GRAFIKA – PROJEKT

Temat 04: Funkcje 4D

Damian Raczyński
Kacper Osuch
Michał Domin

1) Założenia wstępne przyjęte w realizacji projektu

Celem projektu było napisanie programu, który dla zadanej funkcji 4D $f(x, y, z)$ będzie rysował szereg przekrojów wzdłuż kierunku zdefiniowanego wektorem $[a, b, c]$ w odległościach d między przekrojami.

Przyjęliśmy, że do zobrazowania funkcji na płaszczyźnie wykorzystamy metodę dwuwymiarowych przekrojów. Wykonany program pozwala na wykonanie przekrojów wektorami $[1, 0, 0]$, $[0, 1, 0]$, $[0, 0, 1]$, $[1, 1, 0]$, $[1, 0, 1]$, $[0, 1, 1]$. Wartości funkcji są reprezentowane jako punkty o barwie zmieniającej się od niebieskiego do czerwonego, gdzie niebieski jest wartością najniższą w zbiorze punktów.

Przekroje można zmieniać poprzez wciskanie przycisków na ekranie. Przycisk ze strzałką w lewo $<$ przenosi na przekrój „poprzedni" a przycisk ze strzałką w prawo $>$ przenosi na przekrój następny. Dla każdego zestawu danych tworzone jest 5 przekrojów.

Wprowadzana do programu funkcja 4D jest w postaci pliku tekstowego $*.dat$. Każda z linii musi zawierać 4 liczby – współrzędne x, y, z oraz wartość funkcji w punkcie o podanych współrzędnych. Ponadto punkty funkcji powinny być rozłożone w regularnej sześciennej siatce o stałym skoku dz, dy, dz wzdłuż każdej z osi.

Istnieje również przycisk, który pozwala na zapisanie aktualnego przekroju do pliku w formacie $*.bmp$.

2) Analiza projektu

a) Specyfikacja danych wejściowych

Program wczytuje plik `*.dat` zawierający cztery kolumny danych, oddzielonych spacją, po jednej linii dla każdego punktu:

x	y	z	$value$

Oczekujemy, że wartości będą rozmieszczone równomiernie, w sześciennej siatce. Wartości współrzędnych mogą się zaczynać od dowolnej wartości ≥ 0. Wartości danych punktów (value) mogą być dowolne.

b) Opis oczekiwanych danych wyjściowych

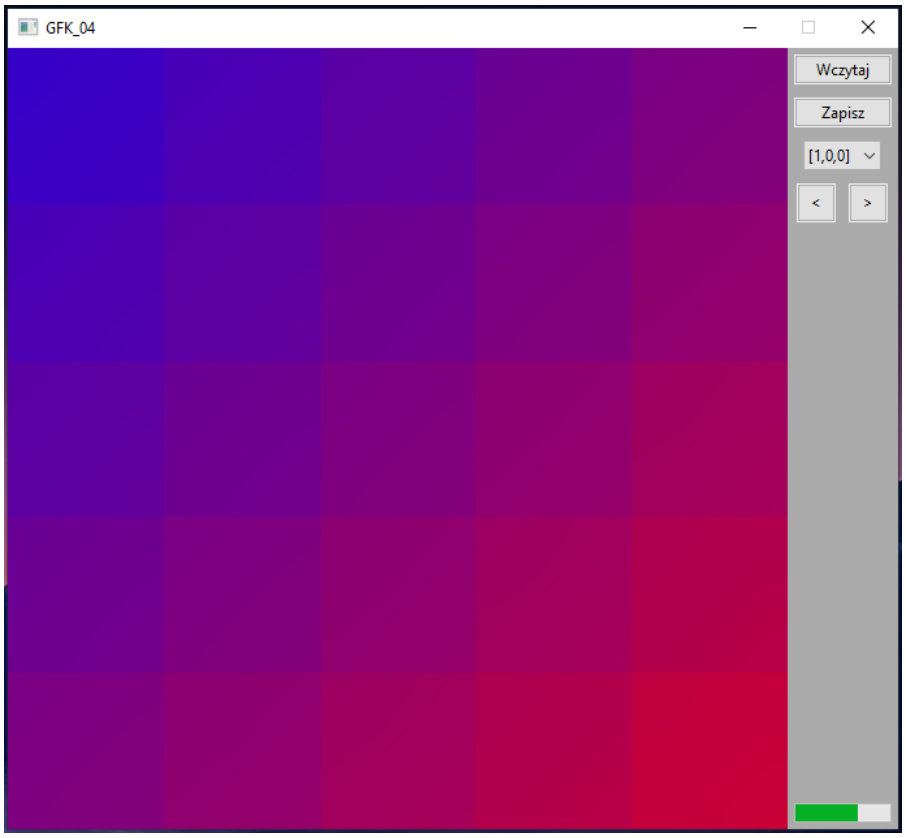
Zapisanie przekroju skutkuje stworzeniem bitmapy (`*.bmp`) o rozmiarze 600×600 pikseli. Bitmapa zawiera aktualnie wyświetlany przekrój.

c) Zdefiniowanie struktur danych

Zdecydowana większość danych przechowywana jest w strukturach typu `std::vector` gdzie każdy punkt jest reprezentowany jako 4-elementowy `std::array`. Punkty są sortowane przy wczytywaniu aby móc szybciej się do nich dostać.

d) Specyfikacja interfejsu użytkownika

Interfejs programu składa się z dwóch głównych elementów – pola wyświetlającego przekrój oraz menu bocznego zawierającego wszystkie elementy do obsługi programu.



Rysunek 1 Interfejs użytkownika w omawianym projekcie.

Panel wyświetla wybrany przekrój o rozmiarze 600×600 pikseli. Kolor danego punktu zależy od jego wartości. Płynnie przechodzi od niebieskiego dla najniższych do czerwonego dla najwyższych. Wartość największa i najniższa pozyskiwane są z całego pliku, więc można porównywać wartości pomiędzy różnymi przekrojami.

Menu główne zawiera następujące elementy:

- Przycisk Wczytaj – otwiera okno dialogowe pozwalające wybrać i wczytać plik `*.dat`.
- Przycisk Zapisz – otwiera okno dialogowe pozwalające zapisać obecnie wyświetlany przekrój do pliku `*.bmp`.
- Lista wyboru wektora – Domyślnie wektor $[1, 0, 0]$ – pozwala wybrać wektor, wzdłuż którego będziemy pobierać kolejne przekroje. Do wyboru są następujące wektory: $[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 1, 0], [1, 0, 1], [0, 1, 1]$. Wybranie wektora skutkuje przesunięciem obecnie wyświetlanego przekroju na pierwszy.
- Przyciski " $<$ " i " $>$ " – poruszanie się pomiędzy kolejnymi przekrojami. Dostępnych jest 5 przekrojów dla każdego wektora.

- Pasek postępu – Obrazuje postęp w tworzeniu przekroju. Jest widoczny tylko wtedy, gdy program obecnie przetwarza przekrój. Warto zaznaczyć, że nie wyświetla się on w fazie wczytywania danych z pliku, co jest szczególnie widoczne gdy wczytujemy bardzo duże zestawy danych (`132k_points.dat` oraz `center.dat` z plików przykładowych).

Elementy sterujące automatycznie „wyszarzają się" kiedy nie powinny być dostępne – np. strzałki gdy dotrzemy do ostatniego dostępnego przekroju lub przycisk „Zapisz" gdy nie wczytaliśmy jeszcze żadnego pliku.

e) Wyodrębnienie i zdefiniowanie zadań

Najważniejszymi modułami projektu są:

- Wczytywanie pliku – Wczytanie danych do pamięci, sortowanie, wyodrębnienie potrzebnych minimalnych i maksymalnych wartości, odblokowanie przycisków sterujących.
- Tworzenie płaszczyzny punktów dookoła których będziemy poszukiwać punktów w zestawie danych. Płaszczyzna ta jest zgodna z obecnie wybranym wektorem i przekrojem.
- Poszukiwanie punktów zestawu danych które są w pobliży punktów wyznaczonej płaszczyzny, wyciąganie i przetwarzanie ich wartości
- Wyświetlanie przekroju na ekranie

f) Decyzja o wyborze narzędzi programistycznych

Korzystaliśmy ze środowiska Visual Studio 2019 oraz biblioteki wxWidgets, głównie ze względu doświadczenia nabytego podczas laboratoriów z grafiki komputerowej. Do wygenerowania GUI posłużył nam program wxFormBuilder.

3) Opis projektu, podział pracy i analiza czasowa

a) Michał Domin
- Tworzenie płaszczyzny – 1 dzień
- Algorytm określający wartości punktów – 8 godzin
- Uzupełnienie siatki wartościami – 3 godziny
- Porządki w kodzie – 1 godzina

b) Kacper Osuch
- Interfejs programu – 3 godziny
- Dokumentacja – 1 dzień
- Przejścia między płaszczyznami – 2 godziny
- Analiza wydajności programu – 4 godziny

c) Damian Raczyński
- Blokady przycisków na interfejsie – 2 godziny
- Przyciski odczytu, zapisu – 3 godziny
- Wypisywanie na ekran – 5 godzin
- Dokumentacja – 1 dzień

4) Opracowanie i opis niezbędnych algorytmów

Metoda Sheparda[1] - sposób aproksymacji wielowymiarowej dla rozproszonych zbiorów znanych punktów aproksymacyjnych. Ogólna postać metody Sheparda dla znalezienia wartości aproksymowanej u dla danego punktu $\boldsymbol{x}$ ma formę funkcji:

$$u(\boldsymbol{x}) = \frac{\sum_{k=0}^{N} w_k(\boldsymbol{x}) u_k}{\sum_{k=0}^{N} w_k(\boldsymbol{x})}, gdzie$$

$$w_k(\boldsymbol{x}) = \frac{1}{d(\boldsymbol{x}, \boldsymbol{x}_k)^p},$$

- $\boldsymbol{x}$ – dowolny punkt aproksymowany,
- $\boldsymbol{x}_k$ – znany punkt aproksymacyjny,
- d – określony operatorem metryki,
- N – całkowita liczba punktów aproksymacyjnych,
- p – parametr.

W tym przypadku wartość współczynnika wagowego zmniejsza się wraz ze wzrostem odległości pomiędzy punktem aproksymowanym $\boldsymbol{x}$ a punktem aproksymującym $\boldsymbol{x}_k$. Dla $0 < p < 1$ $u(\boldsymbol{x})$ ma ostre wierzchołki nad punktami aproksymującymi, a dla $p > 1$ jest gładka. Najczęściej przyjmuje się $p = 2$.

Pozostałe utworzone/zmodyfikowane przez nasz zespół algorytmy w programie zostały przedstawione w następnej części raportu (**5) Kodowanie**).

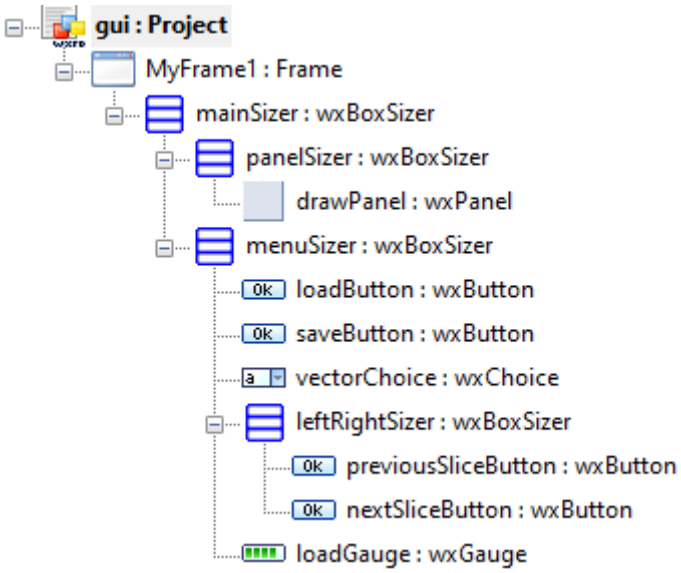
5) Kodowanie

Baza programu została utworzona przy użyciu programu wxFormBuilder[2], dlatego też pliki $main.cpp$, $gui.h$ oraz $gui.cpp$ z projektu nie zostaną dokładnie opisane – są plikami automatycznie wygenerowanymi przez wyżej wymieniony program.

a) Interfejs programu

Jedyną ramką programu jest $MyFrame1$ ($Frame$), które zawiera w sobie wszystkie elementy interfejsu widocznego dla użytkownika programu. Ramka jest stałego rozmiaru, nie można rozciągnąć programu na cały ekran. Istnieje jednak opcja minimalizacji okna oraz zamknięcia programu.

[1] Metoda Sheparda - Wikipedia, wolna encyklopedia
[2] wxFormBuilder - Wikipedia, the free encyclopedia



Rysunek 2 Schemat utworzonego projektu w programie wxFormBuilder.

Szczegółowy opis elementów (pomijając funkcje, które zostały dokładnie opisane w dalszej części):

- Główny horyzontalny sizer $mainSizer$ ($wxBoxSizer$), który dba o prawidłowe rozłożenie wszystkiego w interfejsie. Jego rozmiar jest automatycznie ustalany według tego, co znajduje się w jego wnętrzu. Obszar okna jest podzielony między dwa niżej opisane sizer'y.
- Wertykalny sizer $panelSizer$ $wxBoxSizer$), który odpowiada za prawidłowe ustawienie elementu opisanego niżej, na którym zostaną wyrysowane przekroje wprowadzanych do programu wartości funkcji.
- $drawPanel$ ($wxPanel$) o rozmiarze 600×600 na którym rysowane są przekroje funkcji wprowadzanej do programu. Podczas dokonywania zapisu wyniku do pliku, do pliku zostaje przekazane, co jest w danym momencie widoczne na opisywanym panelu.
- Kolejny wertykalny sizer $menuSizer$ ($wxBoxSizer$), który zawiera w sobie wszystkie przyciski oraz kontrolki, które użytkownik może wcisnąć lub obserwować. Rozmiar tego sizera również jest automatyczny.
- Przycisk $loadButton$ ($wxButton$), który pozwala na wczytanie do programu danych w formacie pliku $*.dat$. Po jego wciśnięciu wyskakuje okno wyboru pliku, który może się znajdować w dowolnym miejscu na komputerze. Program resetuje wybrany przekrój do najwcześniejszego możliwego.
- Przycisk $saveButton$ ($wxButton$), który pozwala na zapis aktualnie obserwowanego przekroju do pliku w formacie $*.bmp$. Po jego wciśnięciu wyskakuje okno wyboru lokalizacji zapisu pliku oraz nadania mu dowolnej nazwy.
- Menu wyboru $vectorChoice$ ($wxChoice$), który jest listą możliwych do wyboru przekrojów. Z założenia jest do wyboru jeden z sześciu wektorów ($[1,0,0], [0,1,0], [0,0,1], [1,1,0], [1,0,1], [0,1,1]$), gdzie domyślnie jest wybrany pierwszy wektor. Po wyborze nowego wektora, program pokazuje pierwszy przekrój funkcji.
- Horyzontalny sizer $leftRightSizer$ ($wxBoxSizer$), który zawiera w sobie przyciski do przechodzenia między płaszczyznami.
- Przyciski $previousSliceButton$, $nextSliceButton$ ($wxButton$), które odpowiednio przesuwają płaszczyzny na poprzednią/następną.
- Wstępnie niewidoczny przez użytkownika pasek postępu $loadGauge$($wxGauge$), który pokazuje postęp generowania płaszczyzny rysowanej na ekran.

Zmienne w klasie $guiMyFrame1$:

- $MemoryBitmap$ – zmienna typu $wxBitmap$, która reprezentuje to, co zostaje wypisywanie na ekranie. Przy wywołaniu funkcji rysującej $Repaint()$, to właśnie ta bitmapa zostaje przedstawiona na ekranie.
- $image$ – zmienna typu $wxImage$, która służy do przygotowania punktów siatki 600×600 w metodzie $DrawMap()$, widocznej później na ekranie. Obraz ten zostaje przypisany do bitmapy $MemoryBitmap$.
- $minVal, maxVal$ – zmienne typu $double$, które reprezentują odpowiednio minimalną/maksymalną wartość, jaką osiąga wprowadzona do programu funkcja.
- $data$ – zmienna typu $std::vector < std::array < double, 4 >>$, która przechowuje w sobie wszystkie wartości otrzymane z pliku wejściowego $*.dat$. Każdy element wektora zawiera przyjęte z dokumentu parametry $x, y, z, f(x, y, z)$.
- $datas$ – zmienna typu $std::vector < std::vector < std::vector < std::array < double, 2 >>>>$, która jest posortowanym 3-wymiarowym wektorem punktów. Dla każdego z parametrów można się dostać do odpowiadającej mu dowolnej wartości:
    - $datas[0 - i][0][0][0]$ – wartości x,
    - $datas[0 - i][1 - (i + 1)][0][0]$ – wartości y,
    - $datas[0 - i][1 - (i + 1)][1 - (i + 1)][0]$ – wartości z,
    - $datas[0 - i][1 - (i + 1)][1 - (i + 1)][1]$ – wartości $f(x, y, z)$,

    gdzie i to ilość wartości w danym wymiarze.

- ix, iy, iz – zmienne typu int, które są wykorzystywane w metodzie $getVal(\dots)$. Reprezentują one punkty pomocnicze o tych samych wartościach względem wprowadzanego do metody punktu $x0$.
- $minDim, maxDim$ – zmienne typu $double$, które reprezentują odpowiednio minimalne/maksymalne położenie punktu we wprowadzanej funkcji.
- $modifier$ – zmienna typu $double$, która reprezentuje wartość niezerowej współrzędnej punktu należącego do danego przekroju. To ta wartość jest zmieniana podczas przechodzenia między płaszczyznami.
- $currentSlice$ – zmienna typu int, informująca o tym, który przekrój jest aktualnie widoczny. Ta wartość jest wykorzystywana do kontroli na którym przekroju się przebywa w danym momencie.

Metody w klasie $guiMyFrame1$:

- $loadButtonClick(wxCommandEvent\& event)$ – metoda pokazuje okno z opcją wczytania do programu pliku o rozszerzeniu $*.dat$. W przypadku anulowania procesu, program zamyka okno i nic się nie dzieje. Gdy nastąpi pomyślne wybranie pliku, następuje jego odczyt w następujący sposób:
    - Resetowane zostają wektory $data, datas$,
    - Ustawione wstępnie zostają wartości ix, iy, iz,
    - Dla pierwszego wiersza z pliku zostają wpisane wartości $x, y, z, f(x, y, z)$ oraz ustawione $minVal, maxVal, minDim, maxDim$,
    - Wykonana zostaje pętla przez cały plik, która dla każdej linii:
        - Wpisuje do $data$ wartości $x, y, z, f(x, y, z)$,
        - Jeżeli jest spełniona zależność $f(x, y, z) < minVal$ następuje aktualizacja $minVal$ oraz analogicznie dla $f(x, y, z) > maxVal$ nastąpiłaby aktualizacja $maxVal$,

        - Wykonywana jest operacja $minDim = std::min(\{x, y, z, minDim\})$ oraz $maxDim = std::max(\{x, y, z, maxDim\})$.
    - Następuje sortowanie $data$,
    - Następuje uzupełnienie $datas$, czyli posortowanego trójwymiarowego wektora wartościami z $data$ (przekształcanie wektora punktów na posortowany trójwymiarowy wektor punktów),
    - W trybie debugowania na konsoli pojawia się informacja o wartościach $minDim, maxDim$,
    - Ustawiane są parametry $currentSlice$, $modifier$, po czym zostaje wykonana metoda $DrawMap()$,
    - Następuje odblokowanie elementów na interfejsie: $nextSliceButton$, $vectorChoice$, $saveButton$ oraz zablokowanie $previousSliceButton$ (zaczynamy od najwcześniejszego przekroju).
- $saveButtonClick(wxCommandEvent\&\ event)$ – metoda otwiera okno zapisu pliku, gdzie użytkownik może zapisać do pliku $*.bmp$ to, co jest w danym momencie widoczne na $drawPanel$. Plik może zostać utworzony w dowolnym miejscu, jak również można nadpisać już istniejący plik.
- $OnChoice(wxCommandEvent\&\ event)$ – metoda obsługująca dokonanie przez użytkownika wyboru wektora z listy. Gdy zostanie wybrany wektor, program ustawia odpowiednią wartość $modifier$, ustawiając również $currentSlice$ na wartość 1 – pierwszy, najwcześniejszy przekrój. Aby wszystko działało prawidłowo, zostaje zablokowany przycisk $previousSliceButton$ oraz odblokowany $nextSliceButton$. Podczas debugowania programu, metoda wypisuje na konsoli informację o zresetowaniu wartości $vectorChoice$.
- $previousSliceClick(wxCommandEvent\&\ event)$ – metoda obsługuje zmianę płaszczyzny na poprzednią, zmniejszając wartość $currentSlice$ o 1. Gdy ten parametr jest mniejszy bądź równy 1, metoda blokuje przycisk, sygnalizując, że jest się na najwcześniejszym przekroju oraz w przypadku, gdyby się wróciło z ostatniego przekroju, aktywuje opcję wyboru następnego przekroju. Następnym krokiem jest zmienienie wartości $modifier$ oraz wykonanie metody $DrawMap()$ aby wyrysować nowy przekrój na ekran.
- $nextSliceClick(wxCommandEvent\&\ event)$ – metoda obsługuje zmianę płaszczyzny na następną, zwiększając wartość $currentSlice$ o 1. Gdy ten parametr jest większy bądź równy 5, metoda blokuje przycisk, sygnalizując, że jest się na ostatnim przekroju oraz w przypadku, gdyby się ruszyło z pierwszego przekroju, aktywuje opcję wyboru poprzedniego przekroju. Następnym krokiem jest zmienienie wartości $modifier$ oraz wykonanie metody $DrawMap()$ aby wyrysować nowy przekrój na ekran.
- $guiMyFrame1(wxWindow*\ parent)$ – konstruktor klasy $guiMyFrame1$, który wykonuje następujące czynności:
    - Ustawia wartości $minVal, maxVal, minDim, maxDim$ na zero,
    - Ukrywa pasek postępu,
    - Tworzy pusty $MemoryBitmap$ o rozmiarze 600×600 oraz pusty $image$ o tym samym rozmiarze,
    - Ustawia zmienną $modifier$ na 1,
    - Uniemożliwia używanie wszystkich przycisków poza $loadButton$, przez co użytkownik musi wczytać dane z funkcji, aby zrobić cokolwiek innego (poza ewentualnym zamknięciem programu).
- $DrawMap()$ – metoda przygotowuje $MemoryBitmap$ do bycia wyrysowaną na ekran. Tworzone jest $wxMemoryDC$, które wskazuje na $MemoryBitmap$. Następnie następuje czyszczenie tego elementu przez wpisaniem nowych wartości. Tworzona jest płaszczyzna

z elementami z przekroju metodą $plaszczyzna()$, po czym następuje wpisanie wartości do tablicy $points$ $(unsigned\ char\ *)$, która jest danymi z $image$ (to zostanie przekazane do $MemoryBitmap$) w postaci parametrów R, G, B dla każdego punktu. Każdy punkt z tablicy ma wyznaczaną wartość $color$ a następnie przypisywane są wartości R, G, B w postaci punktów idących od barwy niebieskiej (najniższa wartość) do czerwonej (wartość najwyższa). Podczas tego procesu w prawym dolnym rogu widoczny jest pasek postępu, pokazujący jaka część $image$ została już zmieniona. Po zakończeniu tego procesu pasek postępu znika i do $MemoryBitmap$ zostaje wpisany $image$ ze zmienionymi wartościami. Metoda przy debugowaniu informuje o zakończeniu tego procesu, po czym wykonuje operację $Repaint()$.

- $Repaint()$ – metoda rysująca na $drawPanel$ to, co jest aktualnie przechowywane w $MemoryBitmap$. Założenie jest, że bitmapa jest niezmiennego rozmiaru 600×600. Metoda w trybie debugowania informuje o pomyślnym wyrysowaniu bitmapy na ekran.
- $getVal(std::array < double, 3 > \& x0)$ – metoda wyszukuje 8 najbliższych punktów a później wylicza wartość w punkcie nas interesującym używając metodę Sheparda. Zmienne w funkcji:
    - s – część górna w równaniu w metodzie Sheparda,
    - so – część dolna w równaniu w metodzie Sheparda,
    - o – odległość między punktem szukanym a utworzonym, otrzymana metodą $odl(v, v2)$,
    - v – przepisanie wprowadzanego do metody punktu 3-elementowego do punktu o czterech elementach (dodawane $f(x, y, z) = 0$). Niezbędne do poprawnego działania metody,
    - $v2$ – nowy punkt o współrzędnych najbliższego z punktów względem wprowadzanego punktu.

    Wszystkie operacje $else\ if$ oraz $else$ pozwalają na szybsze wyznaczenie wartości szukanego punktu.

- $plaszczyzna()$ – metoda wyznacza wartości przekroju dla aktualnego wektora i wybranego w danym momencie przekroju, zwracając wartości w przekroju w postaci płaszczyzny 600×600. Zmienne w metodzie:
    - $mnoznik$ – wartość od której zależy wysokość płaszczyzny o przedziale $(minDim, 2 * maxDim - minDim)$. W ramach zabezpieczenia, przekroczenie tych wartości przekierowuje na środek układu,
    - w – wszystkie punkty płaszczyzny, które zostają zwracane w metodzie,
    - Switch od aktualnej wartości wybranej w $vectorChoice$ dzieli się w zasadzie na dwa rodzaje przypadków (różniących się wyłącznie $w.push\_back(...)$ na końcach):
        - $case\ 0, 1, 2$ – sprawdzany jest najpierw zakres $mnoznik$, wyznaczana jest wartość $delta$ – odległość między punktami już w danym przekroju a na koniec następuje wpisywanie wartości do w,
        - $case\ 3, 4, 5$ – $mnoznik$ tworzy punkt $x0$ od którego są wyznaczane pozostałe wartości w przekroju. Następnie wyznaczane są wartości: max – szerokość przekroju, $delta$ – odległość między punktami już w danym przekroju, która jest zależna od tego, który z kolei jest to przekrój oraz $delta2$ – odległość między punktami już w danym przekroju na niezmiennej długości.

Na koniec metoda zwraca utworzoną płaszczyznę w.

- $odl(std::array<double,4>\ \&\ a, std::array<double,4>\ \&\ b)$ – odległość między punktami trójwymiarowymi $\sqrt{(a_x-b_x)^2+\left(a_y-b_y\right)^2+(a_z-b_z)^2}$. Metoda zwraca wartość -1 gdy podane jako argumenty punkty są identyczne.
- $compare(std::array<double,4>\ a, std::array<double,4>\ b)$ – metoda wykonująca porównanie współrzędnych punktów trójwymiarowych. Porównanie przebiega następująco:
    - Jeżeli $|a_x - b_x|$ jest mniejszy od 0.01:
        - Jeżeli $|a_y - b_y|$ jest mniejszy od 0.01:
            - Jeżeli $a_z < b_z$ zwracana jest prawda,
            - W przeciwnym wypadku zwracany jest fałsz.
        - W przeciwnym wypadku, jeżeli $a_y < b_y$ zwracana jest prawda,
        - W przeciwnym wypadku zwracany jest fałsz.
    - W przeciwnym wypadku, jeżeli $a_x < b_x$ zwracana jest prawda,
    - W przeciwnym wypadku zwracany jest fałsz.

6) Testowanie

Aby przetestować działanie naszego programu, wygenerowaliśmy za pomocą prostego skryptu różne zestawy danych:

- `simple.dat` – Prosty, mały zestaw danych w którym każdy punkt jest sumą współrzędnych,
- `132k_points.dat` – Zestaw danych o takim samym zakresie i wzorze jak powyższy, ale z dużo większą ilością punktów (około 132 tysiące),
- `center.dat` – Również bardzo duży zestaw danych, który ma tym wyższą wartość, im dalej od środka sześcianu się znajdujemy,
- `from1.dat` – Testowy zestaw danych, którego współrzędne nie zaczynają się od zera,
- `just_x.dat` – Zestaw danych, który dobrze przedstawia zmiany wektorów – wartość w danym punkcie zależy tylko od współrzędnej x,
- `powers.dat` – Zestaw danych w którym każdy punkt przyjmuje wartość zależną od współrzędnych tego punktu według wzoru $x^{1.5} + y^{1.2} + z$,
- `noise.dat` – Zestaw danych w którym wartość każdego punktu jest pseudolosowa.

Program działa prawidłowo z każdym z przetestowanych plików, z jednym wyjątkiem. Jeśli korzystamy z programu uruchomionego bezpośrednio, nie poprzez Visual Studio, bardzo duże pliki wczytują się, ale nie rysują pierwszej płaszczyzny automatycznie. Aby zobaczyć jakąkolwiek płaszczyznę, należy zmienić wektor lub kliknąć przycisk następnego przekroju – od tego momentu wszystko będzie działać poprawnie. Problem ten nie występuje kiedy uruchamiamy program korzystając z Visual Studio. Dzieje się tak prawdopodobnie dlatego, że wczytywanie pliku zajmuje kilka sekund, system Windows oznacza okno jako „nie odpowiada” i z jakiegoś powodu nie uruchamia się funkcja pokazująca pierwszy przekrój, jedyne co się dzieje to wczytanie pliku do pamięci.

Aby testować wydajność programu i sprawdzać które elementy są najmniej zoptymalizowane, korzystaliśmy z narzędzi wbudowanych w Visual Studio. Pozwoliły nam one znacznie skrócić czas wykonywania poszczególnych funkcji.

7) Wdrożenie, raport i wnioski

Wszystkie założenia projektu zostały zrealizowane. W przyszłości można by poprawić wydajność programu, tak aby wczytywanie i zmienianie plików oraz przekrojów było natychmiastowe. Obecnie trzeba na te operacje poczekać kilka sekund, lecz jest to i tak o wiele lepszy wynik niż uzyskiwaliśmy w pierwszych iteracjach programu, gdzie od wybrania pliku do wyświetlenia przekroju mijało kilkadziesiąt sekund. Nie udało nam się zrealizować wymagań rozszerzonych, czyli korzystania z wektorów o dowolnych współrzędnych, obsługi punktów nieregularnie rozłożonych w przestrzeni, map konturowych oraz drukowania.