

Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Paweł Tomaszewski

Nr albumu: 292647

**Interaktywna eksploracja i
dopasowanie lokalnie optymalnych
struktur biopolimerów z
wykorzystaniem metod wirtualnej
rzeczywistości**

**Praca licencjacka na kierunku
BIOINFORMATYKA I BIOLOGIA SYSTEMÓW**

Praca wykonana pod kierunkiem
dr. Pawła Daniluka
Instytut Fizyki Doświadczalnej
Zakład Biofizyki

Maj 2017

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data

Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data

Podpis autora pracy

Streszczenie

W ramach niniejszej pracy została przedstawiona implementacja aplikacji narzędziowej z pogranicza biofizyki molekularnej i wirtualnej rzeczywistości. Program służący do interaktywnej eksploracji struktur biopolimerów w poszukiwaniu regionów optymalnych ze względu na jakość lokalnego dopasowania strukturalnego pomiędzy wzorcem (*ang. template structure*) i wybranymi regionami cząsteczki celu (*ang. target structure*). Jakość znalezionej dopasowania odwzorowuje zwrotna projekcja momentów sił w urządzeniu haptycznym.

Program został zrealizowany w formie wtyczki rozszerzającej możliwości pakietu PyMOL. Do swojego działania wykorzystuje urządzenia i metody wirtualnej rzeczywistości - urządzenie haptyczne Phantom Omni oraz bibliotekę Virtual Reality Peripheral Network - a także bioinformatyczne algorytmy wyszukujące optymalne lokalne dopasowania strukturalne (*ang. local structure alignment*), algorytm maksymalizujący stopień dopasowania struktur i minimalizujący wartość RMSD (algorytm Kabsch'a).

W pracy zostały także przedstawione podstawy teoretyczne leżące u jej podstaw, szczegółowe opisy wykorzystanego urządzenia i metod wirtualnej rzeczywistości, a także perspektywy dalszego rozwoju i wykorzystania praktycznego opracowanego oprogramowania.

Kod źródłowy stworzonego oprogramowania stanowi integralny załącznik do niniejszej pracy.

Słowa kluczowe

bioinformatyka, wirtualna rzeczywistość, podobieństwo strukturalne, RNA, DNA, białka, biopolimery, dopasowanie strukturalne

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka, nauki komputerowe

13.1 Biologia

13.2 Fizyka

Klasyfikacja tematyczna

D. Software

D.127. Blabalgorithms

D.127.6. Numerical blabalalysis

Spis treści

Wprowadzenie	5
1. Podstawy teoretyczne	7
1.1. Metody dopasowania struktur chemicznych	7
1.1.1. Globalne dopasowanie struktur	8
1.1.2. Lokalne dopasowanie struktur	8
1.2. RMSD i algorytm Kabsch’a	9
1.3. Przekształcenia geometryczne i ich reprezentacje	12
1.3.1. Skalowanie	13
1.3.2. Translacje	14
1.3.3. Rotacje i kwaterniony	15
1.4. Pole siłowe	17
2. Urządzenie haptyczne Sensable Phantom Omni	19
2.1. Opis urządzenia	20
2.2. Wymagania sprzętowe	20
2.3. OpenHaptics Toolkit v3.0	21
2.3.1. Phantom Device Drivers	21
2.3.2. Haptic Device API	21
2.3.3. Haptic Library API	22
2.3.4. QuickHaptics Micro API	23
2.4. Servo Loop	24
3. Virtual Reality Peripheral Network	25
3.1. Opis pakietu	25
3.2. Opis użycia VRPN	26
4. Implementacja i uruchomienie oprogramowania	27
4.1. Opis stanowiska laboratoryjnego	27
4.2. Rozszerzanie funkcjonalności pakietu PyMOL	28
4.3. Dane wejściowe	28
4.4. Opis oprogramowania	29
5. Podsumowanie	37
Bibliografia	39
Kod źródłowy	41

Wprowadzenie

Biopolimery, a w szczególności kwasy nukleinowe i białka są podstawowymi elementami życia. W każdej komórce oraz całym organizmie są odpowiedzialne za funkcje związane z odżywianiem, rozmnażaniem czy obroną przed patogenami. Dzięki kwasom nukleinowym możliwe jest przenoszenie informacji genetycznej. Biorą one udział w syntezie białek, a także mają funkcję enzymatyczne. Białka są zaś składnikami budulcowymi wielu organów, pełnią funkcje hormonalne i regulatorowe.

Pomimo odmiennej budowy tych dwóch klas cząsteczek w obu przypadkach ich funkcja wynika bezpośrednio ze struktury przestrzennej, która zgodnie z hipotezą Anfinsena [1] jest ściśle zdeterminowana przez sekwencję nukleotydów czy aminokwasów.

Struktura przestrzenna biopolimerów stanowi obecnie przedmiot niezwykle intensywnych badań na całym świecie. Jest to bardzo ważne zagadnienie integrujące ze sobą grupy naukowców z dziedzin, które jeszcze na przełomie wieków miały ze sobą niewiele wspólnego. Do grona biologów i chemików dołączyli matematycy, fizycy oraz informatycy wspólnie tworząc nowatorskie narzędzia ułatwiające modelowanie i tworzenie symulacji zachodzących w skali molekularnej. W badania nad strukturami przestrzennymi biopolimerów zaangażowane są największe na świecie ośrodki naukowe, korporacje farmaceutyczne czy agencje rządowe. Badanie interakcji receptorów z ligandami, projektowanie nowych leków czy terapie celowane to tylko wąski wycinek zagadnień związanych z tymi pracami.

Intensywny rozwój narzędzi bioinformatycznych znacznie ułatwił i przyspieszył te badania. Przy pomocy superkomputerów prowadzi się obliczenia struktur natywnych białek, kwasów nukleinowych czy prowadzi symulacje dynamiki molekularnej. Jest to dziedzina, w której niemalże każdego roku dokonuje się znaczących odkryć i prawdopodobnie jeszcze przez długi czas to się nie zmieni. Wystarczy przytoczyć wyniki odbywającego się co dwa lata międzynarodowego eksperymentu CASP polegającego na przewidywaniu struktur białek, który za każdym razem przynosi wyniki coraz bardziej zbliżone z konformacjami natywnymi [2].

Ważnym aspektem tych badań jest prezentacja wyników szerokiemu gronu odbiorców. Same efektowne wizualizacje nie zawsze są wystarczające, coraz częściej chcemy wchodzić w bezpośrednią interakcję ze światem do którego nie mieliśmy nigdy wcześniej dostępu. W związku z tym w ostatnim czasie coraz częściej sięga się do rozwiązań z zakresu wirtualnej rzeczywistości.

Mianem *wirtualnej rzeczywistości* (*ang. virtual reality, VR*) określamy sztuczne, wykreowane przy pomocy technologii informatycznych, multimedialne projekcje przestrzeni, przedmiotów lub zdarzeń. Na obecnym poziomie rozwoju technologii wirtualna rzeczywistość umożliwia człowiekowi wchodzenie w interakcję z tym środowiskiem przede wszystkim za pośrednictwem zmysłów wzroku, słuchu czy dotyku wykorzystując urządzenia klasy HCI (*ang. human computer interface*).

Wirtualna rzeczywistość w dzisiejszym świecie zyskuje coraz większą popularność w wielu dziedzinach życia, od zastosowań czysto rozrywkowych po zaawansowane projekty naukowe, przemysłowe, a także wojskowe. Postępująca od wielu lat miniaturyzacja, rozwój nowych algo-

rytmów czy drastyczne zwiększenie wydajności obliczeniowej sprzętu komputerowego jedynie przyspiesza ten proces.

Elementami niezbędnymi do prawidłowego wykreowania wirtualnego środowiska jest zarówno dedykowane oprogramowanie jak i sprzęt konieczny do przekazywania informacji zwrotnych do użytkownika. Rzeczywistość wirtualna, aby zostać możliwie najlepiej zinterpretowana przez ludzki mózg musi jak najbardziej przypominać rzeczywistość, w której żyjemy na co dzień. Aby sprostać temu zadaniu, najczęściej reprezentuje się ją w postaci trójwymiarowych scen. Już tylko ten jeden czynnik powoduje, że do poprawnej symulacji niezbędne są nowoczesne, wysokowydajne procesory i karty graficzne będące w stanie przeprowadzić niezbędne obliczenia.

Jak już wspomniano wirtualna rzeczywistość może znajdować zastosowanie także w nauce w szczególności w dziedzinach, w których obiekty zainteresowań są zbyt małe, aby być widoczne gołym okiem, takie jak cząsteczki chemiczne lub pojedyncze atomy. Istnieje cały szereg programów przeprowadzających np. symulacje oddziaływań międzycząsteczkowych, zwijania białek (*ang. protein folding*) czy przeprowadzających obliczenia dynamiki molekularnej (*ang. molecular dynamics*), a także umożliwiających wizualizację tych symulacji. Stosunkowo niewiele jednak jest dedykowanych rozwiązań wirtualnej rzeczywistości, które mogłyby umożliwić interakcję z użytkownikiem za pośrednictwem zmysłu dotyku.

Pracownie Laboratorium Biofizyki na Wydziale Fizyki Uniwersytetu Warszawskiego dysponują sprzętem niezbędnym do realizacji takich zadań. Urządzenie haptyczne Sensable Phantom Omni jest przykładem trójwymiarowego wskaźnika ze zwrotną projekcją momentów sił, którego wykorzystanie otwiera całe spektrum nowych możliwości związanych z realizacją projektów wirtualnej rzeczywistości w biofizyce, biologii molekularnej czy chemii [3].

Rozdział 1

Podstawy teoretyczne

W tym rozdziale zaprezentowano teorię leżącą u podstaw niniejszej pracy. Skupiono się tutaj przede wszystkim na metodach dopasowania (uliniawiania) struktur biopolimerów, sposobach optymalnego nakładania struktur i oceny jego jakości (algorytm Kabsch’a i RMSD) oraz zagadnieniach związanych z przekształceniami geometrycznymi (skalowanie, translacje i rotacje).

1.1. Metody dopasowania struktur chemicznych

Celem poszukiwania optymalnych metod dopasowania strukturalnego (*ang. structural alignment*) jest znalezienie homologii pomiędzy cząsteczkami polimerów lub ich fragmentami jedynie na podstawie kształtu, bez znajomości sekwencji.

Metody dopasowań strukturalnych pierwotnie odnosiły się do cząsteczek polipeptydów i białek jako podstawowych biopolimerów. Szybko jednak ich zastosowanie zostało rozszerzone także na kwasy nukleinowe, w szczególności niekodujący RNA gdyż posiada on ważne funkcje biologiczne (np. enzymatyczne) oraz analogiczne do polipeptydów formy drugo- i trzeciorzędowe.

Z uwagi na znacznie wyższą ewolucyjną trwałość struktury przestrzennej w porównaniu do sekwencji (zarówno aminokwasowej jak i nukleotydowej) poszukiwanie dopasowań strukturalnych często bywa dużo skuteczniejszą metodą znajdowania związków ewolucyjnych pomiędzy organizmami niż klasyczne uliniawianie sekwencji.

Istotnym aspektem tych metod jest efektywna ocena jakości dopasowań. Niestety nie ma jednej uniwersalnej miary podobieństwa struktur. Istnieje jednak kilka dobrze opisanych algorytmów służących do ich szacowania. Najczęściej wykorzystywaną do tego celu metryką jest RMSD (*ang. root-mean-squared deviation*), szczegółowo opisana w dalszej części pracy.

Obliczenie dopasowania strukturalnego ponadto implikuje powstanie dopasowania sekwencyjnego pomiędzy przyrównywanymi merami w każdym z łańcuchów. Ocena takiego jednowymiarowego uliniowania sekwencji również może dać nam wiedzę o bliskości ewolucyjnej występującej pomiędzy strukturami.

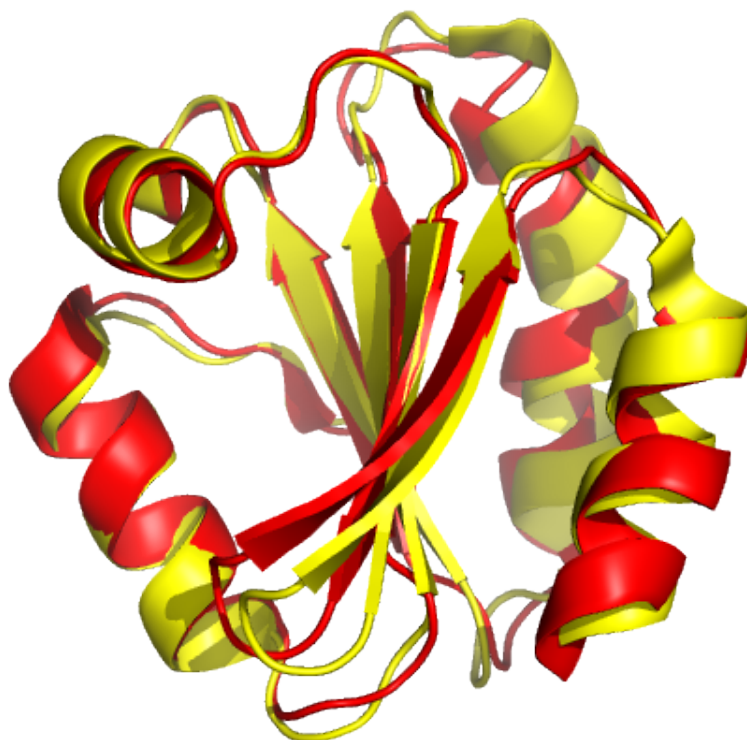
Z powodu dużej złożoności biopolimerów zostało opracowanych wiele metod upraszczających ich reprezentacje do celów obliczeniowych. Przede wszystkim dąży się do rezygnacji z bezpośredniego rozpatrywania lokalizacji wszystkich atomów na rzecz jedynie tych należących do szkieletu (*ang. backbone*) cząsteczki (np. α -węgle aminokwasów czy pentozy kwasów nukleinowych), z pominięciem lub daleko idącym ograniczeniem roli łańcuchów bocznych.

Metody dopasowania możemy podzielić na globalne, których celem jest porównywanie całych struktur trzeciorzędowych oraz lokalne polegające na poszukiwaniu najlepszego dopa-

sowania fragmentu cząsteczki wzorca (np. struktury drugorzędowej) do wybranego regionu (miejsca w obrębie cząsteczki celu).

1.1.1. Globalne dopasowanie struktur

Globalne dopasowanie polega na obliczaniu najlepszego uliniowienia dwóch lub więcej struktur. Jest ono najbardziej użyteczne, gdy podobieństwo porównywanych struktur jest wysokie. Jest to często stosowana metoda służąca do porównywania różnych konformacji tego samego polimeru, na przykład do oceny jakości algorytmów przewidujących strukturę białek lub porównywanie struktur analogicznych białek pochodzących z różnych organizmów.



Rysunek 1.1: Superpozycja dwóch struktur tioredoksyny (białko o długości 105 aminokwasów) ludzkiej (kolor czerwony) i pochodzącej z muszki owocowej *Drosophila melanogaster* (kolor żółty), RMSD=1.244Å

Wadą podejścia globalnego jest to, że jego wykorzystanie dla zróżnicowanych cząsteczek może być bezcelowe z powodu niemożności obliczenia miarodajnej wartości podobieństwa pomiędzy takimi strukturami. Innymi słowy obliczona wartość RMSD pomiędzy takimi cząsteczkami może być pozbawiona sensownej interpretacji. Wówczas należy rozważyć podejście lokalne.

1.1.2. Lokalne dopasowanie struktur

Jak już wspomniano, lokalne dopasowanie polega na poszukiwaniu w obrębie cząsteczki celu (*ang. target*) regionów, które „są podobne” do wybranych struktur zwanych wzorcami lub szablonami (*ang. template*). Przez podobieństwo należy tutaj rozumieć taką odpowiedniość pomiędzy ww. strukturami, że wyznaczona dla ich superpozycji wartość RMSD nie przekracza

zadanego progu. Wzorcami mogą być na przykład struktury drugorzędowe, miejsca wiążące, całe domeny białkowe lub inne wybrane fragmenty cząsteczek.

Istnieje wiele opracowanych metod i algorytmów realizujących takie obliczenia. Do najpopularniejszych należy zaliczyć DALI[4], SSAP[5], CE[6], VAST[7], MATRAS[8], GANGSTA[9] i inne. Każda z nich ma inne podejście do dekompozycji struktury i sposobów poszukiwania lokalnych podobieństw.

Szczególną uwagę musimy jednak zwrócić na metodę lokalnych deskryptorów, zaproponowaną przez Krzysztofa Fidelisa [10][11] (jednego z twórców i organizatorów eksperymentu CASP). Stanowi ona podstawowy algorytm generujący dane wejściowe dla oprogramowania będącego przedmiotem niniejszej pracy. Metoda lokalnych deskryptorów w przeciwieństwie do standardowych metod korzystających z ciągłych segmentów, bazuje na lokalnym przestrzennym otoczeniu aminokwasu. Metoda szczegółowo została opisana w cytowanych wyżej publikacjach.

Wynikiem przeprowadzonych uliniowień i superpozycji struktur jest indeks miejsc (wraz z ich miarą dopasowania) analizowanej struktury, do których wzorzec jest podobny. Należy pamiętać, że jeden wzorzec może pasować do wielu regionów w obrębie tej samej cząsteczki, przez co może występować w wielu miejscach indeksu mapującego. Innymi słowy dzięki takiej operacji uzyskujemy kompletną mapę struktury celu z wyróżnionymi regionami, których superpozycja z wzorcem daje stopień podobieństwa nie gorszy od zadanego.

1.2. RMSD i algorytm Kabsch’a

Efektywna ocena podobieństwa strukturalnego jest jednym z kluczowych elementów procesu dopasowania. W bioinformatyce istnieje kilka sposobów oszacowania tej wartości: obok GDT (*ang. global distance test*) i TM-score (*ang. template modeling score*) [13] najpopularniejsza i stosunkowo prosta w zastosowaniu jest miara odchylenia średniokwadratowego - RMSD (*ang. root-mean-squared deviation*) [12].

Ocena podobieństwa metodą RMSD polega na obliczeniu średniokwadratowej odległości pomiędzy współrzędnymi odpowiadających sobie atomów (lub innych punktów charakterystycznych) zawartych w strukturze wzorca (*ang. template*) i cząsteczce celu (*ang. target*). RMSD wyraża się wzorem:

$$\text{RMSD}(p, q) = \sqrt{\frac{1}{N} \sum_{i=1}^N \|p_i - q_i\|^2}$$

gdzie:

p - wektor współrzędnych struktury wzorca

q - wektor współrzędnych wybranego regionu w strukturze celu

N - długość wektorów współrzędnych p i q

jednostką najczęściej jest Å(angstrom)

O ile same obliczenia są trywialne, to dobór danych wejściowych do algorytmu RMSD może stanowić poważne wyzwanie. Tutaj pochodzą one z zewnętrznej aplikacji i są wynikiem przeprowadzonej wcześniej procedury lokalnego dopasowania wzorca do struktury celu.

W 1976 roku w swojej pracy [14][15] Wolfgang Kabsch opisał algorytm wyznaczający macierze transformacji (rotacji i translacji) optymalizujące nałożenie struktur przestrzennych. Algorytm ten jest powszechnie wykorzystywany do minimalizacji wartości RMSD.

Algorytm Kabsch'a startuje z dwoma wektorami współrzędnych p i q o długości N :

$$p = \begin{pmatrix} x_{p1} & y_{p1} & z_{p1} \\ x_{p2} & y_{p2} & z_{p2} \\ \vdots & \vdots & \vdots \\ x_{pN} & y_{pN} & z_{pN} \end{pmatrix}$$

$$q = \begin{pmatrix} x_{q1} & y_{q1} & z_{q1} \\ x_{q2} & y_{q2} & z_{q2} \\ \vdots & \vdots & \vdots \\ x_{qN} & y_{qN} & z_{qN} \end{pmatrix}$$

gdzie:

p wektor punktów struktury wzorcowej o długości N

q wektor punktów wybranego regionu ze struktury celu o długości N

Algorytm Kabsch'a składa się z 3 kroków (<http://cnx.org/contents/HV-RsdwL@23/Molecular-Distance-Measures>):

1. Translacja

W pierwszym kroku należy dokonać obliczenia centroidów (Cp i Cq) obu struktur i dokonać przesunięcia struktury wzorca o wektor \vec{T} rozpięty pomiędzy tymi punktami tak, aby nałożyły się one na siebie. Do obliczenia centroidu można posłużyć się wzorem na średnią arytmetyczną wartości poszczególnych współrzędnych:

$$Cp = (Cp_x, Cp_y, Cp_z)$$

gdzie:

$$Cp_x = \frac{1}{N} \sum_{i=1}^N x_{pi}$$

$$Cp_y = \frac{1}{N} \sum_{i=1}^N y_{pi}$$

$$Cp_z = \frac{1}{N} \sum_{i=1}^N z_{pi}$$

oraz

$$Cq = (Cq_x, Cq_y, Cq_z)$$

gdzie:

$$Cq_x = \frac{1}{N} \sum_{i=1}^N x_{qi}$$

$$Cq_y = \frac{1}{N} \sum_{i=1}^N y_{qi}$$

$$Cq_z = \frac{1}{N} \sum_{i=1}^N z_{qi}$$

zatem wektor translacji \vec{T} taki, że:

$$\vec{T} = |Cp - Cq| = (|Cp_x - Cq_x|, |Cp_y - Cq_y|, |Cp_z - Cq_z|)$$

możemy użyć do wykonania przesunięcia wszystkich punktów w p i q :

$$p'_i = p_i + \vec{T}$$

gdzie p_i jest konkretnym punktem w p .

2. Macierz kowariancji

Po wykonanym przesunięciu należy obliczyć zależność liniową pomiędzy współrzędnymi wektorów p' i q . Poprawne wyliczenie macierzy kowariancji A będzie także stanowiło podstawę do ustalenia optymalnej macierzy rotacji (w kolejnym kroku):

$$A = \text{cov}(p', q)$$

lub równoważnie w zapisie macierzowym:

$$A = p'q^T$$

3. Optymalna macierz rotacji

Optymalna macierz rotacji powstaje z *dekompozycji według wartości szczególnych* macierzy A . SVD (*ang. singular value decomposition*) to taki rozkład zadanej macierzy na trzy specyficzne macierze U , Σ oraz V , że zachodzi zależność:

$$A = U\Sigma V^T$$

gdzie:

U i V to macierze ortogonalne (takie, że $U^{-1} = U^T$ oraz $V^{-1} = V^T$)

Σ macierz diagonalna (taka, że na przekątnej mamy nieujemne liczby rzeczywiste, będące wartościami szczególnymi macierzy A)

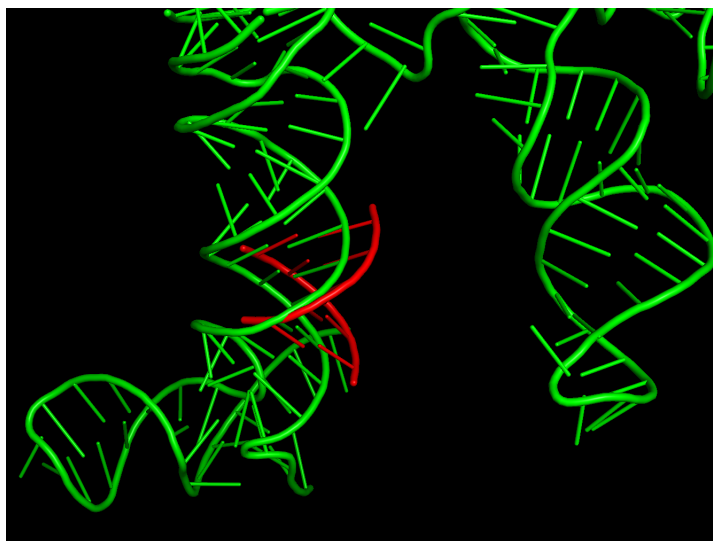
Sprawdzając znak wartość wyznacznika:

$$s = \text{sign}(\det(VU^T))$$

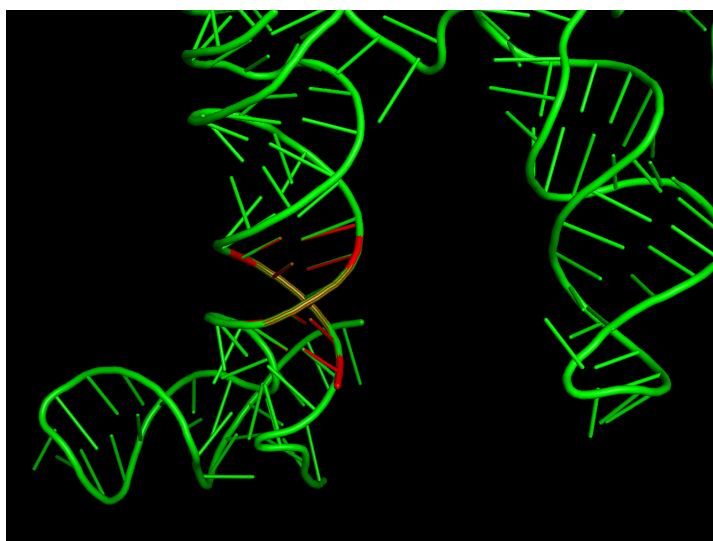
upewniamy się, że operujemy w ramach prawoskrętnego układu współrzędnych. Ostatecznie uzyskujemy optymalną macierz rotacji R :

$$R = V \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix} U^T$$

Zastosowanie translacji i rotacji wyznaczonych zgodnie z algorytmem Kabsch'a prezentują poniższe wizualizacje. Prezentują one różne wartości RMSD dla superpozycji tych samych struktur przed i po zastosowaniu algorytmu:



Rysunek 1.2: Przykładowe dopasowanie struktury wzorca (czerwona) do podobnego regionu struktury celu (zielona), RMSD=3.985



Rysunek 1.3: Przykładowe dopasowanie struktury wzorca (czerwona) do podobnego regionu struktury celu (zielona), RMSD=0

1.3. Przekształcenia geometryczne i ich reprezentacje

Przekształcenia geometryczne w przestrzeni trójwymiarowej są elementarnymi operacjami używanymi w niniejszej pracy [16]. Zaliczamy do nich przede wszystkim skalowanie, translacje i rotacje. Wszystkie te operacje można praktycznie zrealizować na kilka sposobów. Ze względu na to, że mogą operować na milionach punktów w przestrzeni trójwymiarowej (pikseli), optymalne metody numeryczne które je realizują są niezwykle ważne i muszą w maksymalnym stopniu wykorzystywać możliwości oferowane przez komputerowe jednostki obliczeniowe i procesory graficzne.

Współpraca bibliotek programistycznych pochodzących od różnych dostawców może wymagać wielu przekształceń pomiędzy różnymi formatami opisu transformacji. Szczególną uwagę należy zwrócić na konwersję orientacji wskaźnika urządzenia haptycznego z kwaternionów - postaci dostarczanej przez bibliotekę VRPN - na oś i kąt obrotu (*ang. axis-angle rotation*) - akceptowany przez PyMOL - i odwrotnie.

Z uwagi na konieczność dostosowania transformacji do wykonywania na współczesnych komputerach niezbędne było opracowanie efektywnych obliczeniowo metod numerycznych i reprezentacji tych operacji. Bezpośrednia ich implementacja byłaby dość kłopotliwa gdyż nie dość, że każda transformacja musiałaby zostać wykonana oddzielnie (sekwencyjnie), to przede wszystkim nie byłoby to optymalne obliczeniowo rozwiązanie. Do tego celu wybrano ujednolicone narzędzia rozwiązujące powyższe problemy: elementarne macierze transformacji i współrzędne jednorodne.

Elementarne macierze transformacji stanowią łatwe narzędzie matematyczne do operowania przekształceniami geometrycznymi. Każda z transformacji może zostać opisana jako prosta operacja macierzowa (dodawanie albo mnożenie) modyfikująca zbiór punktów w przestrzeni.

Często pojawia się sytuacja w której kilka przekształceń geometrycznych chcemy wykonać jednocześnie. Można sobie wyobrazić przypadek w którym obiekt chcemy jednocześnie przesunąć i dokonać jego obrotu. Niestety stosowanie „surowych” elementarnych macierzy transformacji nie umożliwia wykonania takich operacji w tym samym czasie, dopiero współrzędne jednorodne pozwoliły w pełni wyeliminować ten problem.

Współrzędne jednorodne są sposobem reprezentacji punktów przestrzeni n -wymiarowej za pomocą układu $n + 1$ współrzędnych. Zostały one wprowadzone przez niemieckiego matematyka Augusta Möbiusa w 1827 roku i opisane w jego pracy [17].

Współrzędne jednorodne to narzędzie matematyczne stanowiące pewne udoskonalenie elementarnych macierzy transformacji. Umożliwiają wykonywanie wielu przekształceń jednocześnie, za pomocą jednej operacji macierzowej - mnożenia. Współrzędne jednorodne dają możliwość skumulowania wszystkich transformacji jakie chcemy wykonać na strukturze przestrzennej w jednej odpowiednio zbudowanej macierzy. W dzisiejszych czasach zostały one docenione w wielu dziedzinach nie tylko bezpośrednio związanych z grafiką komputerową ale także w robotyce i biofizyce.

Poniżej szczegółowo opisane zostały wykorzystane w pracy transformacje: skalowanie, translacja oraz rotacja. Dla każdej z nich podane zostały ich reprezentacje za pomocą elementarnych macierzy transformacji oraz współrzędnych jednorodnych. Dodatkowo dla rotacji została przedstawiona reprezentacja w kwaternionach.

1.3.1. Skalowanie

Skalowanie to operacja polegająca na mnożeniu współrzędnych obiektu określonego w przestrzeni trójwymiarowej przez współczynniki skalowania. Współczynniki skalowania są dodatnią liczbą rzeczywistą albo dodatnio określonym wektorem liczb rzeczywistych. W ogólności możemy skalować wszystkie składowe wektora współrzędnych niezależnie przez stosowanie wektora o różnych współczynnikach, jednak dla potrzeb niniejszej pracy ograniczymy się do mnożenia wektora współrzędnych wyłącznie przez skalar.

Geometryczna intuicja stojąca za operacją skalowania polega na takim przekształceniu współrzędnych danego obiektu, że w zależności od wartości współczynnika skalującego S może on zostać:

- niezmienniony, gdy $S = 1$,
- pomniejszony proporcjonalnie do S , gdy $0 < S < 1$,
- powiększony proporcjonalnie do S , gdy $S > 1$.

Współrzędne dowolnego punktu $P = (x, y, z)$ lub $\vec{P} = [x, y, z]^T$ po operacji skalowania o współczynnik S są równe $P' = (x', y', z')$, gdzie:

$$\begin{aligned}x' &= Sx \\y' &= Sy \\z' &= Sz\end{aligned}$$

Reprezentacja skalowania za pomocą elementarnych macierzy transformacji polega na stworzeniu takiej macierzy \hat{S} , że wartość współczynnika S zostanie wpisane do niej w następujący sposób:

$$\hat{S} = \begin{bmatrix} S & 0 & 0 \\ 0 & S & 0 \\ 0 & 0 & S \end{bmatrix}$$

zatem w zapisie macierzowym otrzymujemy:

$$\vec{P}' = \hat{S}\vec{P}$$

$$\vec{P}' = \begin{bmatrix} S & 0 & 0 \\ 0 & S & 0 \\ 0 & 0 & S \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} Sx \\ Sy \\ Sz \end{bmatrix}$$

Aby przedstawić operację skalowania we współrzędnych jednorodnych należy stworzyć nową macierz \bar{S} o rozmiarze o 1 większym niż macierz elementarna \hat{S} oraz nowy wektor $\dot{\vec{P}}$ o długości o 1 większej niż \vec{P} , taki że:

$$\bar{S} = \begin{bmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & S & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \dot{\vec{P}} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

zatem:

$$\begin{aligned}\dot{\vec{P}}' &= \bar{S}\dot{\vec{P}} \\ \dot{\vec{P}}' &= \begin{bmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & S & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} Sx \\ Sy \\ Sz \\ 1 \end{bmatrix}\end{aligned}$$

1.3.2. Translacje

Translacja jest elementarnym przekształceniem geometrycznym. Translacją nazywamy takie przekształcenie, które przesuwa każdy punkt zbioru określony w przestrzeni o dowolny wektor \vec{T} . W odróżnieniu od skalowania, w przypadku przesunięć w przestrzeni do wektora współrzędnych dodajemy wektor współczynników (liczb rzeczywistych). W niniejszej pracy operacja przesunięcia wykonywana jest przy każdej zmianie położenia wskaźnika urządzenia haptycznego oraz podczas wykonywania operacji nałożenia i dopasowania struktur.

Dla zadanego punktu $P = (x, y, z)$ lub $\vec{P} = [x, y, z]^T$ efektem operacji przesunięcia o wektor $\vec{T} = [t_x, t_y, t_z]^T$ są współrzędne $\vec{P}' = (x', y', z')$:

$$\begin{aligned}x' &= x + t_x \\y' &= y + t_y \\z' &= z + t_z\end{aligned}$$

Z powyższego jasno wynika, że translacje można reprezentować jako operację sumy dwóch wektorów:

$$\vec{P}' = \vec{P} + \vec{T}$$

$$\vec{P}' = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \end{bmatrix}$$

Translacje można przedstawić także za pomocą współrzędnych jednorodnych. Wymaga to stworzenia macierzy \bar{T} oraz wektora $\dot{\vec{P}}$, takich że:

$$\bar{T} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \dot{\vec{P}} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

zatem:

$$\dot{\vec{P}}' = \bar{T} \dot{\vec{P}}$$

$$\dot{\vec{P}}' = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \\ 1 \end{bmatrix}$$

1.3.3. Rotacje i kwaterniony

Trzecią elementarną transformacją jest rotacja, która w odróżnieniu od skalowania czy translacji jest operacją bardziej złożoną. Należy zwrócić uwagę, że kolejność wykonywania rotacji ma znaczenie ($R_x R_y \neq R_y R_x$), po drugie samych reprezentacji rotacji jest co najmniej kilka. Najpopularniejszym z nich są obroty o zadany kąt wokół jednej z osi układu współrzędnych lub arbitralnej osi obrotu (*ang. axis-angle*).

Współrzędne punktu $P' = (x', y', z')$ będącego wynikiem rotacji punktu $P = (x, y, z)$ wokół poszczególnych osi (OX , OY , OZ) układu współrzędnych o zadany kąt (odpowiednio α , β i γ) wyrażają się następująco:

rotacja P wokół osi OX o kąt α :

$$\begin{aligned}x' &= x \\y' &= y \cos \alpha - z \sin \alpha \\z' &= y \sin \alpha + z \cos \alpha\end{aligned}$$

rotacja P wokół osi OY o kąt β :

$$\begin{aligned}x' &= z \sin \beta + x \cos \beta \\y' &= y \\z' &= x \cos \beta - z \sin \beta\end{aligned}$$

rotacja P wokół osi OZ o kąt γ :

$$\begin{aligned}x' &= x \cos \gamma - y \sin \gamma \\y' &= x \sin \gamma + y \cos \gamma \\z' &= z\end{aligned}$$

Elementarne macierze przekształceń w tym przypadku wyglądają następująco:

$$\hat{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}$$

$$\hat{R}_y(\beta) = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix}$$

$$\hat{R}_z(\gamma) = \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

gdzie:

$\hat{R}_x(\alpha)$ obrót wokół osi OX o kąt α

$\hat{R}_y(\beta)$ obrót wokół osi OY o kąt β

$\hat{R}_z(\gamma)$ obrót wokół osi OZ o kąt γ

Jak wszystkie inne transformacje, rotacje również można przedstawić we współrzędnych jednorodnych. Odpowiednie macierze mają wówczas następującą postać:

$$\bar{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\bar{R}_y(\beta) = \begin{bmatrix} \cos \beta & 0 & \sin \beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \beta & 0 & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\bar{R}_z(\gamma) = \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 & 0 \\ \sin \gamma & \cos \gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotacje reprezentowane przez współrzędne jednorodne można dowolnie ze sobą łączyć poprzez mnożenie odpowiednich macierzy, należy jednak mieć na uwadze fakt (jak już wcześniej wspomniano), że rotacje nie są przemienne - kolejność wykonywanych operacji ma znaczenie.

Rotacje posiadają jeszcze jedną być może najważniejszą z punktu widzenia programisty reprezentację: kwaterniony [18], które są one strukturą algebraiczną stanowiącą rozszerzenie liczb zespolonych. Kwaterniony zostały wprowadzone przez Williama Hamiltona [19], który poszukiwał wygodnego sposobu opisu mechaniki w przestrzeni trójwymiarowej.

Kwaternion q to taka czwórka liczb rzeczywistych x, y, z i w spełniająca równanie:

$$q = w + xi + yj + zk$$

gdzie:

i, j, k - współczynniki urojone takie, że $i^2 = j^2 = k^2 = ijk = -1$

w - część skalarna kwaterniona

x, y, z - część wektorowa kwaterniona

Kwaterniony mogą stanowić alternatywną wobec współrzędnych jednorodnych lub macierzy transformacji formę opisu rotacji. Możemy z powodzeniem przeprowadzać konwersję pomiędzy różnymi reprezentacjami rotacji.

Dzięki kwaternionom można przedstawić rotację wokół arbitralnie wybranej osi (wektora) obrotu. Załóżmy, że chcemy stworzyć kwaternion q reprezentujący obrót o kąt α wokół wektora $\vec{v} = [v_x, v_y, v_z]$. Kwaternion taki tworzymy w następujący sposób:

$$\begin{aligned} w &= \cos \frac{\alpha}{2} \\ x &= v_x \sin \frac{\alpha}{2} \\ y &= v_y \sin \frac{\alpha}{2} \\ z &= v_z \sin \frac{\alpha}{2} \end{aligned}$$

wówczas:

$$q = \cos \frac{\alpha}{2} + v_x i \sin \frac{\alpha}{2} + v_y j \sin \frac{\alpha}{2} + v_z k \sin \frac{\alpha}{2}$$

Kwaterniony mogą reprezentować nie tylko rotację (operację obrotu bryły) ale także orientację (stan bryły). Biblioteka VRPN (wykorzystana w niniejszej pracy) wydaje informację o bieżącej orientacji (stanie) wskaźnika, natomiast pakiet PyMOL akceptuje jedynie rotacje (zmiany orientacji).

Prawidłowe wykonanie rotacji pomiędzy dwoma kolejnymi kwaternionami q_n i q_{n+1} niosącymi informację o orientacji (stanie) polega na zastosowaniu kwaterniona odwrotnego q_n^{-1} (powracającego strukturę do orientacji początkowej, zerowej), a następnie kwaterniona q_{n+1} . Kwaternion odwrotny jest zdefiniowany następująco:

$$q^{-1} = \frac{w - xi - yj - zk}{w^2 + x^2 + y^2 + z^2}$$

Taki ciąg operacji należy wykonywać cyklicznie w miarę odbierania kolejnych informacji o orientacji wskaźnika.

1.4. Pole siłowe

Biblioteka VRPN dostarcza kilku metod sterowania zwrotną projekcją sił (szerszy opis w dalszej części). W tej pracy wykorzystano tzw. metodę pola siłowego (*ang. force field*), która podobnie jak w pozostałych przypadkach (np. symulacja powierzchni lub brył) realizowana jest przez lokalną aproksymację. Jej wywołanie polega na dostarczeniu funkcji trzech parametrów: punktu zaczepienia (*ang. origin*), wektora siły oraz macierzy Jacobiego.

Punktem zaczepienia wektora siły jest w tym przypadku środek geometryczny wzorcowej struktury chemicznej obliczany jako średnia arytmetyczna wartości poszczególnych współrzędnych.

Wektor siły \vec{F} to wektor wskazujący kierunek i zwrot siły działającej na wskaźnik. W tym przypadku jest on rozpięty pomiędzy środkami geometrycznymi struktury wzorcowej,

a najbliższym „podobnym” regionem - wyznaczonym przez algorytm lokalnego dopasowania strukturalnego. Zwrot siły jest skierowany w stronę owego regionu. Do dalszych rozważań możemy traktować wektor \vec{F} jako iloczyn wektora jednostkowego (wersora) \vec{F} oraz skalarnej wartości siły f :

$$\vec{F} = \dot{F}f = [\vec{F}_x f, \vec{F}_y f, \vec{F}_z f]$$

Macierz Jacobiego $J_{\vec{F}}$ jest macierzą zbudowaną z pochodnych cząstkowych pierwszego rzędu funkcji definiującej pole siłowe:

$$J_{\vec{F}} = \begin{bmatrix} \frac{\delta \vec{F}_x}{\delta x} & \frac{\delta \vec{F}_y}{\delta x} & \frac{\delta \vec{F}_z}{\delta x} \\ \frac{\delta \vec{F}_x}{\delta y} & \frac{\delta \vec{F}_y}{\delta y} & \frac{\delta \vec{F}_z}{\delta y} \\ \frac{\delta \vec{F}_x}{\delta z} & \frac{\delta \vec{F}_y}{\delta z} & \frac{\delta \vec{F}_z}{\delta z} \end{bmatrix}$$

Tutaj różniczkowaniu poddawane są poszczególne składowe wektora siły \vec{F} , w wyniku czego macierz upraszcza się do postaci:

$$J_{\vec{F}} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & f \end{bmatrix}$$

Wyznaczenie powyższych wartości jest niezbędne do prawidłowego wysterowania momentów sił wskaźnika haptycznego za pośrednictwem biblioteki VRPN.

Rozdział 2

Urządzenie haptyczne Sensable Phantom Omni

Nadrzędnym celem wirtualnej rzeczywistości jest możliwie wierne odtworzenie świata rzeczywistego lub fikcyjnego w formie programu komputerowego czy multimedialnej prezentacji. Ponadto metody i narzędzia wirtualnej rzeczywistości umożliwiają użytkownikowi wchodzenie w czynną interakcję z tak wykreowanymi scenami. Obecny stan rozwoju techniki pozwala na tworzenie wydajnych urządzeń klasy HCI (*ang. human computer interface*) będących interfejsami pomiędzy światem realnym a wirtualnym. Przykładem takiego urządzenia jest Phantom Omni, opracowane przez firmę SensAble (obecnie 3D Systems).



Rysunek 2.1: Urządzenie haptyczne SensAble Phantom Omni

Niniejsza praca w znacznym stopniu opiera się na wykorzystaniu tych urządzeń do odtwarzania rzeczywistych ruchów ręki użytkownika w świecie wirtualnym. Pracownie Zakładu Biofizyki Wydziału Fizyki Uniwersytetu Warszawskiego są wyposażone w tego typu urządzenia, zostały one udostępnione autorowi celem realizacji niniejszej pracy.

2.1. Opis urządzenia

Urządzenie haptyczne (*ang. haptic device*) Sensable Phantom Omni jest trójwymiarowym wskaźnikiem o 6 stopniach swobody wskazywania pozycji i orientacji. Urządzenie ma także możliwość programowego sterowania zwrotną projekcją (*ang. force feedback*) sił w trzech stopniach swobody (sterowanie pozycją wskaźnika). Ponadto w urządzeniach dostępne są dwa przyciski monostabilne do dowolnego wykorzystania.

Phantom Omni jest urządzeniem znajdującym zastosowanie w różnego rodzaju aplikacjach działających na styku świata wirtualnego z rzeczywistym. Umożliwia ono użytkownikowi wchodzenie w interakcję z cyfrowym światem za pomocą zmysłu dotyku. Dzięki dostarczonym przez producenta sterownikom i bibliotece OpenHaptics możliwe jest tworzenie oprogramowania w pełni wykorzystującego możliwości oferowane przez urządzenie.

Specyfikację techniczną urządzenia prezentuje poniższa tabela [20]:

Wymiary pola roboczego	szerkość: 160 mm głębokość 70 mm wysokość 120 mm
Rozdzielczość	450 dpi (~ 0.055 mm)
Maksymalny moment obrotowy	3.3 Nm
Sztywność	oś X: 1.26 N/mm oś Y: 2.31 N/mm oś Z: 1.02 N/mm
Dane o pozycji i orientacji	osi X, Y i Z kąty α , β i γ (6 stopni swobody)
Zwrotna projekcja sił	osi X, Y i Z (3 stopnie swobody)
Interfejs	IEEE-1394a Ethernet

2.2. Wymagania sprzętowe

Urządzenia Sensable Phantom Omni w które wyposażone są Pracownie Biofizyki F UW posiadają dwa rodzaje interfejsów: Ethernet oraz FireWire (IEEE-1394a).

Interfejs Ethernet jest powszechnie spotykany w większości komputerów i systemy operacyjne bezproblemowo radzą sobie z jego obsługą. Użycie urządzenia Phantom Omni z interfejsem sieciowym wiąże się jednak z instalacją dedykowanych sterowników oraz pakietu OpenHaptics v3.0, które jak podaje producent są wspierają jedynie systemy do Windows 8 włącznie, co uniemożliwia korzystanie z nich na nowszych platformach.

FireWire jest standardem łącza szeregowego opracowanym w 1995 roku, który umożliwia szybką transmisję danych. Został on zaprojektowany przede wszystkim do szybkiego przesyłania danych o dużym rozmiarze, zatem jest często wykorzystywany przez producentów sprzętu multimedialnego. Jednak w związku z systematycznym wycofywaniem się (od 2011 roku) producentów sprzętu i oprogramowania ze wspierania interfejsu FireWire, korzystanie z urządzeń w niego wyposażonych rodzi wiele problemów. Użytkownik jest zmuszony do poszukiwania dedykowanych (spełniających specyficzne wymagania producenta) kart rozszerzeń do stacji roboczych mających obsługiwać urządzenia - na dodatek firma SenseAble zaleca korzystanie wyłącznie z kontrolerów IEEE-1394a opartych o chipset firmy NEC lub VIA co może dodatkowo utrudnić proces wdrażania urządzeń przy stanowisku pracy.

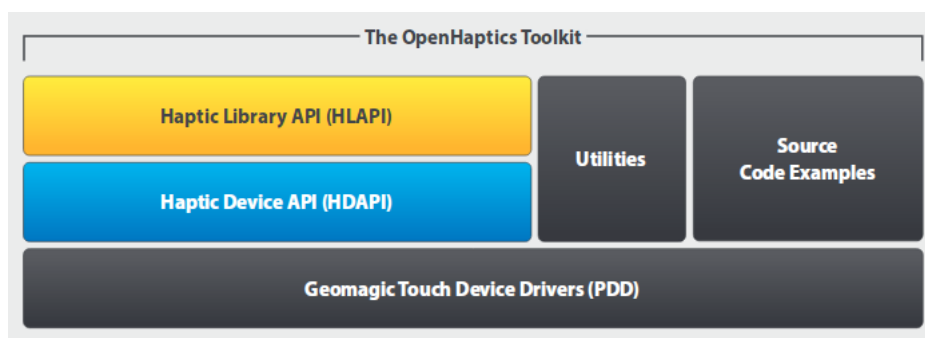
Powyższe trudności powodują, że sama instalacja i uruchomienie urządzenia staje się dość kłopotliwa. Stacje robocze korzystające z urządzeń Phantom Omni w Pracowniach Biofizyki F UW działają na systemach operacyjnych CentOS w wersji 6.0 ze zmodyfikowanym jądrem GNU/Linux, który pozwalał na stosunkowo stabilne działanie urządzenia.

Wraz ze sterownikami i pakietem OpenHaptics v3.0 w pracowniach uruchomiony jest framework Virtual Reality Peripheral Network (opisany w dalszej części pracy), dostarczający uniwersalną i łatwą w użyciu warstwę API nad wieloma rodzajami urządzeń klasy HCI.

Proces instalacji i konfiguracji urządzenia SensAble Phantom Omni nie był przedmiotem niniejszej pracy. Autor pracy miał do dyspozycji przygotowane stanowisko pracy, wyposażone w uruchomione urządzenie haptyczne, zainstalowany zestaw sterowników, bibliotek i pakiet VRPN.

2.3. OpenHaptics Toolkit v3.0

OpenHaptics Toolkit v3.0 jest bogatym pakietem oprogramowania dostarczanego przez producenta urządzenia. Zawiera on zestaw sterowników, bibliotek, narzędzi i przykładowych kodów źródłowych ułatwiających programiście wdrożenie rozwiązań haptycznych w dowolnym programie komputerowym. Biblioteki programistyczne dają dostęp do szerokiego zakresu niskopoziomowych funkcji urządzenia Phantom Omni tworząc przyjazną dla programisty warstwę abstrakcji [21][22].



Rysunek 2.2: Architektura pakietu OpenHaptics

Składnia bibliotek programistycznych pakietu OpenHaptics Toolkit jest wzorowana na składni biblioteki OpenGL i umożliwia tworzenie programów w językach C oraz C++.

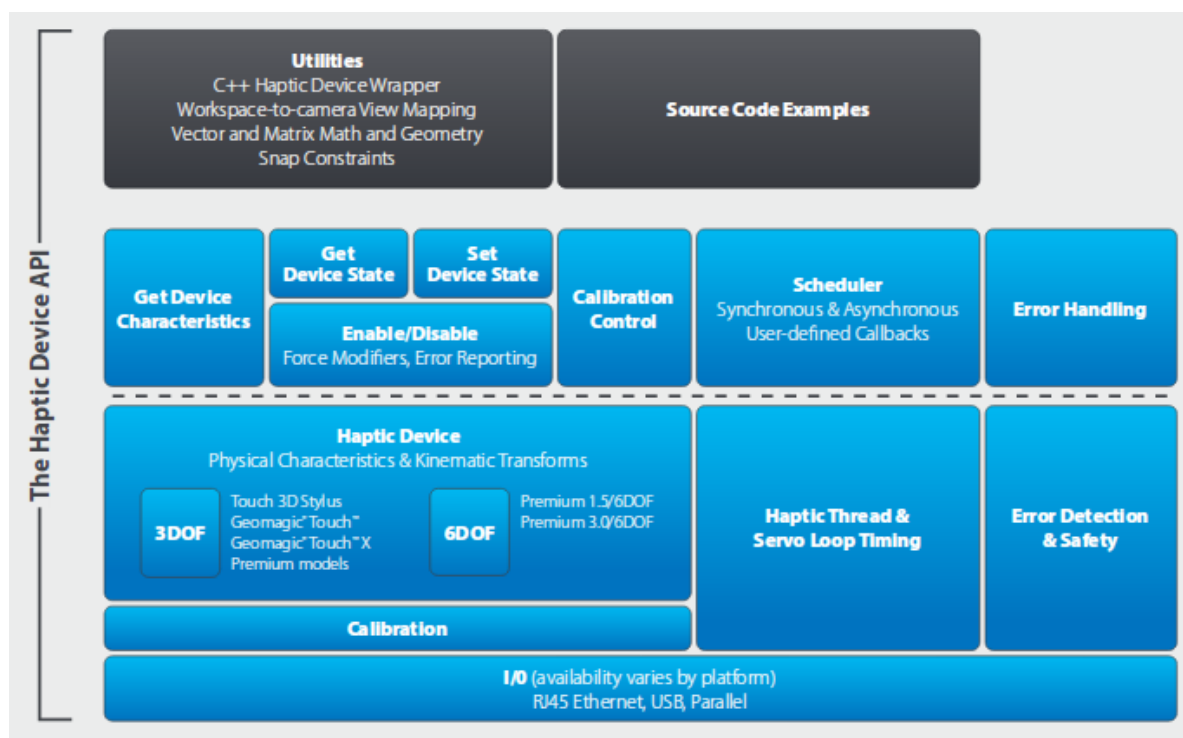
2.3.1. Phantom Device Drivers

Producent oprócz Phantom Omni dostarcza także innego rodzaju urządzenia haptyczne, skierowane do innych grup odbiorców. Phantom Device Drivers (PDD) to zestaw sterowników do wszystkich urządzeń haptycznych oferowanych przez producenta.

2.3.2. Haptic Device API

Haptic Device API (HDAPI) to niskopoziomowy interfejs programistyczny umożliwiający bezpośredni dostęp do wszystkich funkcji związanych z urządzeniami haptycznymi.

Poniższy diagram przedstawia kompletną architekturę modułu HDAPI:



Rysunek 2.3: Architektura HDAPI

Poza bardziej zaawansowanymi funkcjami związanymi z takimi aspektami jak wielowątkowość czy wywołania synchroniczne i asynchroniczne, HDAPI zwraca dane o pozycji i orientacji, umożliwia sterowanie sprzężeniem zwrotnym czy wyzwalanie funkcji (*ang. callback*) w odpowiedzi na wciskanie wbudowanych przycisków. HDAPI dostarcza także rozbudowany interfejs programistyczny systemu detekcji i przechwytywania błędów, kalibracji urządzenia oraz zestaw przykładowych kodów źródłowych demonstrujących wykorzystanie poszczególnych jego funkcji.

2.3.3. Haptic Library API

Haptic Library API (HLAPI) to biblioteki wysokopoziomowego interfejsu programistycznego zaprojektowane głównie pod kątem zgodności z OpenGL API. HLAPI umożliwia daleko idącą integrację z istniejącym kodem i bytami (strukturami, funkcjami, itp.) pochodzącymi z OpenGL. Ponadto ułatwia współpracę z zewnętrznymi bibliotekami dostarczającymi funkcjonalności obliczeń fizycznych (dynamika, detekcja kolizji, itp.).

W odróżnieniu od prostego sterowania momentami sił działającymi na urządzenie (jak w przypadku HDAPI), tutaj możemy definiować bardziej złożone obiekty przestrzenne (bryły, powierzchnie, pola siłowe) na które urządzenie haptyczne może oddziaływać. Również sposób reprezentacji sił dostarczony przez HLAPI jest bardziej rozbudowany, do dyspozycji programisty jest symulacja lepkości, sprężyny, grawitacji czy tarcia.

Podobnie jak w przypadku HDAPI programista dostaje bogaty zestaw przykładowych kodów źródłowych, umożliwiających natychmiastowe przetestowanie interesujących rozwiązań.

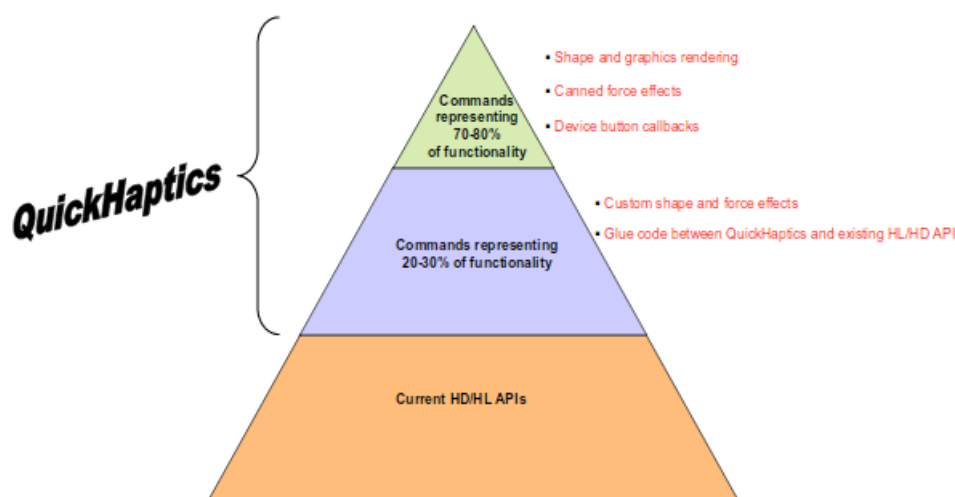
Poniższy diagram przedstawia kompletną architekturę modułu HLAPI:



Rysunek 2.4: Architektura HLAPI

2.3.4. QuickHaptics Micro API

QuickHaptics Micro API jest rozwiązaniem działającym „na szczycie piramidy,” OpenHaptics Toolkit.



Rysunek 2.5: QuickHaptics Micro API

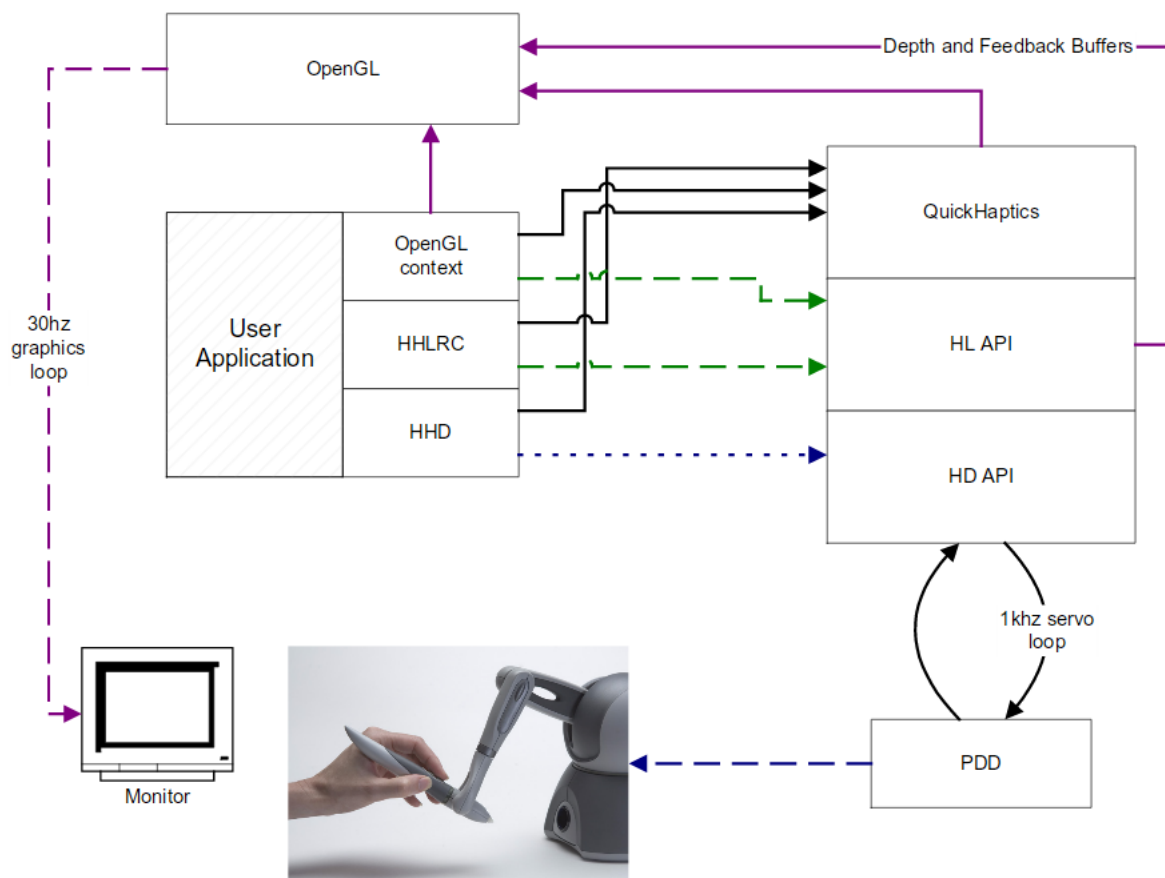
Zgodnie z zapewnieniami producenta QuickHaptics dostarcza jeszcze szybszych i łatwiejszych rozwiązań do tworzenia aplikacji wykorzystujących możliwości jego urządzeń. QuickHaptics Micro API stanowi kolejny, jeszcze wyższy poziom abstrakcji ponad HDAPI oraz

HLAPI, który integrując je dostarcza interfejsy w postaci klas, metod i funkcji języka C++. Umożliwia tworzenie w pełni funkcjonalnych programów przez osoby nie posiadające dużego doświadczenia w dziedzinie wirtualnej rzeczywistości, programowania grafiki komputerowej czy obsługi urządzeń haptycznych.

Wszystkie elementy składające się na OpenHaptics Toolkit mogą być ze sobą łączone i używane zamiennie.

2.4. Servo Loop

Servo Loop to specjalna, działająca w osobnym wątku z wysokim priorytetem pętla, którą musi posiadać każdy program współpracujący z urządzeniami haptycznymi za pośrednictwem bibliotek OpenHaptics.



Rysunek 2.6: Servo Loop

Zapewnienie stabilnego i realistycznego odwzorowania efektów sprzężenia zwrotnego jest tutaj najwyższym priorytetem. Wymagane jest aby pętla Servo Loop była wykonywana z częstotliwością co najmniej 1kHz.

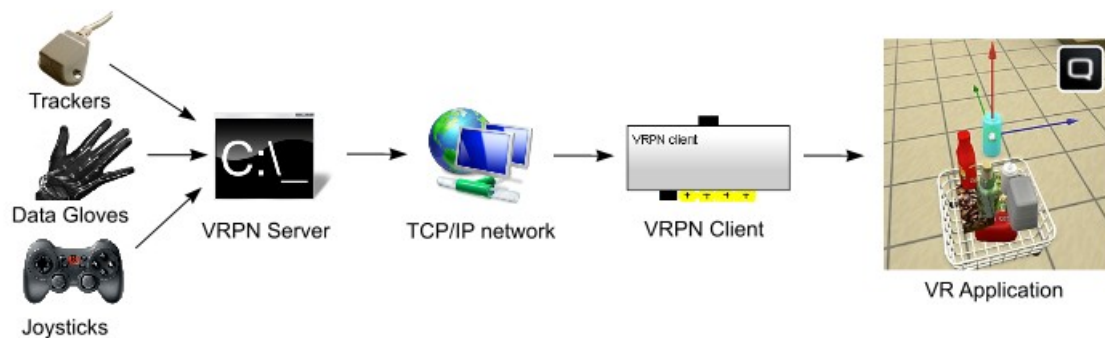
Rozdział 3

Virtual Reality Peripheral Network

Virtual Reality Peripheral Network (w skrócie VRPN) jest rozbudowaną biblioteką, szkieletem aplikacji (*ang. application framework*) i zbiorem narzędzi ułatwiających tworzenie programów komputerowych wykorzystujących urządzenia używane w systemach wirtualnej rzeczywistości. VRPN dostarcza abstrakcyjnych interfejsów programistycznych i serwerów uniezależniających programistę konkretnych rozwiązań sprzętowych. Umożliwia stosunkowo proste budowanie aplikacji obsługujących urządzenia HCI w wielu popularnych językach programowania (np. C++, Python czy Java) [23].

3.1. Opis pakietu

Pakiet VRPN jest od podstaw zaprojektowany jako oprogramowanie sieciocentryczne, tzn. takie w którym sieć komputerowa odgrywa kluczową rolę.



Rysunek 3.1: Architektura VRPN

VRPN definiuje kilka abstrakcyjnych klas wspieranych urządzeń: Analog, Button, Dial, ForceDevice, Imager, Sound, Text oraz Tracker. Przy czym konkretne urządzenie może należeć jednocześnie do jednej lub kilku klas (np. Phantom Omni to Tracker, Button oraz ForceDevice). Istnienie ww. klas powoduje, że wszystkie urządzenia danego typu zgłaszają zmiany stanów jako asynchroniczne wywołania funkcji (*ang. callback*) przekazując w ten sposób obiekty lub wartości parametrów swoich klas. Zadaniem programisty jest odbiór i interpretacja przychodzących danych. Pozwala to na wygodne i szybkie wdrażanie rozwiązań opartych o urządzenia HCI nawet przez osoby bez wcześniejszego doświadczenia z systemami wirtualnej rzeczywistości.

VRPN jest napisany w języku C++, jednak posiada wbudowane mechanizmy generujące kod dla języków Python i Java. W trakcie kompilacji biblioteki z zamiarem jej użycia po stronie klienckiej należy zwrócić szczególną uwagę na docelowy język którego będziemy używać i podać odpowiednie parametry kompilacji zgodnie z wymogami dokumentacji VRPN.

3.2. Opis użycia VRPN

W niniejszej pracy wykorzystuje się integrację pakietu VRPN z urządzeniem Phantom Omni do otrzymywania danych statusowych o pozycji, orientacji i wciśniętych przyciskach oraz sterowania sprzężeniem zwrotnym [24].

Stacja robocza obsługująca urządzenie haptyczne posiada skonfigurowany do jego obsługi proces `vrpn_server` będący serwerem TCP/IP oczekującym na przychodzące połączenia od zainteresowanych aplikacji klienckich. Konfiguracja serwera jest przechowywana w tekstowym pliku konfiguracyjnym `vrpn.cfg`, który jest ładowany każdorazowo podczas uruchamiania.

Stworzona aplikacja, w tym przypadku wtyczka do programu PyMOL, jest klientem TCP/IP nawiązującym połączenie z serwerem VRPN. W trakcie jej uruchamiania tworzone są obiekty klas `vrpn_Tracker`, `vrpn_Button` i `vrpn_ForceDevice` posiadające wskaźniki do funkcji-uchwyków (*ang. handler*) odpowiednio przetwarzających odebrane dane.

Zarejestrowanie funkcji-uchwyty w obiekcie klasy `vrpn_Button` i obsługa przychodzących zdarzeń jest trywialna i sprowadza się do rozpoznania zmiany stanu jednego z dwóch przycisków. W ramach niniejszej pracy przyciski zostały oprogramowane tak, że wciśnięcie pierwszego z nich powoduje wykonanie superpozycji (nałożenia) struktury wzorca z najbliższym możliwym do dopasowania regionem - zgodnie z danymi pochodzącymi z pliku mapującego. Wciśnięcie drugiego przycisku powoduje wykonanie zbliżenia na linię łączącą środki ciężkości ww. struktur.

Dane udostępniane przez obiekt klasy `vrpn_Tracker` to pozycja i orientacja wskaźnika. Odebrane współrzędne należy przeliczyć na jednostki i wielkości obsługiwane przez pakiet PyMOL. W przypadku orientacji należy przejść z reprezentacji w kwaternionach (wydawana przez VRPN) na postać (akceptowaną przez PyMOL) oś-kąt (*ang. axis-angle*), ta operacja została opisana w części teoretycznej pracy. Współrzędne pozycji wskaźnika należy przemnożyć przez eksperymentalnie wyznaczony współczynnik skalujący, który spowoduje realistyczne odwzorowanie przesunięć z przestrzeni rzeczywistej do wirtualnej.

Klasa `vrpn_ForceDevice` umożliwia sterowanie serwomechanizmami wbudowanymi w urządzenie Phantom Omni. Biblioteka VRPN dostarcza szereg metod umożliwiających wykonanie takich operacji na kilka sposobów: począwszy od najprostszego podania wektora pola siłowego, poprzez zdefiniowanie wirtualnej sprężyny zaczepionej w zadanym punkcie, aż do tworzenia wirtualnych powierzchni czy brył.

Integracja oraz przetwarzanie danych dostarczanych przez pakiet VRPN były znaczącą częścią pracy. Użycie pakietu VRPN zamiast OpenHaptics dało znaczący wzrost niezależności projektu od konkretnego urządzenia haptycznego i otworzyło wiele możliwości dalszego rozwoju i wykorzystania stworzonego w ramach niniejszej pracy oprogramowania.

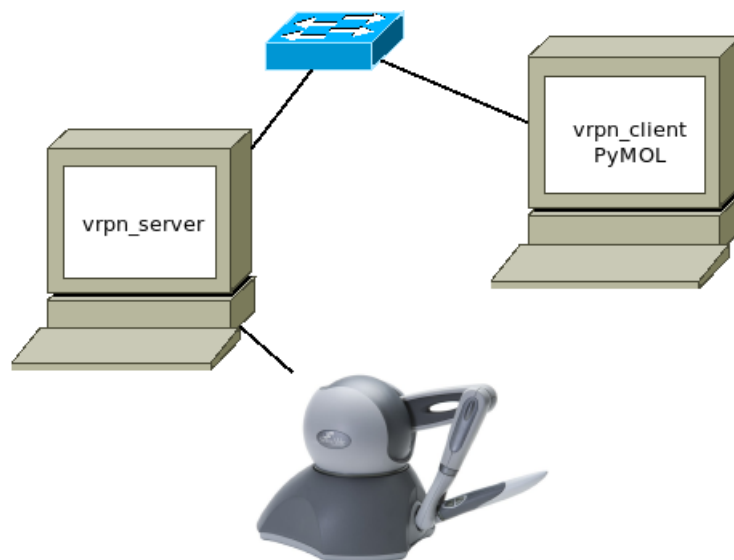
Rozdział 4

Implementacja i uruchomienie oprogramowania

Oprogramowanie będące przedmiotem niniejszej pracy zostało stworzone jako rozszerzenie (*ang. plugin*) popularnego pakietu PyMOL. Wykorzystano w nim możliwości wizualizacyjne pakietu w połączeniu z danymi o pozycji i orientacji urządzenia Phantom Omni odbieranymi dzięki bibliotece VRPN. Dane wejściowe do programu - plik mapujący struktury biopolimerów - został dostarczony przez zewnętrzne oprogramowanie (opisane w części teoretycznej pracy).

4.1. Opis stanowiska laboratoryjnego

Jak już wspomniano prawidłowe uruchomienie oprogramowania wymaga wcześniejszego zestawienia odpowiednio skonfigurowanego stanowiska laboratoryjnego zgodnie z poniższym diagramem:



Rysunek 4.1: Schemat stanowiska laboratoryjnego

Stanowisko składa się z połączonych siecią komputerową stacji roboczych, gdzie jedna stanowi host urządzenia Phantom Omni, druga natomiast to łączący się z nią klient ze stworzonym oprogramowaniem.

Komputer-host musi posiadać kompatybilną z urządzeniem Phantom Omni kartę rozszerzeń z portem IEEE-1394 (FireWire), sterowniki wraz z pakietem OpenHaptics oraz VRPN ze skonfigurowanym procesem `vrpn_server` nasłuchującym połączeń przychodzących na wybranym porcie TCP. Urządzenie powinno być skonfigurowane zgodnie z zaleceniami producenta oraz dokumentacją pakietu VRPN.

Kliencka stacja robocza jest wyposażona w pakiet PyMOL rozszerzony o oprogramowanie - przedmiot niniejszej pracy. Po jego uruchomieniu, załadowaniu danych wejściowych oraz wprowadzeniu adresu sieciowego hosta urządzenia Phantom Omni, program próbuje nawiązać z nim połączenie i rozpocząć pracę.

W szczególności serwer VRPN oraz pakiet PyMOL mogą być uruchomione na jednej maszynie i wykorzystywać interfejs sieciowy `localhost`.

4.2. Rozszerzanie funkcjonalności pakietu PyMOL

PyMOL jest oprogramowaniem służącym głównie do wizualizacji struktur chemicznych oraz przeprowadzania na nich prostych obliczeń. Jego historia sięga roku 2000, kiedy to Warren Lyford DeLano zainicjował powstanie projektu. Przez lata intensywnych prac nad rozwojem program stał się standardowym narzędziem wykorzystywanym przez wiele wiodących ośrodków naukowo badawczych. Od roku 2010 nad jego rozwojem czuwa firma Schrödinger Inc.

Funkcjonalność pakietu PyMOL może być w łatwy sposób rozszerzalna poprzez wbudowany system obsługi wtyczek (*ang. plugin*) tworzonych w języku Python. Programista ma dostęp do rozbudowanego API (*ang. application programming interface*) zawierającego szeroki wachlarz funkcji umożliwiających tworzenie wizualizacji oraz obliczeń na załadowanych reprezentacjach struktur chemicznych. Ponadto programista tworząc wtyczki może z powodzeniem korzystać z biblioteki standardowej języka Python oraz dowolnych innych bibliotek pochodzących od zewnętrznych dostawców.

Podczas tworzenia niniejszego oprogramowania wykorzystano zarówno API wbudowane w PyMOL jak również pochodzące z zewnętrznych bibliotek takich jak VRPN do efektywnej obsługi urządzenia haptycznego.

Proces tworzenia wtyczek do pakietu PyMOL został szczegółowo opisany w jego dokumentacji.

4.3. Dane wejściowe

Oprogramowanie do poprawnego uruchomienia wymaga podania dwóch plików w formacie PDB (*ang. Protein Data Bank*) oraz pliku mapującego.

Pierwszy plik wejściowy definiuje *strukturę celu*. To najczęściej cała, duża cząsteczka białka lub kwasu nukleinowego. W programie zostaje ona umieszczona w początku układu współrzędnych i jest nieruchoma względem wskaźnika urządzenia haptycznego. Stanowi bazową strukturę na której mapowane są regiony podobne do struktury wzorcowej.

Drugi plik - *wzorzec* - może stanowić krótki (najczęściej kilkanaście lub kilkadziesiąt merów) wycinek cząsteczki celu lub dowolną inną podjednostkę czy fragment struktury drugorzędowej (np. α -helisa). Zostaje on na stałe związany z ruchem wskaźnika urządzenia haptycznego. Każde jego przesunięcie czy zmiana orientacji zostaje w czasie rzeczywistym odwzorowana na ekranie.

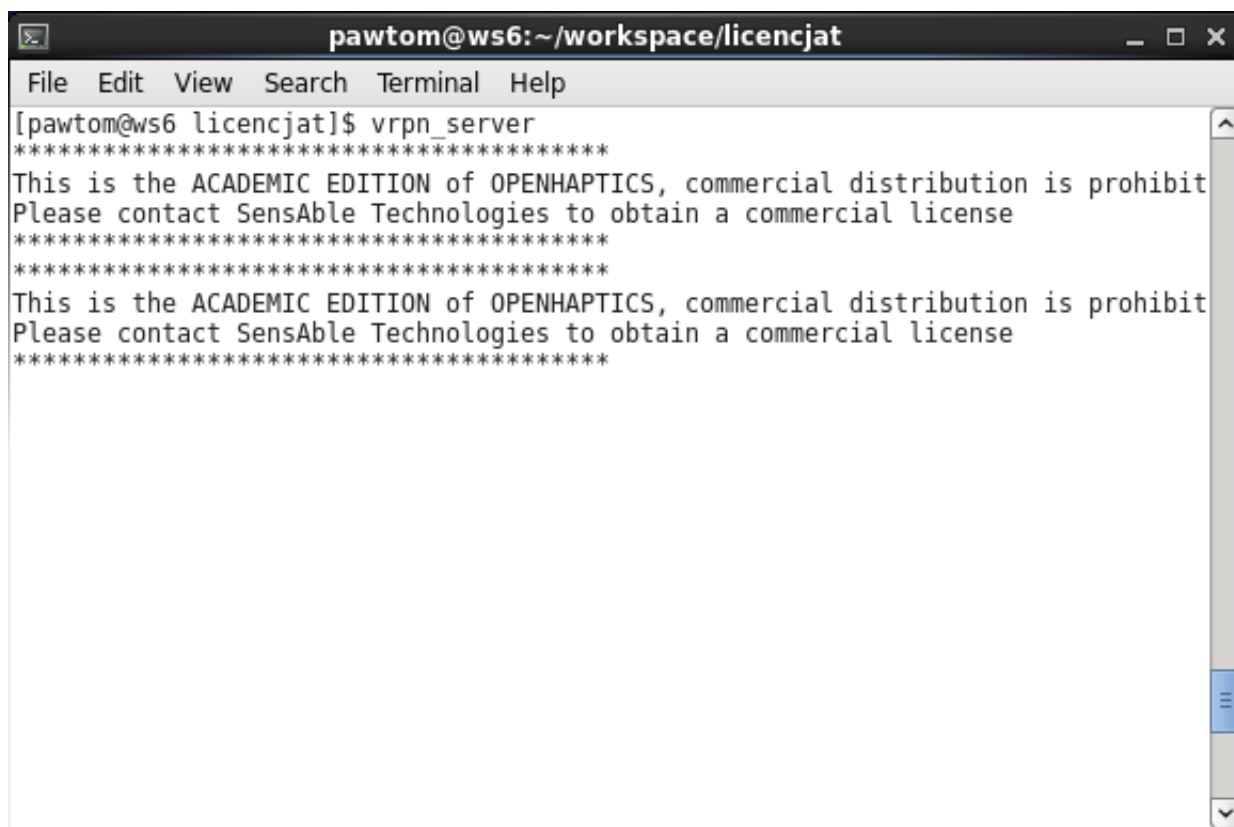
Plikiem mapującym nazywamy dane uzyskane z obliczeń przeprowadzonych przez zewnętrzne oprogramowanie, zawierających informacje o takich regionach struktury celu, których stopień podobieństwa (dopasowania) do struktury wzorcowej nie przekracza zadanego

progu. Dane te są kluczowe do działania niniejszego oprogramowania, gdyż na ich podstawie wyliczane są odległości i momenty sił projektowane do urządzenia Phantom Omni. Szczegóły generowania pliku mapującego zostały opisane we teoretycznej części pracy.

Na ekranie PyMOL wyświetlany jest także wektor łączący wzorcową strukturę z najbliższym optymalnie pasującym regionem struktury głównej - zgodnie z danymi mapującymi. Wektor ten odwzorowuje w sposób wprost proporcjonalny wartość i kierunek momentów sił ustawianych w urządzeniu haptycznym.

4.4. Opis oprogramowania

Po zestawieniu i skonfigurowaniu stanowiska laboratoryjnego zgodnie z wcześniejszym opisem, na komputerze będącym hostem urządzenia Phantom Omni należy uruchomić proces `vrpn_server`:

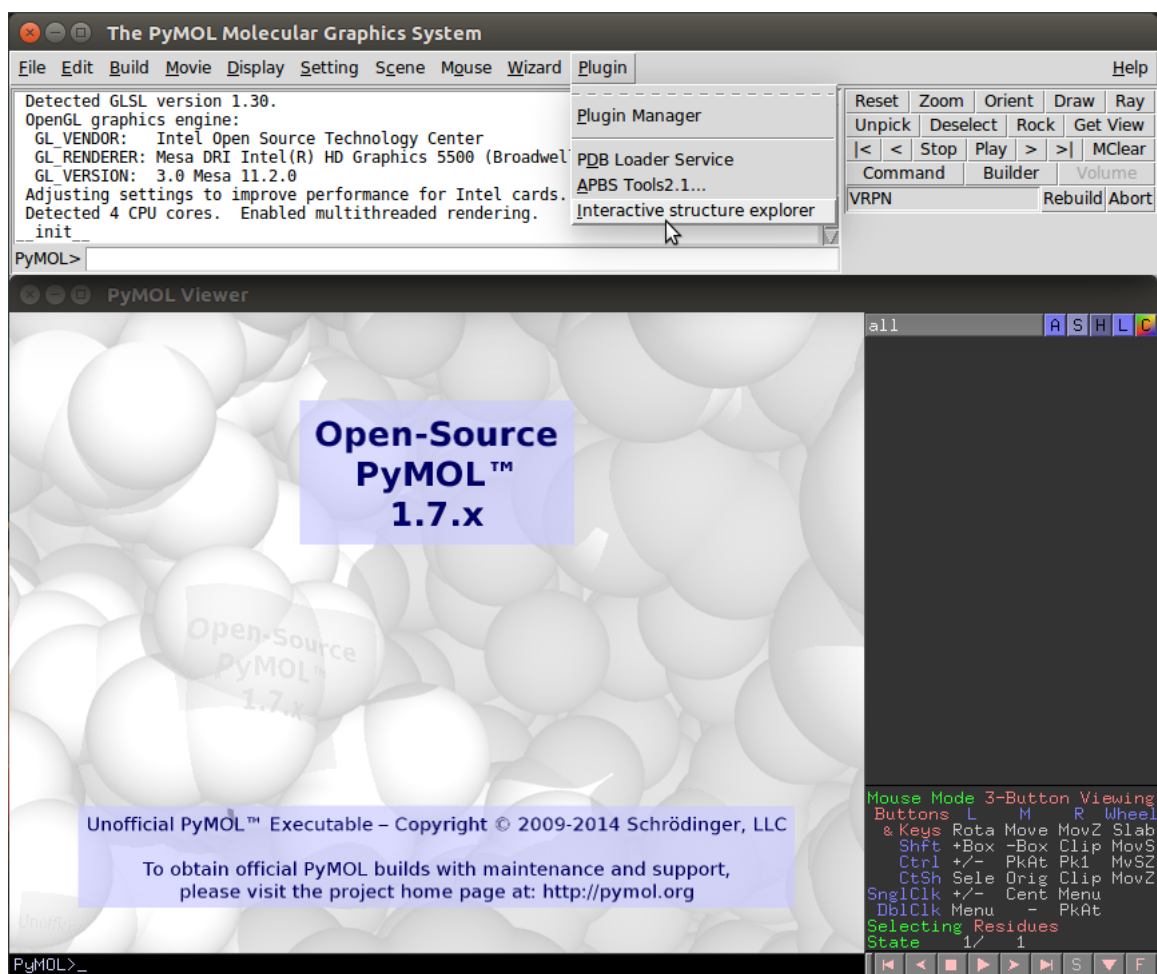


```
pawtom@ws6:~/workspace/licencjat
File Edit View Search Terminal Help
[pawtom@ws6 licencjat]$ vrpn_server
*****
This is the ACADEMIC EDITION of OPENHAPTICS, commercial distribution is prohibit
Please contact SensAble Technologies to obtain a commercial license
*****
*****
This is the ACADEMIC EDITION of OPENHAPTICS, commercial distribution is prohibit
Please contact SensAble Technologies to obtain a commercial license
*****
```

Rysunek 4.2: Poprawne uruchomienie procesu `vrpn_server`

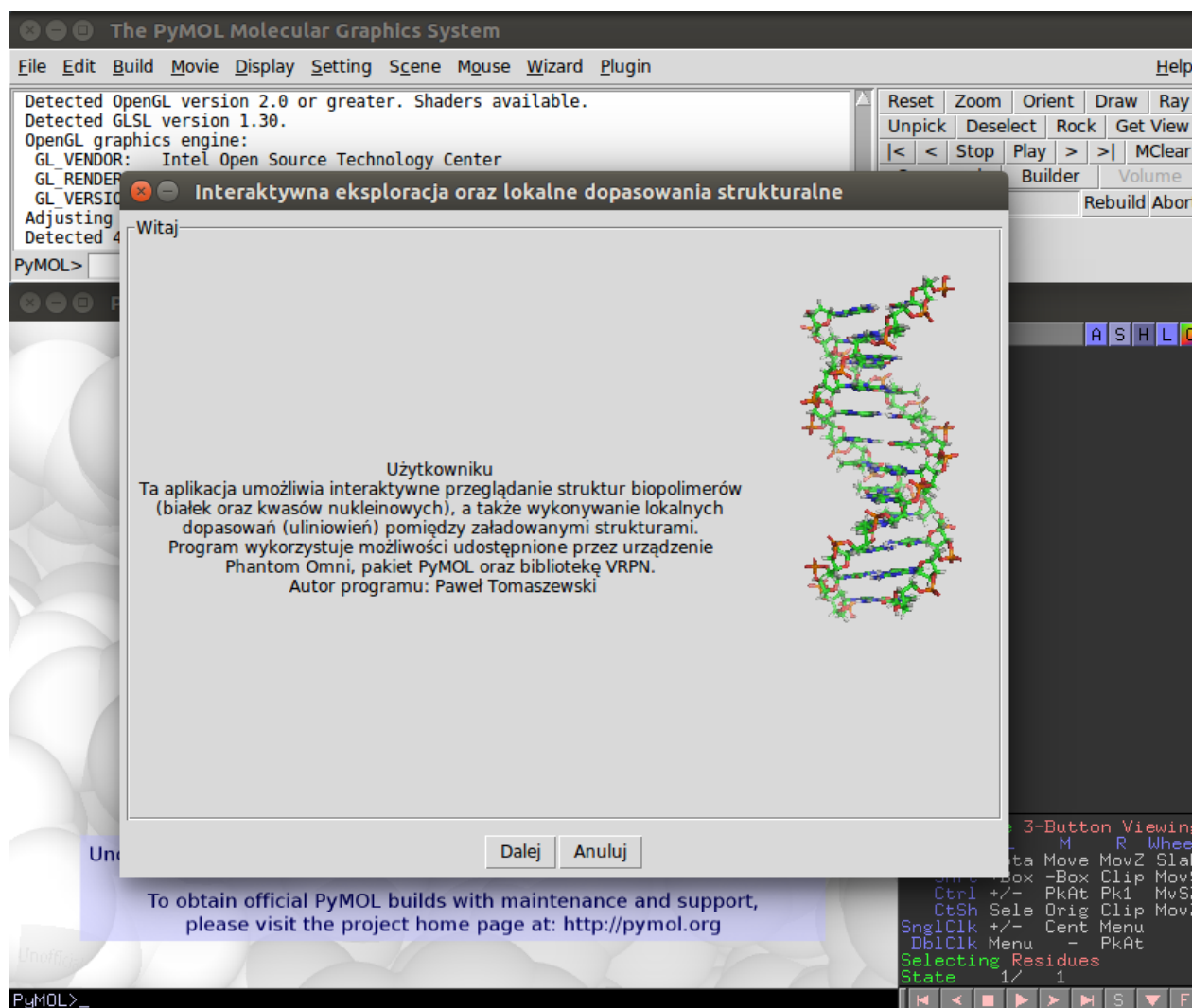
Powyższy ekran pokazuje komunikat z informacją licencyjną. Gdy proces `vrpn_server` uruchomi się pomyślnie, nie powinno być żadnych innych komunikatów.

Na komputerze z oprogramowaniem PyMOL należy uruchomić zainstalowaną wtyczkę przez kliknięcie odpowiedniej pozycji w menu **Plugin**:



Rysunek 4.3: Uruchamianie oprogramowania

W kolejnym kroku pojawi się ekran powitalny z krótkim opisem możliwości oferowanych przez oprogramowanie.



Rysunek 4.4: Ekran powitalny

Po kliknięciu na przycisk DALEJ przechodzimy do kolejnego okna, które daje nam możliwość przeglądania dysku w poszukiwaniu odpowiednich plików wejściowych. Ponadto należy podać adres komputera, na którym uruchomiony jest proces `vrpn_server`. W przypadku pracy na lokalnej maszynie w oknie należy podać adres `phantom0@127.0.0.1` lub `phantom0@localhost`:

Interaktywna eksploracja oraz lokalne dopasowania strukturalne

Wzorzec
Wzorzec jest struktura, która będziemy próbowali dopasować do cząsteczki bazowej.
Wzorzec może stanowić wycinek cząsteczki bazowej, np. jakaś struktura drugorzędowa

Plik mapowania
Tutaj wybierz plik mapowania

Identyfikator PDB
Tutaj wybierz plik

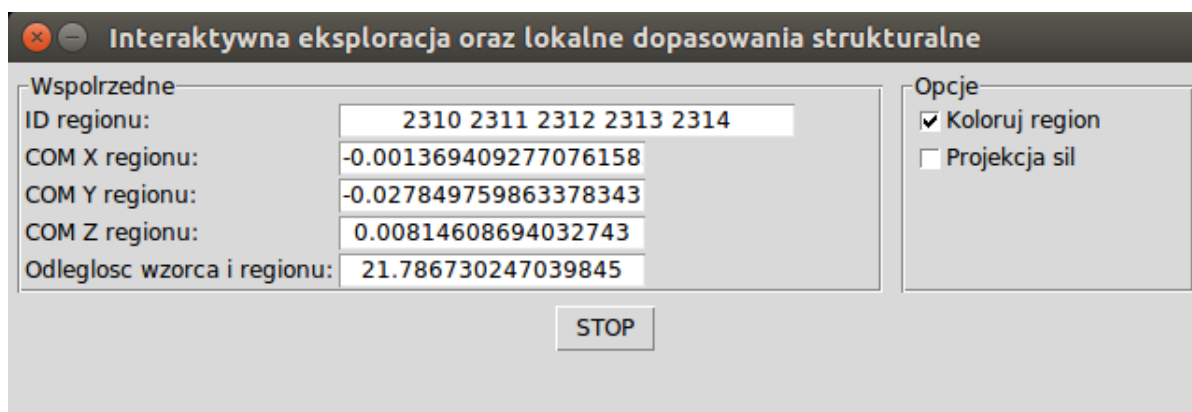
Adres IP serwera VRPN

Rysunek 4.5: Wprowadzanie danych wejściowych

Program zakłada poprawność wprowadzonych danych, tzn. nie jest przeprowadzane sprawdzenie, czy załadowane pliki są zgodne z formatem PDB czy prawidłowym formatem pliku mapującego.

W kolejnym kroku program próbuje nawiązać połączenie TCP z hostem urządzenia haptycznego. Prawidłowe nawiązanie połączenia rozpoczyna wymianę danych pomiędzy aplikacjami. Proces `vrpn_server` wysyła komunikaty o pozycji oraz orientacji wskaźnika, natomiast PyMOL za pośrednictwem sieci wysyła informację o sprzężeniu zwrotnym do urządzenia Phantom Omni.

Na ekranie widać wzorcową strukturę poruszającą się zgodnie z ruchami urządzenia haptycznego, natomiast urządzenie haptyczne dostaje polecenia sterujące momentami sił, przyciągając wskaźnik do najbliższego (wskazywanego na ekranie przez biały odcinek) podobnego regionu w ramach struktury celu.



Rysunek 4.6: Ekran główny 1

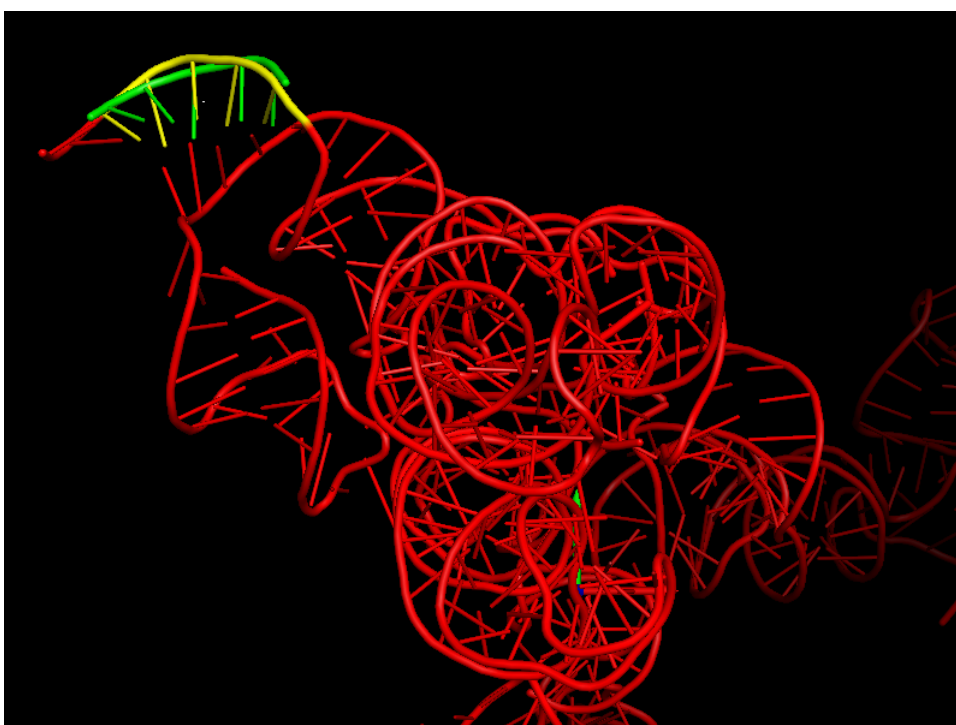
W trakcie normalnej pracy programu, na jego ekranie głównym wyświetlane jest w czasie rzeczywistym szereg informacji:

1. ID **regionu** - identyfikator aktualnie wskazywanego regionu będący najczęściej listą identyfikatorów, numerów **resid** aminokwasów lub nukleotydów pobraną z pliku PDB definiującego strukturę celu.
2. COM (ang. center of mass) - współrzędne X, Y, Z środka ciężkości wskazywanego regionu
3. Odległość pomiędzy aktualną pozycją wzorca i wskazywanego regionu
4. Koloruj **region** zaznaczenie tej opcji powoduje osobne kolorowanie aktualnie wskazywanego regionu w celu jeszcze lepszego jego uwidocznienia na ekranie
5. Projekcja **sił** w programie istnieje możliwość wyłączenia projekcji sił. Ta opcja przydaje się podczas problemów z wydajnością podczas pracy na słabszych komputerach lub w wolniejszej sieci.

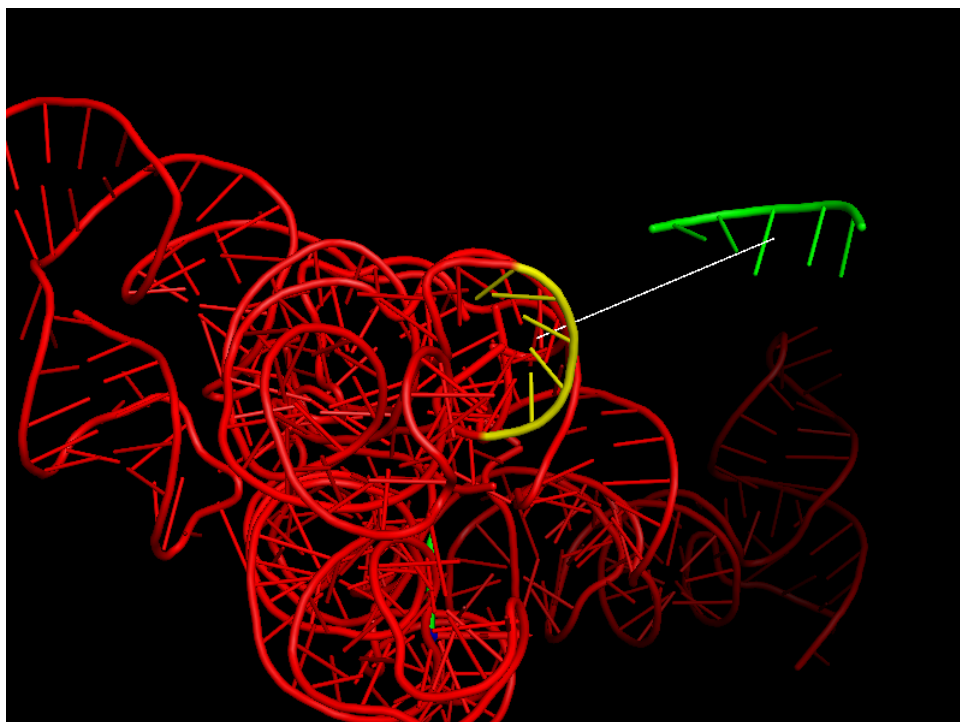
Poniższe zrzuty ekranu przedstawiają przykładowy przebieg procesu eksploracji oraz lokalnego dopasowywania struktur do siebie. Bazuje on na cząsteczce celu (kolor czerwony) będącej rybosomalną podjednostką RNA o identyfikatorze PDB: **1fg0**, krótkim pięcionukleotydowym fragmencie helisy (kolor zielony) RNA - będącej strukturą szablonu.



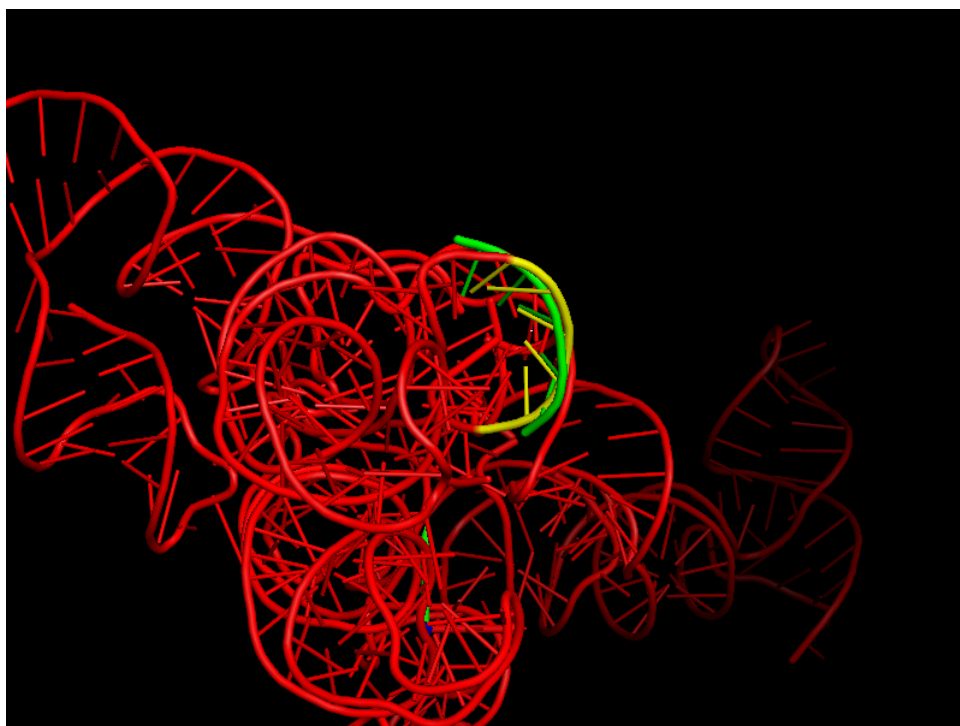
Rysunek 4.7: Podświetlony region (kolor żółty) wraz z pasującą strukturą wzorcową (kolor zielony)



Rysunek 4.8: Superpozycja wzorca i najbliższego regionu



Rysunek 4.9: Podświetlony inny region (kolor żółty) wraz z pasującą strukturą wzorcową (kolor zielony)



Rysunek 4.10: Superpozycja wzorca i najbliższego regionu

Rozdział 5

Podsumowanie

W niniejszej pracy została przedstawiona implementacja oprogramowania, którego funkcjonalność obejmuje interaktywne przeglądanie struktur biopolimerów (białek i kwasów nukleinowych) w poszukiwaniu regionów podobnych do wybranej struktury wzorcowej oraz zwrotną projekcję momentów sił pomiędzy tymi elementami. Program wykorzystuje do tego celu dane wejściowe pochodzące z zewnętrznych źródeł - pliki z opisem struktur w formacie PDB oraz plik mapujący.

Oprogramowanie wykorzystuje metody i urządzenia wirtualnej rzeczywistości w szczególności pakiet VRPN oraz urządzenie haptyczne Phantom Omni. Powstało ono w formie wtyczki rozszerzającej funkcjonalność pakietu PyMOL wykorzystując jego możliwości wizualizacji struktur chemicznych.

Poza oprogramowaniem w pracy zostały także przedstawione podstawy teoretyczne, na które składały się globalne i lokalne metody dopasowania (uliniowienia) strukturalnego oraz algorytm Kabsch'a maksymalizujący stopień tego dopasowania. Opisane także zostały elementarne przekształcenia geometryczne wykorzystywane w pracy oraz sposób konwersji kwaternionów na macierze rotacji (wykorzystywane przez pakiety VRPN i PyMOL). Szczegółowo zostało opisane urządzenie haptyczne Phantom Omni i jego integracja z pakietem Virtual Reality Peripheral Network. Kod źródłowy stworzonego oprogramowania wraz z obszernymi komentarzami stanowi załącznik do niniejszego opracowania.

Ostateczna forma oprogramowania jest zaledwie prototypem i może mieć dość ograniczone wykorzystanie praktyczne. Dobrze jednak nadawje się do celów naukowo-edukacyjnych np. demonstrujących możliwości zastosowania metod wirtualnej rzeczywistości w takich dziedzinach nauki jak biofizyka, biologia molekularna czy chemia.

Wykorzystanie integracji urządzenia Phantom Omni z pakietem VRPN i PyMOL w połączeniu z różnymi metodami bioinformatycznymi tworzy cały szereg nowych i być może nieznanych dotąd możliwości. Łatwość wchodzenia w interakcję z wykreowanym, wirtualnym, molekularnym światem i odzwierciedlanie sił obecnych na tym poziomie może znacznie ułatwić i uatrakcyjnić proces edukacji przyszłych adeptów różnych nauk przyrodniczych.

Dalszy rozwój niniejszego oprogramowania mógłby polegać na dodaniu funkcjonalności, które w chwili obecnej realizuje oprogramowanie zewnętrzne. Wymagałoby to rozbudowy obecnego programu o część realizującą obliczenia dopasowań strukturalnych implementując przykładowe metody opisane we wcześniejszych rozdziałach. Ponadto warto rozważyć dodanie bardziej zaawansowanych metod bioinformatycznych związanych z dokowaniem molekularnym i projektowaniem leków.

W przypadku dalszego znacznego rozwoju funkcjonalności oprogramowania zgodnie z powyższymi lub innymi propozycjami możliwe jest wyobrażenie sobie wykorzystania go na szerszą skalę w ośrodkach naukowo-badawczych uczelni lub przemysłu.

Bibliografia

- [1] Christian B. Anfinsen, *Principles that govern the folding of protein chains*, Science, 1973
- [2] *Critical Assessment of protein Structure Prediction*, <http://predictioncenter.org/casproll/results.cgi>
- [3] Paweł Daniluk, *Metody wirtualnej rzeczywistości. Urządzenia haptyczne.*, Uniwersytet Warszawski, 2011
- [4] L. Holm, C. Sander, *Protein structure comparison by alignment of distance matrices*, J Mol Biol, 233(1):123–38, 1993
- [5] C. A. Orengo, W. R. Taylor, *SSAP: sequential structure alignment program for protein structure comparison*, Methods Enzymol, 266:617–35, 1996
- [6] I. N. Shindyalov, P. E. Bourne, *Protein structure alignment by incremental combinatorial extension (CE) of the optimal path*, Protein Eng, 11(9):739–47, 1998
- [7] T. Madej, J. F. Gibrat, S. H. Bryant, *Threading a database of protein cores*, Proteins, 23(3):356–69, 1995
- [8] T. Kawabata, K. Nishikawa, *Protein structure comparison using the markov transition model of evolution*, Proteins, 41(1):108–22, 2000
- [9] A. Guerler, E. W. Knapp, *Novel protein folds and their nonsequential structural analogs*, Protein Sci, 17(8):1374–82, 2008
- [10] Paweł Daniluk, *Analiza podobieństwa struktur przestrzennych białek przy użyciu dekrptorów lokalnej struktury*, Uniwersytet Warszawski, 2011
- [11] Krzysztof Fidelis, *A novel approach to fold recognition using sequence-derived properties from sets of structurally similar local fragments of proteins*, Bioinformatics, 2003
- [12] Irina Kufareva, Ruben Abagyan, *Methods of protein structure comparison*, Methods in Molecular Biology, 2012
- [13] Yang Zhang, Jeffrey Skolnick, *TM-align: a protein structure alignment algorithm based on the TM-score*, Nucleic Acids Research, 2005
- [14] Wolfgang Kabsch, *A solution for the best rotation to relate two sets of vectors*, Acta Crystallographica, 1976
- [15] Wolfgang Kabsch, *A discussion of the solution for the best rotation to relate two sets of vectors*, Acta Crystallographica, 1978

- [16] Andries van Dam, *How Are Geometric Transformations (T,R,S) Used in Computer Graphics?*, Introduction to computer graphics, 2000
- [17] A. F. Mobius, *Der Barycentrische Calcul : ein neues Hilfsmittel zur analytischen Behandlung der Geometrie*, 1827
- [18] Y. Magarshak, *Quaternion representation of RNA sequences and tertiary structures*, Bio-Systems, 1993
- [19] William Hamilton, *On Quaternions; or on a new System of Imaginaries in Algebra*, list do Johna T. Graves'a, 1843
- [20] 3D Systems, *Haptic Devices. Haptic devices that add the sense of Touch to your digital world*
- [21] 3D Systems, *OpenHaptics Developer Edition*, Broszura informacyjna producenta
- [22] 3D Systems, *OpenHaptics Toolkit version 3.0 Programmers Guide*, przewodnik programisty, 2015
- [23] Russell M. Taylor II, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, Aron T. Helser, *VRPN: A Device-Independent, Network-Transparent VR Peripheral System*, University of North Carolina, 2001
- [24] M. Cuevas-Rodriguez, D. Gonzalez-Toledo, L. Molina-Tanco, A. Reyes-Lecuona, *Contributing to VRPN with a new server for haptic devices*, University of Malaga, 2015

Kod źródłowy

```
1 # -*- coding: utf-8 -*-
3 """
4     Kod źródłowy pracy licencjackiej zatytułowanej:
5     "Interaktywna eksploracja i dopasowanie lokalnie optymalnych struktur biopolimerów
6     z wykorzystaniem metod wirtualnej rzeczywistości.".
7     Praca powstała na kierunku Bioinformatyka i Biologia Systemów
8     Wydziału Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego.
9
10    Autorem programu jest Paweł Tomaszewski. Praca powstała w pracowni udostępnionej
11    przez Zakład Biofizyki Wydziału Fizyki Uniwersytetu Warszawskiego.
12 """
13
14 import os
15 import sys
16 sys.path.append("/home/crooveck/workspace/LICENCJAT/python_vrpn")
17 sys.path.append(".")
18 from transformations import *
19 from Tkinter import *
20 from tkFileDialog import *
21 from pymol import *
22 from pymol.cgo import *
23 from time import *
24 import vrpn_Tracker
25 import vrpn_Button
26 import vrpn_ForceDevice
27 from math import *
28
29 # Flagi wykorzystywane w pętli głównej programu.
30 IS_RUNNING = False          # Flaga sterująca działaniem pętli głównej programu
31 AUTO_ZOOMING = False        # Flaga sterująca opcją auto-zoom
32 REGION_COLORING = True      # Flaga sterująca kolorowaniem regionów
33 FORCES_ENABLED = False      # Flaga sterująca sprzężeniem zwrotnym
34
35 # Współrzędne w przestrzeni urządzenia Phantom Omni
36 trackerX = trackerY = trackerZ = 0
37 # Współrzędne w przestrzeni PyMOL
38 x = y = z = 0
39 # Współczynnik skalujący translacje pomiędzy przestrzeniami urządzenia Phantom Omni
40 #    oraz PyMOL
41 scale=750
42
43 # Środek ciężkości całej cząsteczki
44 molecule_com=[0,0,0]
45 # Środek ciężkości struktury wzorca
46 templateCOM=(0,0,0)
47 # Środek ciężkości struktury regionu
48 regionCOM=(0,0,0)
49
50 # Kwaternion reprezentujący poprzednią (zachowaną) orientację
51 previous_orientation=0
52
53 mapping=[]
54 regions={}
55
```

```

55 currentWindow=0
phantomIp=0
57
# Zmienne przechowujące dane wejściowe
59 targetStructureFile=0
templateStructureFile=0
61 structureMappingFile=0

63 # Zmienne pomocnicze do UI
regionId=regionX=regionY=regionZ=regionTemplateDistance=rmsdEntry=0
65
# Prosta funkcja obliczająca środek masy (COM) zadanych punktów w przestrzeni
67 def simple_com(region_coordinates):
    # pobiera jako parametr liste z krotkami ze współzrzednymi i zwraca srodek masy
69     x, y, z = 0,0,0
    size = len(region_coordinates)

71     for coordinate in region_coordinates:
73         x += coordinate[0]
        y += coordinate[1]
75         z += coordinate[2]

77     return (x/size, y/size, z/size)

79 # Handler obsługujący zdarzenia dot. trackera (rotacje, translacje)
def tracker_handler(u, tracker):
81     global trackerX, trackerY, trackerZ
    trackerX = tracker[1]
83     trackerY = tracker[2]
    trackerZ = tracker[3]

85     # print trackerX,trackerY,trackerZ
87
    # TRANSLACJE:
89     x0 = trackerX*scale
    y0 = trackerY*scale
91     z0 = trackerZ*scale
    # funkcja dokonujaca przekształcenia - transjacji
93     global x, y, z
    cmd.translate(vector=[(x0-x), (y0-y), (z0-z)], object="template", camera=1)
95     x = x0
    y = y0
97     z = z0

99 # ROTACJE
global previous_orientation
101 # bierzacy stan - orientacja
orientation=(tracker[7], tracker[4], tracker[5], tracker[6]) # inny format
kwaterniona do transformations.py niz dostaje z VRPN
103 # przy pierwszym uruchomieniu
# gdy nie ma poprzedniej orientacji
105 if(previous_orientation == 0):
    previous_orientation=orientation
107
    rotation_quaternion=quaternion_multiply(quaternion_inverse(previous_orientation),
orientation)
109     previous_orientation=orientation

111     rotation_matrix = quaternion_matrix(rotation_quaternion) # Return homogeneous
rotation matrix from quaternion.
(rotation_angle, rotation_axis, point) = rotation_from_matrix(rotation_matrix)
113
    cmd.rotate(axis=[rotation_axis[0], rotation_axis[1], rotation_axis[2]],
115         angle=(rotation_angle*180/math.pi), origin=[templateCOM[0], templateCOM[1],
templateCOM[2]], object="template", camera=1)

117 # Handler obsługujący zdarzenia dot. przycisków
def button_handler(u, button):
119     # button[0] - numer przycisku (0-gorny,1-dolny)

```

```

# button[1] - status przycisku (0-puszczony,1-wcisniety)
121
122 global AUTO_ZOOMING
123 if (button[0]==0 and button[1]==0):
124     AUTO_ZOOMING = False
125 elif (button[0]==0 and button[1]==1):
126     AUTO_ZOOMING = True
127
128 # wykonuje dopasowanie i translacje wzorca nad regionem
129 if (button[0]==1 and button[1]==1):
130
131     cmd.align("template","region")
132
133     reg_com=cmd.centerofmass("region")
134     temp_com=cmd.centerofmass("template")
135
136     cmd.translate(object="template", vector=[reg_com[0]-temp_com[0],
137         reg_com[1]-temp_com[1],reg_com[2]-temp_com[2]], camera=0)
138
139 # Handler obsługujący zdarzenia związane ze sprzężeniem zwrotnym
140 # (brak implementacji, patrz: pętla główna programu)
141 def force_handler(u, force):
142     return None
143
144 # Funkcja rysująca osie układu współrzędnych
145 def draw_xyz_axes(x0, y0, z0):
146     w = 0.5 # cylinder width
147     l = 10 # cylinder length
148     h = 2 # cone height
149     d = w * 1.618 # cone base diameter
150
151     axes = [
152         CYLINDER, x0, y0, z0, l, 0.0, 0.0, w, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0,
153         CYLINDER, x0, y0, z0, 0.0, l, 0.0, w, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0,
154         CYLINDER, x0, y0, z0, 0.0, 0.0, l, w, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0,
155         CONE, l, 0.0, 0.0, (h+1), 0.0, 0.0, d, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0,
156         CONE, 0.0, l, 0.0, 0.0, (h+1), 0.0, d, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0,
157         CONE, 0.0, 0.0, l, 0.0, 0.0, (h+1), d, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0,
158         1.0]
159
160     cmd.load_cgo(axes, "axes")
161
162 # Funkcja rysuje strukturę wzorcową (template)
163 def draw_template_structure(template_pdb_file):
164     cmd.load(template_pdb_file,"template")
165     cmd.hide("lines","template")
166     cmd.show("cartoon","template")
167     cmd.color("green","template")
168
169     template_com=cmd.centerofmass("template")
170     # centruje wskaźnik (przenosze go do zera)
171     cmd.translate(vector=[-template_com[0], -template_com[1], -template_com[2]],
172         object="template", camera=0)
173
174 # Funkcja rysuje strukturę celu (target)
175 def draw_target_structure(target_pdb_file):
176     # ładuje czasteczkę
177     cmd.load(target_pdb_file,"target")
178     cmd.hide("lines","target")
179     cmd.show("cartoon","target")
180     cmd.color("red","target")
181
182     # licze jej centrum masy
183     global molecule_com
184     molecule_com=cmd.centerofmass("target")
185     # przesuwam ja na srodek ekranu, srodek ciężkości na (x,y,z)=(0,0,0)
186     cmd.translate(vector=[-molecule_com[0], -molecule_com[1], -molecule_com[2]],

```

```

185     object="target", camera=0)
186
187     # pobieram liste pozycji 3D atomow w czasteczce
188     stored.pos = []
189     cmd.iterate_state(1, "target", "stored.pos.append((x,y,z,elem,chain,resi))")
190
191 # Funkcja obsługująca ładowanie pliku mapującego
192 def load_mapping_file(mapping_file):
193     file=open(mapping_file,"r")
194
195     global mapping
196     mapping=[]
197
198     for line in file:
199         mapping_symbol=line.split()[0]
200         chain=line.split()[1][0]
201         resi=line.split()[1][1:]
202         mapping.append([mapping_symbol,chain,resi,0])
203     file.close()
204
205     # na ostatnie pole w mapping wstawiam COM obliczony dla kazdego nukleotydu
206     for nucleotide in mapping: # iteruje po wszystkich liniach z pliku mapującego
207         # pobieram atomy nalezace do poszczegolnych nukleotydow
208         nucl_coords = [[atom[0],atom[1],atom[2]] for atom in stored.pos if (atom[5]==
209         nucleotide[2])]
210         nucleotide[3]=simple_com(nucl_coords) # srodek masy dla nukleotydu
211
212     print "SKONCZYLEM ŁADOWANIE MAPOWANIA"
213
214 # Obliczam srodek ciężkości wszystkich wykrytych regionów
215 def calculate_regions_com():
216     # zliczam ilosc nukleotydow w helisie wzorcowej
217     # UWAGA: tutaj zakladam, ze wzorcowa helisa (czy struktura) sklada sie z dwoch
218     lancuchow o
219     # tej samej dlugosci (ilosci nukleotydow). To oznacza, ze szukam regionow
220     # o dlugosci jednego lancucha helisy, czyli polowy wszystkich nukleotydow
221     # z tej helisy.
222
223     unique_nucl=set()
224     cmd.iterate_state(1,"template","unique_nucl.add((chain,resi))",space={'unique_nucl
225     ':unique_nucl})
226     M=len(mapping) # ilosc nukleotydow w BADANEJ czasteczce
227     N=len(unique_nucl)/2 # polowa wszystkich nukleotydow z WZORCOWEJ czasteczki
228
229     # wyszukiwanie regionow
230     for m in range(0,M-N+1): # -1 jesli blad
231         region=() # inicjuje pusty tuple
232         region_coords=[]
233         for n in range(0,N): # -1 jesli blad
234             if mapping[m+n][0]=='0':
235                 break
236             region=region+(mapping[m+n][2],)
237             region_coords.append(mapping[m+n][3])
238         if n==(N-1):
239             #znaleziono region w badanej czasteczke pasujacy do czasteczki
240             wzorcowej
241             print "JEST REGION: ", m, region, region_coords
242             # dodaje znaleziony region do tablicy regions
243             regions[region]=simple_com(region_coords)
244
245     # print regions
246
247 # Obliczam srodek ciężkości wszystkich wykrytych regionów (po nowemu)
248 def new_calculate_regions_com():
249     unique_nucl=set()
250     cmd.iterate_state(1,"template","unique_nucl.add((chain,resi))",space={'unique_nucl
251     ':unique_nucl})
252     M=len(mapping) # ilosc nukleotydow z czasteczki celu
253     N=len(unique_nucl) # wszystkie nukleotydy z WZORCOWEJ czasteczki

```

```

249 # wyszukiwanie regionow
251 for m in range(0,M-N+1): # -1 jesli blad
    region=() # inicjuje pusty tuple
    region_coords=[]
253     for n in range(0,N): # -1 jesli blad
        if mapping[m+n][0]!='0':
255             break
        region=region+(mapping[m+n][2],)
257         region_coords.append(mapping[m+n][3])
        if n==(N-1):
259             # znaleziono region w badanej czasteczce pasujacy do czasteczki
                wzorcowej
#             print "JEST REGION: ", m, region, region_coords
261             # dodaje znaleziony region do tablicy regions
                regions[region]=simple_com(region_coords)
263
265 # Funkcja znajdujaca najblizszy atom do zadanej pozycji
def find_closest_atom(x0,y0,z0):
267     if (len(stored.pos)==0):
        return [0,0,0]
269
        minimumDistance=sys.float_info.max # poczatkowa wartosc minimalna powinna byc
        duza
271         closestAtomDistance=0 # index najblizszego atomu
        # liczymy najblizszy atom
273         for atomNumber in xrange(0,len(stored.pos)):
            xDistance = (stored.pos[atomNumber][0]-molecule_com[0])-x0
275             yDistance = (stored.pos[atomNumber][1]-molecule_com[1])-y0
            zDistance = (stored.pos[atomNumber][2]-molecule_com[2])-z0
277             distance=sqrt(math.pow(xDistance,2)+math.pow(yDistance,2)+math.pow(zDistance
,2))
279
            if (distance<minimumDistance):
                minimumDistance=distance
281                 closestAtomDistance=atomNumber
283
        # zapamietuje wsp. nablizszego atomu
        closestAtomX=stored.pos[closestAtomDistance][0]-molecule_com[0]
285         closestAtomY=stored.pos[closestAtomDistance][1]-molecule_com[1]
        closestAtomZ=stored.pos[closestAtomDistance][2]-molecule_com[2]
287
        return (closestAtomX/scale, closestAtomY/scale, closestAtomZ/scale)
289
# Funkcja znajduje najblizszy region do zadanej pozycji
291 def find_closest_region(x0,y0,z0):
    # jesli nie ma regionow, to zwroc najblizszy atom
293     if len(regions)==0:
        return find_closest_atom(x0, y0, z0)
295
        min_distance=sys.float_info.max # startujemy od najwiekszej mozliwej wielkosci
297         closestX,closestY,closestZ = 0,0,0
        closestRegionId=0
299
        for region in regions: # iteruje po regionach
            coords=regions[region] # pobieram z mapy regionow wspolrzedne
301             x_dist = (coords[0]-molecule_com[0])-x0
            y_dist = (coords[1]-molecule_com[1])-y0
303             z_dist = (coords[2]-molecule_com[2])-z0
            distance=sqrt(math.pow(x_dist,2)+math.pow(y_dist,2)+math.pow(z_dist,2))
305
            if (distance<min_distance):
                min_distance=distance
309                 closestRegionId=region
                closestX = coords[0]-molecule_com[0]
311                 closestY = coords[1]-molecule_com[1]
                closestZ = coords[2]-molecule_com[2]
313

```

```

315     # tworze nowy selection do najblizszego regionu
316     selectCommand="resi "
317     for resi in closestRegionId:
318         selectCommand+=resi+" "
319
320     if REGION_COLORING:
321         cmd.color("red","target")
322         cmd.select("region",selectCommand)
323         cmd.color("yellow","region")
324     else:
325         cmd.color("red","target")
326         cmd.select("region",selectCommand)
327
328     return [closestRegionId,(closestX/scale),(closestY/scale),(closestZ/scale),
329            min_distance]
330
331 # Główna funkcja wykonawcza programu
332 def vrpn_client():
333     global structureMappingFile,templateStructureFile,phantomIp
334
335     tracker = vrpn_Tracker.vrpn_Tracker_Remote(phantomIp.get())
336     vrpn_Tracker.register_tracker_change_handler(tracker_handler)
337     vrpn_Tracker.vrpn_Tracker_Remote.register_change_handler(tracker, None,
338     vrpn_Tracker.get_tracker_change_handler())
339
340     button = vrpn_Button.vrpn_Button_Remote(phantomIp.get())
341     vrpn_Button.register_button_change_handler(button_handler)
342     vrpn_Button.vrpn_Button_Remote.register_change_handler(button, None, vrpn_Button.
343     get_button_change_handler())
344
345     forceDevice = vrpn_ForceDevice.vrpn_ForceDevice_Remote(phantomIp.get())
346     vrpn_ForceDevice.register_force_change_handler(force_handler)
347     vrpn_ForceDevice.vrpn_ForceDevice_Remote.register_force_change_handler(forceDevice
348     , None, vrpn_ForceDevice.get_force_change_handler())
349
350     draw_xyz_axes(0,0,0)
351
352     draw_template_structure(templateStructureFile.get())
353     draw_target_structure(targetStructureFile.get())
354     load_mapping_file(structureMappingFile.get())
355
356     new_calculate_regions_com()
357
358     global regionId, regionX, regionY, regionZ
359     global x,y,z
360     global templateCOM,regionCOM
361
362     sleep(1) # czekam az sie wszystko polaczy i narysuje
363
364 # główna pętl programu
365 while IS_RUNNING:
366     if (not AUTO_ZOOMING):
367         cmd.zoom('all')
368
369         tracker.mainloop()
370         forceDevice.mainloop()
371         button.mainloop()
372
373         # obliczam aktualny srodek ciiezkości wzorca
374         templateCOM=cmd.centerofmass("template")
375
376         # znajduje nablizszy dla wzorca region w czasteczce celu
377         # do ktorego bede przyciagac wzorzec/wzказnik
378         region=find_closest_region(templateCOM[0],templateCOM[1],templateCOM[2])
379
380         regionCOM=cmd.centerofmass("region")
381
382         print "RMSD: ", cmd.rms_cur("template","region")

```



```

379         # aktualizacja interfejsu
regionId.delete(0,'end') #todo: zmienic na zmienna textvariable
381         regionId.insert(0,region[0])
regionX.delete(0,'end')
383         regionX.insert(0,region[1])
regionY.delete(0,'end')
385         regionY.insert(0,region[2])
regionZ.delete(0,'end')
387         regionZ.insert(0,region[3])
regionTemplateDistance.delete(0,'end')
389         regionTemplateDistance.insert(0,region[4])

391         if FORCES_ENABLED:
            force=100 # skalarna wartosc sily |F|
393             forceX = (region[1]-trackerX) # wersor X sily
            forceY = (region[2]-trackerY) # wersor Y sily
395             forceZ = (region[3]-trackerZ) # wersor Z sily
            forceDevice.setFF_Origin(trackerX, trackerY, trackerZ)
397             forceDevice.setFF_Force(force*forceX, force*forceY, force*forceZ)
            forceDevice.setFF_Jacobian(force,0,0, 0,force,0, 0,0,force)
399             forceDevice.setFF_Radius(0.1)
            forceDevice.sendForceField()
401         else:
            forceDevice.setFF_Origin(0,0,0)
403             forceDevice.setFF_Force(0,0,0)
            forceDevice.setFF_Jacobian(0,0,0, 0,0,0, 0,0,0)
405             forceDevice.setFF_Radius(0.0)
            forceDevice.sendForceField()
407
            # rysuje linie laczaca wzorzec/wskaznik z najblizszym regionem/atomem
409             cmd.delete('link')
            cmd.load_cgo([CYLINDER, templateCOM[0],templateCOM[1],templateCOM[2], (
regionCOM[0]), (regionCOM[1]), (regionCOM[2]), 0.1, 255, 255, 255, 255, 255],
'link')
411 #             cmd.load_cgo([CYLINDER, templateCOM[0],templateCOM[1],templateCOM[2], (region
[1]*scale), (region[2]*scale), (region[3]*scale), 0.1, 255, 255, 255, 255, 255,
255], 'link')

413             cmd.delete("*")
x=y=z=0
415
# Zatrzymuje działanie programu i powracam do okna konfiguracji
417 def stop():
    global IS_RUNNING
419     IS_RUNNING = False
    sleep(1)
421     configWindow()

423 # Zmieniam flagę kolorowania regionu struktury
def doColorRegion():
425     global REGION_COLORING

427     if REGION_COLORING:
        REGION_COLORING=False
429     else:
        REGION_COLORING=True
431
# Zmieniam flagę zwrotnej projekcji sił
433 def doEnableForces():
    global FORCES_ENABLED
435
    if FORCES_ENABLED:
        FORCES_ENABLED=False
437     else:
        FORCES_ENABLED=True
439

441 # Wyświetlam okno ze statystykami
def statsWindow():
443     global currentWindow,IS_RUNNING

```

```

IS_RUNNING = True
445 thread.start_new_thread(vrpn_client, ())

447 currentWindow.destroy()

449 w=640
h=400
451 currentWindow=Tk()
currentWindow.title("Interaktywna eksploracja oraz lokalne dopasowania
strukturalne")
453 x=currentWindow.winfo_screenwidth()/2 - w/2
y=currentWindow.winfo_screenheight()/2 - h/2
455 currentWindow.geometry("%dx%d+%d+%d" % (w,h,x,y))
currentWindow.attributes('-topmost', 1)
457 currentWindow.resizable(False, False)
currentWindow.grid_columnconfigure(0, weight=1)
459 currentWindow.grid_columnconfigure(1, weight=1)

461 global regionId, regionX, regionY, regionZ, regionTemplateDistance, rmsdEntry
group=LabelFrame(currentWindow, text="Wspolrzedne")
463 group.grid(column=0,padx=5,pady=5,sticky='WE')
Label(group, text="ID regionu:", anchor=W).grid(row=0,column=0,sticky='WE')
465 regionId=Entry(group, width=30, justify=CENTER)
regionId.grid(row=0,column=1,sticky='W')
467 Label(group, text="COM X regionu:", anchor=W).grid(row=1,column=0,sticky='WE')
regionX=Entry(group, width=20, justify=CENTER) # todo: odswiezac te pola po
textvariable zamiast tego co jest
469 regionX.grid(row=1,column=1,sticky='W')
Label(group, text="COM Y regionu:", anchor=W).grid(row=2,column=0,sticky='WE')
471 regionY=Entry(group, width=20, justify=CENTER) # todo: jw.
regionY.grid(row=2,column=1,sticky='W')
473 Label(group, text="COM Z regionu:", anchor=W).grid(row=3,column=0,sticky='WE')
regionZ=Entry(group, width=20, justify=CENTER) # todo: jw.
475 regionZ.grid(row=3,column=1,sticky='W')
Label(group, text="Odleglosc wzorca i regionu:", anchor=W).grid(row=4,column=0,
sticky="WE")
477 regionTemplateDistance=Entry(group, width=20, justify=CENTER) # todo: jw.
regionTemplateDistance.grid(row=4,column=1,sticky='W')
479

group=LabelFrame(currentWindow, text="Opcje")
481 group.grid(column=1,row=0,sticky='NSWE',pady=5,padx=5)
colors=Checkbutton(group, text="Koloruj region", command=doColorRegion)
483 colors.select()
colors.grid(row=0,sticky="W")
485 Checkbutton(group, text="Projekcja sil", command=doEnableForces).grid(row=1,sticky="
W")

487

# RMSD
489 # group=LabelFrame(currentWindow, text="Wykres RMSD (Root-mean-square diviation)")
# group.grid(row=1, columnspan=2, sticky="NSWE", padx=5, pady=5)
491 # rmsdEntry=Entry(group, width=20, justify=CENTER)
# Label(group, text="\n\ntu bedzie wykres...\n\n").grid(sticky="WE")
493

# spacer
495 Frame(currentWindow).grid(sticky="NSWE")

497 # przyciski
group=Frame(currentWindow)
499 group.grid(row=3, columnspan=2)
stopButton=Button(group, text="STOP", command=stop)
501 stopButton.grid()

503 currentWindow.mainloop()

505 # Otwiera okno wyboru pliku wzorca (*.pdb)
def chooseTemplateStructureFile():
507     global templateStructureFile
     templateStructureFile.set(askopenfilename(filetypes=((("PDB", "*.pdb")), ("All

```

```

509         files", ".*")) ))
    print templateStructureFile.get()

511 # Otwiera okno wyboru pliku mapującego (*.map)
def chooseStructureMappingFile():
513     global structureMappingFile
    structureMappingFile.set(askopenfilename( filetypes=(("MAP", "*.map"), ("All files", ".*")) ))
515     print structureMappingFile.get()

517 # Otwiera okno wyboru pliku struktury
def chooseTargetStructureFile():
519     global targetStructureFile
    targetStructureFile.set(askopenfilename( filetypes=(("PDB", "*.pdb"), ("All files", ".*")) ))
521     print targetStructureFile.get()

523 # Otwiera okno konfiguracji programu
def configWindow():
525     global currentWindow, templateStructureFile, structureMappingFile, phantomIp, targetStructureFile

527     currentWindow.destroy()

529     w=640
    h=480
531     currentWindow=Toplevel()
    currentWindow.title("Interaktywna eksploracja oraz lokalne dopasowania strukturalne")
533     x=currentWindow.winfo_screenwidth()/2 - w/2
    y=currentWindow.winfo_screenheight()/2 - h/2
535     currentWindow.geometry("%dx%d+%d+%d" % (w,h,x,y))
    currentWindow.attributes('-topmost', 1)
537     currentWindow.resizable(False, False)

539 #     Wybor wzorca
    message="Wzorzec jest struktura, ktora bedziemy probowali dopasowac do czasteczki bazowej.\nWzorzec moze stanowic wycinek czasteczki bazowej, np. jakas struktura drugorzadowa"
541     group=LabelFrame(currentWindow, text="Wzorzec", padx=5, pady=5)
    group.pack( fill=BOTH, padx=5, pady=5)
543     Label(group, text=message, anchor=W).pack( fill=BOTH)
    Entry(group, textvariable=templateStructureFile, width=50, state="readonly").pack(
545     pady=5, side=LEFT)
    Button(group, text="Wybierz plik", command=chooseTemplateStructureFile).pack( side=LEFT)

547 #     Wybor pliku mapowania
    group=LabelFrame(currentWindow, text="Plik mapowania", padx=5, pady=5)
549     group.pack( fill=BOTH, padx=5, pady=5)
    Label(group, text="Tutaj wybierz plik mapowania", anchor=W).pack( fill=BOTH)
551     Entry(group, textvariable=structureMappingFile, width=50, state="readonly").pack( pady=5, side=LEFT)
    Button(group, text="Wybierz plik", command=chooseStructureMappingFile).pack( side=LEFT)

553 #     Wybor identyfikatora PDB
    group=LabelFrame(currentWindow, text="Identyfikator PDB", padx=5, pady=5)
555     group.pack( fill=BOTH, padx=5, pady=5)
    Label(group, text="Tutaj wybierz plik ", anchor=W).pack( fill=BOTH)
557     Entry(group, textvariable=targetStructureFile, width=50, state="readonly").pack( pady=5, side=LEFT)
    Button(group, text="Wybierz plik", command=chooseTargetStructureFile).pack( side=LEFT)

561 #     ustawianie adresu IP serwera VRPN
563     group=LabelFrame(currentWindow, text="Adres IP serwera VRPN", padx=5, pady=5)
    group.pack( fill=BOTH, padx=5, pady=5)

```

```

565     Entry ( group , justify=CENTER, textvariable=phantomIp , width=30) . pack ( pady=5, side=LEFT)
567 #     spacer
568     Frame ( currentWindow ) . pack ( padx=5, expand=TRUE)
569 #     przyciski
570     group=Frame ( currentWindow )
571     group . pack ( padx=5, pady=5)
572     Button ( group , text=" Anuluj " , command=currentWindow . destroy ) . pack ( side=RIGHT)
573     Button ( group , text=" Dalej " , command=statsWindow ) . pack ()
574
575     currentWindow . mainloop () ;
576
577 # Otwiera okno powitalne programu
578 def helloWindow () :
579     global currentWindow , templateStructureFile , structureMappingFile , phantomIp ,
580     targetStructureFile
581
582     w=640
583     h=480
584     currentWindow=Toplevel ()
585     currentWindow . title (" Interaktywna eksploracja oraz lokalne dopasowania
586     strukturalne")
587     x=currentWindow . wininfo _screenwidth () /2 - w/2
588     y=currentWindow . wininfo _screenheight () /2 - h/2
589     currentWindow . geometry ( "%dx%d+%d+%d" % ( w, h, x, y))
590     currentWindow . attributes ( '-topmost' , 1)
591     currentWindow . resizable ( False , False)
592
593     helloMsg="\
594     Użytkowniku \
595     \nTa aplikacja umożliwia interaktywne przeglądanie struktur biopolimerów \
596     \n(białek oraz kwasów nukleinowych) , a także wykonywanie lokalnych \
597     \ndopasowań (uliniowień) pomiędzy załadowanymi strukturami . \
598     \nProgram wykorzystuje możliwości udostępnione przez urządzenie \
599     \nPhantom Omni , pakiet PyMOL oraz bibliotekę VRPN . \
600     \nAutor programu: Paweł Tomaszewski\
601     "
602     group=LabelFrame ( currentWindow , text=" Witaj " , padx=5, pady=5)
603     group . pack ( fill=BOTH, padx=5, pady=5, expand=True)
604     Label ( group , text=helloMsg ) . pack ( fill=BOTH, side=LEFT)
605     dnaImage=PhotoImage ( file="dna . gif")
606     Label ( group , image=dnaImage ) . pack ( fill=BOTH)
607
608     group=Frame ( currentWindow )
609     group . pack ( padx=5, pady=5)
610     Button ( group , text=" Anuluj " , command=currentWindow . destroy ) . pack ( side=RIGHT)
611     Button ( group , text=" Dalej " , command=configWindow ) . pack ()
612
613 #     inicjalizacja zmiennych globalnych
614     templateStructureFile=StringVar ( value=os . getcwd ()+" / helix _chain _a . pdb")
615     structureMappingFile=StringVar ( value=os . getcwd ()+" / 1 fg0 _ helix . map")
616     targetStructureFile=StringVar ( value=os . getcwd ()+" / 1 fg0 . pdb")
617     phantomIp=StringVar ( value="phantom0@10 . 21 . 2 . 136")
618
619     currentWindow . mainloop () ;
620
621 # Dodaje pozycję w menu PyMOL i uruchamia aplikację
622 def __init__ ( self ) :
623     self . menuBar . addmenuitem ( ' Plugin ' , ' command ' , ' VRPN ' ,
624     label = ' Interactive structure explorer ' , command = lambda s:self : helloWindow ())

```