Wizualizacja pola wektorowego

Autorzy: Łukasz Gut, Grzegorz Litarowicz, Piotr Moszkowicz

Opis projektu

Projekt polegał na napisaniu programu, który pozwala wizualizować funkcję wektorową $\vec{V}=f(\vec{r})$. Całość została napisana w języku C++ z wykorzystaniem open-source'owego środowiska graficznego Qt.

Założenia wstępne przyjęte w realizacji projektu

Głównymi założeniami przy tworzeniu aplikacji było stworzenie prostego i intuicyjnego w obsłudze programu, który pozwoliłby w elegancki sposób wizualizować funkcje wektorowe. Dodatkowo bardzo zależało nam na napisaniu programu, który działałby możliwie szybko. Istotnym i ambitnym wymogiem, który sobie postawiliśmy było napisanie aplikacji wieloplatformowej - wspieramy systemy Windows, Linux oraz MacOS.

Od początku mieliśmy w planach spełnienie zarówno wymagań podstawowych jak i rozszerzonych.

Analiza projektu

Specyfikacja danych wejściowych i specyfikacja interfejsu użytkownika

Nasz program przyjmuje stosunkowo dużo danych wejściowych. Zostaną opisane zgodnie z kolejnością ich wprowadzania od góry do dołu.

• Sekcja "Przedział zmienności argumentów" - W tej sekcji można wpisać minimalne oraz maksymalne współrzędnie, dla których wizualizacja będzie rysowana. Dane należy wpisać jako liczba dziesiętna z kropką (na przykład: "5.5").

• Sekcja "Ilość podprzedziałów" - Dla każdej osii należy wpisać ilość podprzedziałów - od ilości podprzedziałów stricte zależy ilość rysowanych wektorów dla każdej osii. Dane należy wpisać jako liczba naturalna (na przykład "10").

• Sekcja "Długość wektora" - W tej sekcji można wybrać tryb długości wektora: "automatycznie" - długości wektorów są automatycznie wyliczone, "stała" - wektory będą o stałej długości, "Podana przez użytkownika" - użytkownik może wybrać długość z pomocą "slidera" tuż pod listą wybieraną.

• Sekcja “Wybierz funkcję” - Użytkownik może z listy wybrać wzór funkcji, jaki program wyrysuje. Następnie poniżej można wpisać stałe stojące przed danymi zmiennymi. Dane należy wpisać jako liczba dziesiętna z kropką (na przykład “5.5”).

• Sekcja “Wybierz motyw” - W tym miejscu można wybrać motyw wykresu, co poprawia widoczność.

• Sekcja “Odcięcie płaszczyzny” - W tej sekcji można włączyć odcięcie płaszczyzną - wszystkie wektory, które znajdują się nad nią nie będą rysowane. Pod opcją “Włącz odcięcie płaszczyzną” można uzupełnić parametry płaszczyzny w zgodności ze wzorem: $Ax + By + Cz + D = 0$. Dane należy wpisać jako liczba dziesiętna z kropką (na przykład “5.5”).

• Sekcja “Zapis do pliku” - Naciskając button “Zapis do pliku” wyświetla się okienko, z pomocą którego możemy zapisać wykres jako obrazek. W okienku należy wybrać lokalizację oraz nazwę zapisywanego pliku.

Opis oczekiwanych danych wyjściowych

Jako dane wyjściowe program rysuje wizualizację pola wektorowego jako obrazek, po lewej stronie interfejsu. Użytkownik również może wybrać opcję zapisania wizualizacji do pliku.

Struktura programu

W naszym programie utworzyliśmy jedynie klasę Scatter, która odpowiada za renderowanie całego wykresu oraz posiada metody, które są wywoływane w momencie zmian parametrów programu przez użytkownika. W pliku main natomiast znajduje się cała implementacja interfejsu.

Struktura przechowywanych danych

Wszystkie parametry aplikacji są przechowywane jako zmienne danych prostych (najczęściej typu float, w ostateczności int, przykładowo dla slidera), jedynie przedziały zmienności argumentów przechowywane są jako pary (struktura QPair z biblioteki Qt), oraz aktualnie renderowana funkcja przechowywana jest jako typ std::function. Aktualnie rysowane strzałki znajdują się w vectorze.

Wyodrębnienie i zdefiniowanie zadań

W procesie planowania zdefiniowaliśmy następujące zadania do wykonania:

- Utworzenie projektu z pomocą Qt Creator'a
- Przejrzenie dokumentacji Qt, wyszukanie potrzebnych kontrolek, z pomocą których w prosty sposób możemy rozwiązać postawione problemy
- Utworzenie podstawowej wersji programu, która jest w stanie narysować wektory
- Dodanie możliwości definicji przedziału zmienności argumentów
- Dodanie opcji wyboru podprzedziałów
- Dodanie możliwości zmiany długość renderowanego wektora
- Dodanie możliwości wyboru wizualizowanej funkcji oraz opisującej jej stałych
- Dodanie opcji odcięcia płaszczyzną
- Dodanie możliwości zapisu wizualizacji do pliku
- Dodanie dodatkowych funkcjonalności - opcja zmiany motywu
- Ostateczne formatowanie kodu i sprawdzenie jego poprawności
- Wykonanie ostatecznych testów manualnych wszystkich funkcjonalności na trzech systemach operacyjnych (Windows, Linux, MacOS).
- Napisanie dokumentacji

Decyzja o wyborze narzędzi programistycznych

Od początku postanowiliśmy korzystać z open-source'owej części biblioteki Qt. Jest to wieloplatformowa biblioteka z pomocą której można pisać aplikację desktopowe na wszystkie systemy operacyjne w języku C++ lub Java. Z powodu naszego doświadczenia oraz możliwości optymalizacji wybraliśmy język C++. Jako preferowane IDE z powodu korzystania z Qt wybraliśmy Qt Creator - IDE powstałe właśnie specjalnie w celu pisania aplikacji w tejże bibliotece. Również korzystaliśmy z narzędzia Git w celu wersjonowania kodu oraz portalu Github jako miejsca, w którym dzieliliśmy się aktualną wersją kodu. Również od razu pisaliśmy dokumentację wszystkich funkcji - narzędzie, które wybraliśmy do automatycznego generowania dokumentacji to doxygen. Ostatecznie kod został automatycznie sformatowany z pomocą clang-tidy, jako część programu CLion. Program był kompilowany za pomocą g++ na platformie Linux/MacOS oraz MinGW na platformie Windows.

Podział pracy i analiza czasowa

Przy projekcie staraliśmy się podzielić obowiązki po równo. W efekcie każdy z nas pracował zarówno przy tworzeniu interfejsu użytkownika, jak i przy pisaniu algorytmów renderujących pola wektorowe.

Opracowanie i opis niezbędnych algorytmów

Istotny algorytm, na którym bazuje całe działanie aplikacji znajduje się w metodzie generateAndRenderVectors klasy Scatter. Z początku czyścimy aktualnie wyświetlane dane, następnie obliczane są długości wektorów oraz znalezienie najkrótszego i najdłuższego z nich. Następnie w kolejnych pętlach wektory są generowane na podstawie danych oraz opcji wybranych przez użytkownika. Finalnie każdy z nich jest odpowiednio obracany oraz dodawany do wektora elementów do wyrenderowania. Pozostałe metody głównie odpowiadają za kontakt z użytkownikiem i ich implementacja jest trywialna - z reguły jest to zapisanie danych w instancji klasy.

Kodowanie

Cały kod został udokumentowany z pomocą narzędzia doxygen. W archiwum z kodem źródłowym oraz skompilowanym programem znajduje się folder doc, w który można podejrzeć wygenerowaną dokumentację kodu.

Testowanie

Program był testowany po każdym etapie dodawania nowych funkcjonalności. Z tego powodu błędy nie piętrzyły się, były od razu wykrywane i naprawiane. Po zakończeniu tworzenia aplikacji dokonaliśmy finalnych testów na wszystkich platformach, które mieliśmy w planie obsługiwać.

Wdrożenie, raport i wnioski

Wdrożenie przebiegło bez większych problemów. Aplikacja działa na platformach Windows, Linux oraz MacOS, zarówno na architekturze 32, jak i 64 bitowej. Jedynym problemem, z którym nie byliśmy sobie w stanie poradzić był problem dotyczący czyszczenia bufora, zawierającego dane o wektorach gotowych do wyrysowania. W efekcie podczas korzystania z aplikacji niektóre strzałki czasem nie są usuwane z wspomnianego wcześniej bufora, przez co są niepotrzebnie widoczne. W przyszłości zdecydowanie należałoby się temu dokładniej przyjrzeć.