Politechnika Warszawska
Wydział Matematyki i Nauk
Informacyjnych

PRACA DYPLOMOWA INŻYNIERSKA
NA KIERUNKU INFORMATYKA

WIZUALIZACJA ZJAWISK POGODOWYCH

Autor:
Adam Babik,
Alicja Celigowska

Promotor:
mgr inż. Paweł Rzążewski

Opiekun naukowy:
prof. dr hab. inż. Bohdan Macukow

Warszawa, kwiecień 2015

....

podpis promotora podpis autora

Spis treści

Wstęp **3**

1. Wymagania funkcjonalne **5**
1.1. Ogólne wymagania 5
1.2. Wymagania funkcjonalne aplikacji serwerowej 6
1.3. Wymagania funkcjonalne aplikacji webowej 7
1.4. Wymagania funkcjonalne aplikacji na platformę Android 9

2. Technologie **11**
2.1. Technologie serwerowe 11
2.2. Technologie webowe 13
2.3. Technologie użyte do stworzenia aplikacji na platformę Android 16

3. Dokumentacja techniczna **19**
3.1. Dokumentacja techniczna serwera 19
3.2. Dokumentacja techniczna aplikacji webowej 23
3.3. Dokumentacja techniczna aplikacji na Androida 26

4. Algorytmy **29**
4.1. Algorytm mapowania danych na model Ziemi w 3D 29
4.2. Algorytm wizualizacji wiatru na Androidzie 30
4.3. Algorytm tworzenia heap mapy 33

5. Analiza porównawcza (Benchmarking) **37**
5.1. Wydajność aplikacji webowej 37
5.2. Wydajność aplikacji na Androida 39

6. Dokumentacja użytkownika **43**
6.1. Aplikacja webowa 43
6.2. Aplikacja na platformę Android 44

7. Przebieg pracy **49**
7.1. Model wytwórczy 49
7.2. Podział prac 49
7.3. Harmonogram 50

8. Bibliografia **53**

Streszczenie

Niniejsza praca dyplomowa opisuje proces tworzenia kompleksowego systemu do pobierania oraz wizualizacji danych meteorologicznych w 3D. System ten jest zbudowany z trzech komponentów: aplikacji działającej po stronie serwera oraz dwóch aplikacji klienckich. Pierwsza z nich to aplikacja webowa uruchamiana w przeglądarce internetowej, zaś druga to aplikacja natywna działająca na urządzeniach mobilnych pod kontrolą systemu Android.

W pierwszym rozdziale szczegółowo opisano wszystkie wymagania funkcjonalne dla każdego komponentu. Jeśli chodzi o aplikację serwerową to najważniejszymi wymaganiami są pobieranie, przechowywanie i udostępnianie danych meteorologicznych dla aplikacji klienckich. Aplikacja webowa oraz aplikacja na platformę Android muszą potrafić wyświetlać model Ziemi w 3D oraz przetwarzać otrzymane dane i na ich podstawie prezentować wizualizację temperatur, wiatrów, deszczu oraz burz.

Kolejne trzy rozdziały zostały poświęcone użytym technologiom, architekturze całego systemu oraz wzajemnym relacjom między poszczególnymi komponentami. Opisano w jaki sposób aplikacja serwerowa pobiera dane meteorologiczne z zewnętrznych źródeł, archiwizuje je w bazie relacyjnej SQL i udostępnia za pomocą protokołu HTTP. Szczegółowo przedstawiono sposób budowy aplikacji webowej z wykorzystaniem języka JavaScript zgodnego ze standardem ECMAScript 6 i technologii WebGL oraz budowę aplikacji na platformę Android wraz z integracją z OpenGL. Opisano także najważniejsze algorytmy do przetwarzania oraz wizualizacji danych pomiarowych.

W rozdziale piątym przeprowadzono analizę wydajności każdej z aplikacji klienckich w zależności od liczby punktów pomiarowych i liczby wierzchołków modelu geometrycznego. Wydajność zmierzono dla poszczególnych wizualizacji jako liczbę wyświetlanych klatek na sekundę.

Podsumowaniem pracy jest krótka dokumentacja użytkownika przedstawiająca efekt końcowy oraz opis modelu wytwórczego i przebiegu pisania pracy.

Wstęp

Wizualizacja danych to zagadnienie ich obrazowego przedstawienia. Jego celem jest skuteczne i zrozumiałe przekazanie informacji oraz pozyskanie uwagi potencjalnego odbiorcy. Odpowiedni sposób przekazania danych pozwala na poprawne i szybkie odnalezienie zależności między nimi.

Niniejsza praca dyplomowa opisuje proces tworzenia kompleksowego rozwiązania do pobierania oraz wizualizacji danych meteorologicznych w 3D. System składa się z trzech komponentów: aplikacji działającej po stronie serwera, aplikacji klienckiej działającej w przeglądarce oraz natywnej aplikacji na platformę Android.

System pozwala na obserwowanie oraz śledzenie zmian warunków atmosferycznych w skali całego globu. Inspiracją do stworzenia aplikacji był projekt Earth (*http://earth.nullschool.net/*), który pozwala na wizualizację wielu zjawisk meteorologicznych, ale działa w 2D.

Dane meteorologiczne pochodzą z serwisu OpenWeatherMap. Serwis ten udostępnia wygodny interfejs oparty na protokole HTTP. Charakteryzuje się także niskimi limitami liczby zapytań oraz przestępną dokumentacją.

System zbudowany jest w nowoczesnej architekturze klient – serwer, pozwalającej na niezależne działanie każdej aplikacji wchodzącej w jego skład. Dynamiczny rozwój technologii webowych oraz standardów sieciowych pozwolił na zbudowanie aplikacji 3D działającej w przeglądarce bez dodatkowych wtyczek i rozszerzeń. Obecnie dużą uwagę skupiają na sobie także urządzenia mobile, działające pod kontrolą systemu operacyjnego Android. Z tego powodu zbudowaliśmy także aplikację na tę platformę.

Z biznesowego punktu widzenia, architektura naszego systemu jest dobrze skalowalna oraz łatwa w rozbudowie, pozwala również na dotarcie do szerokiej grupy potencjalnych użytkowników.

Rozdział 1

Wymagania funkcjonalne

Celem niniejszej pracy dyplomowej było zbudowanie dwóch aplikacji klienckich, które przedstawiają wizualizacje wybranych danych meteorologicznych na modelu Ziemi w 3D. Pierwsza z nich jest aplikacją webową, działającą w przeglądarce internetowej. Druga jest aplikacją natywną zbudowaną na platformę Android i uruchamianą na telefonach komórkowych.

1.1. Ogólne wymagania

Dane meteorologiczne muszą zawierać następujące pomiary:

- temperatura powietrza,
- prędkość i kierunek wiatru,
- suma opadów i miejsca wystąpień burz.

Wymienione pomiary muszą być aktualne, z możliwością odtworzenia danych archiwalnych z kilku poprzednich dni. Pomiary są zbierane z różnych źródeł co pewien określony czas.

Interfejs obu aplikacji umożliwia użytkownikowi wybór aktualnie wyświetlanej wizualizacji jednego zjawiska meteorologicznego. Użytkownik ma także możliwość przełączania się pomiędzy wizualizacjami w różnych dniach i godzinach.

Na modelu Ziemi znajdują się kontury lądów i państw. Model ma możliwość obracania, powiększania oraz pomniejszania w celu obserwacji dowolnego miejsca. Te funkcje są

wykonywane płynnie z użyciem typowych narzędzi wskazujących dostępnych dla danej platformy. W przypadku interfejsu przeglądarkowego jest to kursor, natomiast w przypadku aplikacji na Androida są to gesty.

W celu pobierania i przechowywania danych meteorologicznych oraz aby dostarczyć użytkownikowi niezbędne pliki do uruchomienia aplikacji webowej, niezbędnym wymaganiem była budowa aplikacji serwerowej wraz z lokalną bazą danych. Aplikacja serwerowa potrafi zbierać i zapisywać dane z jednego lub wielu dostępnych źródeł danych, aby następnie udostępnić je aplikacjom klienckim.

Poniżej zostały przedstawione szczegółowe wymagania funkcjonalne dla poszczególnych aplikacji.

1.2. Wymagania funkcjonalne aplikacji serwerowej

Aplikacja serwerowa realizuje dwie najważniejsze funkcje:

- pobiera i zapisuje dane meteorologiczne,
- udostępnia zebrane dane aplikacjom klienckim.

Do realizacji pierwszego punktu wykorzystano relacyjną bazę danych, zgodną ze standardem SQL, która jest dostępna jedynie w sieci lokalnej.

Komunikacja pomiędzy aplikacjami klienckimi i aplikacją serwerową została zrealizowana za pomocą protokołu HTTP w wersji 1.1, natomiast dane są kodowane i przesyłane w formacie JSON.

Aplikacja serwerowa musi posiadać co najmniej dwa zasoby, dostępne za pomocą metody GET protokołu HTTP. Pierwszy z nich powinien udostępniać najświeższe zapisane dane meteorologiczne. Natomiast drugi powinien oferować możliwość pobierania archiwalnych danych meteorologicznych na podstawie przesłanego parametru daty.

Dodatkową funkcją aplikacji serwerowej jest przesyłanie niezbędnych plików statycznych do aplikacji działającej w przeglądarce. Pliki statyczne są to np. pliki JavaScript, pliki kaskadowych arkuszy styli CSS oraz pliki HTML.

Aplikacja serwerowa oraz baza danych działają pod kontrolą systemu operacyjnego z rodziny Linux.

1.3. Wymagania funkcjonalne aplikacji webowej

Poza ogólnymi wymaganiami opisanymi w sekcji 1.1 Ogólne wymagania, każda z aplikacji klienckich ma dodatkowe, specyficzne dla siebie wymagania funkcjonalne. Spowodowane jest to różnymi możliwościami sprzętowymi, wielkościami ekranów oraz chęcią przeprowadzania niezależnych eksperymentów.

Szczególnymi wymaganiami aplikacji webowej jest możliwość uruchomienia jej na konkretnych przeglądarkach. Niestety z powodu wykorzystania nowej technologii jaką jest WebGL, jest ona dostępna jedynie w najnowszych wersjach popularnych przeglądarek: Google Chrome 42, Mozilla Firefox 37, Safari 8 oraz Internet Explorer 11.

Ważne jest także, aby karta graficzna zainstalowana w danej stacji roboczej była wpierana przez przeglądarki wymienione powyżej. W przypadku, gdy przeglądarka nie potrafi wyświetlić aplikacji korzystającej z technologii WebGL, zaprezentowany zostaje odpowiedni komunikat.

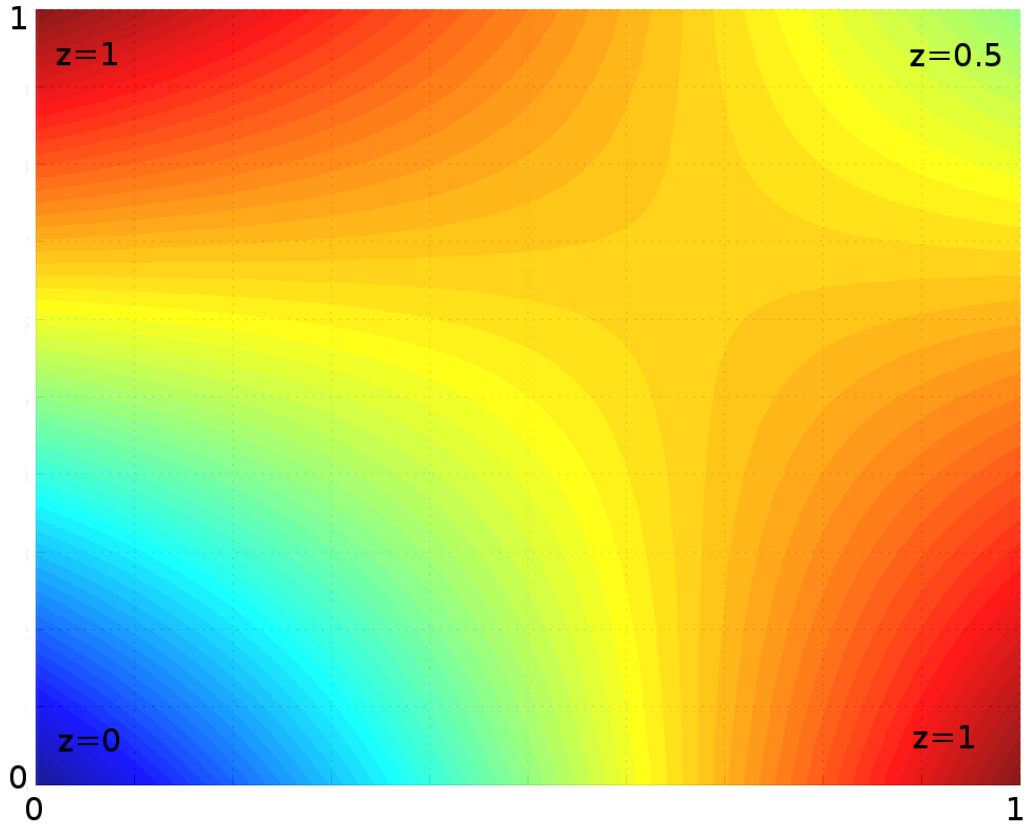
1.3.1. Wizualizacja temperatur

Wizualizacja rozkładu temperatur powietrza jest pokazana na modelu Ziemi za pomocą tzw. *heat mapy* – zobacz Rysunek 1.1. Kolor czerwony odpowiada najwyższym temperaturom z danego zbioru pomiarów, a kolor niebieski i fioletowy temperaturom najniższym. Podczas kolorowania płaszczyzny, kolor dla każdego piksela jest ustalany za pomocą interpolacji liniowej.

1.3.2. Wizualizacja prędkości i kierunku wiatrów

Prędkość wiatrów została zrealizowana niemal identycznie jak zostało to opisane w sekcji 1.3.1 Wizualizacja temperatur. Jedyną różnicą są kolory użyte do reprezentowania poszczególnych wartości. Punkty z najniższymi wartościami prędkości wiatru będą pokolorowane na niebiesko, natomiast punkty z wartościami najwyższymi będą miały kolor zielony. Wartości pomiędzy punktami także powinny być interpolowane liniowo.

Kierunek wiatru został przedstawiony za pomocą linii stycznych do punktów pomiarowych.



Rysunek 1.1: Przykład heap mapy wykonanej za pomocą interpolacji dwuliniowej

1.3.3. Wizualizacja sumy opadów i miejsca występowania burz

Wizualizacja sumy opadów deszczu różni się od wizualizacji przedstawionych powyżej. Jeden pomiar przedstawiony jest za pomocą prostopadłościanu ustawionego prostopadle do punkty sfery. Jego wysokość zależy od wielkości sumy opadów. Kolor zmienia się od niebieskiego do fioletowego, gdzie fioletowy oznacza większą wartość pomiaru.

Burze zostały przedstawione w analogiczny sposób, z tą różnicą, że mają kolor żółty, aby odróżnić się od reprezentacji pomiaru opadu.

1.4. Wymagania funkcjonalne aplikacji na platformę Android

Wymagania funkcjonalne dla aplikacji na platformę Android zostały dobrze opisane w rozdziale 1.1 Ogólne wymagania. Różnicą jest interfejs, który został odpowiednio przystosowany do urządzeń mobilnych: implementacja sterowania gestami, menu aplikacji uruchamiane standardowym przyciskiem do zmiany menu na urządzeniu.

Szczególnymi wymaganiami są wymagania odnośnie procesora, wersji SDK oraz wersji OpenGL ES. Aplikacja na platformę Android została częściowo stworzona w języku C++, a napisany w nim moduł został skompilowany pod kątem procesorów ARMv7. Dlatego urządzenie klienta powinno zawierać procesor wspierający odpowiednie ABI. Minimalna wersja SDK to 11 (wersja systemu Ice Cream bądź Sandwich), natomiast wymagana wersja OpenGL ES to wersja numer 2. Zdecydowana większość urządzeń średniobudżetowych powinna sprostać tym wymaganiom.

W przypadku niespełnienia wymienionych wyżej wymagań, pojawi się błąd podczas próby instalacji aplikacji.

1.4.1. Wizualizacja temperatur

Wizualizacja rozkładu temperatury powietrza jest pokazana na modelu Ziemi za pomocą tzw. *heat mapy*. Kolor czerwony odpowiada najwyższym temperaturom z danego zbioru pomiarów, a kolor niebieski i fioletowy temperaturom najniższym. Podczas kolorowania płaszczyzny, kolor dla każdego piksela jest ustalany za pomocą interpolacji liniowej.

1.4.2. Wizualizacja prędkości i kierunku wiatrów

Prędkość wiatrów powinna być zrealizowana niemal identycznie jak zostało to opisane w sekcji 1.4.1 Wizualizacja temperatur. Jedyną różnicą są kolory użyte do reprezentowania poszczególnych wartości. Punkty z najniższymi wartościami prędkości wiatru będą pokolorowane na niebiesko, natomiast punkty z wartościami najwyższymi będą miały kolor zielony. Wartości pomiędzy punktami także powinny być interpolowane liniowo.

Kierunek wiatru jest przedstawiony za pomocą przesuwających się punktów po powierzchni planety.

1.4.3. Wizualizacja sumy opadów i miejsca występowania burz

W przypadku wizualizacji opadów zastosowano tę samą technikę co we wcześniej wspomnianych zjawiskach, przy czym zakres kolorów jest od białego (brak opadów) do niebieskiego (silne opady). Dodatkowym elementem graficznym są równoległe do powierzchni Ziemi opadające krople deszczu, występujące w miejscach opadów.

Burze zostały przedstawione za pomocą żółtych okręgów naprzemiennie rosnących i malejących.

Rozdział 2

Technologie

W tym rozdziale przedstawimy technologie, które wybraliśmy do realizacji wymagań opisanych w rozdziale 1 Wymagania funkcjonalne. Każda z trzech aplikacji stworzonych podczas pisania tej pracy dyplomowej działa w innym środowisku, dlatego też każda z nich wymaga użycia innych technologii oraz języków programowania.

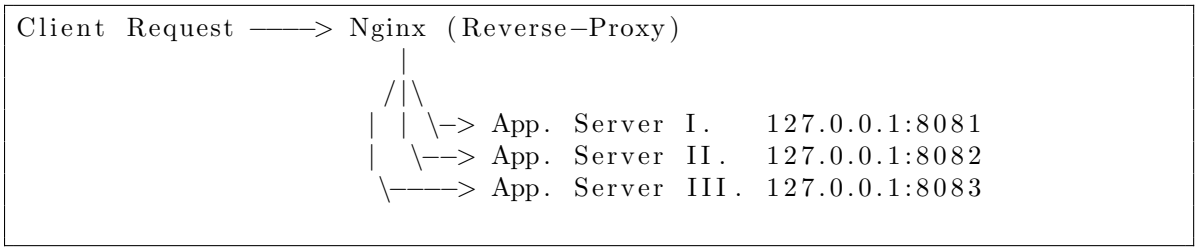
2.1. Technologie serwerowe

Aplikacja serwerowa wchodzi w skład technologi back-endowych, czyli działających na serwerze. Zdecydowaliśmy się skorzystać z VPS-a (ang. *Virtual Private Server*), dzięki czemu mieliśmy możliwość zainstalowania własnego systemu operacyjnego oraz programów, a także dowolność w ich konfiguracji.

Zgodnie z wymaganiami funkcjonalnymi zainstalowaliśmy system operacyjny Ubuntu 12.10 należący do rodziny systemów Linux.

W skład technologii serwerowych wykorzystanych do realizacji tej pracy dyplomowej wykorzystano:

1. nginx jako reverse-proxy serwer,
2. uWSGI jako serwer do uruchomienia aplikacji napisanych w języku Python,
3. Django jako framework do budowania aplikacji internetowych,
4. PostgreSQL jako relacyjna baza danych.



Rysunek 2.1: Zasada działania serwera reverse-proxy

2.1.1. nginx

nginx jest serwerem WWW oraz serwerem reverse-proxy. Serwer reverse-proxy ma za zadanie przekierować zapytania z jednego adresu IP na inny, zwykle niedostępny z sieci Internet.

Zasada działania serwera reverse-proxy została przedstawiona na Rysunku 2.1. Przechwytuje on zapytania napływające od klientów i przekazuje je do odpowiednich procesów działających na różnych portach.

W naszej pracy dyplomowej nginx został wykorzystany właśnie jako serwer reverse-proxy, który przekazuje zapytania HTTP do aplikacji napisanej w języku Python z wykorzystaniem frameworka Django oraz jako serwer plików statycznych.

Wykorzystanie tego typu serwera umożliwia znacznie lepsze zarządzanie zasobami. Pozwala m. in. na rozdzielanie ruchu na wiele procesów oraz odciąża aplikację internetową z części zadań jak np. serwowanie statycznych plików.

Innymi zaletami nginx-a jest bardzo dobra dokumentacja, łatwa instalacja i konfiguracja oparta o pliki konfiguracyjne, szybkość działania oraz małe zużycie pamięci operacyjnej.

2.1.2. uWSGI

uWSGI jest serwerem WWW, który uruchamia aplikację internetową napisaną w Pythonie. Nie jest to niezbędny element całej architektury, ale znacząco wspomaga pracę oraz zarządzanie procesami aplikacji internetowej.

Najważniejszą z jego cech jest możliwość uruchamiania wielu instancji tej samej aplikacji. Pozwala to na np. wyłączenie części tych procesów, które wymagają modyfikacji,

jednocześnie zapewniając, że strona internetowa lub usługi będą ciągle dostępne dla użytkowników końcowych.

2.1.3. Django

Django jest frameworkiem do tworzenia aplikacji internetowych na licencji BSD, czyli ogólnie pojęte open source. Został napisany w języku Python i jest to jedyna zależność tego frameworku. Django wymaga zainstalowanego Pythona w wersji co najmniej 2.7. Oprócz tego zaletą Django jest bardzo przyjazna dokumentacja, powszechność i prostota języka Python.

Django posiada wbudowany ORM, który obsługuje wiele baz SQL, jak np. SQLite, PostgreSQL, MySQL, Oracle.

2.1.4. PostgreSQL

Jako serwer bazy danych wybraliśmy open sourcową, relacyjną bazę danych PostgreSQL. Do współpracy z Django wymaga pakietu psycopg2, który zapewnia komunikację między nią, a aplikacją napisaną w języku Python. Obie technologie świetnie ze sobą współpracują i są powszechnie używane w aplikacjach naukowych oraz biznesowych.

2.1.5. cron

Dane z zewnętrznych źródeł są pobierane cyklicznie, bez ingerencji administratora, za pomocą skryptów napisanych w języku Python. Za uruchamianie tych skryptów odpowiada mechanizm cron, który jest standardowo wbudowany w systemy z rodziny Linux.

2.2. Technologie webowe

Od kilku ostatnich lat aplikacje webowe przeżywają prawdziwy rozkwit. Większość usług jest przenoszona do Internetu, a najłatwiejszym sposobem, aby je udostępnić szerokiemu gronu użytkowników jest przeglądarka internetowa. Jednak rosnąca liczba użytkowników oraz coraz nowocześniejsze i bardziej dynamiczne interfejsy spowodowały, że należało zmienić sposób tworzenia tego typu aplikacji.

Aplikacje internetowe nie są już jedynie aplikacjami serwerowymi, które odpowiadają zarówno za logikę biznesową jak i generowanie interfejsu użytkownika. Taka architektura jest niezwykle trudna w utrzymaniu, źle się skaluje, wymaga dodatkowej mocy obliczeniowej oraz generuje duże transfery danych.

W nowoczesnych architekturach aplikację internetową buduje się jako dwa niezależne komponenty: aplikację działającą po stronie serwera oraz aplikację działającą po stronie klienta, w przeglądarce. Pierwsza z nich odpowiedzialna jest jedynie za logikę biznesową oraz udostępnienie danych. Natomiast aplikacja przeglądarkowa buduje i prezentuje interfejs użytkownikowi, korzystając z różnych form komunikacji z aplikacją serwerową.

W naszej pracy dyplomowej rolę aplikacji serwerowej pełni aplikacja napisana w języku Python z wykorzystaniem frameworka Django. Udostępnia ona dwa zasoby do pobierania danych meteorologicznych.

Aplikacja webowa napisana w języku JavaScript pełni rolę aplikacji działającej po stronie klienta, która prezentuje użytkownikowi otrzymane dane. Dane te pobierane są asynchronicznie przy użyciu technologii AJAX. Sam interfejs jest hybrydowy, tzn. łączy HTML i CSS oraz technologię do prezentacji grafiki 3D – WebGL. Taki podział zadań powoduje, że cała praca polegająca na wyświetleniu interfejsu użytkownika jest realizowana na komputerach klienckich.

2.2.1. JavaScript

JavaScript jest wieloparadygmatowym, dynamicznie typowanym językiem programowania, który służy do pisania aplikacji webowych działających w przeglądarkach internetowych. JavaScript jest implementacją standardu ECMAScript. Aktualna wersja tego standardu jest oznaczona numerem 5.1 i jest wpierana przez najnowsze wersje wszystkich popularnych przeglądarek internetowych. Nowa wersja tego standardu jest oznaczona numerem 6 i jest w finalnej fazie prac. Oficjalne wydanie wersji ECMAScript 6 planowane jest na połowę 2015 roku.

Aplikacja webowa napisana na potrzeby niniejszej pracy inżynierskiej została napisana z użyciem standardu ECMAScript 6. Kod aplikacji, przed uruchomieniem w przeglądarce, jest kompilowany do standardu ECMAScript 5.

Zdecydowaliśmy się skorzystać z wersji beta standardu ECMAScript 6 z powodu wielu usprawnień. W szczególności nowa wersja wprowadza możliwość definiowania modułów

oraz wiele nowych elementów składniowych języka, które znacznie redukują ilość kodu oraz sprawiają, że jest on bezpieczniejszy.

Nowe elementy składni języka ze standardu ECMAScript wykorzystane w aplikacji webowej to:

- `export`, `import` do zarządzania modułami,
- `class` do definiowania klas (konstruktorów),
- `=>` (ang *arrow function*) do tworzenia funkcji anonimowych,
- nowe słowa kluczowe do deklarowania zmiennych `let` oraz `const`.

Kod JavaScript napisany w standardzie ECMAScript 6 jest kompilowany do standardu ECMAScript 5 za pomocą narzędzia Babel. Narzędzie to wspiera także kompilację wielu plików źródłowych napisanych w języku JavaScript do jednego pliku wynikowego oraz obfuskację kodu i generowanie map źródłowych (ang. *source maps*) służących do debugowania.

2.2.2. AJAX

Dane pomiędzy aplikacją serwerową oraz aplikacją webową przesyłane są za pomocą technologi AJAX (ang. *Asynchronous JavaScript and XML*). Pomimo obecności słowa *XML* w rozwinięciu skrótu, wcale nie jest to jedyny wpierany format danych. Dane mogą być zapisane w dowolnym formacie tekstowym takim jak np. JSON, YAML. Za nawiązanie asynchronicznego połączenia oraz interpretację i parsowanie danych odpowiada klasa `XMLHttpRequest`. Tutaj nazwa także wskazuje na XML jako jedyny format, ale również wspierane są dowolne tekstowe formaty danych. Obecnie najpowszechniejszym formatem jest JSON i w naszym projekcie skorzystaliśmy właśnie z tego formatu. Jest on także wpierany przez Django i język Python.

Połączenie asynchroniczne charakteryzuje się tym, że jest wykonywane w tle i nie powoduje przeładowania aktualnie załadowanej strony WWW. Aplikacja informowana jest o zakończeniu połączenia przez wywołanie odpowiedniej funkcji (callback) przypisanej do właściwości obiektu `XMLHttpRequest` podczas tworzenia zapytania.

2.2.3. WebGL

Interfejs naszej aplikacji webowej jest hybrydowy, tzn. zbudowany w oparciu o język znaczników HTML (ang. *HyperText Markup Language*) oraz technologię tworzenia grafiki 3D w przeglądarce internetowej znaną pod nazwą WebGL.

WebGL jest stosunkowo nową technologią webową, dedykowaną do tworzenia i wyświetlania grafiki 3D. Zbudowany jest na podstawie standardu Open GL ES 2.0 znanym z systemów wbudowanych. Dostęp do WebGL w przeglądarce internetowej jest zrealizowany za pomocą dedykowanego API dla języka JavaScript. Natomiast samo wyświetlenie grafiki 3D odbywa się z wykorzystaniem taga HTML `canvas`.

WebGL wspiera OpenGL Shading Language dzięki czemu możliwe jest tworzenie własnych programów uruchamianych bezpośrednio na GPU.

2.3. Technologie użyte do stworzenia aplikacji na platformę Android

Smartphone'y oraz tablety stają się nieodłącznymi elementami naszego życia. Wzrastają ich możliwości, moce obliczeniowe oraz liczba dostępnych funkcji. W związku z tym zdecydowaliśmy, że nasza aplikacja będzie dostępna także na tych urządzeniach. Z uwagi na dość ograniczone zasoby sprzętowe oraz narzut środowiska przeglądarki trudno uruchomić na nich aplikację webową. Dlatego też zdecydowaliśmy się stworzyć osobną aplikację natywną. Wybraliśmy system Android, ponieważ jest to najpopularniejszy system operacyjny dedykowany tej grupie urządzeń.

2.3.1. SDK na platformę Android

SDK dla Androida jest zbiorem bibliotek udostępniających API systemu operacyjnego w języku Java. Jednym z ważniejszych i nieodłącznych elementów tego systemu jest maszyna wirtualna Dalvik, która zarządza aplikacjami napisanymi w Javie. Jest to analogiczna maszyna do tej tradycyjnej występującej na standardowych komputerach, z tym wyjątkiem, iż została stworzona i zoptymalizowana specjalnie na potrzeby platformy Android.

Nasza natywna aplikacja na Androida jest napisana głównie w języku Java i wykorzystuje wiele komponentów z wspomnianego SDK. Wykorzystywany jest standardowy szablon aplikacji, obsługa gestów oraz API do połączenia z internetem.

SDK dla Androida jest obecnie najpowszechniejszym narzędziem do budowania aplikacji na tą platformę.

2.3.2. NDK na Android (JNI)

Choć SDK na Androida jest standardowym narzędziem do budowania aplikacji na tą platformę, czasem zachodzi potrzeba skorzystania z technologii NDK (Native Development Kit). Z powodów wydajnościowych i ograniczeń narzuconych przez SDK, nasza aplikacja będzie wykorzystywać w większym stopniu zasoby sprzętowe (szczególnie przy pobieraniu i parsowaniu danych pogodowych), w związku z czym zdecydowano się wykorzystać także NDK.

NDK to technologia umożliwiająca pisanie komponentów w języku C oraz C++ kompilowanych do kodu maszynowego na platformę Android. Dzięki NDK programista ma znacznie szerszy i szybszy dostęp do zasobów sprzętowych w porównaniu z SKD. Do komunikacji pomiędzy kodem maszynowym i kodem napisanym w języku Java, NDK wykorzystuje JNI (Java Native Interface).

2.3.3. OpenGL ES

OpenGL ES jest otwartą biblioteką graficzną do tworzenia i wyświetlania grafiki 3D dedykowaną dla aplikacji na platformy wbudowane. Biblioteka ta jest standardem dla systemu Android, dlatego wykorzystaliśmy ją w celu renderowania trójwymiarowego obrazu kuli ziemskiej wraz z wizualizacjami zjawisk pogodowych.

Rozdział 3

Dokumentacja techniczna

W tym rozdziale skupimy się na opisie architektury oraz implementacji poszczególnych komponentów naszej pracy dyplomowej.

3.1. Dokumentacja techniczna serwera

Serwer musi realizować dwa konkretne zadania: udostępnianie danych pogodowych oraz ich pobieranie. Do pierwszego zadania służy aplikacja WWW, do drugiego zadania napisano odpowiedni moduł wraz ze skryptami w języku Python, które są uruchamiane za pomocą narzędzia cron.

3.1.1. Aplikacja WWW

Do stworzenia aplikacji WWW wykorzystano framework Django oraz serwer baz danych PostgreSQL. Django wspiera wzorzec MTV (*Model – Template – View*). Jest to model dość podobny do wzorcowego modelu MVC (*Model – View – Controller*).

Kontroler w aplikacjach MVC służyłby do wywołania odpowiedniej funkcji widoku w zależności od danych wejściowych. W przypadku Django tę pracę wykonuje framework, stąd używa się tutaj określenia MTV.

Model służy do określenia formatu danych, ich zależności oraz definiuje sposób walidacji. Widok operuje na warstwie modelu i przekazuje dane do odpowiedniego szablonu. Szablon to plik HTML wraz ze specjalnymi tagami, który jest kompilowany i wyświetlany w przeglądarce WWW.

Aplikacje Django

Projekt w Django składa się z plików konfiguracyjnych oraz zbioru modułów, które w terminologii Django są nazywane aplikacjami (ang. *apps*). Przykładami aplikacji mogą być czat, blog, system komentarzy itp. Aplikacje w projekcie Django powinny być osobnymi, niezależnymi bytami, choć często zdarza się, że jedna aplikacja może wymagać innej. Z kolei każda aplikacja zawiera modele, widoki oraz szablony.

Nasza aplikacja WWW zawiera jedną aplikację o nazwie *meteoviz_app* wraz z:

- modelami, które definiują jak zapisane są dane meteorologiczne, zależności między nimi oraz udostępniają API do zapisu/odczytu danych,
- widokami, które udostępniają dane meteorologiczne,
- szablonem, który pozwala na pobranie i start aplikacja webowej.

Udostępnianie danych

Dane są udostępniane za pomocą protokołu HTTP. Udostępnione są dwa publiczne zasoby do pobierania danych:

- *http://server_ip/meteoviz_app/get_current_weather/* służy do pobierania najnowszych danych pogodowych,
- *http://server_ip/meteoviz_app/get_weather/?time=%d.%m.%Y_%H-%M* służy do pobierania danych pogodowych z określonej daty i czasu.

Czas w przypadku drugiego zasobu musi być sformatowany w następujący sposób: `%d.%m.%Y_%H-%M`, gdzie `%d` należy zastąpić dniem miesiąca (wartości od 01 do 31), `%m` miesiącem (wartości od 01 do 12), `%Y` rokiem (czterocyfrowym), `%H` godziną (wartości od 00 do 23), `%M` minutą (wartości od 00 do 59). Jest to format zgodny z standardem C i taki format jest używany przez serwer.

Zwrócone dane są zakodowane w formacie JSON. Jest to słownik o dwóch kluczach: `data` oraz `time`. Wartość pod kluczem `time` zawiera datę i godzinę wraz z milisekundami (`%f`) oraz strefą czasową (`%z`) pobranych danych meteorologicznych. Pełny format daty i czasu to: `%Y-%m-%d %H:%M:%S.%f+%z`.

Wartość pod kluczem `data` zawiera tablicę pomiarów pogodowych. Pojedynczy pomiar dotyczy pogody w konkretnym punkcie określonym za pomocą współrzędnych geograficznych.

Jeden pomiar jest przedstawiony za pomocą słownika o następującej budowie:

- `lg`: informacja o długości geograficznej (liczba zmiennoprzecinkowa).
- `lt`: informacja o szerokości geograficznej (liczba zmiennoprzecinkowa).
- `r`: informacja o opadzie (liczba całkowita określająca ilość milimetrów opadu w ciągu 3 godzin).
- `ws`: informacja o prędkości wiatru (liczba zmiennoprzecinkowa, określa prędkość w milach na sekundę).
- `t`: informacja o temperaturze (liczba zmiennoprzecinkowa, określa temperaturę w kelwinach).
- `wd`: informacja o kierunku wiatru (liczba zmiennoprzecinkowa, w stopniach meteorologicznych, wartości od 0 do 360)
- `i`: informacja o ID pogody (liczba całkowita, kody między 200 a 300 oznaczają burze).
- `c`: informacja o zachmurzeniu (liczba zmiennoprzecinkowa, podana wartość jest w procentach).

Nazwy kluczy zostały skrócone w celu zmniejszenia ilości przesyłanych danych.

Moduł pobierania danych meteorologicznych

Aplikacja WWW udostępnia publicznie tylko metody służące do odczytywania danych. Do zapisu służy osobny moduł, który wykorzystuje warstwę modelu aplikacji WWW i jest uruchamiany okresowo za pomocą mechanizmu cron. cron uruchamia skrypt napisany w języku Python, który inicjuje pobieranie danych meteorologicznych z zewnętrznych serwisów.

Na początku swojego działania skrypt odczytuje z konfiguracji informację na temat klasy (konkretnie jej nazwę), która odpowiada za pobranie danych pogodowych. Następnie obiekt tej klasa jest inicjowany i uruchamia się procedura poboru danych. Taka konstrukcja pozwala na dopisywanie kolejnych klas służących do pobierania danych, a każda z nich może pobierać je z innych źródeł oraz w różny sposób.

W przypadku niepowodzenia pobrania danych zostanie podjęta kolejna próba, a aplikacje klienckie będą wyświetlać starsze dane.

Podstawowym źródłem danych pogodowych jest serwis OpenWeatherMap.org. Spośród wszystkich znalezionych źródeł danych pogodowych tylko to jedno spełnia nasze wymagania.

Serwis udostępniający dane pogodowe musi dawać możliwość pobierania sporej ilości danych. OpenWeatherMap jest jedynym, którego limity są wystarczająco wysokie. W przypadku innych serwisów częstym ograniczenie jest limit na ilość dziennych zapytań, przez co dokładność wizualizacji staje się dosyć niska. Klasa pobierająca dane z OpenWeatherMap nazywa się *OpenWeatherMapDownloader* i znajduje się w pliku *weather_downloader.py* w aplikacji *meteoviz_app*.

Dodanie nowego źródła danych jest dosyć proste. Polega na dopisaniu nowej klasy w aplikacji *weather_app*, w pliku *weather_downloader.py*. Nowa klasa musi spełniać następujące wymagania:

- musi dziedziczyć z klasy `BaseDownloader`,
- metoda `__init__` musi wywołać metodę bazową (nie przyjmuje żadnych argumentów),
- musi ustawić parametr `_measurements_per_latitude`,
- musi implementować metodę `download_weather` przyjmującą dwa argumenty: pierwszy to instancja klasy `WeatherGlobal`, drugi to lista współrzędnych geograficznych, dla których należy pobrać pogodę i utworzyć obiekt typu `WeatherLocal`.

W pliku `settings.py` (w katalogu `meteoviz_server`) należy podmienić opcję `WEATHER_DOWNLOADER` na nazwę nowej klasy, jeśli chcemy by była ona domyślna.

3.1.2. Modele

Aplikacja WWW udostępnia dwa modele: *WeatherGlobal* oraz *WeatherLocal*. Są one mapowane na dwie tabele w bazie danych.

WeatherGlobal posiada następującą konstrukcję:

- `time`: pole typu `datetime` ze wsparciem stref czasowych, z kluczem `unique`,
- `json_data`: pole tekstowe zawierająca dane pogodowe zapisane w formacie JSON.

WeatherLocal posiada następującą konstrukcję:

- `weather_global`: klucz obcy na rekord w tabeli WeatherGlobal,
- `latitude`: wysokość geograficzna, liczba zmiennoprzecinkowa,
- `longitude`: szerokość geograficzna, liczba zmiennoprzecinkowa,
- `temperature`: temperatura, liczba zmiennoprzecinkowa,

- `wind_speed`: prędkość wiatru, liczba zmiennoprzecinkowa,
- `wind_deg`: kierunek wiatru, liczba zmiennoprzecinkowa,
- `weather_cond_id`: ID pogody, liczba całkowita,
- `rain`: wskaźnik opadów, liczba całkowita,
- `clouds`: poziom zachmurzenia, liczba zmiennoprzecinkowa.

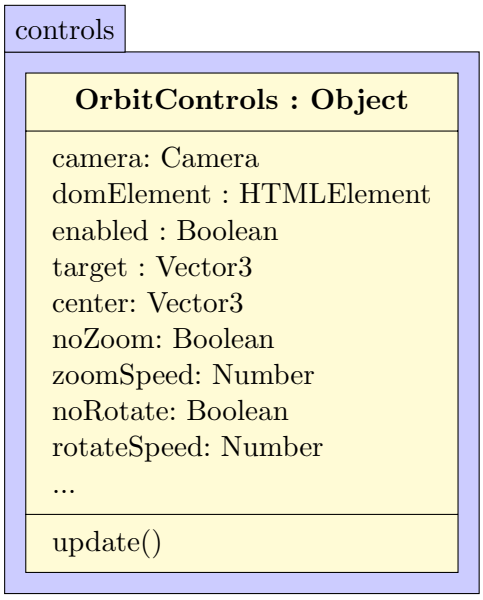
3.2. Dokumentacja techniczna aplikacji webowej

3.2.1. Moduły

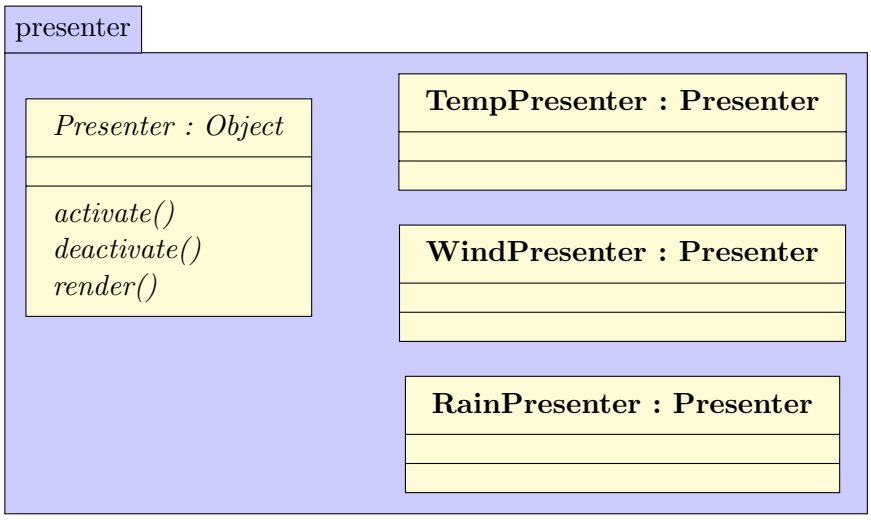
Kod aplikacji webowej został podzielony na kilka modułów. Moduły są od siebie niezależne i definiują klasy implementujące podobne funkcje. Można wyróżnić następujące moduły: *controls*, *presenters*, *ui*.

Moduł *controls* (zobacz Rysunek 3.1) zawiera klasę, która jest odpowiedzialna za sterowanie kamerą, tzn. obracaniem, przybliżaniem i oddalaniem modelu 3D.

Moduł *presenter* (zobacz Rysunek 3.2) zawiera klasy, które odpowiadają za wyświetlanie poszczególnych wizualizacji. Implementują one wspólny interfejs, dzięki czemu można łatwo definiować i dodawać nowe sposoby prezentacji danych.



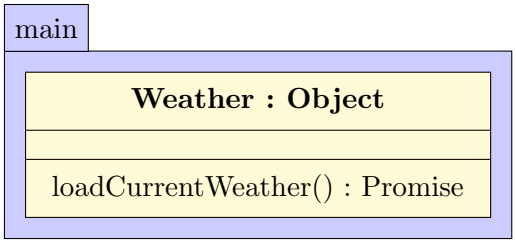
Rysunek 3.1: Klasa OrbitControls



Rysunek 3.2: Klasy modułu presenter

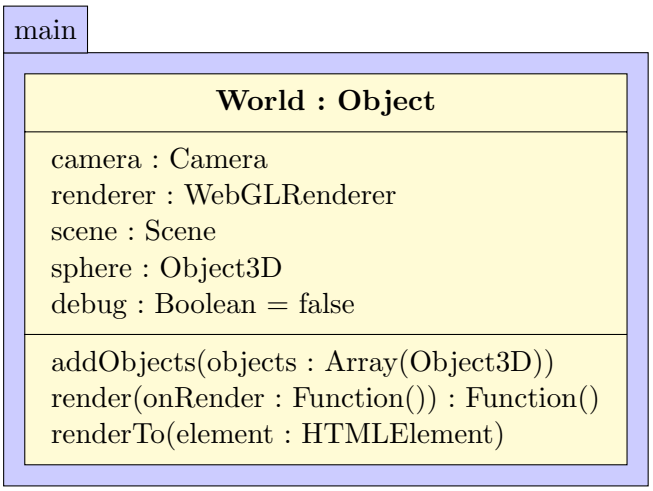
Moduł *ui* zawiera klasy odpowiedzialne za różne elementy interfejsu użytkownika. Są to:

- `DateNav`: odpowiada za interakcję z datą,
- `Loader`: odpowiada za pokazywanie i ukrywanie widoku podczas ładowania danych,
- `Marker`: odpowiada za pokazywanie aktualnych danych dla wybranego punktu na modelu 3D.

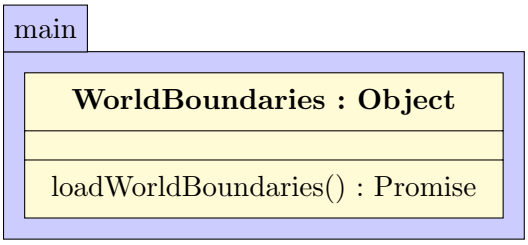


Rysunek 3.3: Klasa Weather

Istnieje także moduł *main*, który zawiera głównej klasy aplikacji, którymi są: *Weather* (zobacz Rysunek 3.3), *World* (zobacz Rysunek 3.4) oraz *WorldBoundaries* (zobacz Rysunek 3.5). Pierwsza klasa definiuje metody potrzebne do pobrania danych meteorologicznych oraz parsuje je do odpowiednich struktur danych. Druga klasa odpowiedzialna jest za stworzenie modelu Ziemi w 3D, definiuje kamerę, scenę oraz odpowiada za ich rysowanie. Ostatnia z wymienionych klas odpowiada za parsowanie i rysowanie kontur lądów oraz państw na modelu 3D.



Rysunek 3.4: Klasa World



Rysunek 3.5: Klasa WorldBoundaries

Plikiem wejściowym jest *index.js*, który importuje klasy z innych modułów, tworzy odpowiednie obiekty i rozpoczyna renderowanie scen.

3.2.2. Kompilacja i budowanie kodu JavaScript

Kompilacji kodu JavaScript odbywa się za pomocą zadań (ang. *tasks*) wykonywanych za pomocą tzw. *task runnera* o nazwie Gulp. Podstawowym zadaniem jest konkatenacja wszystkich plików źródłowych do jednego pliku JavaScript za pomocą narzędzia browserify. Zabieg ten pozwala zmniejszyć ilość niezbędnych zapytań wykonywanych do serwera oraz umożliwia zarządzanie modułami. Podczas konkatenacji każdy plik jest także kompilowany ze standardu ECMAScript 6 do standardu ECMAScript 5 z wykorzystaniem kompilatora Babel.

3.2.3. Three.js

WebGL jest niskopoziomowym API dla języka JavaScript dostępnym w środowisku przeglądarki. Zbudowany jest na podstawie API napisanym w języku C, dlatego wpiera

paradygmat programowania proceduralnego. Daje to ogromne możliwości w kwestii optymalizacji i dostosowania środowiska do konkretnych wymagań, ale sprawia też, że budowanie i rysowanie nawet prostych brył wymaga znacznej ilości kodu oraz ogromnej wiedzy. Pisanie dużej ilości kodu sprawia, że program staje się trudniejszy w utrzymaniu oraz bardziej narażony na błędy logiczne.

Znacznie lepszym rozwiązanie, z punktu widzenia inżynierii oprogramowania, jest wykorzystanie biblioteki, która oparta jest na niskopoziomowym API, ale dostarcza wysokopoziomowe struktury oraz pozwala na pracę w środowisku obiektowym.

Do stworzenia aplikacji webowej została użyta biblioteka Three.js. Three.js jest biblioteką napisaną zgodnie z zasadami programowania obiektowego. Poza obiektowym interfejsem dostarcza wiele abstrakcyjnych klas i obiektów, których nie znajdziemy w standardowym API dostarczonym przez WebGL. Są to np. scena, kamera, wiele modeli geometrycznych brył, oświetlenie. Biblioteka udostępnia także narzędzia do ładowania i kompilowania programów napisanych w języku GLSL oraz do ich debugowania.

3.3. Dokumentacja techniczna aplikacji na Androida

3.3.1. Moduły

Aplikacja składa się z następujących modułów:

Klasa główna

Składa się z jednej klasy *MainActivity.java*, która jest klasą wejściową uruchamiającą pozostałe moduły. Odpowiada także za komunikację z użytkownikiem (obsługuje menu oraz okna dialogowe).

Moduł pobierania danych pogodowych

Moduł ten składa się z dwóch klas: *WeatherDataProvider.java* i *MeteovizWeatherDataProvider.java*. Odpowiada za nawiązanie połączenia z serwerem WWW oraz pobranie danych meteorologicznych. Następnie są one przekazane i parsowane w innym module: *GlobalWeatherData*. Moduł do pobierania danych działa asynchronicznie.

Moduł prezentujący widżet posiadający kontekst OpenGL

SDK dla platformy Android udostępnia klasę *GLSurfaceView*, która odpowiada za wyświetlanie widżetu z kontektem OpenGL. Napisano własną klasę *EarthSurfaceView* która ją rozszerza. *EarthSurfaceView* odpowiada za rozpoczęcie renderingu oraz obsługuje gesty użytkownika.

Moduł odpowiedzialny za rendering

W skład tego modułu wchodzą pliki: *EarthRenderer.java* i *EarthRendererOpenGLES2.java*. Pisząc aplikację na platformę Android z wykorzystaniem OpenGL musimy napisać własną klasę renderującą, której instancja musi zostać przypisana do odpowiedniej klasy typu *GLSurfaceView*. Do tego stworzona została klasa *EarthRendererOpenGLES2*, która implementuje renderer wykorzystujący OpenGL ES w wersji drugiej.

Moduł odpowiedzialny za rendering planety

Moduł ten na początku działania aplikacji generuje sferę przedstawiającą kulę ziemską. W następnej kolejności wczytuje z pliku kontury lądów i granic państw. W trakcie działania aplikacji odpowiada za cykliczne rysowanie ziemi na ekranie. W jego skład wchodzą pliki: *Earth.java*, *EarthOpenGLES2.java*.

Moduł odpowiedzialny za kamerę

Moduł kamery (*Camera.java*) jest odpowiedzialny za ustawienie kamery obserwującej kulę ziemską. Udostępnia API do obracania jej wokół planety oraz przybliżania/oddalania się od niej w określonym zakresie.

Moduł odpowiedzialny za przechowywanie danych pogodowych

Moduł *GlobalWeatherData* odpowiada za parsowanie danych meteorologicznych, przechowywanie oraz udostępnianie ich dla modułów do wizualizacji zjawisk pogodowych na podstawie zadanych koordynatów w świecie 3D.

Moduły odpowiedzialne za wyświetlanie zjawisk pogodowych

Każdy moduł składa się z jednego pliku: *WeatherPresenterOpenGLES2.java*, *WindPresenter.java*, *StormPresenter.java*, *RainPresenter.java*, *TempPresenter.java* implementujący ten sam interfejs. Moduły do wizualizacji zjawisk pogodowych odpowiadają za przedstawienie efektów temperatury, deszczu, burz oraz wiatrów.

Pomocnicze komponenty

Plik *Utils.java* zawiera szereg przydatnych funkcji jak konwersja współrzędnych geograficznych na trójwymiarowy punkt oraz na odwrót. Zawiera także funkcje do konwersji między formatami kolorów, wczytywania zasobów i inne matematyczne funkcje niedostępne w bibliotece standardowej.

3.3.2. NDK

Podczas pisania aplikacji zaszła potrzeba wykorzystania technologii NDK (*Native Development Kit*), która umożliwia pisanie modułów w językach kompilowanych bezpośrednio do kodu maszynowego (C oraz C++) na platformę Android.

W naszym przypadku pożądany jest dostęp do bezpośredniego zarządzania pamięcią. Nasza aplikacja pokazuje zjawiska pogodowe na całej planecie, w związku z czym wszystkie dane meteorologiczne zakodowane w formacie JSON muszą być trzymane w pamięci operacyjnej. Limit pamięci którą może zaalokować aplikacja napisana w Javie jest bardzo niska. Bez wykorzystania natywnego zarządzania pamięcią aplikacja przestałaby działać zaraz po uruchomieniu. Warto zauważyć, że problemem nie jest brak fizycznej pamięci sprzętu, ale zwykłe ograniczenia narzucone na Javę przez maszynę wirtualną Dalvik.

Użycie języka C++ pozwala obejść nie tylko ten limit, ale faktycznie zmniejszyć zapotrzebowanie na pamięć, ponieważ obiekty stworzone w Javie powodują duże narzuty. Dodatkową zaletą jest zwiększenie przepustowość parsowania danych. Dlatego też moduł *GlobalWeatherData* został napisany w języku C++ z wykorzystaniem JNI do komunikacji z kodem w języku Java.

Rozdział 4

Algorytmy

4.1. Algorytm mapowania danych na model Ziemi w 3D

Wizualizacje zjawisk pogodowych tworzone są na powierzchni całej planety, dlatego proces wymaga przejścia przez wszystkie wierzchołki całego modelu 3D i przypisania do nich punktów pomiarowych. Niestety nie możemy użyć standardowej metody, ponieważ otrzymamy złożoność $O(n \cdot m)$, gdzie n – liczba wierzchołków modelu, m – liczba punktów pomiarowych. Dla najlepszego efektu n i m powinny mieć zbliżone wartości. Dla większej liczby punktów jest to zdecydowanie zbyt duża złożoność, dlatego należało wymyślić inny algorytm.

Ponadto warto zauważyć, że generowanie modelu sfery oraz siatki punktów, dla których pobierane są dane meteorologiczne są procesami niezależnymi. Serwer WWW nie potrzebuje informacji o rozmieszczeniu wierzchołków od aplikacji klienckich, udostępnia jedynie listę z pomiarami, która dodatkowo może zawierać pewne puste miejsca w przypadku niepowodzenia pobrania pomiaru.

Nowy algorytm działa w następujący sposób: punkty pomiarowe ze współrzędnymi geograficznymi konwertowane są na punkty w przestrzeni 3D, a następnie każdy z tych punktów dodawany jest do dwupoziomowej mapy (mapa map). Dodawanie punktów do tej struktury odbywa się w taki sposób, że współrzędna y jest kluczem pierwszego poziomu, a konkatenacja wartości współrzędnych x oraz z staje się kluczem drugiego poziomu. Wartością mapy na drugim poziomie jest punkt z przestrzeni 3D.

Przykład 4.1. Załóżmy, że mamy dwa punkty pomiarowe ze współrzędnymi geograficznymi: $P_1 = \{lg_1, lt_1\}$ oraz $P_2 = \{lg_2, lt_2\}$., gdzie lg oznacza długość geograficzną, a lt szerokość geograficzną. W pierwszym kroku konwertujemy te punkty do przestrzeni 3D, aby umieścić je na naszym modelu Ziemi. Otrzymujemy $P_1 = \{x_1, y_1, z_1\}$ oraz $P_2 = \{x_2, y_2, z_2\}$. W drugim kroku dodajemy punkty do dwupoziomowej mapy. Oznaczmy tę mapę literą M. Jeśli $y_1 \neq y_2$ to:

$$M = \begin{cases} \overbrace{y_1}^{\text{klucz pierwszego poziomu}} : \begin{cases} \overbrace{x_1 \oplus y_1}^{\text{klucz drugiego poziomu}} : P_1 \\ \underbrace{x_2 \oplus y_2}_{\text{klucz drugiego poziomu}} : P_2 \end{cases} \\ \underbrace{y_2}_{\text{klucz pierwszego poziomu}} : \end{cases}$$

Jeśli $y_1 = y_2$ to:

$$M = \begin{cases} y_1 : \begin{cases} x_1 \oplus y_1 : P_1 \\ x_2 \oplus y_2 : P_2 \end{cases} \end{cases}$$

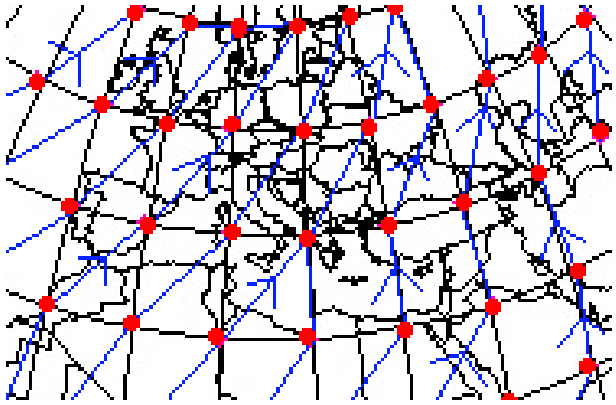
Taka struktura umożliwia odnalezienie pomiaru dla wierzchołka sfery w czasie stałym.

4.2. Algorytm wizualizacji wiatru na Androidzie

Wizualizacja wiatru polega na przesuwaniu pojedynczych, małych, żółtych punktów po powierzchni planety, zgodnie z kierunkiem wiatru. Nazwijmy te punkty **punktami wiatru**. Każdy taki punk jest jest powiązany z wierzchołkiem modelu kuli. Nazwijmy te wierzchołki **punktami kontrolnymi**. Każdy punkt kontrolny posiada informację o swojej współrzędnej w świecie 3D i sile wiatru, a także referencję do następnego punktu kontrolnego, w kierunku którego wieje wiatr. Tak więc punkty kontrolne tworzą listy punktów, gdzie każdy z nich wskazuje na swój następnik, bądź pusty punkt, jeśli siła wiatru w tym miejscu jest na tyle jest słaba, iż możemy uznać, że wiatr zanika.

Na Rysunku 4.1 widzimy punkty kontrolne (czerwone kropki) oraz niebieskie linie, które symbolizują kierunki wiatru. Kierunki te zostały wyznaczone na podstawie punktów pomiarowych w taki sposób, by łączyły w linii prostej punkty kontrolne.

Na początku animacji, przy każdym punkcie kontrolnym, gdzie istnieje dość silny wiatr, pojawia się żółty punkt wiatru (czyli w miejscach czerwonych kropek z Rysunku 4.1). Dany punkt wiatru podąża zgodnie z wyznaczonym kierunkiem do następnego punktu



Rysunek 4.1: Punkty kontrolne i kierunki wiatru

kontrolnego. Gdy dotrze do niego, przenoszony jest na pozycję startową i ponownie wykonuje tę samą animację.

Ruch punktów wiatru ma dwie charakterystyczne cechy: jeśli do punktu kontrolnego, w którym jest słaby wiatr dotrze animowany punkt, to wtedy on znika. Druga cecha, to jeśli do punktu kontrolnego, gdzie jest silny wiatr nie przybędzie żaden punkt wiatru (czyli w tym miejscu wiatr dopiero przybiera na sile), to w tym punkcie kontrolnym punkty wiatru zaczną się pojawiać.

Przykład 4.2. Mamy 3 punkty kontrolne: A, B, C. Punkt wiatru z A podąży do B, a punkt wiatru z B do C. Załóżmy, że w punkcie kontrolnym C wiatr traci na sile i dalej nie wieje, tj. nie pojawi się tam żaden punk wiatru. Natomiast w punkcie A wiatr zwiększa swoją siłę. Zatem w praktyce na ekranie widać 2 punkty wiatru. Użytkownik zauważy, że w punkcie A pojawia się żółty punkt, która zwiększa rozmiar i przesuwa się do punktu B. Punkt wiatru z B będzie w tym czasie przesuwać się do C zmniejszając swoją średnicę, aż zniknie do zera. Oba punkty wiatru dotrą do swojego celu w tym samym momencie. Punkt wiatru z B wróci do pozycji A przywracając swój pierwotny rozmiar. Natomiast punkt wiatru z C wróci do pozycji B również przywracając swój pierwotny rozmiar. Użytkownik będzie jedynie widział, że w punkcie kontrolnym A powstają żółte punkty wiatru, a w C znikają. Zastosowanie takiego algorytmu sprawia, że jeśli do punktu kontrolnego B podąża kilka punktów wiatru, które scalą się w jeden, to użytkownik nie zauważy zmiany liczby tych animowanych punktów. Dzięki takiej technice można utrzymywać stałą liczbę punktów wiatru, bez potrzeby tworzenia i usuwania obiektów z bufora.

4.2.1. Algorytm krok po kroku

Model kuli ziemskiej jest zbudowany z odpowiedniej listy wierzchołków oraz krawędzi. W pierwszym etapie algorytm przegląda listę wszystkich wierzchołków modelu ziemi i dla każdego z nich wyznaczany jest najbliższy punkt pomiaru prędkości i kierunku wiatru. W następnym etapie przeglądana jest lista krawędzi modelu ziemi, gdzie dla każdego wierzchołka wyznaczamy krawędź, po którym powinien poruszać się punkt wiatru. Szukamy najbardziej odpowiedniej, jednej krawędzi, gdyż wiatr zawsze wieje w jedną. W ostatnim etapie dla każdego punktu kontrolnego (wierzchołka modelu ziemi) tworzony jest punkt wiatru, chyba że siła wiatru w tym miejscu jest zbyt mała.

```
# V - zbior wierzcholkow modelu ziemi,
# E - zbior krawedzi, gdzie 3 kolejne punkty tworza trojkat
V = earth.getVertices()
E = earth.getIndices()

for v in V:
  v.control_point = ControlPoint()
  v.weather_data = get_weather_data_for_point(v)

for i in range(0, len(E) / 3):
  # a, b i c to 3 kolejne wierzcholki tworzace trojkat
  # kazdy z tych wierzcholkow ma przypisany control_point
  a, b, c = get_control_points(E, i * 3)

  # Ponizej sprawdzamy na ile linia laczaca punkty a oraz b
  # zgadza sie z kierunkiem wiatru, ktory wieje z punktu a.
  # W ten sposob obliczamy do ktorego sasiedniego wierzcholka
  # modelu ziemi powinien poruszac sie punkt wiatru.
  a.check_accuracy_with(b)
  a.check_accuracy_with(c)
  b.check_accuracy_with(a)
  b.check_accuracy_with(c)
  c.check_accuracy_with(a)
  c.check_accuracy_with(b)

for v in V:
  if v.wind_speed < MIN_WIND_SPEED:
    continue
  # tworzymy punkt wiatru
  v.wind_particle = WindParticle(v)
```

Listing 4.1: Pseudokod tworzenia siatki z kierunkami wiatrów

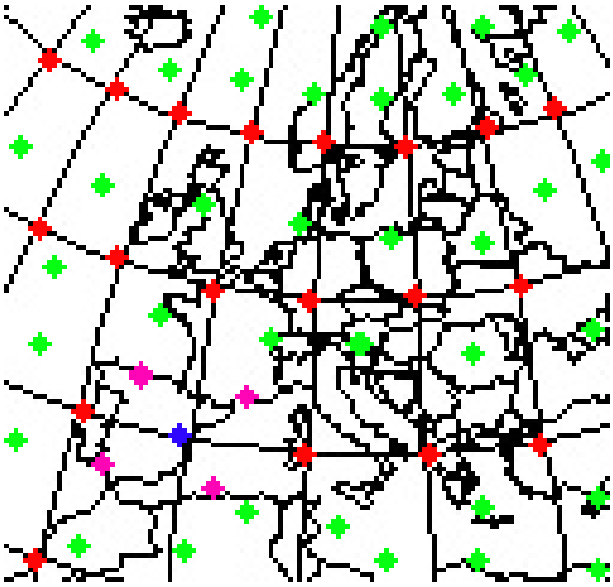
```
# Sprawdzamy na ile polozenie sasiedniego wierzcholka (neighbour)
# zgadza sie z kierunkiem wiatru danego wierzcholka (self).

def check_accuracy_with(self, neighbour):
  A = [self.lat, self.long]
  B = [neighbour.lat, neighbour.long]
  ABd = [B[1] - A[1], B[0] - A[0]]
  for i in range(0, 2):
    if ABd[i] > Math.PI:
      ABd[i] = Math.PI * 2 - ABd[i]
    if ABd[i] < -Math.PI:
      ABd[i] = Math.PI * 2 + ABd[i]
  angle = Math.atan2(ABd[1], ABd[0]) + Math.PI

  # wind_dir zawiera kierunek wiatru podany w stopniach
  wd = Math.toRadians(self.wind_dir)

  # sprawdzamy obliczony kat z rzeczywistym kierunkiem wiatru
  accuracy = Math.abs(angle - wd)
  if accuracy > 180:
    accuracy = 360 - accuracy;
  accuracy = 1 - accuracy / 180
  if accuracy >= self.accuracy:
    self.next_control_point = neighbour
    self.accuracy = accuracy
```

Listing 4.2: Pseudokod funkcji check_accuracy_with

4.3. Algorytm tworzenia heap mapy

Przy różnych wizualizacjach pogodowych mówimy o tak zwanych *heat mapach*, czyli o pewnej technice, którą wykorzystujemy przy kolorowaniu modelu kuli ziemskiej w zależności od danych meteorologicznych.

Na Rysunku 4.2 mamy zaznaczone przykładowe wierzchołki budujące model sfery (punkty czerwone i jeden niebieski) oraz punkty pomiarów pogodowych (punkty zielone i różowe). Oba zbiory są zupełnie niezależne, przy czym jeden otrzymujemy za pośrednictwem aplikacji serwerowej (punkty pomiarowe), a drugi generowany jest przez aplikacje klienckie.



Rysunek 4.2: Wierzchołki modelu ziemi (kolor czerwony i niebieski) oraz miejsca pomiarów (kolor różowy i zielony)

Dla każdego wierzchołka modelu wyznaczane są cztery najbliższe pomiary pogodowe (za wyjątkiem biegunów, gdzie wyznaczamy dwa najbliższe punkty). Sposób tego wyznaczania możemy zobaczyć na Rysunku 4.2 patrząc na cztery punkty różowe i jeden niebieski, odpowiednio punkty pomiarowe i wierzchołek modelu.

```
def interpolate_color(vertex, closest_weather_data):
  distances = 0
  tempDist = 0
  dTemp = MAX_TEMP - MIN_TEMP
  for i in range(0, len(closest_weather_data))
    dist = calc_distance(vertex, closest_weather_data[i])
    temp = closest_weather_data[i].temperature
    if temp < MIN_TEMP:
      temp = MIN_TEMP
    if temp > MAX_TEMP:
      temp = MAX_TEMP
    distances += dist;
    tempDist += dist * (temp - MIN_TEMP) / dTemp
  val = tempDist / distances

  return hsl_to_rgb((1 - val) * 260 / 360, 1, 0.5)
```

Listing 4.3: Pseudokod wyznaczania temperatury dla wierzchołka modelu

Argument *vertex* jest wierzchołkiem modelu kuli ziemskiej, a *closest_weather_data* zbiorem najbliższych punktów pomiaru..

Wykorzystujemy tutaj prostą interpolację liniową. Im punkt pomiaru temperatury leży bliżej wierzchołka, tym ma większy wpływ na na obliczony kolor. Posługujemy się początkowo reprezentacją HSL (*Hue*, *Saturation*, *Lightness*), czyli barwą, nasyceniem i jasnością, a potem konwertujemy ją do RGB. Wykorzystujemy reprezentację HSL dlatego, że pozwala na łatwiejszą zmianę barwy.

W podanym przykładzie obliczamy temperaturę. Temperatury najniższe chcemy pokazać za pomocą koloru fioletowego, a najwyższe za pomocą koloru czerwonego. Jeśli spojrzymy na reprezentację barw w modelu HSL, to widzimy, że tym kolorom odpowiada zakres od 240 do 360 stopni.

Analogicznie możemy przedstawić siłę wiatru, gdzie przestrzeń barw od niebieskiego do zielonego odpowiada zakresowi od 240 do 120 stopni.

Rozdział 5

Analiza porównawcza (Benchmarking)

Testy wydajności obu aplikacji klienckich wykonano w zależności od dwóch czynników:

- liczby punktów pomiarowych,
- liczby punktów modelu geometrycznego Ziemi.

Do pomiaru wydajności obu aplikacji użyto wielkości opisującej liczbę wyświetlanych klatek na sekundę (ang. *fps, frames per second*).

Liczba punktów modelu Ziemi jest uzależniona od liczby podziałów ścian dwudziestościanu. Szczegółowość równa 1 oznacza, że każdą krawędź ściany dwudziestościanu dzielimy na pół. Tak więc z jednej ściany otrzymujemy cztery. Szczegółowość równa 2 oznacza, że każdą ze ścian znowu dzielimy w taki sam sposób itd. Z tego powodu nie jest możliwy dokładny wybór liczby wierzchołków, ale zależy on od wybranej szczegółowości modelu geometrycznego.

Liczba punktów pomiarowych także zależy od sposobu oraz gęstości ich dystrybucji na powierzchni Ziemi.

5.1. Wydajność aplikacji webowej

Testy aplikacji webowej zostały przeprowadzone na komputerze o następujących parametrach:

- System operacyjny: OS X 10.10.3,

- CPU: 1,7 GHz Intel Core i7,
- Pamięć RAM: 8 GB 1600 MHz DDR3,
- Karta graficzna: Intel HD Graphics 5000 1536 MB.

Do pomiarów wydajności użyto najnowszej wersji przeglądarki Google Chrome w wersji 42.0.2311.90.

Otrzymano następujące wyniki dla stałej liczby wierzchołków modelu i zmiennej liczby punktów pomiarowych:

Tabela 5.1: Wyniki pomiaru wydajności przy stałej liczbie 10242 wierzchołków modelu

Liczba pkt. pomiarowych	Fps temperatury	Fps burz i opadów	Fps wiatru
1950	53	51	45
8518	54	48	27
28357	53	43	15

Otrzymano następujące wyniki dla stałej liczby punktów pomiarowych i zmiennej liczby wierzchołków siatki geometrycznej modelu Ziemi:

Tabela 5.2: Wyniki pomiaru wydajności przy stałej liczbie 8518 punktów pomiarowych

Liczba wierzchołków	Fps temperatury	Fps burz i opadów	Fps wiatru
2562	53	51	28
10242	53	51	28
40962	52	52	28

Tabela 5.3: Wyniki pomiaru wydajności przy stałej liczbie 28357 punktów pomiarowych

Liczba wierzchołków	Fps temperatury	Fps burz i opadów	Fps wiatru
2562	53	44	13
10242	51	43	13
40962	52	44	13

5.1.1. Wnioski

Szybkość wizualizacji temperatury, która zależy jedynie od liczby punktów siatki, jest stała. Powyższe pomiary świadczą o tym, że aplikacja byłaby w stanie wyświetlić jeszcze bardziej szczegółowe dane.

Różnice między wydajnością renderowania wizualizacji burz i opadów oraz wiatru, pomiędzy kolejnymi poziomami liczby punktów pomiarowych, spowodowane są dodatkowymi elementami, jakie te wizualizacje przedstawiają. Dodatkowe elementy zależą jedynie od ilości punktów pomiarowych. W przypadku wiatru, dla każdego punktu pomiarowego rysowana jest styczna przedstawiająca kierunek wiatru (dla każdego punktu pomiarowego potrzebne są dwa wierzchołki do narysowania fragmentu stycznej). Natomiast w przypadku deszczu są to obiekty reprezentujące wysokości opadów (dodatkowo 8 punktów dla jednego punktu pomiarowego). Jednak w przypadku wizualizacji opadów i burz liczba wybranych punktów jest ograniczona przez wielkość opadów. Natomiast w przypadku wizualizacji wiatrów, fragment stycznej jest rysowany dla każdego punktu, dlatego też jest ona najmniej wydajna

Spadki w wydajności renderowania wiatrów spowodowane są zwiększoną ilością pamięci potrzebną do narysowania odcinków reprezentujących kierunki wiatrów.

Czas parsowania danych pomiarowych w zależności od ilości tych danych praktycznie się nie zmienia. Bardziej widoczne zmiany można zaobserwować zwiększając liczbę punktów siatki. Potwierdza to, że algorytm do mapowania punktów pomiarowych na punkty siatki jest wydajny.

Optymalnym wyborem jeśli chodzi o szybkość i jakość wizualizacji jest 12649 punktów pomiarowych oraz 10242 wierzchołków siatki.

5.2. Wydajność aplikacji na Androida

Podczas testów wykorzystano urządzenie z systemem Android o następujących parametrach:

- Wersja Androida: 4.1.2,
- API Level: 16,
- CPU ABI: armaebi-v7a,
- RAM: 818 MB,
- Procesor o taktowaniu 1.2 GHz,
- Renderer OpenGL: Adreno (TM) 203,
- Rozdzielczość: 540x960.

Otrzymano następujące wyniki dla stałej liczby wierzchołków modelu i zmiennej liczby punktów pomiarowych:

Tabela 5.4: Wyniki pomiaru wydajności przy stałej liczbie 30726 wierzchołków modelu

Liczba pkt. pomiarowych	Fps temperatury	Fps burz i opadów	Fps wiatru
10000	15	15	8-9
20000	15	15	8-9
40000	15	15	8-9
80000	15	15	8-9

Otrzymano następujące wyniki dla stałej liczby punktów pomiarowych i zmiennej liczby wierzchołków siatki modelu Ziemi:

Tabela 5.5: Wyniki pomiaru wydajności przy stałej liczbie 40000 punktów pomiarowych

Liczba wierzchołków	Fps temperatury	Fps burz i opadów	Fps wiatru
1926	17	17	17
7686	17	17	17
30726	15	15	8-9

5.2.1. Wnioski

Nie występują praktycznie żadne różnice w wydajności aplikacji w zależności od ilości danych meteorologicznych, ponieważ liczba pomiarów nie wpływa na proces wyświetlania grafiki. Występuję jedynie wzrost czasu potrzebnego na parsowanie danych.

Głównym czynnikiem spowalniającym szybkość wyświetlania jest szczegółowość modelu Ziemi. Dzieje się tak dlatego, że właśnie od tej zmiennej zależy ilość pracy jaką musi wykonać karta graficzna oraz ilość niezbędnej pamięci na tej karcie.

Zauważono, że limitem szybkości renderowania grafiki dla aplikacji na Andorida jest liczba około 17 fps. Problem dotyczy jednak całego badanego urządzenia. Mimo to, aplikację obsługiwało się wystarczająco komfortowo i można uznać, że do 7686 wierzchołków nie powinno być problemów z wydajnością.

Problem pojawia się przy wielkości ponad 30000 wierzchołków. Szczególnie wizualizacja ruchu cząsteczek wiatru zajmuje sporo czasu. Z drugiej strony, ruch nawet przy 8-9 klatkach na sekundę wyglądał wystarczająco płynnie, a wykorzystane urządzenie

pozycjonowane jest znacznie poniżej przeciętnej na rynku pod kątem wydajności grafiki 3D.

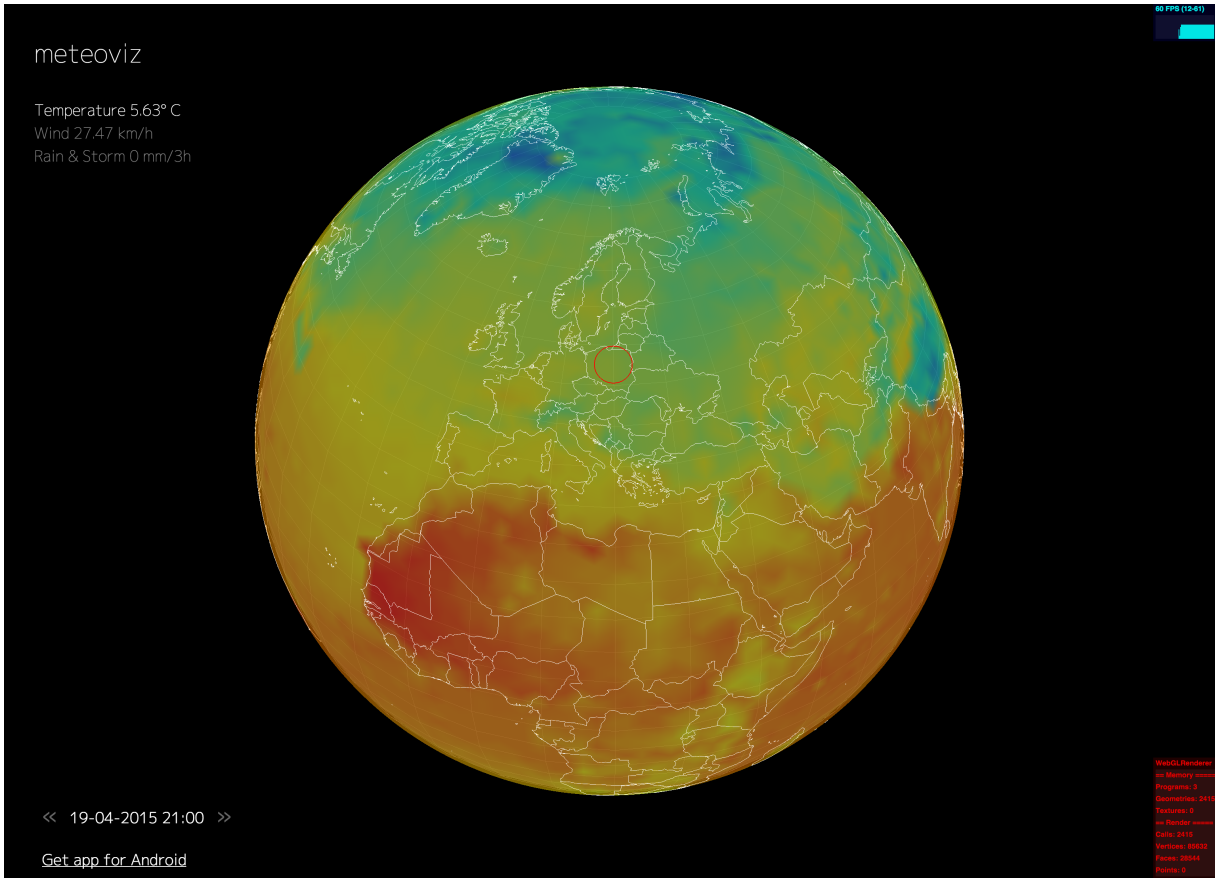
Z uwagi na to, że istnieje bardzo wysoki spadek jakości grafiki przy przejściu z 30000 na 7000 wierzchołków, domyślna liczba wierzchołków modelu ziemi wynosi 30726. Sfera jest zbudowana na podstawie dwudziestościanu i liczba wierzchołków zależy od liczby rekursywnych podziałów tej bryły. Liczba 30726 wierzchołków odpowiada 5 podziałom, 7686 odpowiada 4 podziałom. Zatem niemożliwym było przy tym modelu wybrać wartość pośrednią.

Dzięki zastosowaniu wydajnego algorytmu do wyszukiwania danej pomiarowej odpowiadającej wierzchołkowi modelu, ilość tych danych jest nieistotna. Aczkolwiek jeśli danych pomiarowych jest o wiele więcej niż wierzchołków modelu ziemi, to część danych jest pomijana. Jeśli liczba punktów siatki modelu jest zbyt wielka w porównaniu z liczbą pomiarów, to jeden pomiar pogody zostanie przypisany wielu wierzchołkom modelu. Dlatego optymalnym rozwiązaniem jest utrzymywanie równej liczby obu tych danych.

Rozdział 6

Dokumentacja użytkownika

6.1. Aplikacja webowa



Rysunek 6.1: Wizualizacja temperatury w aplikacji webowej

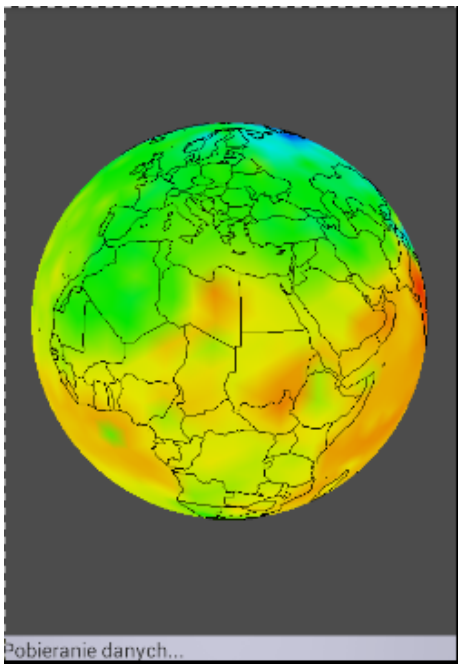
Interfejs aplikacji webowej został przedstawiony na Rysunku 6.1 i został zbudowany w następujący sposób: w centrum znajduje się model kuli ziemskiej, w lewym górnym

rogu jest menu do wyboru typu wizualizacji, a w lewym dolnym rogu menu do zmiany daty. Poza tym w prawym górnym rogu wyświetla się licznik fps, a w dolny statystyki utworzonych obiektów 3D.

W lewym dolnym rogu znajduje się również link do pobrania aplikacji na platformę Android.

Użytecznym elementem jest także marker na modelu Ziemi oznaczony czerwonym okręgiem. Klikając na modelu w dowolnym miejscu, możemy go przesunąć i odczytać szczegółowe dane meteorologiczne.

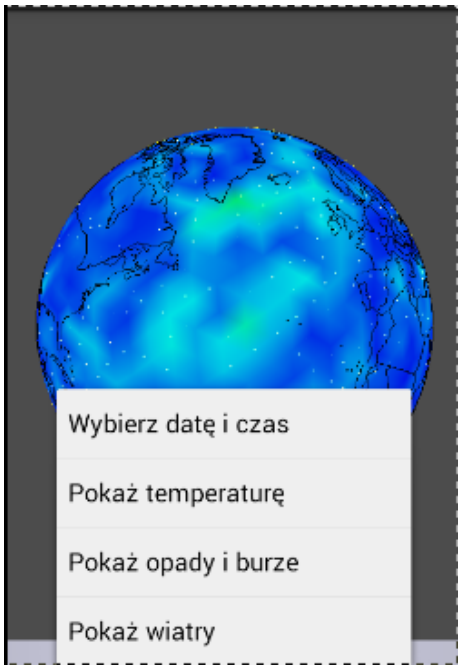
6.2. Aplikacja na platformę Android



Rysunek 6.2: Okno główne aplikacji

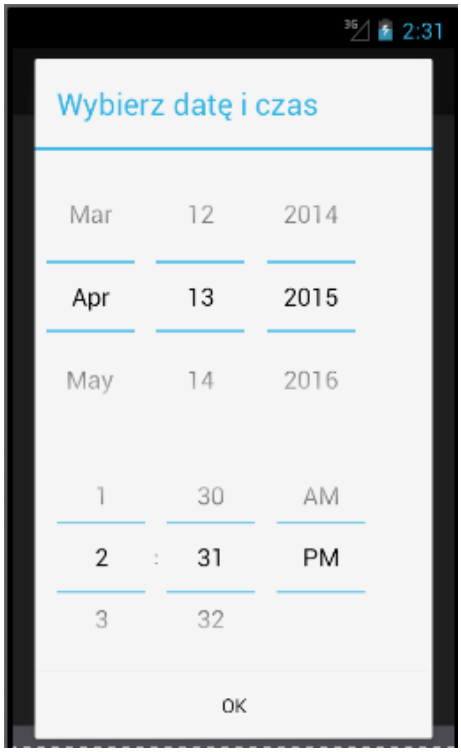
Po otworzeniu i załadowaniu aplikacji zobaczymy główny ekran jak na Rysunku 6.2. Centralną część zajmuje model kuli ziemskiej, który możemy przybliżać, oddalać oraz obracać. Na dole jest dodatkowy pasek z statusem informacji. Może on zawierać informacje:

- *Pobieranie danych*: informuje że aplikacja ładuje dane pogodowe,
- *Ładowanie widoku*: informuje że ładowany jest nowy widok (zmieniana jest wizualizacja),
- *Data i czas*: wskazują datę dla której wyświetlane są dane pogodowe,



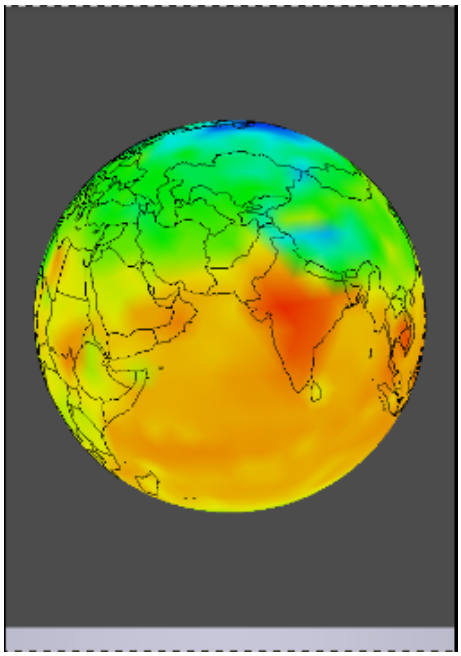
Rysunek 6.3: Menu aplikacji

- ewentualny komunikat o błędzie, np. w przypadku braku połączenia z Internetem.



Rysunek 6.4: Wybór daty i czasu pogody

Po uruchomieniu przycisku menu na telefonie uzyskamy widok jak na Rysunku 6.3. Możemy tutaj zmienić rodzaj wizualizacji (temperatura, wiatry, opady i burze), oraz wybrać czas, dla którego chcemy wyświetlić dane pogodowe.



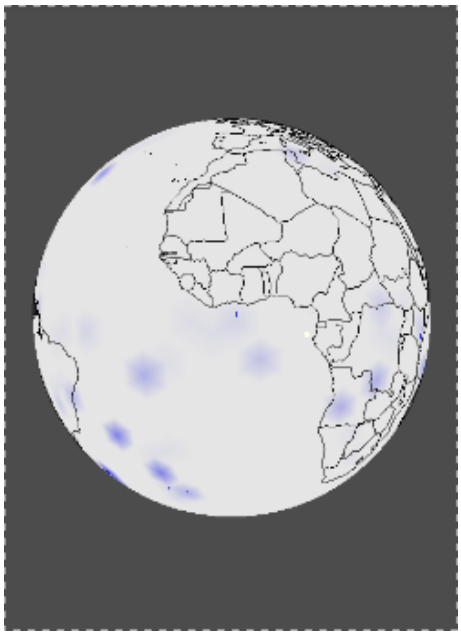
Rysunek 6.5: Wizualizacja temperatury

Po wybraniu opcji „Wybierz datę i czas" pokaże się nam okno dialogowe jak przedstawiono na Rysunku 6.4. Pozwala ono na wybór daty i czasu, dla którego chcemy wyświetlić pogodę. Trzeba zwrócić uwagę, że nie są obsługiwane prognozy pogody: wybranie przyszłej daty spowoduje wskazanie obecnej pogody.

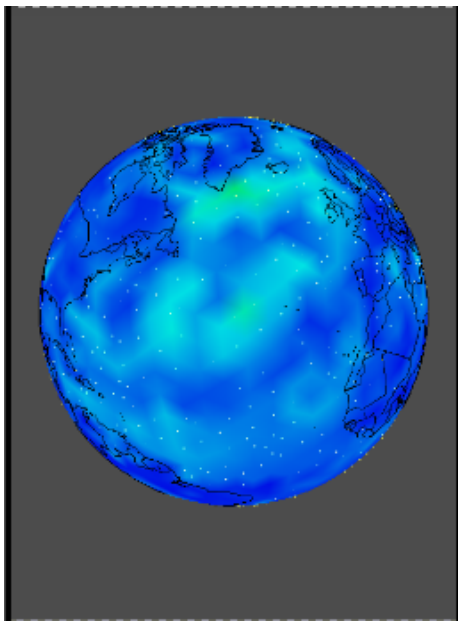
Na Rysunku 6.5 widzimy wizualizację temperatury. Kolory fioletowe i niebieski oznaczają obszary o niskich temperaturach, kolor czerwony o wysokich.

Na Rysunku 6.6 widzimy prezentację deszczu oraz burz. Obszary, gdzie występują opady, zostały oznaczone na niebiesko i razem z animacją opadających kropel deszczu. Im mocniejszy deszcz, tym silniejszy kolor niebieski. Burze są oznaczone żółtymi kulami (jak ta w pobliżu Afryki), które animują się przez powiększanie i pomniejszanie.

Na Rysunku 6.7 powyżej możemy zobaczyć wizualizacje wiatru. Obszary pokolorowane na niebiesko oznaczają ciszę: brak wiatru bądź jest on bardzo słaby. Kolor zielony oznacza silny wiatr. Kierunki wiatru są pokazane za pomocą cząsteczek – żółte punkty przemieszczające się po powierzchni planety.



Rysunek 6.6: Wizualizacja deszczu i burz



Rysunek 6.7: Wizualizacja wiatru

Rozdział 7

Przebieg pracy

7.1. Model wytwórczy

Praca dyplomowa została napisana w trybie iteracyjnym i składała się z czterech etapów (ang. *milestones*). Oznacza to, że każdy etap polegał na stworzeniu kompletnego fragmentu funkcjonalności i zaprezentowaniu jej wraz z kodem źródłowym, testami jednostkowymi oraz dokumentacją techniczną.

Główną przyczyną wyboru metody iteracyjnej jest wykorzystywanie nowych i nieznanych nam bliżej technologii. Dzięki tej metodzie już po pierwszym etapie mogliśmy stwierdzić czy nasze założenia mogą zostać zrealizowane oraz czy wybrane technologie są wystarczające do realizacji całej pracy. Metoda iteracyjna pozwalała nam także na sprawdzenie dostępnych źródeł danych oraz implementację algorytmów, które odpowiadały za ich import już na samym początku pracy nad pracą dyplomową.

7.2. Podział prac

Nasza praca dyplomowa składała się z trzech komponentów i była rozłożona na cztery etapy, każdy trwający od dwóch do trzech tygodni. Osobą odpowiedzialną za aplikacje webową był Adam Babik, natomiast za aplikację na platformę Android Alicja Celigowska. Prace nad aplikacją serwerową były wspólne, ale podzielone tak, że algorytmy pobierające dane stworzyła Alicja Celigowska, a konfigurację aplikacji, serwera WWW oraz instalację na maszynie wirtualnej wykonał Adam Babik.

7.3. Harmonogram

Szczegółowy podział prac zadań znajduje się poniżej. Na liście zadań w nawiasach okrągłych podano osobę odpowiedzialną.

7.3.1. Etap 1

Data zakończenia pracy: 27.11.2014.

Pierwszy etap obejmował pracę nad aplikacją serwerową oraz aplikacją webową. Na tym etapie dane meteorologiczne powinny być pobierane, zapisywana i udostępniane aplikacjom klienckim. Aplikacja webowa powinna potrafić wyświetlić model Ziemi wraz z konturami lądów oraz powinna reagować na operacje użytkownika.

Aplikacja serwerowa

- Pobieranie danych meteorologicznych z wybranego serwisu (Alicja Celigowska),
- Zapis danych do lokalnej bazy danych (Alicja Celigowska),
- API do pobierania danych - podstawowe (Alicja Celigowska).

Aplikacja webowa

- Wyświetlanie modelu Ziemi wraz z południkami, równoleżnikami oraz konturami lądów (Adam Babik),
- Możliwość obracania, przybliżania i oddalania modelu 3D (Adam Babik),
- Weryfikacja połączenia z aplikacją serwerową z wykorzystaniem technologii AJAX (Adam Babik).

Etap został ukończony zgodnie z terminem, a zaplanowane zadania zostały zrealizowane w 100%.

7.3.2. Etap 2

Planowana data zakończenia pracy: 11.12.2014, ostateczna data: 18.12.2014.

Drugi etap obejmował dalszą pracę nad aplikacją webową oraz rozpoczęcie prac nad aplikacją na platformę Android. Zadania dla tej drugiej pokrywają się z zadaniami dla aplikacji webowej z pierwszego etapu.

Aplikacja webowa

- Pobieranie i parsowanie danych z aplikacji serwerowej (Adam Babik),
- Przygotowanie wizualizacji dwóch wybranych danych meteorologicznych na modelu 3D (Adam Babik).

Aplikacja na Androida

- Wyświetlanie modelu Ziemi wraz z południkami, równoleżnikami oraz konturami lądów (Alicja Celigowska),
- Możliwość obracania, przybliżania i oddalania modelu 3D (Alicja Celigowska),
- Weryfikacja połączenia z aplikacją serwerową (Alicja Celigowska).

Czas potrzebny na realizację zadań dla etapu drugiego został przedłużony o tydzień. Zostało to spowodowane zbyt optymistycznym założeniem co do czasu wykonania wizualizacji danych meteorologicznych w aplikacji webowej. Mimo tych trudności zaplanowane zadania zostały zrealizowane w 100%.

7.3.3. Etap 3

Data zakończenia pracy: 8.01.2014.

Trzeci etap zakładał zakończenie prac nad tworzeniem reszty wizualizacji zjawisk meteorologicznych w aplikacji webowej oraz rozpoczęcie prac nad wyświetlaniem tych danych w aplikacji na platformę Android.

Aplikacja webowa

- Stworzenie wizualizacji pozostałych wizualizacji (Adam Babik).

Aplikacja na Androida

- Pobieranie danych z serwera WWW za pomocą protokołu HTTP (Alicja Celigowska),
- Wyświetlanie dwóch wybranych zjawisk meteorologicznych (Alicja Celigowska).

Zadania zostały zrealizowane w 100%, zaplanowano także optymalizacje, które mają na celu poprawienie szybkości oraz redukcję zniekształceń na wizualizacjach opartych o heat mapę.

7.3.4. Etap 4

Data zakończenia pracy: 22.01.2014.

Ostatni etap to ukończenie pozostałych zadań, optymalizacja, poprawienie znalezionych błędów oraz przygotowanie wersji produkcyjnej.

Aplikacja serwerowa

- Przygotowanie maszyny wirtualnej (Adam Babik),
- Instalacja oprogramowania, konfiguracja serwera proxy oraz aplikacji WWW (Adam Babik).

Aplikacja webowa

- Optymalizacje modelu sfery (Adam Babik),
- Testy aplikacji na najnowszych wersjach popularnych przeglądarek (Adam Babik).

Aplikacja na Androida

- Stworzenie pozostałych dwóch wizualizacji (Alicja Celigowska),
- Optymalizacje, (Alicja Celigowska),
- Przygotowanie paczki instalacyjnej (Alicja Celigowska).

Zadania zostały zrealizowane w 100%, a finalne wersje oprogramowania przedstawione do oceny przez promotora pracy dyplomowej.

Rozdział 8

Bibliografia

[1] Strona opisująca algorytm tworzenia sfery
http://blog.andreaskahler.com/2009/06/creating-icosphere-mesh-in-code.html

[2] Dokumentacja Django
https://docs.djangoproject.com/en/1.7/

[3] Dokumentacja SDK Androida
http://developer.android.com/guide/index.html

[4] Dokumentacja NDK Androida
https://developer.android.com/tools/sdk/ndk/index.html

[5] OpenWeatherMap
http://openweathermap.org/

[6] Dokumentacja OpenGL ES
https://www.khronos.org/opengles/sdk/docs/man/

[7] Dokumentacja biblioteki Three.js
http://threejs.org/docs/

[8] WebGL 1.0 Specification
https://www.khronos.org/registry/webgl/specs/latest/1.0/

[9] Draft Specification for ECMAScript 6
http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts

[10] Professional WebGL Programming: Developing 3D Graphics for the Web, Andreas Anyuru, 8 Maj 2012

Adam Babik,
Alicja Celigowska
Nr albumu

Warszawa, 3 maja 2015

Oświadczenie

Oświadczam, że pracę inżynierską pod tytułem „Wizualizacja zjawisk pogodowych", której promotorem jest mgr inż. Paweł Rzążewski wykonałem samodzielnie, co poświadczam własnoręcznym podpisem.

.....

Adam Babik,
Alicja Celigowska