Funkcje 4D

Autorzy: Miłosz Świerad, Bartłomiej Mucha, Oleksandr Shaforostov

1. Podział obowiązków

Miłosz Świerad napisał funkcje do generowania danych oraz funkcję do zapisywania danych z pliku do programu.

Bartłomiej Mucha stworzył GUI programu.

Oleksandr Shaforostov napisał funkcje do rysowania przekroju.

2. Opis projektu

Tematem projektu są funkcje 4D. Celem było stworzenie programu, który wczytuje plik zawierający zapisane w kolumnach kolejne wartości *x, y* oraz *z*, a także wartość funkcji *f(x, y, z)*, a następnie wyświetla przekrój funkcji. Przekroje wykonane wzdłuż wybranego przez użytkownika kierunku, który jest zdefiniowany wektorem [x,y,z]. Każdy przekrój reprezentuje rozkład wartości funkcji 4D w danej płaszczyźnie.

3.Założenia wstępne przyjęte w realizacji projektu

Funkcja 4D ma być wprowadzona do programu w postaci pliku tekstowego. Postawowa wersja programu może obsługiwać wektory: [1,0,0] [0,1,0] [0,0,1] [1,1,0] [1,0,1] [0,1,1]. Przechodzimy między przekrojami za pomocą suwaka. Wybrany przekrój można zapisać do pliku w formacie BMP.

4.Analiza projektu

4.1. Specyfikacja danych wejściowych

Na początku generowany jest plik tekstowy z danymi dla pewnej funkcji *f*. Wartości zapisane są w czterech kolumnach: *x y z f(x, y, z)*. Gdzie: *x, y* oraz *z* – współrzędne; *f(x,y,z)* – wartość funkcji 4D.

4.2. Opis oczekiwanych danych wyjściowych

Wynikiem działania programu jest narysowany przekrój dla danej funkcji 4D przy wybranym wektorze. Każda wartość jest reprezentowana jako punkt o barwie zmieniającej się od czerwonego do niebieskiego. Wynik można zapisać do pliku w formacie BMP.

4.3. Zdefiniowanie struktur danych

Pobrane z pliku dane o wartościach funkcji są przechowane w programie w kontenerze STL – vector.

W programie został zdefiniowany vector: std::vector<vector<double>> Function. W tej strukturze wiersz reprezentuje oś oraz wartość funkcji: indeks 0 – x, 1 – y, 2 – z, 3 – wartość funkcji. Kolumny reprezentują każdą kolejną wartość funkcji i jej argumentów. Wektor std::vector<vector<double>> DataMap jest wektorem pomocniczym został stworzony z myślą o

optymalizacji działania programu. W DataMap znajdują się tylko te dane z Function, które są potrzebne do wygenerowania bitmapy zadanego przekroju.

4.4. Specyfikacja intefejsu użytkownika

Przycisk „Load File” – uruchamia dialog wyboru pliku tekstowego jaki będzie wczytany do programu jako funkcja 4D.

Przycisk „Capture Picture” – uruchamia dialog zapisu aktualnej bitmapy. Dostępne formaty zapisu to BMP, JPG i PNG.

Pola tekstowe „Floating Vector” – pozwalają zdefiniować wektor, po jakim użytkownik będzie mógł się poruszać po zadanej funkcji 4D. Dostępne pola to X, Y i Z odpowiadające osiom układu. Uwaga: program rozróżnia w tych polach ‘0’ lub inna wartość.

Przycisk „Draw” – aktualizuje bitmapę zawierającą aktualny przekrój oraz wyświetla w panelu jej zawartość. Ten przycisk należy zastosować po każdej zmianie w interfejsie, w tym zmianie rozmiaru okna.

Suwak w ramce „State” – pozwala poruszać się po zadanej funkcji 4D względem wektora zdefiniowanego w ramce „Floating Vector”.

Panel – w nim jest wyświetlana bitmapa.

W programie użytkownik może zdefiniować wektor (Floating Vector), względem którego będzie się poruszał po zadanej funkcji 4D za pomocą pól tekstowych X, Y i Z, w które należy wpisać współrzędne. Za pomocą suwaka, operującego na wartościach procentowych, użytkownik może przeglądać przekroje zadanej funkcji poruszając się po wcześniej ustalonym wektorze. Każdorazowo wybór wektora i przekroju należy zatwierdzić przyciskiem „Draw”. Ma to związek z dużym czasem generowania się bitmapy, dlatego nie mogła być ona wyświetlana na bieżąco. Wyświetlanie jej na bieżąco doprowadzało do wyjątkowo powolnego działania programu. Przycisk „Load File” daje możliwość wybrania pliku tekstowego z danymi funkcji 4D i wprowadzenia tych danych do programu. W panelu zostaje wyświetlony wynik działania programu – bitmapa zawierająca narysowany przekrój. Przycisk „Capture Picture” pozwala zapisać aktualnie wyświetlany w panelu przekrój. Bitmapę zawierającą przekrój można zapisać do formatu BMP, JPG i PNG. Program umożliwia manipulowanie rozmiarem okna. Minimalny rozmiar okna to taki, w którym panel ma rozmiar 400x400 pikseli. Maksymalny to taki, na jaki pozwala rozmiar monitora. Po każdej zmianie rozmiaru okna należy odświeżyć panel przyciskiem „Draw”.

4.5. Wyodrębnienie i zdefiniowanie zadań

Pierwsze zadanie: wygenerować plik z danymi. Dalej tworzenie interfejsu programu. Następnie należało napisać funkcje do wprowadzenia danych z wygenerowanego wcześniej pliku do programu. Kolejnym zadaniem było stworzenie funkcji do rysowania przekrojów. I na koniec – zapis wyniku do pliku w formacie BMP.

4.6. Decyzja o wyborze narzędzi programistycznych

Program został napisany w języku C++, z wykorzystaniem biblioteki wxWidgets 2.9.3 i środowiska wxDev-Cpp. Taki wybór był dokonany z kilku przyczyn: każdy członek zespołu zna

język C++; wxWidgets był stosowany na laboratorium PGK, stąd każdy ma doświadczenie w pracy z tą biblioteką. Pierwotnie miał zostać wykorzystany Code::Blocks razem z biblioteką wxWidgets 3.0.1, jednak ze względu na trudności z instalacją tego środowiska uznaliśmy za jedyną łatwo dostępną, pewną i akceptowalną alternatywę środowisko wxDev-Cpp.

Funkcje testowe były generowane za pomocą skryptu napisanego w języku Python.

5. Podział pracy i analiza czasowa

Na początku podział pracy był następujący:

Bartłomiej Mucha: tworzy interfejs użytkownika za pomocą biblioteki wxWidgets, razem obsługą wszystkich dostępnych elementów interfejsu.

Miłosz Świerad: tworzy osobny generator funkcji 4D oraz generuje pliki z danymi, nastepnie pisze funkcje do wprowadzenia danych z pliku do programu w postaci std::vector<std::vector<double>>.

Oleksandr Shaforostov: pisze funkcje Draw() do rysowania przekrojów z wykorzystaniem metody Sheparda.

Pozostały czas: wspólne usunięcie błędów, testowanie programu.

6.Opracowanie i opis niezbędnych algorytmów

Do rysowania przekrojów wykorzystano metodę Sheparda, której wzór ma postać:

$$f(x,y) = \left(\sum_{k=1}^{N} w_k(x,y) f_k(x,y)\right) / \left(\sum_{k=1}^{N} w_k(x,y)\right) \quad (1)$$

$$w_k(x,y) = \frac{1}{\left(\sqrt{(x-x_k)^2 + (y-y_k)^2}\right)^P} \quad (2)$$

Metoda służy do aproksymacji wartości funkcji w dowolnych punktach na podstawie zbioru wartości funkcji w N nieregularnie rozłożonych punktach.

7.Kodowanie

Dane z wartościami funkcji wprowadzamy z pliku do programu za pomocą funkcji fromFile() z fromFile.h. Ta funkcja uzupełnia wbudowany w klasę ProjectGrafikaDevDlg wektor std::vector<vector<double>> Function.

Interfejs był stworzony z pomocą biblioteki wxWidgets 2.9.3.

Dla całego projektu była stworzona klasa ProjectGrafikaDevDlg.

W tej klasie zdefiniowane zostały metody: do rysowania Draw(), do tworzenia vectora pomocniczego DataM().

Zmienne _X, _Y, _Z przechowują odpowiednią wartość z wektora.

Zmienne pomocnicze os1, os2, os3; os1 oraz os2 – osi której rysowane na panelu, a os3 - oś, wzdłuż której zmieniamy wartość suwaka.

Wektor Function przechowuje wszystkie dane podanej funkcji 4D. Wektor DataMap przechowuje tylko te dane, który są potrzebne w zależności od wektora (Floating Vector) oraz wartości suwaka.

Zmienne a, b przechowują dane o granicach zakresu danych; a – początek, b – koniec. Dlatego program może obsługiwać dane o różnej ilości węzłów.

Zmienna fileLoad typu bool mówi, czy plik był już wcześniej wczytany. Razem z uruchomieniem programu fileLoad przyjmuje wartość false, po pierwszym wczytaniu funkcji jej wartość jest zmieniana na true. Bez tej zmiennej razem z uruchomieniem programu doszłoby do naruszenia nieprzypisanej pamięci w związku z czyszczeniem zawartości pustych wektorów Function i DataMap. Bez niej program próbuje uzupełniać wektor DataMap, nawet jeśli wektor Function jest pusty.

Po wprowadzeniu danych zostaje wywołana funkcja Draw (std::vector<vector<double>>, int N), której argumentami są vector danych DataMap.

Do aproksymacji wartości funkcji w dowolnych punktach na podstawie zbioru wartości funkcji w N regularnie rozłożonych punktach była wykorzystana metoda Sheparda.

Funkcje testowe generowane były za pomocą skryptu *generator.py*, w którym należy podać zakres współrzędnych i wzór generowanej funkcji w postaci łańcucha znaków.

8. Testowanie

Pierwsza wersja rysowała bardzo wolno, bo ok. 20 sekund, dla 10 węzłów w jednym kierunku. Wprowadzenie wektora pomocniczego DataMap, jak i usunięcie zbędnych działań w funkjach warunkowych i pętlach pozwoliła zoptymalizować w pewnej kod, w wyniku czego program działał znacznie szybciej, a rysowanie trwało ok. 13 sekund dla 20 węzłów i mniej niż 5 sekund dla 10 węzłów.

Funkcja, w której wartość maksymalna wynosiła ok. 10^{19} błędnie generowała bitmapę. Przyczyną jest duża różnica z minimum (min = 0), oraz zbyt duże maksimum. Program może obsługiwać dane typu double, ale stopień 19 wychodzi poza zakres tego typu.

Na jednym z naszych komputerów zdarzało się, że program okazyjnie się zawieszał po wielokrotnej zmianie rozmiaru okna, albo po zmianie pliku. Były to bardzo rzadkie przypadki, jednak nie udało nam się ustalić przyczyny, możliwe, że nie leży ona w naszym kodzie.

9. Wdrożenie, raport i wnioski

Zrobiony projekt spełnia wymagania podstawowe: rysuje przekroje dla pewnej funkcji 4D, dla podanego wektora ([1,0,0] [0,1,0] [0,0,1] [1,1,0] [1,0,1] lub [0,1,1]). Po rysowaniu wynik można zapisać w formacie BMP, JPG lub PNG.

Głównym niedoskonałością, jaką cechuje się nasz program, jest jego szybkość działania. Dla dużych danych program bardzo wolno działa, nawet po optymalizacji, która znacząco przyspieszyła jego działanie.