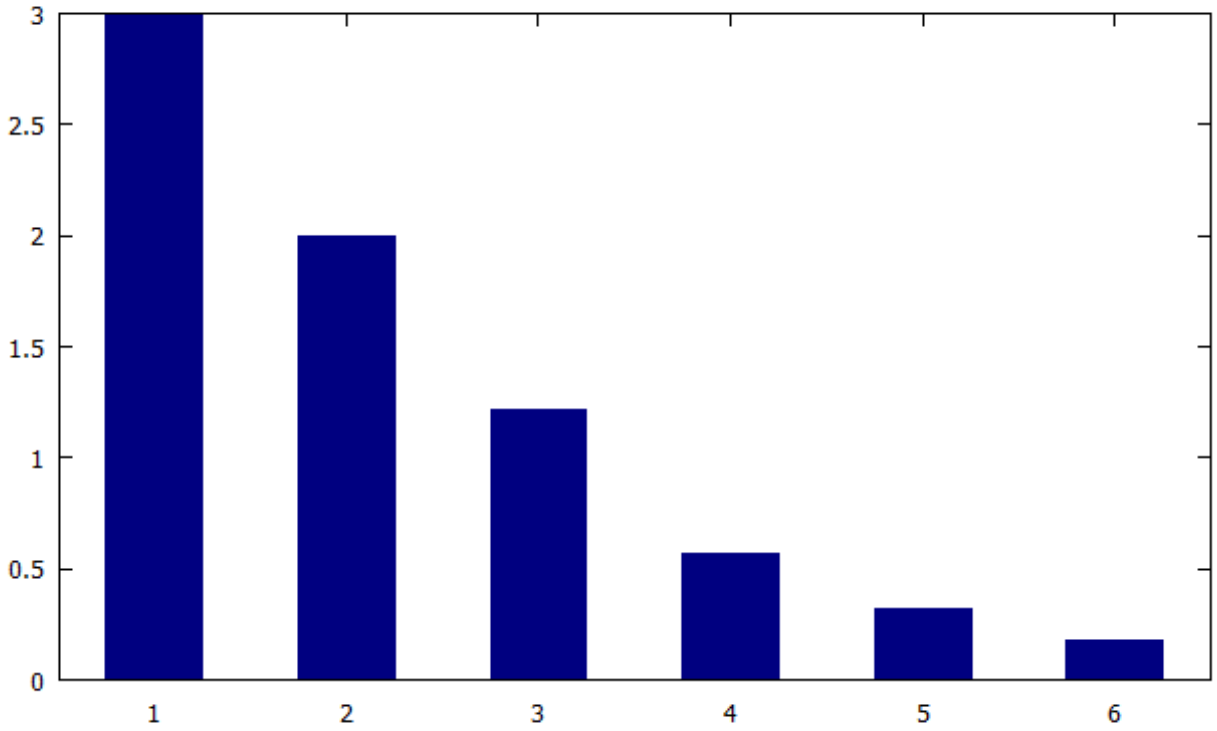
Można przewidywać, że jeśli uruchomiono by symulację obrazującą dłuższy okres np. 50 lat, zanik elektronów w stanie wzbudzonym odbywałby się coraz wolniej. Wynika to z założenia, że elektron może przetunelować tylko jeden raz - oznacza to, że jeśli po pewnym czasie część elektronów przetuneluje do centrum rekombinacji, dla pozostałych elektronów prawdopodobieństwo zajścia tego efektu - które w głównej mierze zależy od odległości pomiędzy pułapką z elektronem, a pułapką z dziurą - będzie zbyt małe.

[1] Wykres zaczyna się od wartości 0 na osi X, ponieważ pomiar był wykonywany począwszy od 2 dnia symulacji



Rysunek 4.2: Porównanie czasu wykonywania programu (w godzinach) z różnymi parametrami wejściowymi

Ważnym parametrem symulacji jest zakres, z którego losowane są wartości położenia dla cząstek. Porównując ze sobą wygenerowane dane na wykresie 3 oraz 4 z rysunku 4.1, gdzie ilość elektronów oraz symulowany czas są takie same, a zakres wartości położeń cząstki znacząco się różni, zaobserwowano, że im większa gęstość ładunków tym efekt tunelowania zachodzi szybciej, co jest zgodne z przewidywaniami wynikającymi ze wzoru 1.2. Dzięki temu oraz wykorzystanej implementacji wraz z założeniem, że po zrekombinowaniu elektron nie może przetunelować po raz kolejny, czas wykonania programu zmniejszył się widocznie z 3 h do ok. 5 min co widać na rysunku 4.2.



Rysunek 4.3: Porównanie czasu wykonywania programu (w godzinach) z różnymi parametrami wejściowymi ze stałą gęstością rozkładu cząstek

Powyższy wykres przedstawia porównanie czasu wykonywania programu z parametrami dobranymi w taki sposób, aby gęstość rozkładu elektronów i dziur była stała. Symulowany czas wyniósł 1000 dni, a parametry były zmniejszane o około 20% przy każdym wykonaniu programu, w stosunku do poprzedniego uruchomienia projektu. Oznaczenie na wykresach:

- 1 - Ilość elektronów: 10^4, ilość dziur elektronowych: 10^4, zakres, w którym losowane są położenia cząstek: [-800,800], symulowany czas: 1000 dni. Czas wykonania: 3 h.
- 2 - Ilość elektronów: 8000, ilość dziur elektronowych: 8000, zakres, w którym losowane są położenia cząstek: [-742,742], symulowany czas: 1000 dni. Czas wykonania: 2 h.
- 3 - Ilość elektronów: 6400, ilość dziur elektronowych: 6400, zakres, w którym losowane są położenia cząstek: [-688,688], symulowany czas: 1000 dni. Czas wykonania: 1 h 13 min.
- 4 - Ilość elektronów: 5120, ilość dziur elektronowych: 5120, zakres, w którym losowane są położenia cząstek: [-592,592], symulowany czas: 1000 dni. Czas wykonania: 37 min.
- 5 - Ilość elektronów: 4096, ilość dziur elektronowych: 4096, zakres, w którym losowane są położenia cząstek: [-549,549], symulowany czas: 1000 dni. Czas wykonania: 19 min.
- 6 - Ilość elektronów: 3276, ilość dziur elektronowych: 3276, zakres, w którym losowane są położenia cząstek: [-509,509], symulowany czas: 1000 dni. Czas wykonania: 11 min.

Jak można zaobserwować przy każdym zmniejszeniu wartości parametrów o około 20% czas wykonania symulacji ulega skróceniu od około 33% przy pierwszym zmniejszeniu ilości cząstek, do 50% przy kolejnych uruchomieniach programu.

Rozdział 5

Dalszy rozwój programu

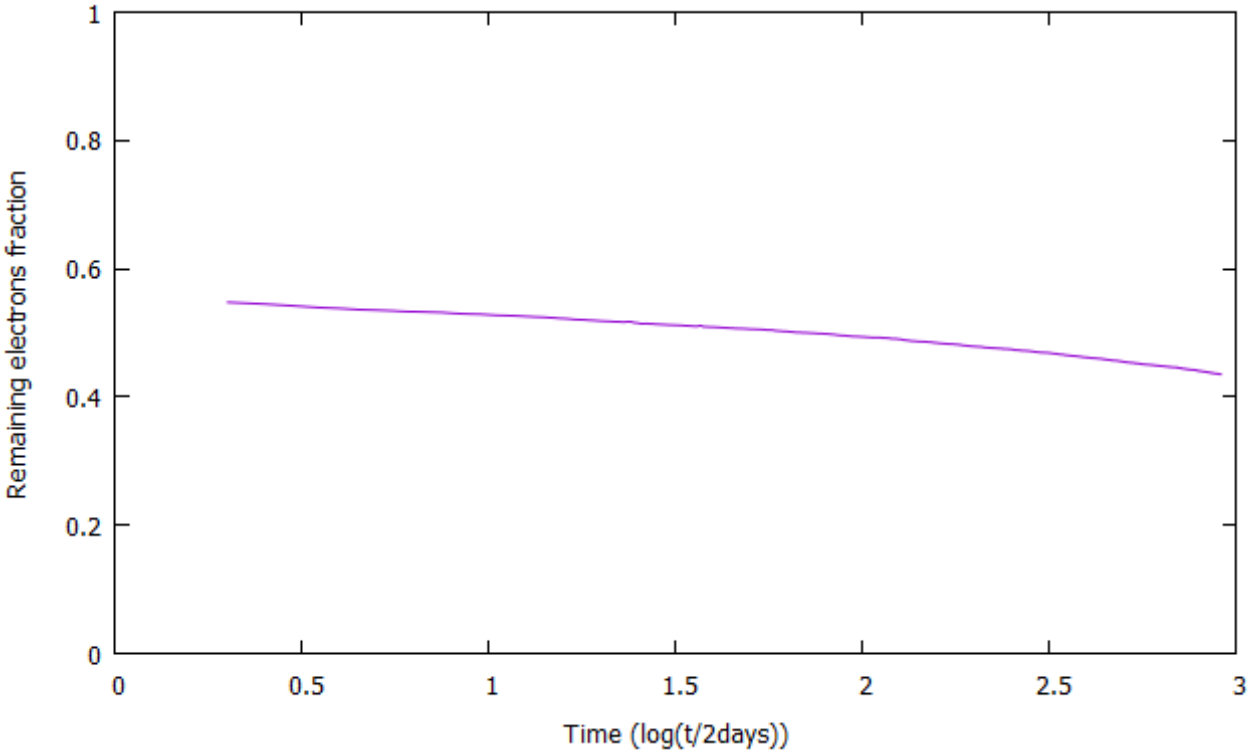
Projekt posiada podstawową funkcjonalność symulującą atermiczny zanik sygnału luminescencyjnego w skaleniach. Może być rozwinięty i ulepszony o dodatkowe funkcje.

Jak zaobserwowano podczas wykonywania symulacji największą trudnością w działaniu programu jest jego złożoność obliczeniowa, a co za tym idzie czas wykonywania programu. Na chwilę obecną podczas trwania symulacji każdy elektron, który może przetunelować do centrum rekombinacji jest porównywany ze wszystkimi dziurami elektronowymi do czasu spełnienia warunku zajścia efektu tunelowego 1.1. Oznacza to, że elektron traktuje każdą dziurę na równi z innymi dziurami. Jak wynika ze wzoru 1.2 na prawdopodobieństwo zajścia efektu tunelowego w znacznym stopniu wpływa odległość między pułapką z elektronem, a centrum rekombinacji. Dlatego też przy dalszym rozwoju programu należy zoptymalizować kod w taki sposób, aby dziury znajdujące się bliżej elektronu miały wyższy priorytet przy wyliczaniu prawdopodobieństwa tunelowania tzn. były rozważane wcześniej niż dziury odległe.

Jednym z pomysłów takiej optymalizacji jest użycie drzewa kd (w tym przypadku k = 3) czyli podział kryształu skalenia na N regiony. Dla każdego elektronu oraz każdej dziury przechowywana byłaby dodatkowa informacja na temat ich przynależności do danego regionu, określana na podstawie współrzędnych cząstki. Gdy dla danego elektronu rozważane byłoby prawdopodobieństwo zajścia efektu tunelowego, do jego wyliczenia w pierwszej kolejności wybierane zostaną dziury uwięzione w tym samym regionie. Jeśli efekt tunelowania nie zaszedł to w zależności od dalszej implementacji program mógłby uznać, że dla tej iteracji elektron nie będzie tunelował lub kontynuowałby sprawdzanie prawdopodobieństwa w sąsiadujących regionach.

Jak zaobserwowano na wykresach w rozdziale 4 na początku działania symulacji otrzymano znaczny spadek ilości wzbudzonych elektronów, a dopiero potem wykres kształtem przypomina funkcję liniową. Jest to spowodowane losowym ustalaniem współrzędnych cząstek, przez co przy starcie programu część elektronów znajduje się bardzo blisko dziur umożliwiając zajście efektu tunelowego. Aby ulepszyć otrzymywany wykres, należałoby zaimplementować funkcję, która będzie monitorowała zapisywanie wyników tak, aby pod uwagę brane były tylko dane po początkowym spadku ilości elektronów wynikającym z losowych współrzędnych startowych.

Przykładowy wykres wyglądałby wtedy następująco:



Inną możliwością rozbudowy programu jest dodanie nowej funkcjonalności. Obecnie kroki czasowe przy sprawdzaniu prawdopodobieństwa zajścia tunelowania są równe. Z funkcji prawdopodobieństwa tunelowania wynika, że dobrym pomysłem byłoby wykorzystanie kroków logarytmicznych - na początku działania symulacji używane są małe przedziały czasowe np. sekundy, następnie minuty, potem godziny, dni, lata, dziesiątki lat itd.

Aby program dostarczał bardziej miarodajne dane, należałoby również umożliwić elektronom wielokrotne tunelowanie. W obecnej implementacji elektron po przetunelowaniu nie jest brany pod uwagę w kolejnej iteracji przy wyliczaniu prawdopodobieństwa zajścia tego efektu. W rzeczywistym krysztale zdarza się jednak, że ten sam elektron może tunelować większą ilość razy.

Istnieje również możliwość łatwej zmiany kodu w celu użycia różnych wzorów na prawdopodobieństwo zajścia efektu tunelowego oraz zmiany sposobu rozkładu ładunków podczas tworzenia obiektu kryształu. Obecnie używany jest rozkład jednostajny ciągły, lecz nic nie stoi na przeszkodzie zmiany go na np. rozkład Gaussa.

Rozdział 6

Podsumowanie

Celem wykonywanej pracy było napisanie programu symulującego atermiczny zanik sygnału luminescencyjnego w skaleniach. Założona podstawowa funkcjonalność została osiągnięta, a zaimplementowany projekt jest prosty do zrozumienia oraz dalszego rozwoju. Dzięki stworzeniu klas odpowiadającym rzeczywistym cząstkom uzyskano program, który - mimo paru ograniczeń wynikających z założeń - dobrze odzwierciedla i symuluje przedstawiane zjawisko w świecie realnym.

Praca została usprawniona dzięki skorzystaniu z profesjonalnego IDE CLion, oraz innym narzędziom użytym w trakcie implementacji. Dzięki nim, kod programu jest przejrzysty, zawiera dokumentację, oraz nie posiada błędów i wycieków pamięci.

Używając programu należy pamiętać, że symuluje on proces długotrwały. W rzeczywistości elektrony znajdujące się w pułapkach mogą być tam przechowywane przez setki lat. Symulacja takiego zjawiska nie jest prosta, gdyż wymaga bardzo dużej ilości obliczeń, co znacznie wydłuża czas potrzebny do otrzymania danych. W celu wykonania symulacji w rozsądnym czasie w prezentowanych wynikach ilość cząstek nie przekroczyła rzędu 10^4 [1], a symulowany czas wynosił najczęściej zaledwie 1000 dni. Pozwoliło to na przeprowadzenie symulacji trwającej maksymalnie około 24 godzin. Czas wykonywania symulacji można zmniejszyć optymalizując kod na przykład używając drzewa *kd*, jak wspomniano w rozdziale 5.

[1] W rzeczywistym krysztale skalenia cząstek może być znacznie więcej

Bibliografia

[1] *Instrukcja do ćwiczenia laboratoryjnego z dozymetrii promieniowania jonizującego dla studentów specjalności Fizyka Medyczna i pokrewnych*, WFiIS AGH, Kraków 1993, s.3 rys. 1

[2] Piotr Psuja, *Właściwosci luminescencyjne i katodoluminescencyjne nanometrycznych kompozytów ITO (In2O3/SnO2) domieszkowanych jonami ziem rzadkich*, Wrocław 2009, s.14 rys. 3.1

[3] D.J. Huntley, *An explanation of the power-law decay of luminescence*, Styczeń 2006, s.1360 równanie 1

[4] `http://en.cppreference.com/w/cpp`

[5] Stephen Prata, *Język C++. Szkoła programowania. Wydanie V*, Helion, 2006

[6] Andrzej Bluszcz, *Datowanie luminescencyjne osadów czwartorzędowych - teoria, ograniczenia, problemy interpretacyjne*, Gliwice 2000