

**Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie**

Wydział Fizyki i Informatyki Stosowanej
KATEDRA INFORMATYKI STOSOWANEJ I FIZYKI KOMPUTEROWEJ



PRACA MAGISTERSKA

ŁUKASZ MADOŃ

**WIZUALIZACJA PROCESÓW DYFUZJI I
PORZĄDKOWANIA W STOPACH
MIĘDZYMETALICZNYCH**

PROMOTOR:
dr Lucjan Pytlik

Kraków, wrzesień 2012

OŚWIADCZENIE AUTORA PRACY

OŚWIADCZAM, ŚWIADOMY ODPOWIEDZIALNOŚCI KARNEJ ZA
POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZĄ PRACĘ DYPLOMOWĄ
WYKONAŁEM OSOBIŚCIE I SAMODZIELNIE, I NIE KORZYSTAŁEM ZE
ŽRÓDEŁ INNYCH NIŻ WYMIESZCZONE W PRACY.

.....

PODPIS

Merytoryczna ocena pracy przez opiekuna:

Merytoryczna ocena pracy przez recenzenta:

Kraków, wrzesień 2012

Tematyka pracy magisterskiej i praktyki dyplomowej Łukasza Madonia, studenta V roku studiów kierunku informatyka stosowana, specjalności informatyka stosowana w nauce i technice

Temat pracy magisterskiej: Wizualizacja procesów dyfuzji i porządkowania w stopach międzymetalicznych.

Opiekun pracy: dr Lucjan Pytlik

Recenzenci pracy: dr hab. inż. Khalid Saeed

Miejsce praktyki dyplomowej: WFiIS AGH, Kraków

Program pracy magisterskiej i praktyki dyplomowej

1. Omówienie realizacji pracy magisterskiej z opiekunem.
2. Zebranie literatury dotyczącej tematu pracy.
 - opracowanie zagadnienia „procesów porządkowania w stopach”
 - przegląd „bibliotek graficznych do wizualizacji modeli 3D”
3. Praktyka dyplomowa:
 - Zebrania wymagań projektowych,
 - Wstępna implementacja,
 - Projekt ostatecznej aplikacji
 - Implementacja i testy.
4. Opracowanie redakcyjne pracy.

Termin oddania w dziekanacie: wrzesień 2012

.....
(podpis kierownika katedry)

.....
(podpis opiekuna)

Spis treści

1. Wprowadzenie	10
1.1. Cele pracy	10
1.2. Zawartość pracy	11
2. Zagadnienia fizyczne.....	12
2.1. Złożone stopy metaliczne	12
2.2. Stop β - Mg ₂ Al ₃	14
2.3. Symulacje dyfuzji i porządkowania w CMA.....	15
3. Przegląd potencjalnych technologii do realizacji projektu	18
3.1. Zdefiniowanie wymagań projektowych.....	18
3.1.1. Model Kaskadowy	19
3.1.2. Model spiralny	21
3.1.3. Model iteracyjny	22
3.1.4. Agile Manifesto	23
3.1.5. The Lean Startup	24
3.2. Specyfikacja programu do wizualizacji CMA.....	25
3.3. Prototypy różnych technologii.....	27
4. Implementacja aplikacji	31
4.1. Struktura aplikacji	31
4.2. Użyte Algorytmy i Struktury Danych	34
4.2.1. Funkcje pomocnicze	35
4.2.2. Klasa Sphere3D	36
4.2.3. Klasa Atom	37
4.2.4. Klasa AtomBuilder	38
4.2.5. MainWindow	39
4.2.6. MouseHelper	40
4.3. Graficzny Interfejs Użytkownika.....	41

4.4. Testy.....	43
5. Podsumowanie.....	48
5.1. Wyniki	48
5.2. Możliwe rozszerzenia.....	54
5.3. Użyte narzędzia i biblioteki.....	54
5.4. Zakończenie.....	54
A. Instrukcja obsługi	56
A.1. Pobieranie aplikacji	56
A.2. Tworzenie modelu i manipulacja.....	56
A.3. Przetwarzanie grupy plików	56

1. Wprowadzenie

W dzisiejszych czasach komputer stał się niezbędnym narzędziem do rozwiązywania problemów współczesnej fizyki. Fizyczne symulacje komputerowe są naturalnym uzupełnieniem fizyki teoretycznej oraz fizyki doświadczalnej. Można w łatwy sposób stworzyć abstrakcyjny model, który analizujemy lub poddajemy symulacji, w warunkach często nieosiągalnych doświadczalnie. Modele i symulacje komputerowe również posiadają swoje wady, takie jak zły zestaw założeń, niewystarczająca dokładność, zbyt krótki czas, dlatego też trudnym wyzwaniem jest optymalne połączenie doświadczenia i symulacji, tak aby uzyskać najlepsze wyniki. Jednym za zagadnień fizycznych intensywnie badanych metodami komputerowymi jest wytłumaczenie struktury i własności złożonych stopów metali - CMA.

CMA (ang. Complex Metallic Alloys) definiuje się jako stop metaliczny, posiadający komórkę elementarną struktury krystalicznej, składającą się z tysięcy atomów. Periodyczność komórki elementarnej jest znacznie większa niż średnia odległość pomiędzy sąsiadami w krysztale, przez co własności fizyczne, a zwłaszcza transport elektronów, różnią się znaczco od typowych kryształów. Istnieje kilka rodzajów CMA. Najintensywniej badane są obecnie stopy oparte na aluminium, takie jak β - Mg₂Al₃[7].

Badanie struktury β - Mg₂Al₃ jest głównym powodem powstania omawianego programu do wizualizacji. Stop ten posiada wiele ciekawych własności, takich jak niska gęstość. Ponad jedna trzecia pozycji atomowych w siatce krystalicznej jest częściowo obsadzona, co może zostać wykorzystane do magazynowania wodoru. lub materiał termoelektryczny. Interesujące są też własności termoelektryczne i wiele innych, które nie zostały jeszcze do końca zbadane[10].

1.1. Cele pracy

Celem pracy jest stworzenie, systemu pozwalającego na wizualizację procesów dyfuzji i porządkowania w stopach międzymetalicznych, takich jak stop β - Mg₂Al₃. Aplikacja będzie miała za zadanie obrazowanie zmian poziomu i typu obsadzenia węzłów sieci krystalicznej. Zjawiska te są na tyle uniwersalne w procesach transformacji, że narzędzie będzie mogło znaleźć zastosowanie przy analizie innych stopów metali. Dodatkowo system będzie w stanie stworzyć klatki animacji, tak aby przedstawiały one ewolucję procesu. Oprogramowanie będzie używane jako

narzędzie wspomagające prace fizyków. W pracy zawarty będzie również opis struktur danych i algorytmów użytych w aplikacji oraz proces jej projektowania.

1.2. Zawartość pracy

W rozdziale drugim opiszę złożone stopy metali, skupiając się głównie na β - Mg_2Al_3 . Następne rozdziały przynoszą informację na temat użytych algorytmów oraz struktur danych. Po staram się również opisać technologie, jakich użyłem w kontekście wymagań projektowych. Pracę zamyka rozdział poświęcony wnioskom z procesu realizacji programu. W pracy znajduje się również dodatek z instrukcją obsługi.

2. Zagadnienia fizyczne

W danym rozdziale przedstawię stan badań doświadczalnych dotyczących własności złożonych stopów metali. Rozpocznę od krótkiej charakterystyki struktur krystalicznych, tworzonych przez metale i ich stopy.

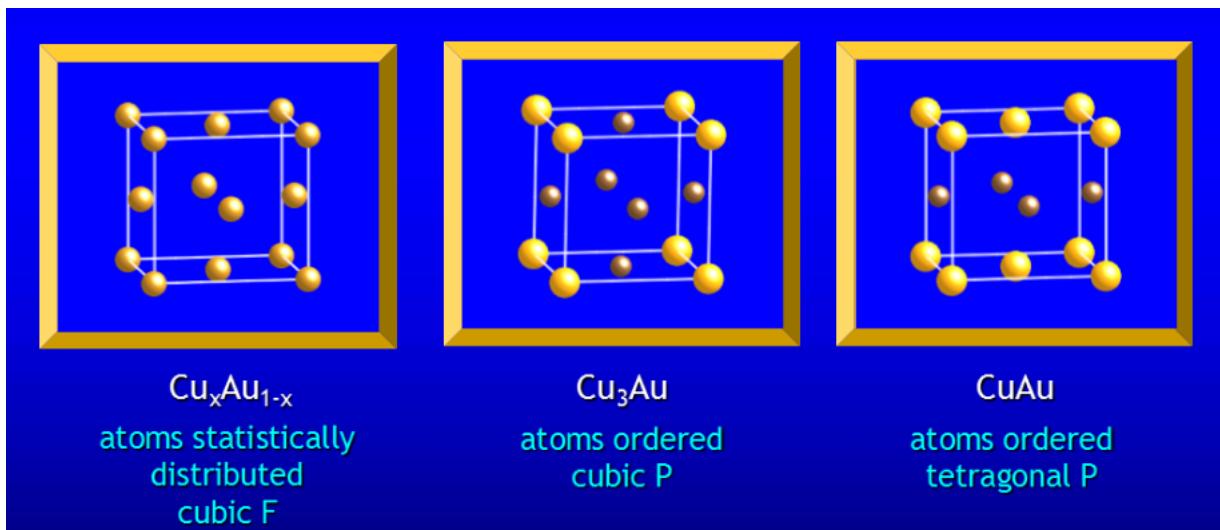
2.1. Złożone stopy metaliczne

Metale i ich stopy w fazie stałej mogą tworzyć bardzo szeroką gamę struktur przestrzennych od dobrze uporządkowanych struktur krystalicznych aż do struktur całkowicie amorficznych. To w jakiej formie obserwujemy fazę stałą, zależy w dużym stopniu od historii obróbki termicznej, czyli od tego, jak wolno dana substancja była schładzana. Możemy obserwować zarówno dobrze uporządkowane fazy krystaliczne przy odpowiednio długim wygrzewaniu, jak i szkła metaliczne przy gwałtownym chłodzeniu cieczy. Pomiędzy tymi dwoma skrajnościami pojawi się cała różnorodność faz częściowo uporządkowanych i o różnym poziomie defektów, takich jak wakancje, podmiany atomów, dyslokacji i błędy ułożenia.

Większość czystych metali krystalizuje w jednej z trzech struktur krystalicznych, charakteryzujących się gęstym upakowaniem i dużą liczbą sąsiadów (l. koordynacyjną). Są to struktury heksagonalne lub struktury regularne objętościowo (BCC) lub ściennie (FCC) centrowane[17]. W tej pracy bardziej będą nas interesować stopy metali, które wykazują znacznie większe bogactwo zachowań.

Stopy metali możemy podzielić ze względu na ich strukturę wewnętrzną. Rozróżniamy:

1. roztwór stały (w pewnym zakresie kompozycji), kiedy stop składa się z dwóch pierwiastków (A i B), które obsadzają zamiennie wspólną sieć krystaliczną (każdy węzeł może być obsadzony przez atom A lub B). Mogą również wystąpić atomy w pozycjach międzywęzłowych, przy większych różnicach promieni atomowych. Po przekroczeniu granicy rozpuszczalności otrzymujemy rozpad na mieszaninę różnych faz. Zachowanie zależy od reguły Hume-Rothery. Przykładem jest stop miedzi i srebra (Cu-Ag).
2. stopy uporządkowane: kiedy po uporządkowaniu poszczególne pozycje w sieci przyporządkowane są danemu typowi atomu A lub B. Sieć krystaliczna pozostaje praktycznie



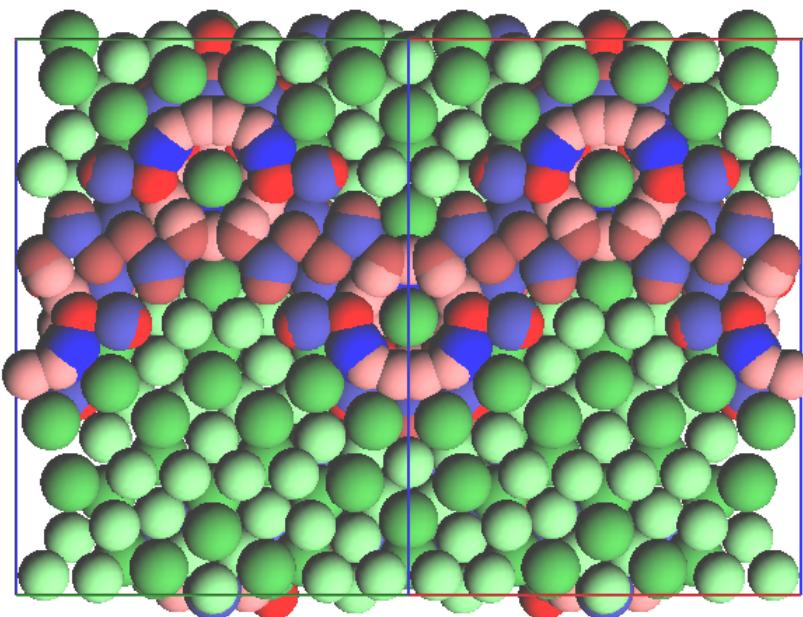
Rysunek 2.1: Układy krystalograficzne dla stopów $\text{Au} - \text{Cu}$, AuCu i Au_3Cu .

taka sama, nową fazę daje tylko uporządkowanie obsadzeń. Przykładem jest stop miedzi i złota(Cu-Au).

3. fazy pośrednie, w których przy określonym zakresie kompozycji (blisko składu idealnego) atomy tworzą fazę ze swoistą budową krystaliczną, nie zawsze będącą w relacji z budową krystaliczną samych składników.
4. związki międzymetaliczne są częstymi przedstawicielami faz z punktu 3, tych ze strukturą bardzo odmienną od struktury samych metali składowych. Posiadają cechy, takie jak wiązania w pełni izotropowe (metal), wysokie liczby koordynacyjne i duża gęstość upakowania. Przykłady: fazy Lavesa, Heuslera, Franka-Kaspera, Zintla
5. złożone stopy międzymetaliczne powstają, kiedy istnieją czynniki zakłócające zbudowanie prostej struktury pośredniej fazy międzymetalicznej. Mogą to być relacje promieni atomowych, odejście od izotropowości wiązań lub pokrewieństwo chemiczne. Efektem może być lokalne zakłócenie budowy prostej struktury i tworzenie klastrów opartych na prostych relacjach geometrycznych i innych liczbach koordynacyjnych, np. fazy Samsona (β - Mg_2Al_3), Bergmana, Taylora. Obecność klastrów powoduje trudności ze wzajemnym wpasowaniem głównej fazy i elementów klastrowych, skutkujące obecnością wakancji, pozycjami międzymetalicznymi itd. Ta klasa ma wiele elementów wspólnych z kwazikryształami. Bardzo wiele stopów międzymetalicznych jest kruchych w niskich temperaturach, jednak po osiągnięciu pewnej temperatury właściwości fizyczne ulegają zmianie, przez co można je zastosować w np. turbinie gazowej samolotu[37].

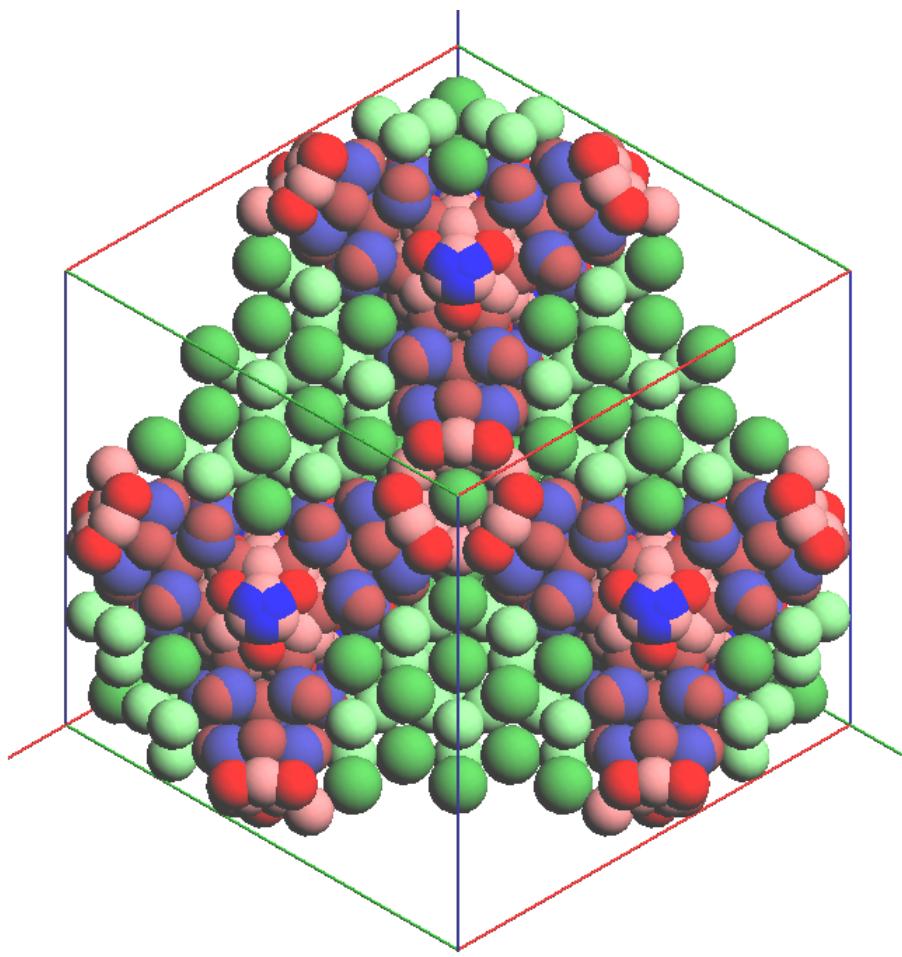
2.2. Stop β - Mg_2Al_3

Faza β - Mg_2Al_3 należy do najbardziej skomplikowanych wśród faz tego stopu. Odkrycia jej dokonał Riederer w 1936 roku na podstawie badań dyfrakcyjnych[31]. Inny naukowiec, Perlitz, odkrył, że stop składa się z 38% magnezu i 62% aluminium (wagowo). Posiada on również kubiczną grupę przestrzenną Fd3m z komórką elementarną o boku równym 2,822nm posiadającą około 1168 atomów[27]. Główną zaletą stopu β - Mg_2Al_3 jest bardzo mała gęstość i duża liczba luk w sieci krystalicznej, co sugeruje możliwość ich wykorzystania jako akumulatorów wodoru[17]. Rysunki 2.2 i 2.3 przedstawiają przekrój komórki elementarnej stopu β - Mg_2Al_3 . Zielonym kolorem oznaczone są atomy, których prawdopodobieństwo obsadzenia wynosi 1 (stabilna siatka). Kolorem niebieskim oznaczona jest pozycja, w której istnieje możliwość obsadzenia atomem aluminium, natomiast kolor czerwony reprezentuje atom magnezu (część klasterowa).



Rysunek 2.2: Przekrój komórki elementarnej β - Mg_2Al_3 . Płaszczyzna prostopadła do kierunku [1,1,0].

W temperaturze poniżej 100°C stop β - Mg_2Al_3 występuje w stanie fazy metastabilnej. W temperaturze pomiędzy 100 a 200°C zachodzi częściowa transformacja do fazy β' , natomiast powyżej 214°C faza β' znika i zaobserwować można tylko fazę β (rysunek 2.4[11]. W komórce elementarnej stopu pozycje atomowe są określanie przez 23 krystalograficznie nierównoważne pozycje Wyckoffa[17]. Generuje to w sumie 1832 możliwe do obsadzenia pozycje atomowe w komórce elementarnej, z których 840 jest obsadzonych z prawdopodobieństwem równym 1 przez atomy magnezu lub aluminium, natomiast prawdopodobieństwo obsadzenia pozostałych

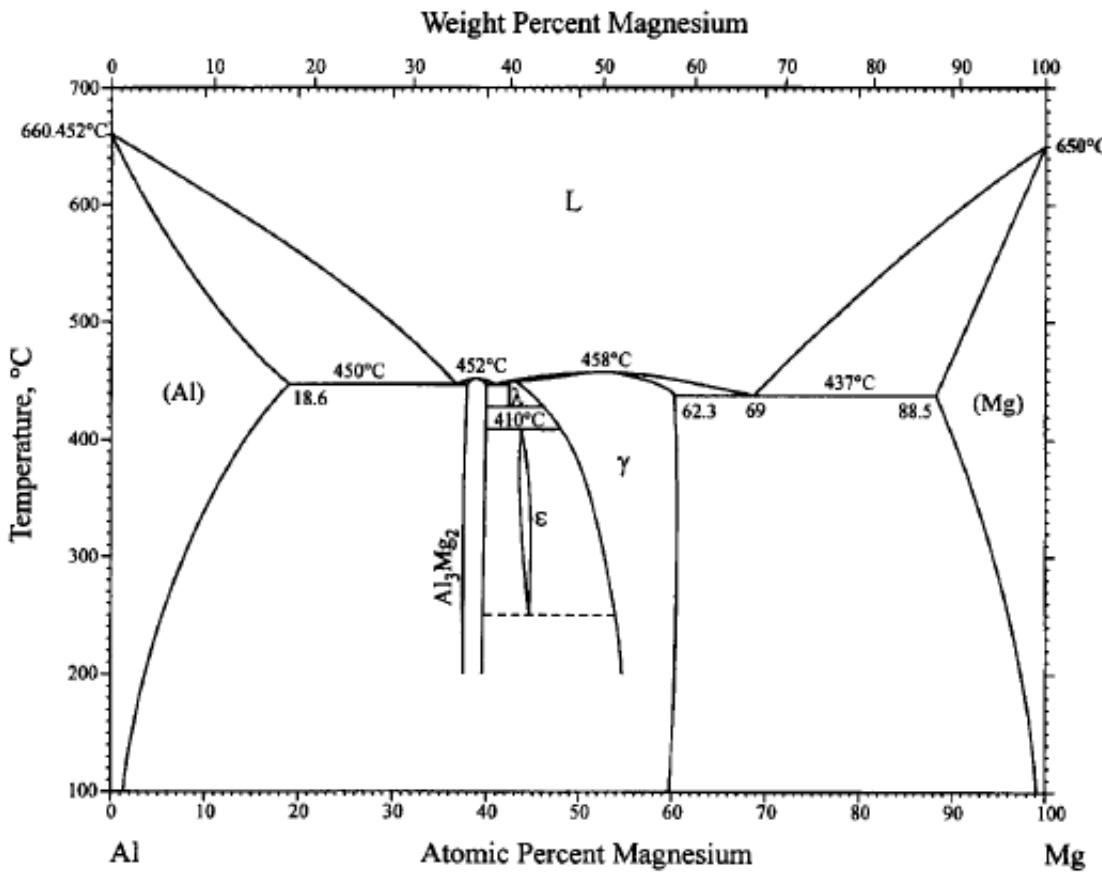


Rysunek 2.3: Przekrój komórki elementarnej β - Mg₂Al₃. Płaszczyzna prostopadła do kierunku [1,1,1].

atomów jest mniejsze od 1. Dla pozycji, których prawdopodobieństwo wynosi 1 istnieje lokalne ograniczenie odległościowe równe 0,25092nm. Pozostałe pozycje nie spełniają tego ograniczenia. Pozycje o prawdopodobieństwie równym 1 układają się w szeregi warstw heksagonalnych, prostopadłych do kierunku [1,1,1] (główna przekątna), tworząc szkielet dla pozycji z dużym udziałem wakancji[17].

2.3. Symulacje dyfuzji i porządkowania w CMA

Pliki wynikowe chmc, wykorzystywane w niniejszej wizualizacji, są plikami wyjściowymi symulacji, realizującej metodę Monte Carlo z zamianą obsadzeń pozycji przez dany typ atomu lub wakancję nazywaną często symulacją gazu sieciowego (Lattice Gas). Metoda wykorzystuje losową zamianę obsadzenia pozycji z częściową akceptacją wyniku takiej zamiany, co prowadzi do znalezienia rozkładu optymalnego w danej temperaturze. Jest to najprostsza z możliwych symulacji, w której nie pozwalamy na zmianę położen przestrzennych pozycji krystalicznych,

Rysunek 2.4: Diagram fazowy stopu β -Mg₂Al₃.

biorąc pod uwagę tylko zmienność obsadzeń tych pozycji (przez atomy Mg, Al lub wakacje). W tym przypadku liczymy energię dla każdego kroku. Jeżeli energia w następnym kroku maleje, to akceptujemy zmianę, jeżeli energia rośnie, to akceptujemy zmianę z prawdopodobieństwem proporcjonalnym do wzoru 2.1[35].

$$P(\text{akc.}) = e^{\frac{-\Delta E}{kT}} \quad (2.1)$$

gdzie ΔE jest przyrostem energii, k stałą Boltzmanna, a T oznacza temperaturę.

Energia jest wyznaczana z potencjałów z odpowiednim uśrednieniem obliczeń kwantowych (potencjał dwuatomowy). Oddziaływanie między elementami układu są opisywane wzorem 2.2.

$$E = \sum_{i,j=1, j>1}^N U(r_{ij}) \quad (2.2)$$

gdzie $r_{ij} = |\vec{r}_i - \vec{r}_j|$ a $U(r_{ij})$ jest energią potencjalną w położeniach r_i oraz r_j [36].

Kolejnym algorytmem używanym w symulacji jest symulowane wyżarzanie. Algorytm ten jest algorytmem heurystycznym, który próbuje znaleźć optymalne rozwiązanie, przeszukując przestrzeń alternatywnych rozwiązań problemu. Algorytm jest bardzo skuteczny, gdy chcemy

ustalić najlepsze ekstremum pośród ekstremów lokalnych. Algorytm przypomina proces wyżarzania w metalurgii - stąd nazwa. Można go z powodzeniem stosować do całej gamy problemów, jak choćby problem komiwojażera. Jest to przykład algorytmu zachłannego[19]. W tym przypadku modyfikujemy nie tylko obsadzenie, ale również pozycje (przesunięcia). Daje to dużo większą elastyczność i szybszą ewolucję systemu. Szukanie najniższej energii odbywa się przez wygrzewanie układu w coraz niższej temperaturze.

Następny algorytm, jaki jest wykorzystywany w symulacji, to dynamika molekularna(MD). Jest to numeryczne rozwiązywanie i komputerowa symulacja przestrzeni fazowej dla zdefiniowanego modelu układu molekuł. Stosuje się metody numeryczne, ponieważ analityczne rozwiązanie byłoby zbyt skomplikowane. Elementarnie stosuje się równania ruchu Newtona, które wzbogaca się o oddziaływanie w celu uzyskania informacji o właściwościach zależnych od czasu. Oddziaływanie między elementami układu są opisywane przez funkcję oraz zespół parametrów dla tej funkcji. Dynamika molekularna znajduje wiele zastosowań, między innymi w biochemii jako narzędzie do poznawania struktury i oddziaływań w białkach, kwasach nukleinowych i innych biomolekułach[22]. Temperatura jest kontrolowana przez całkowitą energię kinetyczną systemu.

Wyniki symulacji, zawarte w zbiorach „chmc”, które stanowią dane wejściowe dla konstruowanej aplikacji, pochodzą z symulacji, która operuje tylko obsadzeniami pozycji krystalicznych przez dwa typy atomów (Mg, Al) lub wakancję. W zbiorach tych zawarte są uśrednione, po pewnych przedziałach czasu, prawdopodobieństwa obsadzeń odpowiednich pozycji krystalicznych oraz ich rozrzutu. Z punktu widzenia użytkownika wizualizacji ważne są przede wszystkim prawdopodobieństwa obsadzeń oraz ich zmiany z czasem i temperaturą. Wizualizacja tych parametrów może dać istotne informacje o ewolucji systemu, np. o tym które pozycje biorą głównie udział w procesach porządkowania, jakimi kanałami zachodzi dyfuzja atomów, w jakich temperaturach zachodzą poszczególne typy procesów porządkowania i do jakich struktur one prowadzą. Uświadamiają one również jaka jest skala czasowa procesów porządkowania i jak długo powinna być prowadzona symulacja, aby uzyskane wyniki były miarodajne dla danych procesów porządkowania. Jak widać z powyższego, aplikacja realizująca wizualizację tych danych może być pomocna na wiele sposobów osobom prowadzącym badania nad tym procesami.

3. Przegląd potencjalnych technologii do realizacji projektu

W rozdziale opiszę technologie użyte do stworzenia prototypów i finalnej aplikacji.

3.1. Zdefiniowanie wymagań projektowych

Jednym z najważniejszych zadań podczas realizacji projektu jest zebranie i analiza wymagań projektowych. Wymagania projektowe to zbiór potrzeb danego produktu lub usługi, a także sposób ich działania. W inżynierii oprogramowania zbiór wymagań jest wykorzystywany w fazie projektowania nowego produktu. Wymagania pokazują, jakie elementy i funkcje są niezbędne w konkretnym przypadku. Istnieje bardzo dużo modeli opisujących ten proces. Większość z nich jest standaryzowana przez organizacje ISO oraz IEC pod numerem 12207 Systems and software engineering[15].

Przed zebraniem wymagań często wykonuje się jeszcze jeden etap - studium wykonalności, które składa się z analizy rynku, analizy ekonomicznej, analizy strategicznej oraz analizy technicznej. W zależności od rozmiaru projektu samo zbieranie wymagań, może zostać podzielone na etapy. Jest to bardzo częsta praktyka w projektach rządowych[38]. Fazę wymagań dzielimy na:

1. gromadzenie wymagań
2. analizowanie
3. specyfikowanie
4. zatwierdzenie

Dobre wymagania projektowe charakteryzuje[38]:

1. Aktualność - wymaganie powinno pozostać aktualne z upływem czasu.
2. Jednoznaczność - wymaganie powinno być jasno sformułowane, bez subiektywnych opinii. Żargon techniczny i akronimy powinny być zdefiniowane w dokumencie.

3. Kompletność - wymaganie powinno być zdefiniowane w jednym miejscu z kompletem niezbędnych informacji.
4. Obowiązkowość - każde niezbędny wymaganie musi zostać zdefiniowane. Brak wymagania oznacza wykluczenie z projektu.
5. Poprawność - wymaganie wypełnia wszystkie lub część potrzeb biznesowych, które są jasno określone.
6. Spójność - wymaganie odnosi się do jednej i tylko jednej sprawy.
7. Wykonalność - wymaganie musi być wykonalne w ramach ograniczeń projektowych.
8. Weryfikowalność - wymaganie powinno być weryfikowalne empirycznie lub poprzez analizę.
9. Zgodność - wymagania nie powinny być sprzeczne ze sobą.

Istnieje wiele różnych cech, które charakteryzują dobre wymagania, ale są one specyficzne dla danej domeny technologicznej. Istnieje jeszcze jeden istotny podział wymagań. Podział na trzy kategorie:

1. Wymagania funkcjonalne - definiują, co ma realizować system np. system powinien generować raporty.
2. Wymagania niefunkcjonalne - są to wymagania jakościowe. Opisują bezpieczeństwo, wydajność itp.
3. Wymagania ograniczeń - określają granice rozwiązania.

Wszystkie wymagania powinny być weryfikowalne, poza wymaganiami ograniczeń typu „system nie powinien”.

3.1.1. Model Kaskadowy

Pierwsze wzmianki o wymaganiach projektowych pochodzą z lat sześćdziesiątych ubiegłego wieku.[34] IBM był pionierem badań dotyczących procesu tworzenia oprogramowania. W 1956 roku jeden z pracowników IBMu, Herbert D. Benington, podczas konferencji na temat zaawansowanych metod tworzenia oprogramowania, opisał model kaskadowy (inaczej zwany wodospadowym)[26].

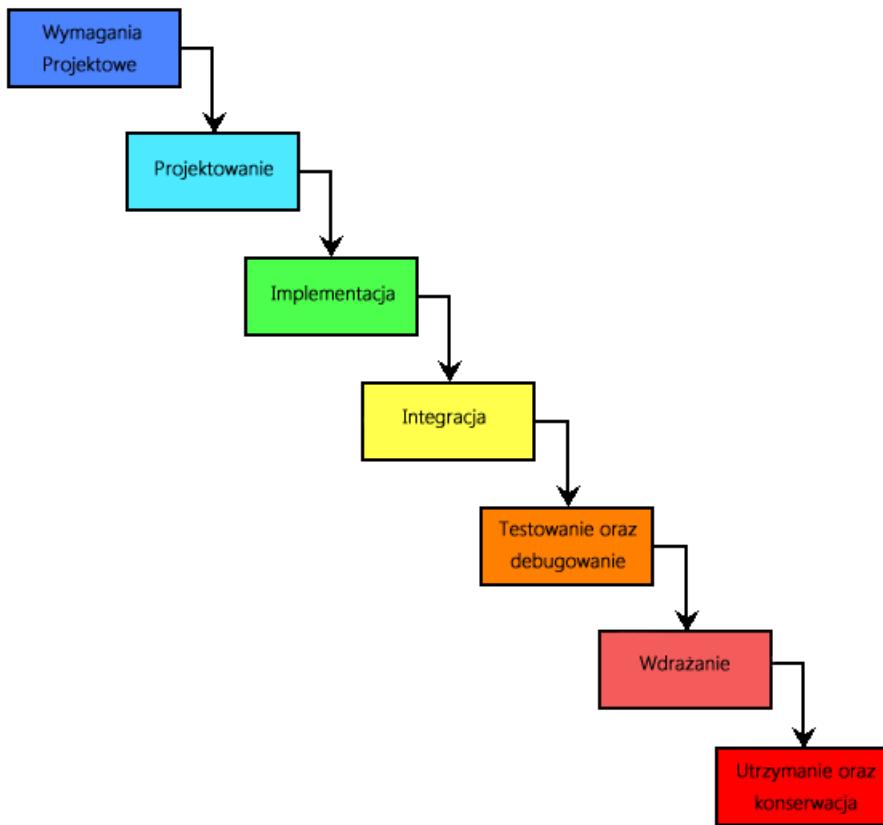
Model kaskadowy podzielony jest na 7 chronologicznych części:[30]

1. Tworzenie wymagań projektowych

2. Projektowanie
3. Implementacja
4. Integracja
5. Testowanie i debugowanie
6. Wdrażanie
7. Utrzymanie i konserwacja

Model kaskadowy jest sekwencyjny. Główną jego cechą jest podział na niezależne części, z których każda wykonywana jest po zakończeniu poprzedzającej. Zaletą takiego podejścia jest wykrywanie błędów we wczesnych etapach. Błąd wykryty podczas projektowania stanowi dużo mniejsze wyzwanie i jest łatwiejszy w naprawie, niż defekt, którego istnienie spostrzeżono dopiero w fazie wdrażania. Jednak takie rozwiązanie ma również swoje wady. Najistotniejszym problemem jest brak możliwości szybkiej zmiany na późnym etapie projektu. W realnych warunkach wymagania zmieniają się bardzo często, co dla nietrywialnych projektów może oznaczać ciągłe powroty do fazy projektowej. Bardziej fatalne w skutkach może okazać się złe określenie wymagań projektowych. Przykładowo: podczas wdrażania klient stwierdzi, że oprogramowanie nie jest dostatecznie szybkie, co może być skutkiem ograniczeń technologicznych. W takim wypadku projekt jest zamkany. Kolejnym problemem jest słabe wykorzystanie zasobów ludzkich - testerzy muszą czekać na zakończenie pracy przez programistów. W 2001 roku przeprowadzono badanie 1027 projektów IT w Wielkiej Brytanii. Okazało się, że techniki zarządzania narzucające metodologię kaskadową były jednym z największych czynników, które wpłynęły na porażkę projektu [21]. Problemy te okazały się na tyle poważne, że dzisiaj nikt już nie korzysta z modelu kaskadowego. W genialny sposób Edward V. Berard podsumował kompleksowe specyfikacje modelu kaskadowego.

Walking on water and developing software from a specification are easy if both are frozen.



Rysunek 3.1: Schemat modelu kaskadowego.

3.1.2. Model spiralny

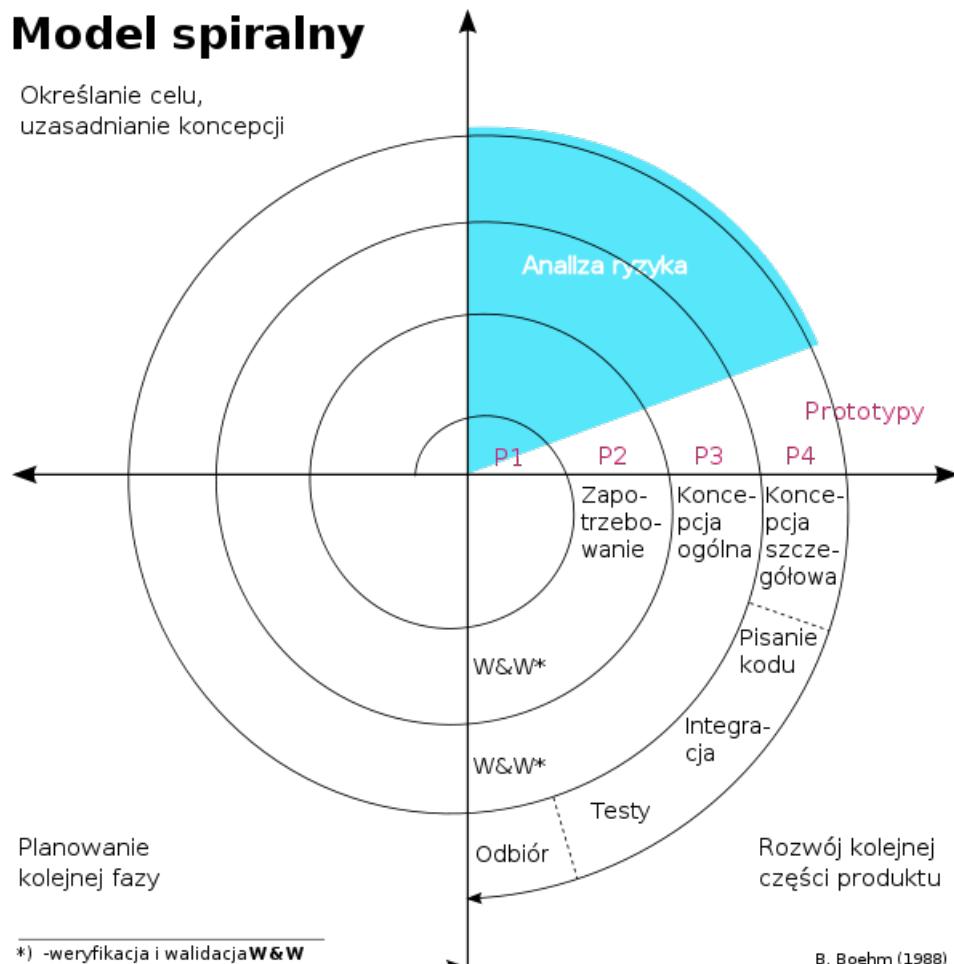
Model spiralny jest to rodzaj procesu tworzenia oprogramowania. Kolejne etapy są podzielone na mniejsze części. Każda z tych części jest cyklicznie powtarzana w pętli, przez co plan jest przedstawiany w postaci spirali. Model spiralny, podobnie jak kaskadowy, posiada ścisłą kontrolę nad projektem. W początkowej fazie łączymy projektowanie z tworzeniem prototypów [3].

Pętle spirali są podzielone na sektory:

1. Definicja celów
2. Rozpoznanie i redukcja zagrożeń
3. Implementacja i zatwierdzanie
4. Ocena i planowanie

Wyraźną zaletą modelu spiralnego jest szczegółowa analiza zagrożeń. Model spiralny nie precyzuje, w jaki sposób mają być realizowane poszczególne pętle np. można użyć modelu

kaskadowego. Podobnie jak w modelu kaskadowym, usuwanie błędów w końcowych etapach jest bardzo kosztowne.

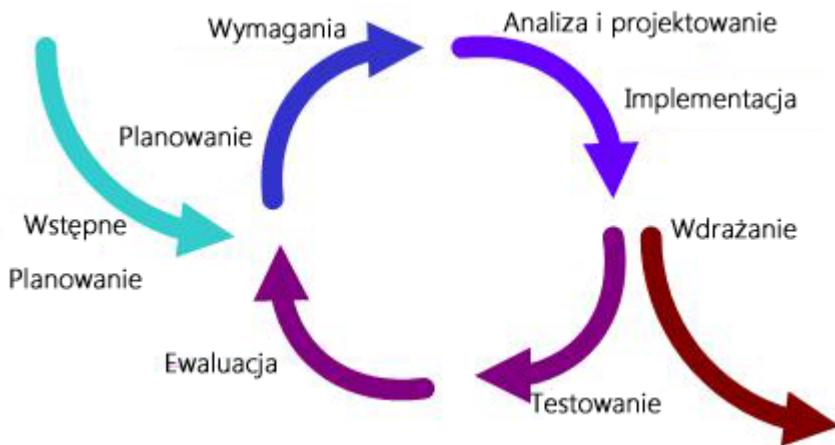


Rysunek 3.2: Schemat modelu spiralnego. Źródło:[3].

3.1.3. Model iteracyjny

W odpowiedzi na wady modelu kaskadowego powstały modele iteracyjne. Pierwszy system wykonany w oparciu o model iteracyjny datuje się na rok 1960. Powstał on w ramach projektu Mercury, który był sponsorowany przez NASA[6]. Podstawowym pomysłem, który definiuje model iteracyjny, jest budowanie systemu poprzez powtarzalne cykle (iteracje). Podejście to pozwala deweloperom wyciągnąć wnioski z dotychczasowej pracy, zwiększa elastyczność wymagań w późniejszym okresie oraz pozwala lepiej wykorzystać zasoby firmy. Model iteracyjny jest często stosowany w wypadku, gdy pełna funkcjonalność systemu nie jest wymagana. Klient może otrzymać częściowo działający system, aby lepiej ocenić jego funkcjonalność.

Oczywiście model iteracyjny posiada również wady, takie jak dodatkowy koszt każdego cyklu czy trudność w podziale na iteracje. Pomimo tego, wszystkie współczesne metody tworzenia oprogramowania bazują na idei iteracji.



Rysunek 3.3: Schemat modelu iteracyjnego.

Przykładem modelu iteracyjnego jest RUP (Rational Unified Process)[1], który składa się z sześciu praktyk:

1. Buduj iteracyjnie z ryzykiem jako najważniejszym czynnikiem
2. Zarządzaj wymaganiami
3. Wprowadź architekturę opartą na komponentach
4. Wprowadź modele wizualne oprogramowania
5. Ciągle sprawdzaj jakość
6. Kontroluj zmiany

Pod koniec lat dziewięćdziesiątych RUP dominował jako model budowania systemów, zwłaszcza w środowiskach korporacyjnych i rządowych. Wadą tego rozwiązania jest spora ilość dokumentacji - często wymagana np. przez regulatora - która wydłuża i usztywnia proces tworzenia systemu.

3.1.4. Agile Manifesto

Jako reakcja na poziom komplikacji i koszty metod, takich jak kaskadowa, powstało wiele tzw. „lekkich” metod. Oto niektóre z nich:[20]

1. Scrum (1995)
2. XP (Extreme Programming) (1996)
3. Crystal Clear

4. Feature Driven Development
5. Dynamic Systems Development Method (1995)

W 2001 roku fundacja non-profit Agile Alliance opublikowała „Agile Manifesto”. Manifest ten zawiera najważniejsze zasady nowo powstałych metodologii. Składa się on z 4 podpunktów:[5]

1. Ludzie i interakcje ponad procesy i narzędzia
2. Działające oprogramowanie ponad wyczerpującą dokumentację
3. Współpraca klienta ponad negocjację umowy
4. Reakcja na zmiany ponad wykonanie planu

Metodologie Agile posiadają bardzo ścisły opis procesu, dlatego zarządzanie projektem wymaga dyscypliny. Inspekcje wymagań są bardzo częste, przez co szybko tworzone jest oprogramowanie wysokiej jakości. Agile kierowany jest głównie do małych i średnich zespołów, dlatego dokumentacja jest ograniczona do minimum. Zespoły programistów Agile są zazwyczaj wielofunkcyjne i samozarządzalne, z bardzo płaską hierarchią. Bardzo duży nacisk kładzie się na komunikację bezpośrednią, dlatego bardzo popularne są codzienne kilkunastominutowe spotkania.

Metody Agile dzielą zadania na małe procesy iteracyjne, które nie zawierają planowania długoterminowego. Iteracje są krótkie. Trwają zazwyczaj od 1 do 4 tygodni. Każda iteracja zawiera pełen proces: planowanie, analiza wymagań, projektowanie, implementacja i testowanie. Na końcu działający produkt jest przedstawiany klientom. Minimalizuje to ryzyko drastycznych zmian w wymaganiach projektu. Bardzo często, aby zminimalizować komunikację, przedstawiciel klientów jest członkiem zespołu.[5].

3.1.5. The Lean Startup

Bardzo ciekawe podejście opisuje Eric Ries w swojej książce „The Lean Startup”. Opisuje on sytuacje, w których często nie można stworzyć specyfikacji, ponieważ klient nie wie dokładnie, jakiego produktu oczekuje lub jest to produkt innowacyjny. Filozofia „The Lean Startup” opiera się na procesie produkcji Lean Manufacturing. Jest to proces opracowany w japońskich fabrykach samochodów, który minimalizuje straty poprzez redukcję kosztów, które nie tworzą wartości dla klienta. W szczególności system koncentruje się na odpowiednim umieszczeniu małych zapasów niezbędnych materiałów(zwanych Kanban) na całej linii produkcyjnej, zamiast przechowywania zapasów w magazynie centralnym. Zmniejsza to straty i zwiększa produktywność, z drugiej strony trudniejsze staje się zarządzanie zasobami(główne surowcami i półprodukami) [32].

Z filozofią Lean Startup wiąże się pięć kluczowych pojęć:

1. Minimum viable product - jest to produkt, który pozwala nam na zebranie jak największej liczby potwierdzonych faktów o klientach, przy jak najmniejszym koszcie. Pozwala rozpoczęć proces badania potrzeb klientów jak najwcześniej.
2. Continuous deployment - jest to proces, w którym każdy nowy kod jest natychmiast wdrażany.
3. Split testing (inaczej zwany A/B test) - jest to rodzaj testu, w którym tej samej grupie klientów pokazujemy dwie lub więcej wersji danego produktu. Dzięki temu możemy wybrać wersję, która odpowiada klientom bardziej oraz określić kierunek rozwoju.
4. Vanity metrics - jest to rodzaj metryki, w jaki sposób firmy próbują określić wzrost biznesu, tak aby był on jak najlepszy. Często daje to fałszywy obraz.
5. Pivot - jest to kontrolowana, gruntowna zmiana produktu lub strategii.

Filozofia Lean Startup jest nie tylko z powodzeniem stosowana do produktów IT - korzystają z niej również inne branże.

3.2. Specyfikacja programu do wizualizacji CMA

Gdy zaczynałem budować program do wizualizacji stopów międzymetalicznych, postanowiłem zebrać wymagania według danego planu:

1. Wymagania powinny być podzielone na podpunkty
2. Każdy podpunkt powinien zawierać, co jest wymagane - bez określenia, jak to zostanie zrobione
3. Każdy podpunkt powinien być „atomowy”. Na przykład: „Program powinien wyświetlić 1000 obiektów oraz tworzyć animacje 1000 obiektów” należy rozbić na 2 podpunkty: „Program powinien wyświetlić 1000 obiektów” oraz „Program powinien tworzyć animacje 1000 obiektów”
4. Powinien być jasno określony cel, wymagania projektowe oraz specyfikacja funkcjonalna
5. Należy określić wymagania bezpieczeństwa
6. Wymagania niefunkcjonalne powinny określać wydajność i dostępność
7. Należy określić, co nie jest wymagane, tak aby odróżnić elementy przypadkowo pominięte

8. Należy określić sposób komunikacji z innymi komponentami lub systemami

Po kilku drobnych zmianach ostateczna wersja specyfikacji składała się z 13 podpunktów dla wymagań projektowych i 11 podpunktów specyfikacji funkcjonalnej:

Cel:

Celem projektu jest stworzenie aplikacji do wizualizacji procesów dyfuzji i porządkowania w stopach międzymetalicznych. Aplikacja powinna korzystać z graficznego interfejsu, tak aby narzędzie do analizy było wygodne w użyciu.

Wymagania projektowe:

1. Program działa pod systemem Windows (zalecany jest Windows 7 lub wyższy, możliwe jest wspieranie innych systemów)
2. Program działa pod kontrolą środowiska .Net
3. Program powinien obsługiwać kilka tysięcy modeli 3D jednocześnie, działając płynnie dla operacji, takich jak obrót
4. Program powinien obsługiwać różne rozdzielcości ekranów
5. Program powinien informować użytkownika o postępie zadań trwających więcej niż 5 sekund
6. Program powinien obsługiwać pliki w formacie .chmc
7. Parametry modeli takie jak kolor, przezroczystość muszą w sposób płynny i adekwatny odwzorowywać się na odpowiednie wartości wyników symulacji (prawdopodobieństwa, typy obsadzeń)
8. Program powinien tworzyć jednolite wizualizacje modeli dla całej grupy plików (np. z różnych temperatur)
9. Dla każdej klatki animacji muszą zostać uwzględnione parametry początkowe ustawione przez użytkownika
10. Program powinien zużywać mniej niż 1GB ramu dla liczby obiektów poniżej 3000
11. Dostępność programu jest wyłącznie zależna od komputera, na którym aplikacja się znajduje
12. Program nie powinien łączyć się z usługami internetowymi lub sieciowymi
13. Program nie powinien łączyć się z innymi komponentami

Specyfikacja funkcjonalna:

1. Użytkownik może otworzyć plik, aby wczytać model
2. Użytkownik ma możliwość filtrowania modeli poprzez ustawienie maski
3. Użytkownik może przetwarzanie grupę plików do stworzenia animacji
4. Użytkownik może obracać modelem 3D
5. Użytkownik może powiększyć model
6. Użytkownik może precyzyjnie wybrać skale modelu
7. Użytkownik może wrócić do parametrów początkowych
8. Użytkownik może zmienić parametry kamery
9. Użytkownik może skalować modele
10. Użytkownik może zapisać zrzut ekranu, na którym znajduje się model
11. Użytkownik ma możliwość filtrowania elementów/składników modeli poprzez odpowiednie ustawianie maski
12. Użytkownik może zmienić wartość kanału alfa modeli

Zaletą tak zwięzkiej specyfikacji jest łatwość ewentualnych zmian oraz krótki czas nauki dla dewelopera.

3.3. Prototypy różnych technologii

Po zebraniu wszystkich wymagań projektowych mogłem przystąpić do budowy kilku prototypów różnych technologii, by sprawdzić, która najlepiej spełnia wymagania. Prototypowanie jest o tyle istotne, że daje nam empiryczny dowód możliwości danej technologii. Jako potencjalnych kandydatów wytypowałem cztery technologie, które posiadają wsparcie dla grafiki 3D:

1. OpenGL
2. XNA
3. WPF
4. DirectX

OpenGL (ang. Open Graphics Library) - jest otwartym standardem API, służącym do generowania grafiki. Biblioteka umożliwia prace na najniższym poziomie, w związku z tym bardzo często korzysta się z rozszerzeń, silników lub frameworków. Filozofia działania OpenGL opiera się na maszynie stanowej [24]. Biblioteka ma dwa główne cele. Stworzyć jednolity interfejs niezależny od sprzętu oraz wpierać standard w pełni, nawet w przypadku braku funkcjonalności w sprzęcie (emulacja oprogramowaniem). OpenGL jest bezpośredniem konkurentem DirectX.

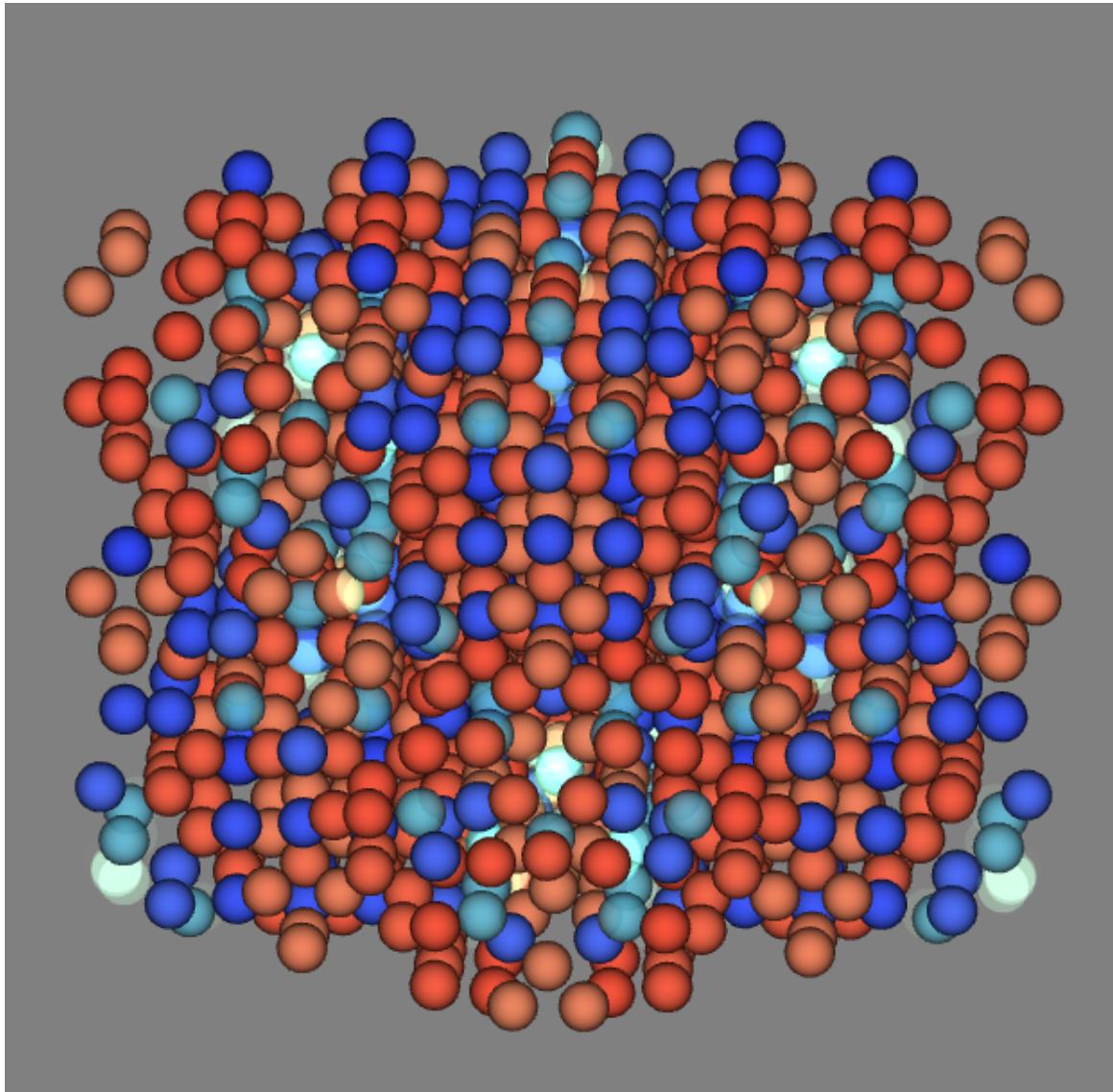
XNA - jest to framework do tworzenia gier w ramach platformy .Net. Początkowo powstał on dla konsoli Xbox, ale został rozszerzony na cały ekosystem Microsoftu. Działa on na relativnie niskim poziomie. Posiada wiele cech wspólnych z technologią DirectX, takich jak wsparcie dla grafiki 3D, jednak kod jest w pełni zarządzany. Dostępna jest również open sourcedowa implementacja Mono.XNA, która działa na innych systemach, takich jak Mac OS X czy Linux. Pomimo iż XNA powstało jako framework do gier, z powodzeniem jest stosowany do innych zadań. Przykładowo XNA Math służy do obliczeń matematycznych na jednostce graficznej[14].

WPF (ang. Windows Presentation Foundation) - jest częścią platformy .Net, służącą do tworzenia aplikacji graficznych. Do definicji obiektów używany jest XAML (pochodna XMLa), który zwiększa możliwości budowania bogatych interfejsów bez ingerencji programisty. Grafik, tworząc interfejs graficzny, tworzył tylko szablon dla programisty, który implementował wszystkie elementy. Korzystając z XAML i narzędzi, takich jak Blend, grafik jest w stanie stworzyć w pełni funkcjonalne GUI bez pomocy programisty. Pośrednio korzysta z DirectX, wyłącznie poprzez interfejs z kodu zarządzanego.[23].

DirectX jest kolekcją API do tworzenia grafiki 2D i 3D. Głównie wykorzystywana w grach i aplikacjach multimedialnych. DirectX jest technologią Microsoftu, która wspiera również efekty dźwiękowe do gier, obsługę urządzeń peryferyjnych itp. DirectX, poza wsparciem dla multimedialnych, oferuje również DirectCompute. Technologia umożliwia wykorzystanie DirectX do obsługi obliczeń GPGPU. Aby korzystać bezpośrednio z DirectX, należy aplikację napisać w języku C lub C++[16].

OpenGL i DirectX z pewnością poradziły sobie z programem do wizualizacji w sensie wydajnościowym. Problemem staje się tworzenie przycisków, suwaków i innych kontrolek. Obie technologie pracują na bardzo niskim poziomie, dlatego wymagają sporych nakładów pracy. W przypadku XNA i WPF nie miałem doświadczenia, które potwierdziły spełnienie wymagań projektowych, dlatego stworzyłem 2 prototypy.

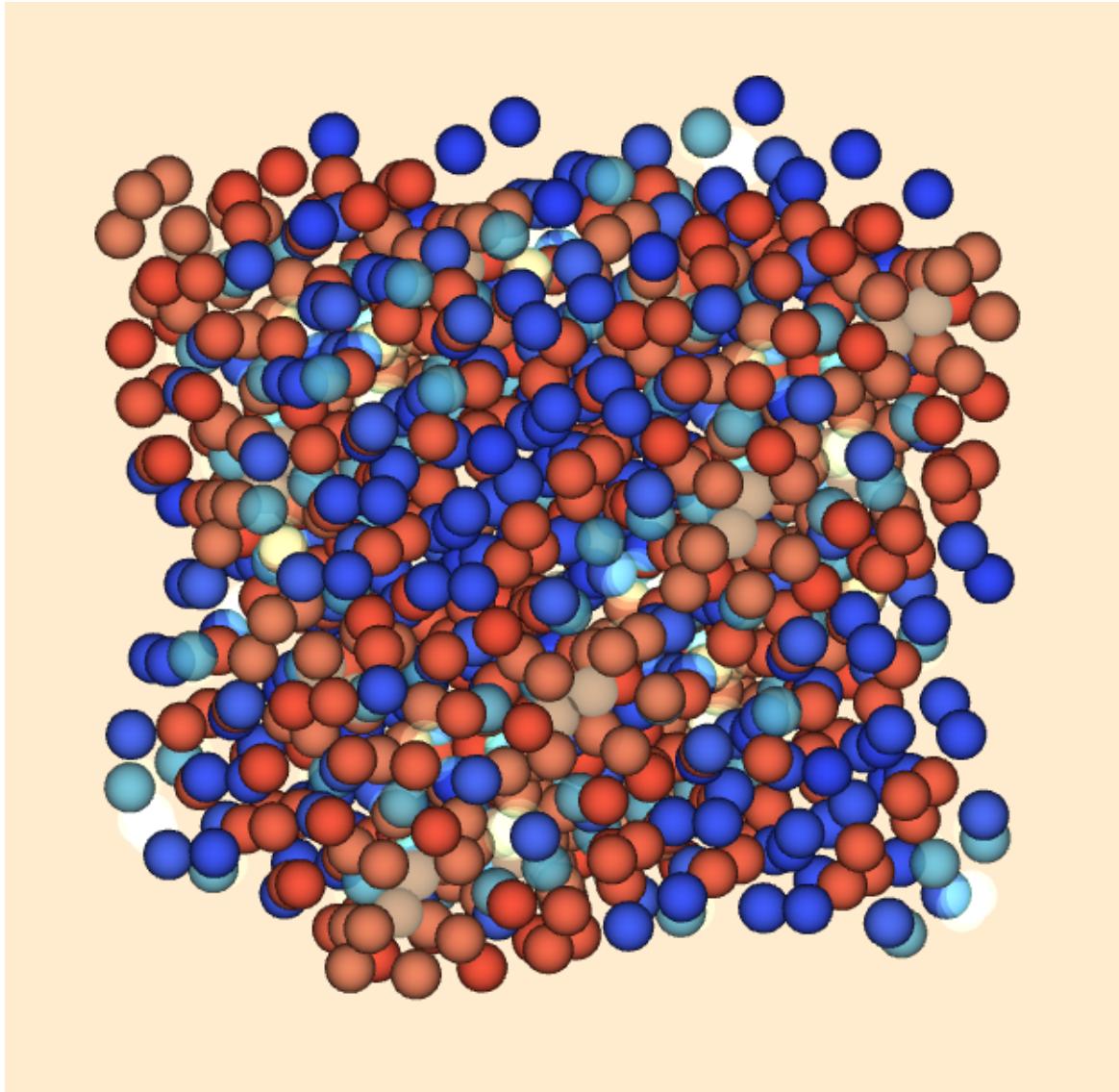
Pierwszy prototyp bazował na XNA. Po załadowaniu testowych plików prototyp spełnił wszystkie wymagania stawiane w specyfikacji.



Rysunek 3.4: Zrzut ekranu pierwszego prototypu (XNA).

Kolejny prototyp powstał w WPF. Po załadowaniu modeli działał on wyraźnie wolniej od pierwszego, jednak na tyle dobrze, że spełnił wszystkie wymagania projektowe.

Reasumując, każda z wyszczególnionych technologii spełniła wszelkie kryteria, postanowiłem jednak wybrać WPF. Jest to technologia najbardziej optymalna pod względem poświęconego czasu inżynieryjnego. Posiada wbudowany układ współrzędnych w trzech wymiarach, kamerę, predefiniowane tekstury i wiele innych. Integracja z graficznym interfejsem nie wymaga żadnej dodatkowej pracy w odróżnieniu od XNA.



Rysunek 3.5: Zrzut ekranu drugiego prototypu (WPF).

4. Implementacja aplikacji

W rozdziale opiszę projekt aplikacji. Omówię strukturę programu, interfejs użytkownika, użyte algorytmy oraz przedstawię, jak aplikacja została przetestowana.

4.1. Struktura aplikacji

Struktura omawianej aplikacji opiera się na wzorcu architektonicznym Model View Controller (MVC). Wzorce architektoniczne określają sprawdzony sposób tworzenia architektury systemu. Definiują one, z jakich elementów składa się system, ogólną strukturę komponentów, komunikację pomiędzy modułami oraz jaki zakres funkcjonalności przypada na każdy komponent.

MVC głównie używany jest w aplikacjach, posiadających interfejs graficzny. Składa się on z 3 części:

1. Model - określa logikę aplikacji
2. View - odpowiada za prezentowanie danych użytkownikowi
3. Kontroler - opisuje, jak Model komunikuje się z View.

Nierozłącznymi elementami każdej aplikacji są struktury danych, algorytmy i komunikacja. MVC silnie separuje każdy z elementów. Przykładowo, w dobrze zaprojektowanej aplikacji kod wyświetlający informacje użytkownikowi nie może znajdować się w modelu. Do zalet tego wzorca należą:

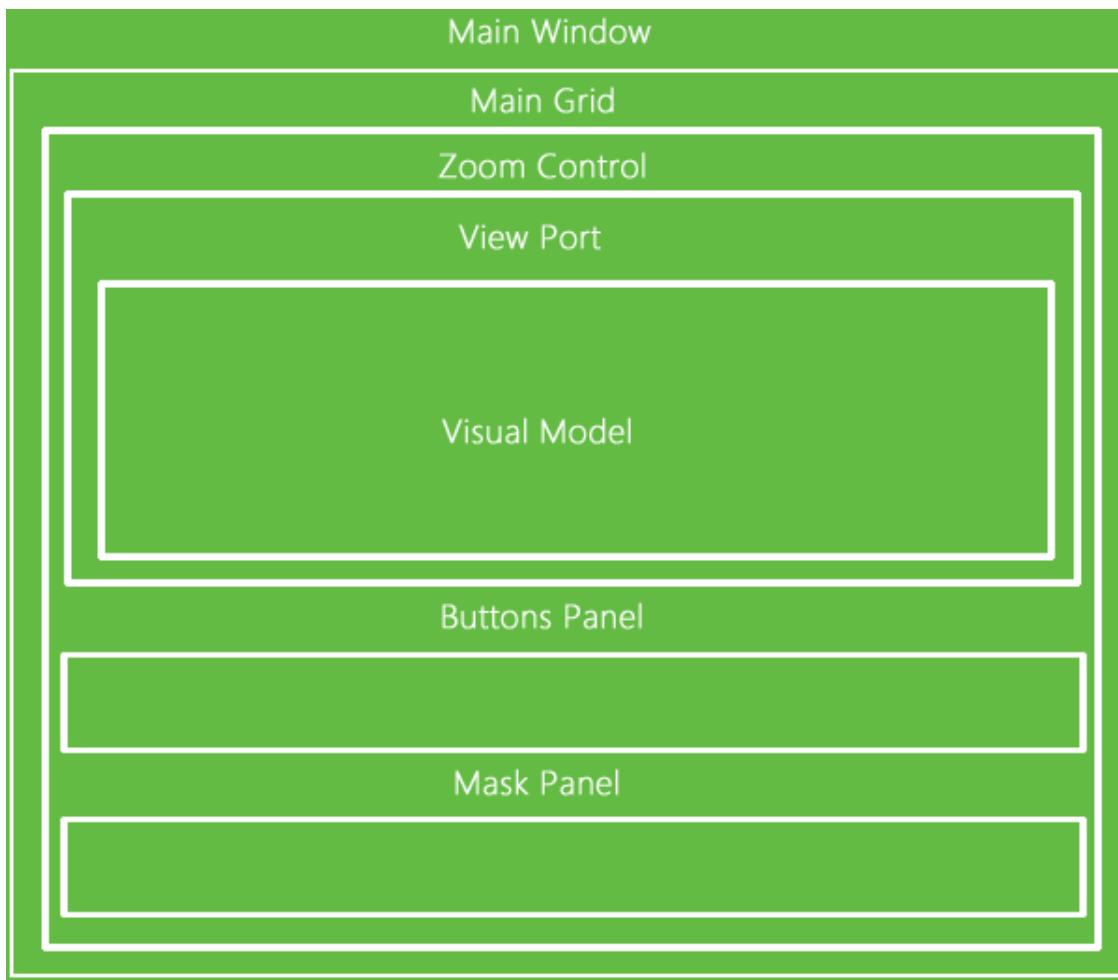
1. Niezależność modelu i widoku - można zmienić wygląd lub dodać inny do aplikacji bez konieczności ingerencji w model
2. Lepsza podatność na zmiany - zespoły pracujące nad widokiem i modelem mogą pracować niezależnie

Natomiast wady to:

1. Złożoność - aplikacje oparte na tym wzorcu bywają rozbudowane, dlatego najczęściej stosuje się go wobec średnich i dużych projektów

2. Kosztowne zmiany interfejsów

MVC jest bardzo popularnym wzorcem pośród aplikacji internetowych, gdzie widok jest definiowany w języku HTML, model określa aplikacja serwerowa i komunikacja na ogół opiera się na żądaniach HTTP[12]. Windows Presentation Foundation opiera się na wzorcu MVVM (Model View View-Model), jednak nie korzystałem z niego w swojej aplikacji - cechy jakie posiada MVVM nie były konieczne. Strukturę widoku wizualizacji najlepiej oddaje rysunek 4.1.

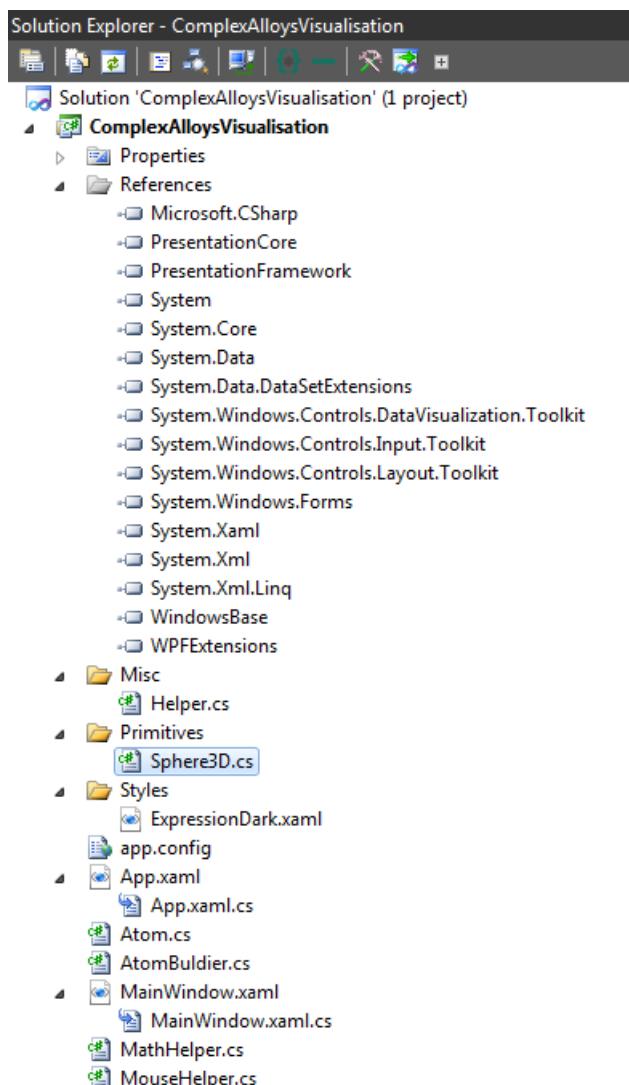


Rysunek 4.1: Struktura widoku - podział na kontenery.

W głównym oknie osadzony jest kontener typu Grid (Main Grid). Jest to rodzaj panelu, który pozwala na umieszczenie wielu elementów wewnętrz. Zoom Control służy do powiększania i pomniejszania modeli. Obok niego jest panel, w którym znajdują się guziki oraz suwaki (Buttons Panel) i panel z filtrem maski (Mask Panel). Najistotniejszy, z punktu widzenia użytkownika, jest panel typu Viewport3D (View Port), który odpowiada za wyświetlanie obiektów w trzech wymiarach. W środku znajduje się Visual Model, do którego przypisany jest model atomów[23]. Dzięki takiej strukturze aplikacja w łatwy sposób dostosowuje się do różnych rozdzielczości ekranu.

W aplikacji można wyszczególnić kilka wzorców projektowych, które są wszechobecne wśród programów. Deweloperzy bardzo często korzystają z wzorców, często niejawnie, aby wykorzystać najlepsze i sprawdzone rozwiązanie. Jednym z wzorców kreacyjnych, z których korzystam jest Budowniczy. Dzięki niemu można oddzielić tworzenie obiektów od logiki, co pozwala nam używać tego samego procesu dla różnych obiektów np. obiektów służących do testów[9]. Na podstawie plików wynikowych i parametrów ustawionych przez użytkownika program tworzy modele atomów w komórce elementarnej. Używana do tego jest klasa AtomBuilder, która z kolekcji atomów tworzy model 3D.

Program do wizualizacji składa się z 5569 niepustych linii kodu, 11 plików i biblioteki WPFEstenstion, która nie wchodzi w skład standardowej biblioteki. Rysunek 4.2 przedstawia strukturę projektu:



Rysunek 4.2: Struktura projektu Complex Alloys Visualisation.

Properties zawierają informacje, takie jak numer identyfikacyjny biblioteki lub pliku wykonywalnego. Folder References określa wszystkie zależności, jakie posiada projekt. W folderze Misc znajduje się klasa Helper, definiująca metody pomocnicze. Następnie folder Primitives posiada klasę Sphere3D, która definiuje teselacje sfery. Plik ExpressionDark.xaml w folderze Styles określa wygląd GUI. App.xaml oraz App.xaml.cs określają punkt startowy aplikacji oraz obiekty istniejące przez cały czas istnienia procesu. Klasa Atom definiuje obiekt reprezentujący pojedynczy atom. AtomBuilder tworzy model z atomów. MainWindow.xaml odpowiada z GUI w oknie głównym. MainWindows.xaml.cs określa interakcje użytkownika z programem. MathHelper posiada funkcję do konwersji stopni na radiany. Klasa MouseHelper określa w jaki sposób myszka może obracać model.

4.2. Użyte Algorytmy i Struktury Danych

Aplikacja w pełni napisana jest w języku obiektowym C#. W książce „Język C# Programowanie” znalazłem bardzo dobrą definicję języka:

C# (wymawiane jako „si szarp” to prosty, nowoczesny, obiektowy i bezpieczny pod względem stosowania typów język programowania. Korzenie C# tkwią w rodzinie języków C, dzięki czemu szybko będą go mogli przyswoić programiści używający C, C++ i Java. Standaryzacją C# zajmuje się organizacja ECMA International, która opracowała normę ECMA-334, oraz IOS/IEC, odpowiedzialna za normę ISO/IEC 23270. Kompilator języka C# oferowany przez firmę Microsoft w ramach platformy .Net spełnia wymogi obydwu wymienionych standardów.

C# jest językiem obiektowym, lecz zapewnia także wsparcie dla programowania komponentowego (ang. component-oriented). Nowoczesne sposoby projektowania oprogramowania w coraz większym stopniu opierają się na komponentach programowych, mających postać niezależnych i samoopisujących się pakietów funkcji. Kluczem do tego typu komponentów jest to, że prezentują one model programowania za pomocą właściwości, metod i zdarzeń, są wyposażone w atrybuty udostępniające deklaratywne informacje na temat komponentu, a także zawierają swoją własną dokumentację. C# umożliwia korzystanie z konstrukcji językowych, które w bezpośredni sposób wspierają wymienione koncepcje, co czyni go niejako naturalnym językiem do tworzenia i używania komponentów programowych[2].

Język ten jest kompilowany do kodu pośredniego Common Intermediate Language (CIL), który jest wykonywany przez środowisko uruchomieniowe. Natywnie wspieranym środowiskiem jest .Net działający pod system Windows. Istnieje również alternatywna, otwarta implementacja Mono, która wspiera wiele systemów operacyjnych. Język C# jest stosunkowo

młodym językiem. Pierwsza wersja powstała w 2001 roku. C# posiada bardzo wiele przydatnych cech, takich jak zarządzanie pamięcią, wyrażenia LINQ czy pełna kontrola typów. Jednak dla mnie najbardziej użyteczna jest wieloparadygnowość języka. C# w wersji 1.0 pozwalał na pisanie kodu obiektowego oraz proceduralnego. C# 2.0 dodał wsparcie dla programowania generycznego. C# 3.0 został wzbogacony o elementy programowania funkcjonalnego poprzez wyrażenia lambda oraz LINQ. C# 4.0 dodał wsparcie dla programowania dynamicznego, natomiast C# 5.0 wspiera programowanie asynchroniczne. Na daną chwilę nie istnieje żaden mainstreamowy język, który wspierałby tak wiele metodologii programowania. Język ten posiada również mało tzw. „gotcha” (sprzeczne z intuicją rozwiążanie, które powoduje błędy w programowaniu), przez co programy zawierają mniej błędów. Moim zdaniem jest to istotna cecha, która wpływa na jakość oprogramowania. Ludzki mózg posiada swoje ograniczenia, więc korzystając z prostszych narzędzi, możemy budować bardziej skompilowane systemy[2].

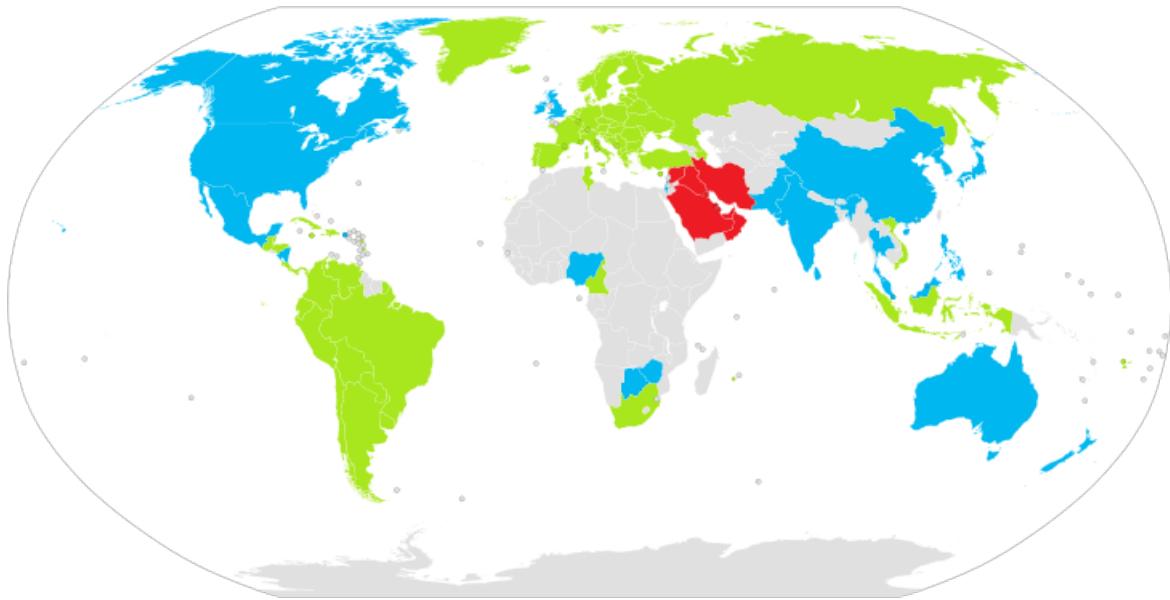
4.2.1. Funkcje pomocnicze

Każda aplikacja średniej lub dużej wielkości posiada zbiór funkcji pomocniczych. Pomagają one w ograniczeniu powtarzającego się kodu w obrębie programu. Dlaczego tworzyć zbiór funkcji zamiast biblioteki? Ponieważ często są zbyt specyficzne dla danej domeny problemu, aby złożyć z nich bibliotekę. Programista ma zawsze możliwość wniesienia zmian według wymagań. Na potrzeby aplikacji stworzyłem 3 funkcje pomocnicze. Pierwsza z nich służy do zamiany stopni na radiany według powszechnie znanego wzoru:

$$\frac{\text{stopnie}}{180} * \Pi \quad (4.1)$$

Dwie kolejne metody służą do parsowania łańcuchów tekstowych i zamiany na liczbę zmiennoprzecinkową. Dlaczego definiować na nowo tak prostą funkcję, która istnieje w wielu bibliotekach? Są ku temu dwa powody: różny separator dziesiętny oraz formatowanie liczb.

Na świecie dominują dwa różne separatory. Pierwszy z nich to kropka, który na powyższej mapce zaznaczony jest kolorem niebieskim. Drugi to przecinek - kolor zielony. Istnieje również trzeci separator, momayyez, który występuje w krajobrazach arabskich. Użycie nieprawidłowego separatora dziesiętnego może mieć różne rezultaty. Najmniej szkodliwe jest oczywiście nieprawidłowe wyświetlanie, natomiast bardzo często aplikacja zmienia swoje działanie lub zawiesza się. Fakt ten wynika z niewiedzy, ignorancji lub błędu w projekcie, tak jak w przypadku platformy .Net. Domyslnie ustawiony jest lokalny separator, dlatego przykładowo aplikacja czytająca plik w Wielkiej Brytanii może działać poprawnie, natomiast w Polsce zawiesza się. Jest kilka rozwiązań dla tego problemu. Jeżeli posiadamy źródła programu, możemy poprawić kod, który sprawia problemy. W przypadku braku dostępu do kodu źródłowego możemy stworzyć moduł, który będzie konwertował pliki. Innym sposobem jest zmiana ustawień użytkownika,



Rysunek 4.3: Rodzaje separatorów dziesiętnych na świecie. Zielony: przecinek. Niebieski: kropka. Czerwony: momayyez. Źródło: Wikipedia na licencji Creative Commons.

jednak może to wpływać na inne aplikacje. Najlepszym rozwiązaniem jest stworzenie nowego użytkownika z odpowiednimi ustawieniami tylko dla tej aplikacji.

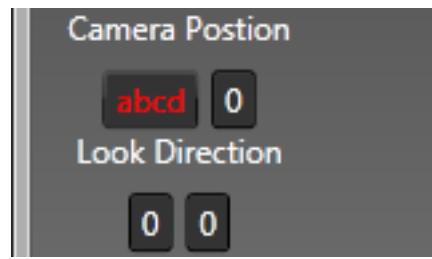
Aby uniknąć tego błędu, moja funkcja korzysta z wbudowanej funkcji z odpowiednimi parametrami:

```
static bool TryParseDouble( this string text , out double value )
{
    return double . TryParse( text , NumberStyles . Any ,
        CultureInfo . InvariantCulture , out value );
}
```

Funkcja akceptuje każdy format liczby zmiennoprzecinkowej. Dobrą praktyką jest akceptowanie większego zbioru danych, aby później odpowiednio je filtrować. Zdefiniowana metoda jest metodą rozszerzeń. Można ją rozpoznać po słowie kluczowym „this”. Dzięki temu można ją użyć, jakby była zdefiniowana w typie `string`. Druga metoda ma to samo zadanie, jednak dodatkowo wyświetla błąd dla użytkownika w postaci czerwonego tekstu. Świadczy on o nieprawidłowym formacie liczby.

4.2.2. Klasa Sphere3D

Model każdego atomu jest przedstawiany jako sfera. Kształt pojedynczej sfery jest tworzony w kodzie przy pomocy algorytmu teselacji, który definiuje poniższy pseudokod:



Rysunek 4.4: Nieprawidłowa pozycja kamery.

1. Ustalamy warunki początkowe: powierzchnia sfery składa się z 32 trójkątów
2. Budujemy sferę, posługując się kątem theta z zakresu $<0; 360>$ oraz zmienną y z zakresu $<-1;1>$
3. Tworzymy pustą siatkę MeshGeometry3D
4. Dla kroków dt oraz dy dodajemy do siatki pozycje punktów, normalne oraz współrzędne tekstury
5. Łączymy punkty trójkątami
6. „zamrażamy” siatkę

Tak stworzona siatka może zostać wykorzystana wielokrotnie, dlatego obliczana jest tylko raz.

4.2.3. Klasa Atom

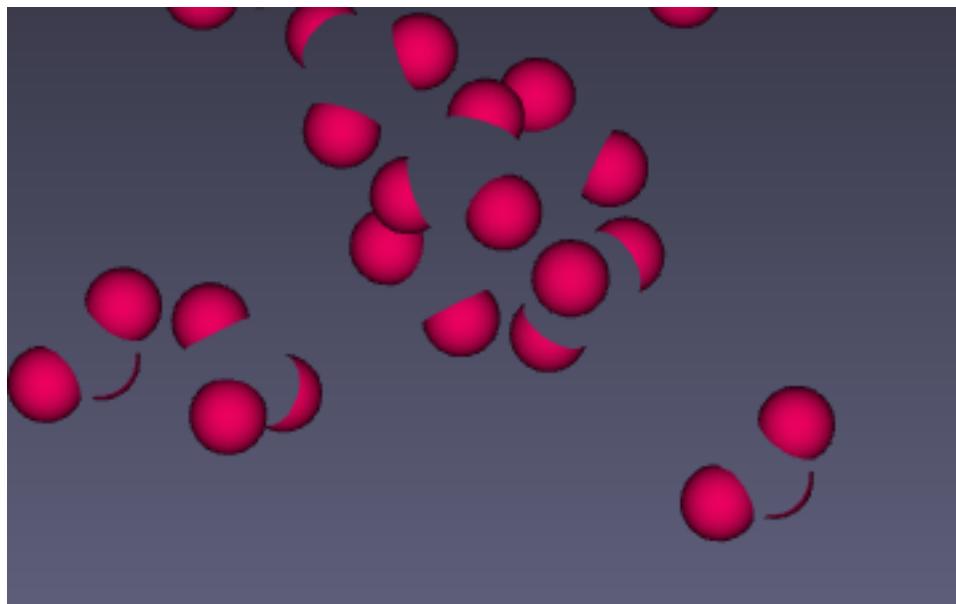
Klasa atom definiuje strukturę danych dla pojedynczego atomu. Seria atomów jest tworzona podczas parsowania pliku z parametrami. Każdy atom ma następujące parametry:

1. Współrzędne x, y oraz z wyrażone jako względna wartość z zakresu $<-0,5; 0,5>$
2. Długość komórki elementarnej
3. Rodzaj atomu (Magnez lub Aluminium)
4. Prawdopodobieństwo wystąpienia wakancji
5. Prawdopodobieństwo obsadzenia atomem Aluminium
6. Prawdopodobieństwo obsadzenia atomem Magnezu
7. Stała konwersji do Angstromów (28,289)

Dzięki tym parametrom możemy zbudować model siatki krystalicznej dla komórki elementarnej.

4.2.4. Klasa AtomBuilder

Metoda CreateModel w klasie AtomBuilder tworzy model 3D na podstawie kolekcji atomów. Podczas implementacji algorytmu napotkałem pierwsze ograniczenie narzucone przez WPF. Jest to brak bezpośredniego dostępu do Z-buffer, który odpowiada za zarządzanie głębią w przestrzeni trójwymiarowej. Dzięki Z-bufferowi rysowane są tylko te elementy, które są widoczne. Problem pojawia się, gdy niektóre elementy są całkowicie lub częściowo przeźroczyste. Domyślnie tekstura DiffuseMaterial używa Z-buffer, przez co przeźroczyste atomy na pierwszym planie zasłaniają te na dalszym[23].



Rysunek 4.5: Materiał używający Z-buffer.

Jako rozwiązanie tego problemu dodałem drobną modyfikację. Jeżeli przeźroczystość atomu jest poniżej 30%, zmieniam materiał na EmissiveMaterial, który nie zapisuje wartości do Z-buffera.

Podczas wizualizacji modeli kolor jest jednym z kluczowych parametrów. Dostarcza użytkownikowi informacje o prawdopodobieństwie obsadzenia. Zależności pomiędzy kolorem a prawdopodobieństwem opisuje dana tabela:

Sumaryczne prawdopodobieństwo każdej ze składowych wynosi 1. Wszystkie wartości pośrednie powstają przez mieszanie się barwy czerwonej i niebieskiej. Niestety kolor składający się z 50% czerwonego oraz 50% niebieskiego nie jest zbyt wyraźny. Dlatego aby polepszyć kontrast i jakość modelu, dodatkowo występuje kanał zielony. Ilość zieleni jest proporcjonalna do różnicy poziomu obsadzenia magnezu i aluminium.

$$I(g) = 1 - \frac{|P(Mg) - P(Al)|}{|P(Mg) + P(Al)|} \quad (4.2)$$

Zmienna $I(g)$ jest poziomem intensywności kanału zielonego.

Kanał Alfa	Prawd. wakancji	Prawd. Obsadze-nia Mg	Prawd. Obsadze-nia Al	Komentarz
0	1	0	0	Brak atomu
1	0	1	0	Atom Mg (kolor nie-bieski)
1	0	0	1	Atom Al (kolor czer-wony)
0,5	0,5	0,5	0	Mg 50%
0,5	0,5	0	0,5	Al 50%
1	0	0,5	0,5	Jest pełne obsadze-nie, ale mieszane

Tablica 4.1: Zależność prawdopodobieństwa i koloru.

4.2.5. MainWindow

W pliku MainWindow.cs znajduje się kod, który definiuje interakcję użytkownika z aplikacją. Architektura WPF opiera się na zdarzeniach, które są obsługiwane przez funkcje obsługi. W przypadku klasy MainWindow jest to 19 funkcji odpowiedzialnych za m.in. obsługę przycisków, powiększanie lub pomniejszanie modelu, zmianę kamery, zmianę wektora kierunkowego kamery, skalowanie, zapis zrzutu ekranu, zmianę maski, przetwarzanie grupy plików oraz zmianę przeźroczystości. Typ MainWindow definiuje również pola, które opisują stan. Pierwszy z nich to lista obiektów typu Atom z początkową pojemnością dla 2000 elementów. Kolejna jest lista masek filtrująca serie pomiarowe. Następnie obiekty typu AtomBuldier, MouseHelper, domyślny promień dla atomów wynoszący 1.0 oraz łańcuch znaków wskazujący użytkownikowi, że żaden plik nie został wybrany.

Funkcje klasy MainWindow można podzielić na 3 grupy: tworzenie modelu, modyfikacja parametrów i przetwarzanie grupy modeli. Tworzenie modelu definiuje poniższy algorytm:

1. Użytkownik naciska przycisk Open File i w oknie wyboru pliku wybiera plik danych z rozszerzeniem .chmc
2. Czytane są dane z wybranego pliku, na podstawie których tworzona są obiekty reprezentujące atomy
3. Lista stworzonych atomów jest filtrowana poprzez maskę
4. Tworzony jest model 3D

Atomy filtrowane są poprzez zapytanie LINQ. LINQ jest nową technologią, opracowaną przez firmę Microsoft. Definiuje zbiór metod, z których tworzone są zapytania dla obiektów, takich jak kolekcje, bazy danych, XML czy inne zbiory danych. Jest to część języka, która bazuje na funkcjonalnym stylu programowania.[2]. Zastosowałem zapytanie LINQ do filtracji serii atomów, które są zgodne z wybraną maską.

```
private void draw()
{
    var filteredAtoms = from atom in atoms
                         where masks.Contains(atom.PositionNumber)
                         select atom;

    visualModel.Content = atomBuldier.CreateModel(filteredAtoms,
                                                   alphaSlider.Value);
}
```

Czytelnik znający język SQL może dostrzec podobieństwo słów kluczowych w zapytaniach LINQ.

Przetwarzanie grupy modeli jest w gruncie rzeczy modyfikacją algorytmu dla pojedynczego modelu:

1. Po naciśnięciu Process Files użytkownik, w oknie wyboru folderu, wybiera folder z plikami
2. Wczytywane są pliki o rozszerzeniu .chmc
3. Dla każdego pliku wczytywane są dane, filtrowane atomy zgodnie z maską, renderowane i ustawiane są parametry, które zdefiniował użytkownik
4. Każdy model zapisany jest w pliku w formacie PNG, gdzie jego nazwa odpowiada temperaturze

Ponieważ przetwarzanie dużej liczby plików może trwać dłuższą chwilę, aplikacja wyświetla kolejne modele, tak aby użytkownik wiedział, że program nie zawiesił działania. Dodatkowo na środku ekranu pojawia się napis „Processing Files”.

4.2.6. MouseHelper

Klasa MouseHelper służy do obrotu obiektu przy użyciu myszki. W tym celu WPF używa kwaternionów. Dla każdego ruchu myszki w poziomie i pionie obliczany jest obrót obiektu, jeżeli wciśnięty jest lewy klawisz a mysz znajduje się w obrębie okna.

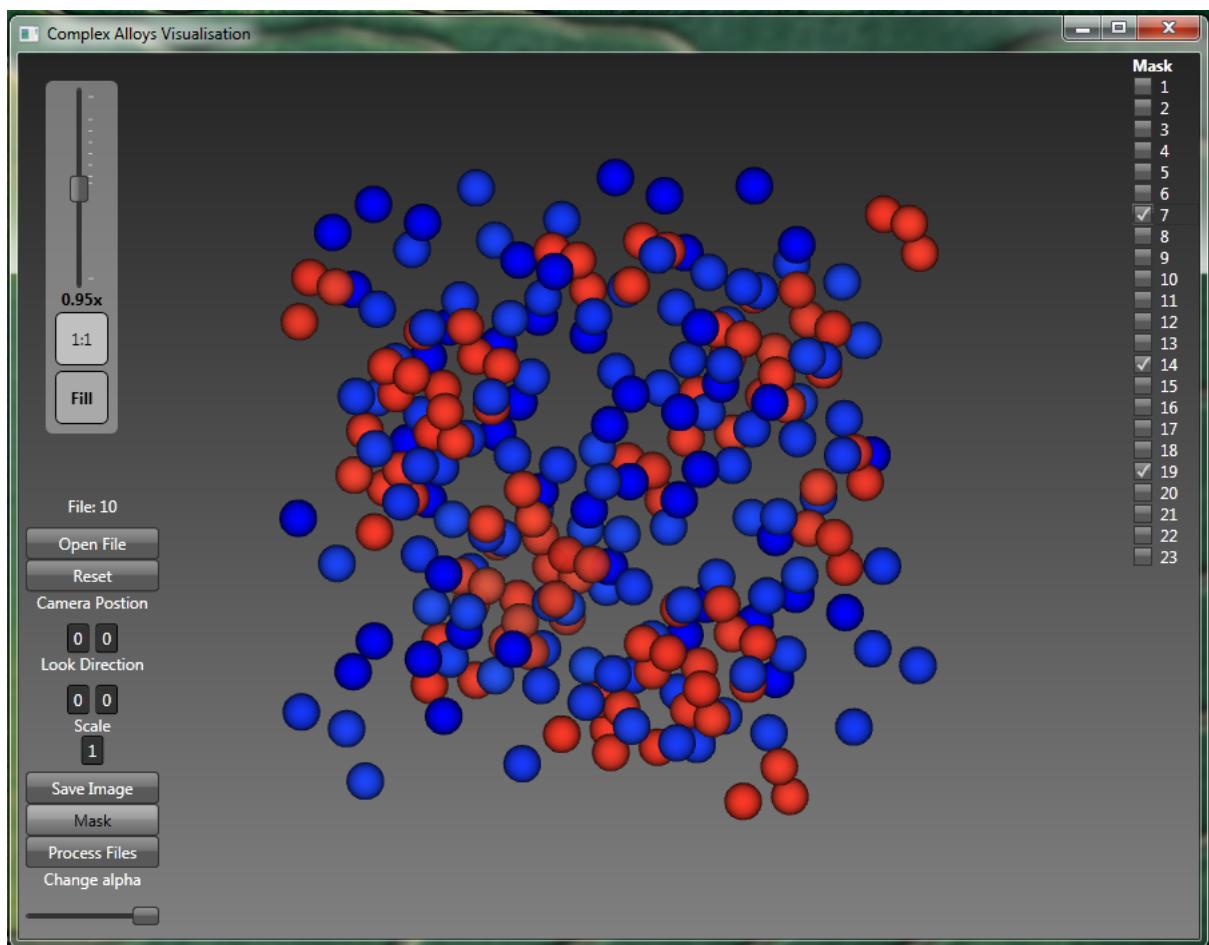
4.3. Graficzny Interfejs Użytkownika

Graficzny interfejs użytkownika (ang. Graphical User Interface), nazywany również środowiskiem graficznym, stanowi ogólne określenie sposobu prezentacji informacji przez komputer oraz sposobu interakcji z użytkownikiem. Głównym sposobem interakcji jest operacja na różnych elementach, które są rysowane na monitorze pod postacią obrazów. Klasycznie użytkownik komunikuje się z komputerem przy pomocy komend tekstowych[28].

Prekursorem wszystkich graficznych interfejsów był Sketchpad. Sketchpad został stworzony z myślą o wsparciu projektów technicznych. Użytkownik przy pomocy specjalnego pióra tworzył figury geometryczne, którymi następnie mógł manipulować poprzez obrót, skalowanie i przesunięcie. [8]

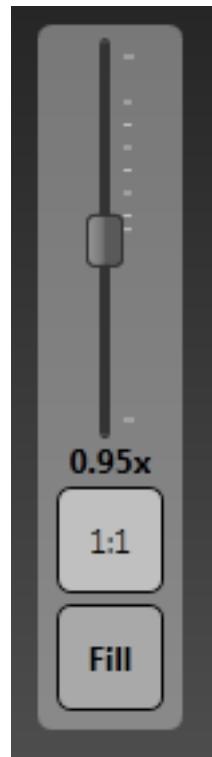
Pierwsze prototypy interfejsów znanych z dzisiejszych systemów powstały w laboratorium PARC firmy Xerox. PARC User Interface zawierał okienka, przyciski, ikony i menu. Dodano obsługę urządzenia wskazującego poza standardową klawiaturą. [13]

GUI mojej aplikacji składa się z głównego okna, w którym wyświetlane są modele atomów i dwóch paneli bocznych.



Rysunek 4.6: GUI programu do wizualizacji CMA.

W lewym panelu znajduje się kontrolka odpowiedzialna za skalowanie modelu.



Rysunek 4.7: Kontrolka skalowania modelu.

Atomy mogą być skalowane od 0.01 do 100 przy użyciu kółka w myszce lub bezpośrednio suwakiem. Kontrolka może również wrócić do domyślnej skali 1:1 lub wypełnić cały ekran. Służą do tego przyciski odpowiednio: „1:1” oraz Fill.

W lewym panelu znajdują się również elementy odpowiedzialne za główną funkcjonalność aplikacji.

Rysunek 4.8: posiada ponumerowane kontrolki, które odpowiednio służą do:

1. Wyświetlania nazwy pliku, z którego stworzony jest model
2. Otwierania pliku z modelem, który następnie jest parsowany do tworzenia modelu
3. Powrotu do domyślnych ustawień
4. Zmiany pozycji kamery w płaszczyźnie xy
5. Zmiany wektora, w jakim kierunku skierowana jest kamera
6. Skalowania promienia atomów
7. Zapisu zrzutu ekranu
8. Otwierania i zamykania panelu masek



Rysunek 4.8: Lewy panel GUI.

9. Przetwarzania grupy plików

10. Zmiany przeźroczystości

Każdy z elementów panelu został zaprojektowany w taki sposób, aby rezultat operacji był natychmiast widoczny dla użytkownika np. brak dodatkowego kroku w postaci odświeżania. Gdy użytkownik wpisze nieprawidłowe dane do pola tekstowego, tekst jest podświetlony kolorem czerwonym. Dane sprawdzane są z każdym kliknięciem klawiatury.

Lewy panel zawiera kontrolki służące do filtracji atomów.

Poprzez kliknięcie odpowiedniego checkboxa dodajemy lub usuwamy serię atomów.

4.4. Testy

Testowanie oprogramowania jest to badanie prowadzone w celu zapewnienia zainteresowanym stronom informacji o jakości produktu lub usługi. Testowanie może również dostarczyć dodatkowych informacji o systemie, takich jak ryzyko związanie z projektem.

Testowanie oprogramowania można podzielić na proces weryfikacji i walidacji składający się z czterech części:

Mask
1
2
3
4
5
6
<input checked="" type="checkbox"/> 7
8
9
10
11
12
13
<input checked="" type="checkbox"/> 14
15
16
17
18
<input checked="" type="checkbox"/> 19
20
21
22
23

Rysunek 4.9: Maski serii atomów.

1. sprawdzenie produktu pod kątem wymagań
2. czy oprogramowanie zachowuje się zgodnie z przewidywaniami
3. czy może być zaimplementowane zgodnie z wymaganiami
4. czy spełnia potrzeby zleceniodawcy

Testowanie tradycyjnie jest jednym z ostatnich etapów w budowaniu aplikacji, jednak nowe metodologie, takie jak Agile, przesuwają znaczną część testów do początkowych etapów.[18]

Przykładem takiego podejścia jest Test-driven development (TDD), który opiera się na krótkich cyklach podzielonych na:

1. tworzenie automatycznych testów początkowych, które opisują nową funkcjonalność
2. pisanie początkowego kodu, który "przechodzi" przez wszystkie testy
3. refactoring kodu do określonych standardów

Programiści często korzystają z TDD, aby poprawić jakość istniejącego już programu. [4]

Testowanie oprogramowania nigdy nie potwierdzi w 100% czy system nie ma wad, lecz daje nam odpowiedź na inne pytanie: Czy w pewnych warunkach oprogramowanie posiada błędy?

Przeanalizujmy poniższą funkcję:

Koszt naprawy		Czas wykrycia				
		Wymagania	Architektura	Programowanie	Testy	Po dostarczeniu
Czas Wystąpienia	Wymagania	1x	3x	5-10x	10x	10-100x
	Architektura	-	1x	10x	15x	25-100x
	Programowanie	-	-	1x	10x	10-25x

Tablica 4.2: Zależności między kosztem naprawy błędu a etapem wystąpienia.

```
int add(int a, int b) {
    return a + b;
}
```

Aby w pełni przetestować tę funkcję, należy sprawdzić wynik dla każdych możliwych danych wejściowych. W przypadku typu liczbowego int daje nam to $4\ 294\ 967\ 296$ możliwości dla pierwszej oraz tyle samo dla drugiej liczby. W sumie daje to $1,84467441 \times 10^{19}$ testów jednostkowych tej trywialnej funkcji! Dlatego nigdy nie tworzy się wszystkich możliwych testów. Funkcję add, w praktyce, należy przetestować dla kilku wartości losowych oraz dla przypadków szczególnych, takich jak maksymalna wartość int.

Bardzo ważnym elementem testowania jest tworzenie testów z uwzględnieniem odbiorcy oprogramowania. Użytkownicy gry video diametralnie różnią się od użytkowników systemu bankowego.

Testowanie oprogramowania jest złożonym procesem, który wymaga sporych nakładów ludzkich. Jak zawsze w takich sytuacjach, pojawia się pytanie o opłacalność prowadzenia testów. W 2002 National Institute of Standards and Technology (NIST) sporządził raport dotyczący wpływu błędów oprogramowania na gospodarkę amerykańską. Roczny koszt nieprawidłowego działania aplikacji w roku 2002 wyniósł 59,5 mld dolarów. Około jedną trzecią tej kwoty da się zaoszczędzić, jeżeli oprogramowanie byłoby lepiej testowane[29]. Wcześniej wykryte błędy są tańsze w naprawie. Potwierdza to analiza wielu projektów, której wyniki opisuje dana tabela. Wartości reprezentują krotności pojedynczego kosztu[25].

Testowanie oprogramowania można podzielić na kilka kategorii. Jednym z nich jest podział na testy statyczne i dynamiczne. Testy statyczne są to testy, które nie wymagają uruchomienia programu np. Code Review, zaś testy dynamiczne są to takie, w których aplikacja jest uruchamiana np. testy jednostkowe.

Metody testowania również dzielą się na Black-box oraz White-box. Przy testowaniu typu White-box tester ma wiedzę o architekturze i wewnętrznej strukturze testowanego modułu. Stosowane są głównie jako testy jednostkowe, testy API. Testy typu Black-box nie korzystają z wewnętrznych informacji. Tester ma tylko informację o spodziewanym rezultacie. Istnieją również hybrydowe metody, nazywane Grey-box.[33]

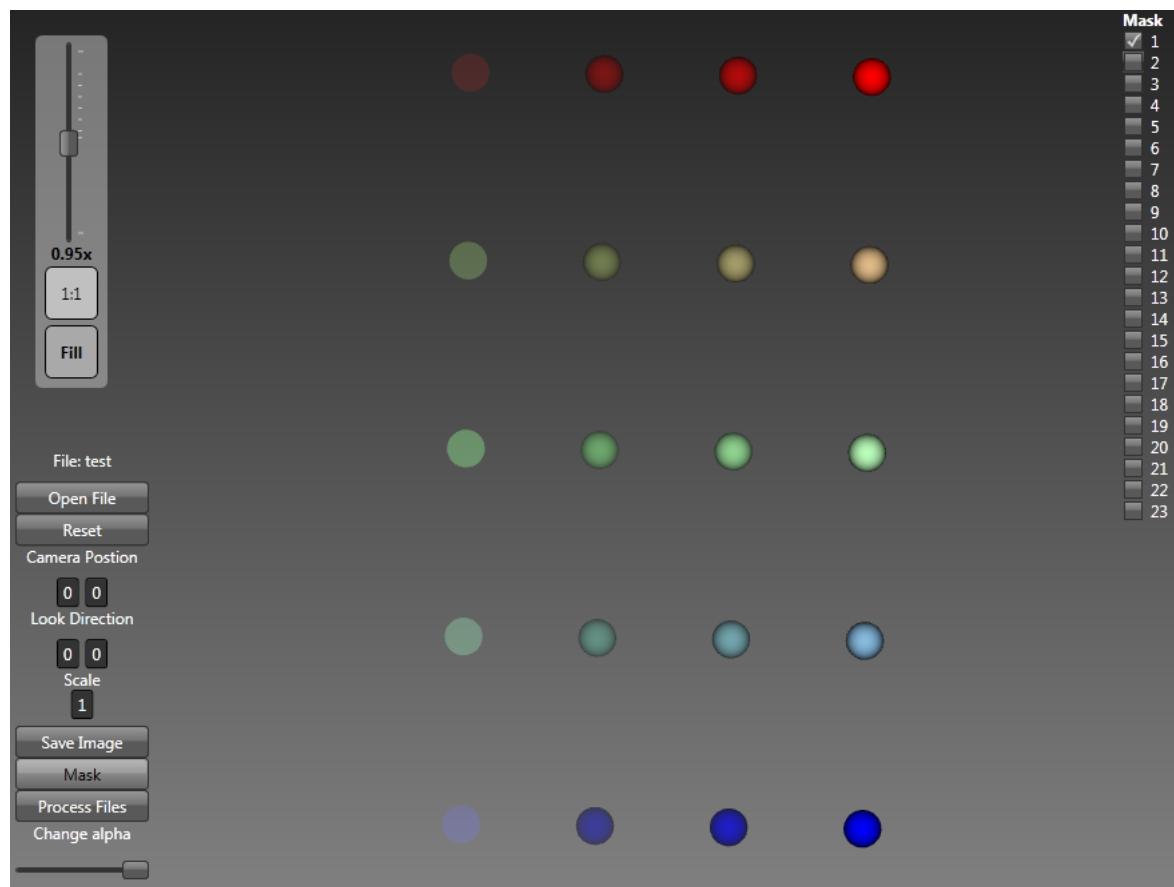
Podczas tworzenia aplikacji do wizualizacji CMA korzystałem głównie z testów GUI, wydajnościowych i regresywnych. Testy GUI mają na celu sprawdzenie działania każdego elementu interfejsu. Istotne nie tylko jest poprawne działanie każdego z elementów, ale również ich zachowanie w określonej sekwencji. Z tego powodu wraz z komplikacją GUI liczba testów rośnie eksponencjalnie. Mój program posiada tylko kilkanaście elementów, dlatego manualne testowanie jest wystarczające.

Testy wydajnościowe mają na celu zbadanie, jak system zachowuje się pod obciążeniem. Możemy określić, czy system spełnia wymagania wydajnościowe, zbadać skalowalność i niezawodność. Testy wydajnościowe mogą również pomóc w wykrywaniu błędów. Podczas testowania odkryłem, że jedna z metod zajmuje około 95% czasu. Był to banalny błąd. Dla każdego atomu obliczałem siatkę na nowo, mimo że jest współdzielona. Testy pomogły również zoptimizować przetwarzanie grupy plików. W pierwszej wersji przetworzenie 30 plików zajmowało około 20 minut. W ostatecznej wersji proces ten trwa około 3 minuty.

Testy regresywne jest to rodzaj testowania, który szuka błędów w istniejącej funkcjonalności po wprowadzeniu zmian. Innymi słowy, chcemy zbadać, czy wprowadzane poprawki nie naruszają działającej poprawnie części systemu.[33] Po każdej większej zmianie, a przed dodaniem kodu do systemu kontroli wersji, przeprowadzałem co najmniej jeden test regresywny.

Ostatnim testem była kontrola wizualizacji prawdopodobieństw obsadzenia. Posłużył do tego plik test.chmc, który zawierał kombinacje obsadzeń ze zmianą od całkowitego Al do całkowitego Mg poprzez obsadzenia mieszane. Generuje to w sumie 20 atomów (rysunek 4.10).

Na rysunku 4.10 widać, że przejścia w palecie są gładkie oraz kolor, zgodnie z oczekiwaniemi, przechodzi od czerwonego do niebieskiego z domieszką zielonej barwy.



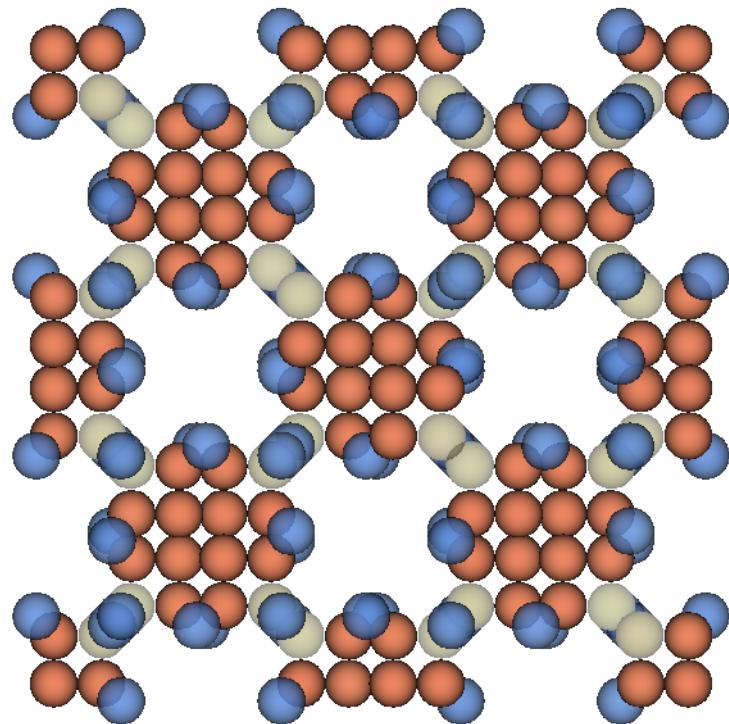
Rysunek 4.10: Zbiór testowy 20 atomów.

5. Podsumowanie

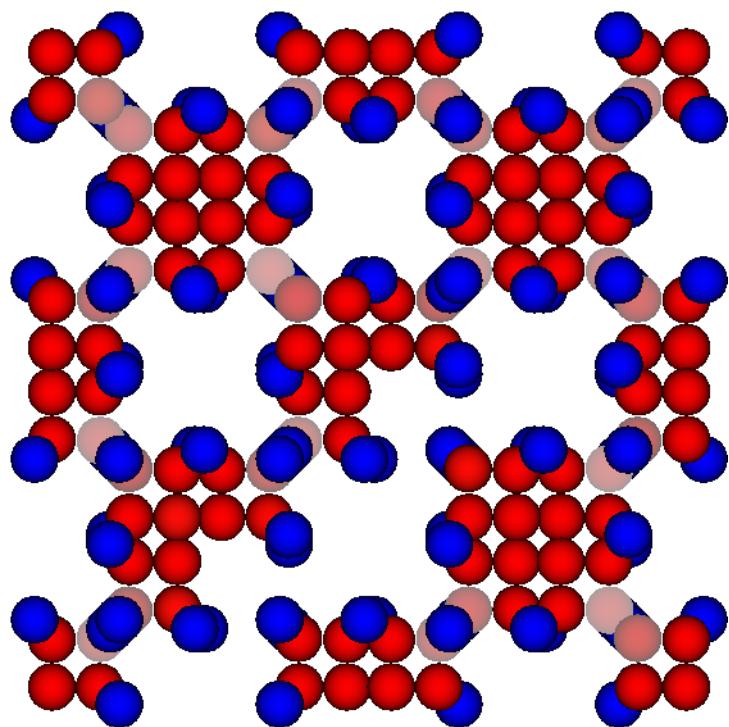
W rozdziale zaprezentuję wyniki, opiszę jak można rozszerzyć aplikację oraz podsumuję zrealizowane zadania.

5.1. Wyniki

Celem pracy było stworzenie narzędzia, dlatego postanowiłem zamieścić kilka obrazów wynikowych ilustrujących możliwości aplikacji. Przykładowe wyniki wykorzystane do prezentacji możliwości aplikacji zostały otrzymane z symulacji przeprowadzonej w 31 temperaturach, stopniowo obniżanych z początkowych 2000K do 400K. W trakcie symulacji obserwujemy najpierw porządkowanie obsadzeń na pozycjach narożnych kластerów (pozycje nr 8, 21) pomiędzy 2000 a 1200K, a następnie obserwujemy dyfuzję i porządkowanie obsadzeń na pozycjach Al w kластerze (nr 8) w zakresie temperatur od 900 do 600K. Poniższe rysunki ilustrują zmiany zachodzące w systemie poprzez zmiany kolorów i przezroczystości odpowiednich obiektów. Użytkownik może w każdej chwili wykonać zrzut ekranu oraz zmieniać parametry modelu (rysunek 5.1 i 5.2).

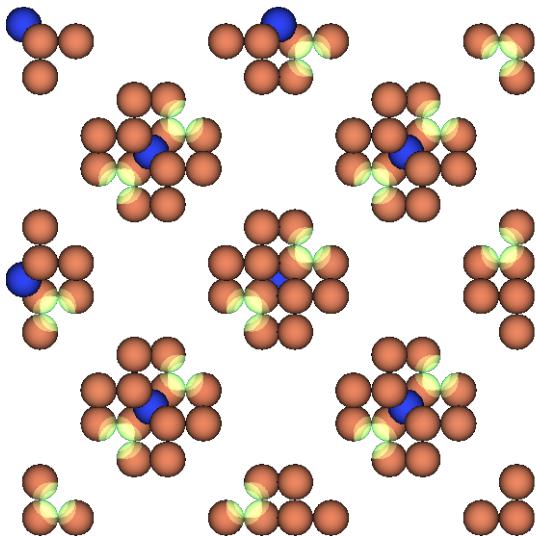


Rysunek 5.1: Seria pozycji nr 3, 14 i 19 atomów β - Mg_2Al_3 przy temperaturze 2000K.

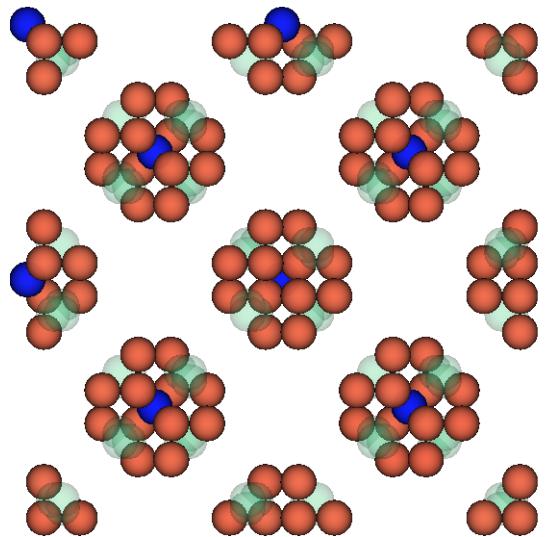


Rysunek 5.2: Seria pozycji 3, 14 i 19 atomów β - Mg_2Al_3 przy temperaturze 400K.

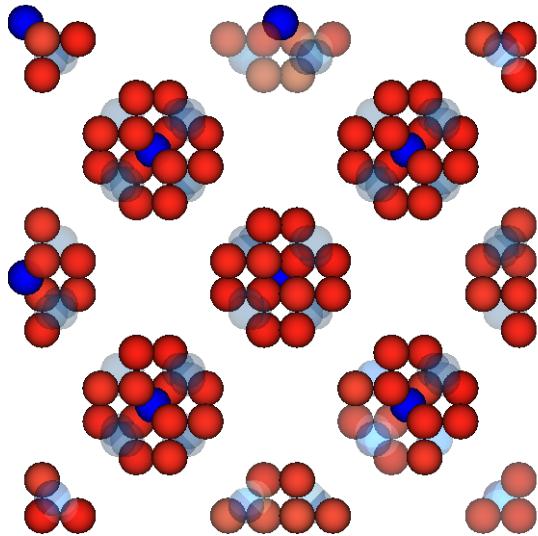
Przy przetwarzaniu wielu plików możemy stworzyć kolejne klatki przedstawiające ewolucję systemu dla klastra głównego. Poniżej kolejne stany począwszy od temperatury 2000 K dla pozycji 1, 8, 11, 12, 13, 14, 21. Wszystkich modeli jest 31, dlatego zaprezentuję tylko kluczowe klatki.



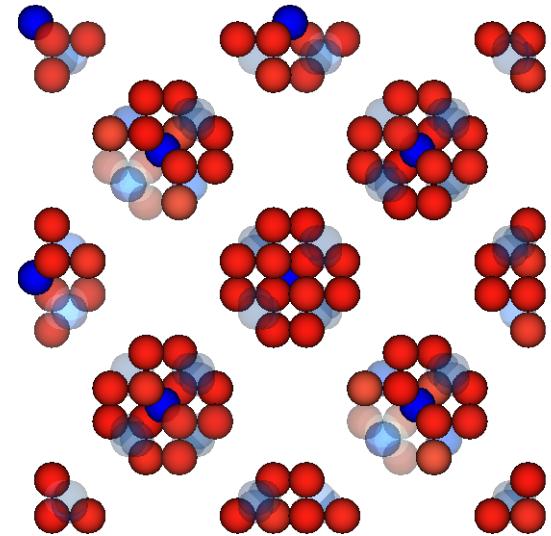
Rysunek 5.3: Stan komórki elementarnej dla temperatury 2000K.



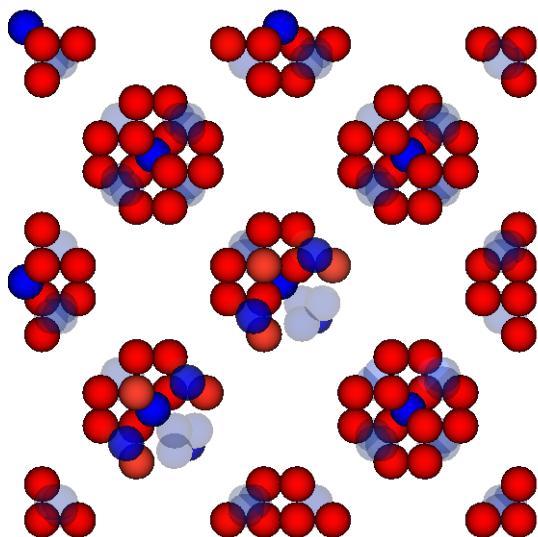
Rysunek 5.4: Stan komórki elementarnej dla temperatury 1500K.



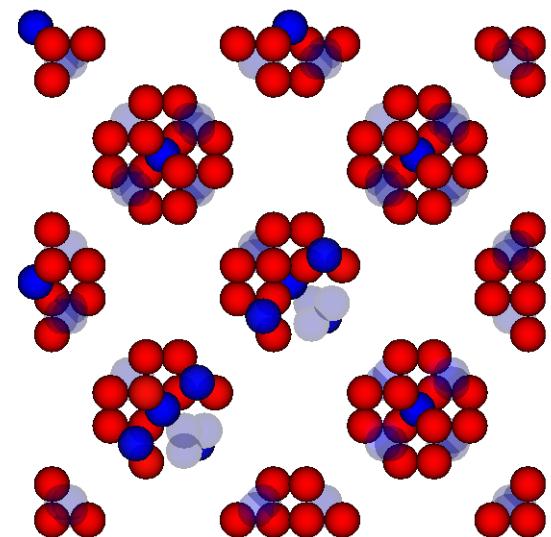
Rysunek 5.5: Stan komórki elementarnej dla temperatury 900K.



Rysunek 5.6: Stan komórki elementarnej dla temperatury 850K.

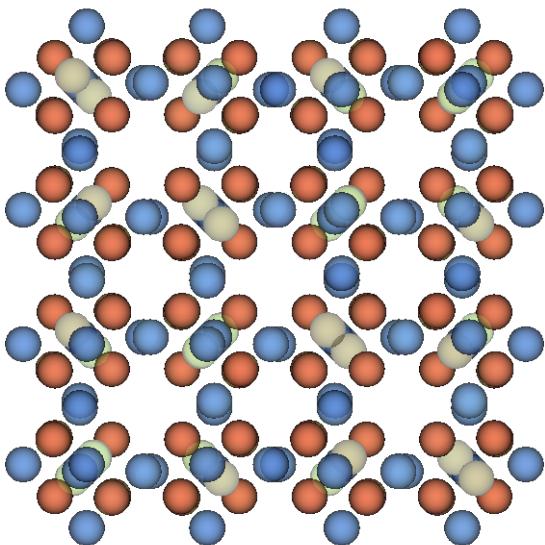


Rysunek 5.7: Stan komórki elementarnej dla temperatury 650K.

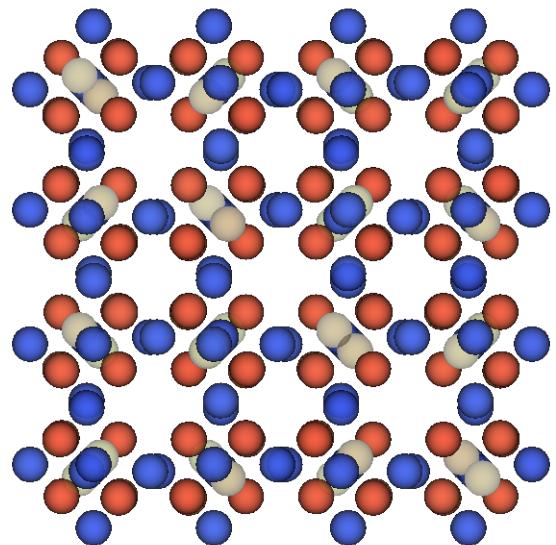


Rysunek 5.8: Stan komórki elementarnej dla temperatury 400K.

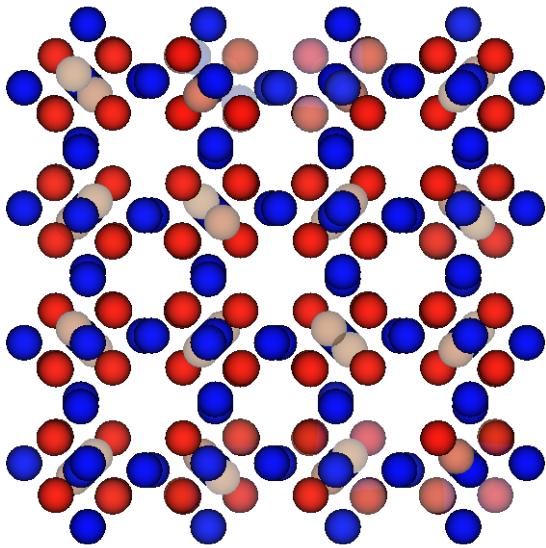
Kolejna seria rysunków przedstawia połączenia dla pozycji 3, 9, 10, 18 i 19.



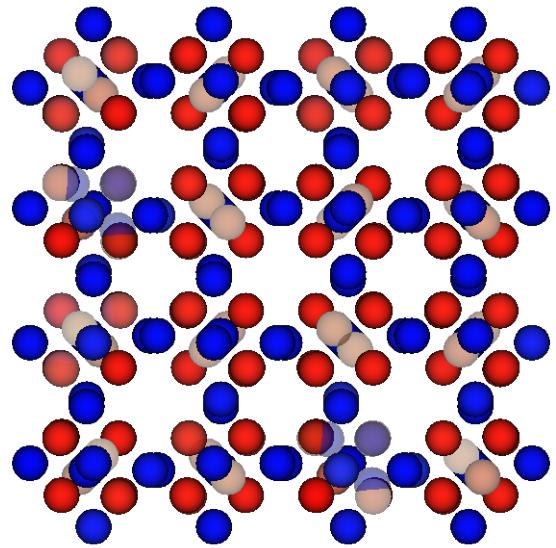
Rysunek 5.9: Stan komórki elementarnej dla temperatury 2000K.



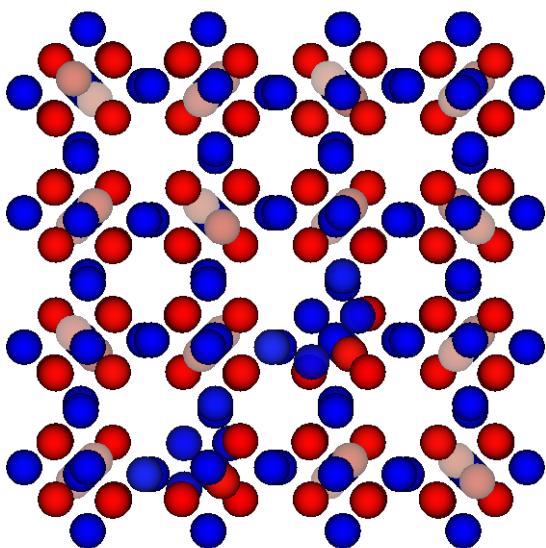
Rysunek 5.10: Stan komórki elementarnej dla temperatury 1500K.



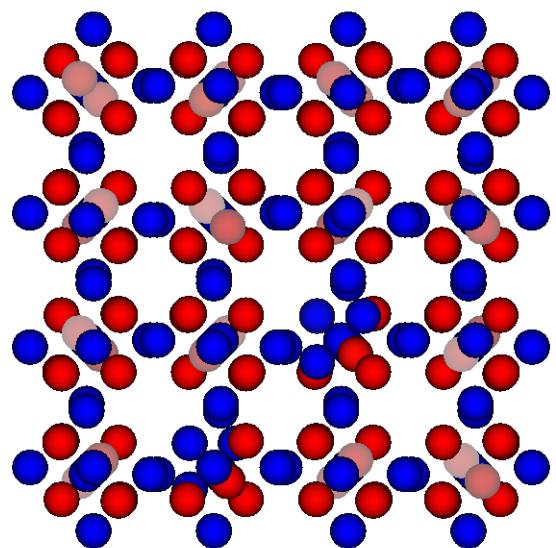
Rysunek 5.11: Stan komórki elementarnej dla temperatury 900K.



Rysunek 5.12: Stan komórki elementarnej dla temperatury 850K.



Rysunek 5.13: Stan komórki elementarnej dla temperatury 650K.



Rysunek 5.14: Stan komórki elementarnej dla temperatury 400K.

Z obrazów wynikowych złożyłem film. Można go obejrzeć pod adresem: <http://www.youtube.com/watch?v=CsOJeIbpqro> (wymagana jest przeglądarka z obsługą HTML5 lub z wtyczką Flash).

5.2. Możliwe rozszerzenia

Projekt znajduje się po adresem <http://alloysvisualisation.codeplex.com/>. Posiada on w pełni otwarte źródła na licencji New BSD. Każdy może zmodyfikować aplikacje według własnych potrzeb. Program może w bardzo łatwy sposób zostać zmieniony, tak aby wyświetlał inny rodzaj modeli 3D. Najłatwiej modyfikować program przy użyciu Visual Studio 2010. Możliwa jest również komplikacja z linii poleceń. Do modyfikacji, poza źródłami, niezbędna jest również biblioteka WPFEextension. Można ją znaleźć na stronie projektu.

5.3. Użyte narzędzia i biblioteki

Podczas implementacji korzystałem z następujących narzędzi i bibliotek:

1. Środowisko programistyczne - Visual Studio 2010 Ultimate
2. Środowisko uruchomieniowe - .Net 4.0
3. Język C# w wersji 4.0
4. System kontroli wersji - Team Foundation Server w serwisie codeplex.com
5. Biblioteka WPFEextensions - <http://wpfextensions.codeplex.com/>
6. Python 2.7

5.4. Zakończenie

Celem pracy było zaprojektowanie i implementacja systemu pozwalającego na wizualizację procesów dyfuzji i porządkowania w stopach międzymetalicznych, takich jak stop β - Mg₂Al₃. Kluczowe były dwa elementy: Przedstawienie modelu siatki krystalicznej z możliwością wygodnej manipulacji obiekttami oraz przetwarzanie grupy plików, aby wyniki pokazywały ewolucję systemu.

Interfejs jest łatwy i intuicyjny w użyciu. Położyłem spory nacisk na wydajność aplikacji. Użytkownik widzi niemal natychmiast rezultat każdej operacji, dlatego wybór technologii miał spore znaczenie. Wybrałem język C# działający w ramach technologii .Net. W tym przypadku

daje on optymalne rozwiązanie pomiędzy sprawnością programowania, a szybkością działania. Jako środowisko programistyczne, ze względu na najlepsze wsparcie, wybrałem Visual Studio 2010 Ultimate.

W Pierwszym etapie pracy skupiłem się na zebraniu wymagań projektowych. Temat złożonych stopów metalicznych (CMA) był dla mnie nowy, dlatego pomoc merytoryczna promotora okazała się niezwykle cenna. W kolejnym etapie stworzyłem odpowiednie prototypy, aby wybrać najlepszą z technologii. W ostatnim etapie zaprojektowałem i zaimplementowałem aplikację, którą następnie przetestowałem.

Cel pracy został osiągnięty. Aplikacja do wizualizacji procesów zachodzących w złożonych stopach metalicznych jest w pełni funkcjonalna zgodnie ze specyfikacją. Interfejs graficzny jest prosty w obsłudze. Program z powodzeniem może być zastosowany do innych celów, nawet bez ingerencji w kod.

A. Instrukcja obsługi

A.1. Pobieranie aplikacji

Aplikacje należy pobrać ze strony alloysvisualisation.codeplex.com (po prawej stronie znajduje się przycisk Download). W skompresowanym pliku znajduje się:

1. Plik wykonywalny ComplexAlloysVisualisation.exe
2. skrypt chmcMergeScript.py
3. biblioteka WPFEextensions.dll
4. folder out

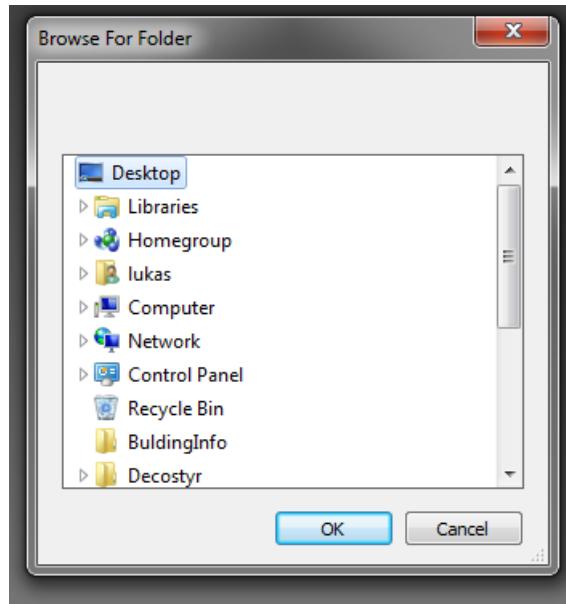
Plik ComplexAlloysVisualisation.exe startuje aplikację.

A.2. Tworzenie modelu i manipulacja

Po uruchomieniu aplikacji należy nacisnąć przycisk Open File i wybrać plik .chmc. W folderze out znajduje się kilka przykładowych plików. Aplikacja domyślnie stworzy model z serii 14. Aby zmienić serie, należy nacisnąć przycisk Mask i w panelu po prawej stronie wybrać interesujące nas serie. W rozdziale 4.3 Graficzny Interfejs Użytkownika znajduje się dokładny opis każdej funkcjonalności.

A.3. Przetwarzanie grupy plików

W celu przetworzenia grupy plików, należy uruchomić aplikację i nacisnąć przycisk Process Files. Po naciśnięciu pojawi się okno, gdzie należy wybrać folder z plikami .chmc.



Rysunek A.1: Okno wyboru folderu.

Przetwarzanie plików przy kilku seriach powinno trwać około 0,6 s dla każdego pliku. Wyniki zapisane zostaną w folderze screens.

Spis rysunków

2.1	Układy krystalograficzne dla stopów $Au - Cu$, $AuCu$ i Au_3Cu	13
2.2	Przekrój komórki elementarnej β - Mg_2Al_3 . Płaszczyzna prostopadła do kierunku [1,1,0].	14
2.3	Przekrój komórki elementarnej β - Mg_2Al_3 . Płaszczyzna prostopadła do kierunku [1,1,1].	15
2.4	Diagram fazowy stopu β - Mg_2Al_3	16
3.1	Schemat modelu kaskadowego.	21
3.2	Schemat modelu spiralnego. Źródło:[3].	22
3.3	Schemat modelu iteracyjnego.	23
3.4	Zrzut ekranu pierwszego prototypu (XNA).	29
3.5	Zrzut ekranu drugiego prototypu (WPF).	30
4.1	Struktura widoku - podział na kontenery.	32
4.2	Struktura projektu Complex Alloys Visualisation.	33
4.3	Rodzaje separatorów dziesiętnych na świecie. Zielony: przecinek. Niebieski: kropka. Czerwony: momayyez. Źródło: Wikipedia na licencji Creative Commons.	36
4.4	Nieprawidłowa pozycja kamery.	37
4.5	Materiał używający Z-buffer.	38
4.6	GUI programu do wizualizacji CMA.	41
4.7	Kontrolka skalowania modelu.	42
4.8	Lewy panel GUI.	43
4.9	Maski serii atomów.	44
4.10	Zbiór testowy 20 atomów.	47
5.1	Seria pozycji nr 3, 14 i 19 atomów β - Mg_2Al_3 przy temperaturze 2000K.	49
5.2	Seria pozycji 3, 14 i 19 atomów β - Mg_2Al_3 przy temperaturze 400K.	49
5.3	Stan komórki elementarnej dla temperatury 2000K.	50
5.4	Stan komórki elementarnej dla temperatury 1500K.	50

5.5	Stan komórki elementarnej dla temperatury 900K.	51
5.6	Stan komórki elementarnej dla temperatury 850K.	51
5.7	Stan komórki elementarnej dla temperatury 650K.	51
5.8	Stan komórki elementarnej dla temperatury 400K.	51
5.9	Stan komórki elementarnej dla temperatury 2000K.	52
5.10	Stan komórki elementarnej dla temperatury 1500K.	52
5.11	Stan komórki elementarnej dla temperatury 900K.	53
5.12	Stan komórki elementarnej dla temperatury 850K.	53
5.13	Stan komórki elementarnej dla temperatury 650K.	53
5.14	Stan komórki elementarnej dla temperatury 400K.	53
A.1	Okno wyboru folderu.	57

Spis tabelic

4.1	Zależność prawdopodobieństwa i koloru.	39
4.2	Zależności miedzy kosztem naprawy błędu a etapem wystąpienia.	45

Bibliografia

- [1] Mark Aked. *RUP in brief*. IBM, 2011.
- [2] Scott Wiltamuth Peter Golde Anders Hejlsberg, Mads Torgersen. *Język C# Programowanie Wydanie III*. Helion, 2010.
- [3] Boehm B. A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(4):14–25, 1986.
- [4] K Beck. *Test-Driven Development by Example*. Addison Wesley, 2003.
- [5] Kent Beck. Manifesto for agile software development. *Agile Alliance*, 2001. Retrieved 2010.
- [6] Victor Basili Craig Larman. *Iterative and Incremental Development: A Brief History*. IEEE Computer, 2003.
- [7] J-M. Dubois and E. Berlin-Ferre. *Complex Metallic Alloys: Fundamentals and Applications*. WILEY-VCH, Weinheim, 2011.
- [8] Ivan Edward. *ketchpad: A Man-Machine Graphical Communication System*. University of Cambridge, 2003.
- [9] Ralph Johnson John Vlissides Erich Gamma, Richard Helm. *Inżynieria oprogramowania: Wzorce projektowe*. WNT, 2008.
- [10] Thomas C. Feuerbacher, M. The samson phase, b-mg2al3, revisited. *Z. Kristallogr.*, 222(6):259–288, 2007.
- [11] Thomas C. Makongo J. P. A. Hoffmann S. Carrillo-Cabrera W. Cardoso R. Grin Y. Kreiner G. Joubert J.-M. Schenk T. Gastaldi J. Nguyen-Thi H. Mangelinck-Noel N. Billia B. Donnadieu P. Czyska-Filemonowicz A. Zielinska-Lipiec A. Dubiel B. Weber T. Schaub P. Krauss G. Gramlich V. Christense J. Lidin S. Fredrickson D. Mihalkovic M. Sikora W. Malinowski J. Bruhne S. Proffen T. Assmus W. Boissieu M. d. Bley F. Chemin J.-L. Schreuer J. Steuer W. Feuerbacher, M. *Z. Kristallogr.*, 222(6):259–288, 2007.

- [12] Martin Fowler. *Architektura systemów zarządzania przedsiębiorstwem. Wzorce projektowe*. Helion, 2005.
- [13] Malcolm Gladwell. *Creation Myth: Xerox PARC, Apple, and the truth about innovation*. The New Yorker, 2011.
- [14] https://www.microsoft.com/en-us/news/press/2004/mar04/03_24xnalaunchpr.aspx. Microsoft: Next generation of games starts with xna. 2004.
- [15] ISO/IEC. Systems and software engineering - software life cycle processes. (12207), 2008.
- [16] Jack Hoxley. Jason Zink, Matt Pettineo. *Practical Rendering and Computation with Direct3D 11*. CRC Press, 2011.
- [17] Komelj M. Klanjsek M. Tkalec U. Vrtnik S. Feuerbacher-M. Dolinsek J. Jeglic, P. Phys. rev. b. 75(1):14–202, 2007.
- [18] Cem Kaner. Exploratory testing,. *Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference*, 2006.
- [19] C. D.; Vecchi M. P. Kirkpatrick, S.; Gelatt. Optimization by simulated annealing. *Science*, 4598(220):671–680, 1983.
- [20] Craig Larman. *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley, 2004.
- [21] Dean Leffingwell. *Scaling Software Agility: Best Practices for Large Enterprises*. Addison-Wesley, 2007.
- [22] D. J. Tildesley M. P. Allen. *Computer simulation of liquids*. Oxford University Pres, 1989.
- [23] Matthew MacDonald. *Pro WPF in C#*. Apress, 2010.
- [24] Kurt Akeley Mark Segal. The opengl graphics system: A specification version 4.0 (core profile). 2010.
- [25] Steve McConnell. *Code Complete*. Microsoft Press, 2004.
- [26] Office of Naval Research. Symposium on advanced programming methods for digital computers. *Navy Mathematical Computing Advisory Panel.*, OCLC(10794738), 1956.
- [27] H. Perlitz. *Nature*, 154(606), 1944.
- [28] Linux Information Project. *GUI definition*. 2008.

- [29] NIST report. *Software errors cost U.S. economy \$59.5 billion annually.*
- [30] Markus Rerych. Wasserfallmodell - entstehungskontext. *Institut für Gestaltungs und Wirkungsforschung*, 2007.
- [31] K. Riederer. *Z. Metalk.*, 28(312), 1936.
- [32] Eric Ries. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Publishing, 2011.
- [33] Patton Ron. *Software Testing*. Sams, 2005.
- [34] Andrew Stellman and Jennifer Greene. *Applied Software Project Management*. O'Reilly Media, 2005.
- [35] Charles E. Leiserson Thomas H. Cormen and Ronald L. Rives. *Introduction to Algorithm*. MIT Press, 2002.
- [36] http://sig3.ecanews.org/isac2010/lectures/18_mihalkovic_simulation.pdf.
- [37] http://spaceflight.esa.int/impress/text/education/Solidification/Intermetallics_Immiscibles.html.
- [38] Karl E. Wiegers. *Software Requirements: Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle, Second Edition*. Microsoft Press, 2003.