POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA
SPECJALNOŚĆ: INŻYNIERIA SYSTEMÓW INFORMATYCZNYCH

PRACA INŻYNIERSKA

Zarządzanie zadaniami w systemie obrazowania wielospektralnego

Task management for hyperspectral imaging system

AUTOR:
Aleksander Cieślak

PROWADZĄCY PRACĘ:
dr inż. Tadeusz Tomczak

OCENA PRACY:

WROCŁAW, 30 października 2016

Spis treści

Spis rysunków

Spis tabel

Rozdział 1

Cel projektu

Celem niniejszej pracy jest projekt i implementacja modułu zarządzania zadaniami dla systemu Gerbil (`http://gerbilvis.org/`). Jest to system do analizy i wizualizacji danych wielospektralnych. Gerbil posiada zestaw potężnych algorytmów przetwarzania obrazów oraz uczenia maszynowego, które przekładają się na szerokie spektrum funkcjonalności. Jednak jego słabym punktem jest warstwa zarządzania danymi oraz potok przetwarzania danych. To z kolei powoduje niestabilność całej aplikacji. W ramach pracy dyplomowej został zaproponowany system, który rozwiązuje wyżej wspomniane problemy. System ten pozwala na bezpieczny dostęp do danych w całej aplikacji oraz gwarantuje zachowanie właściwego potoku przetwarzania danych.

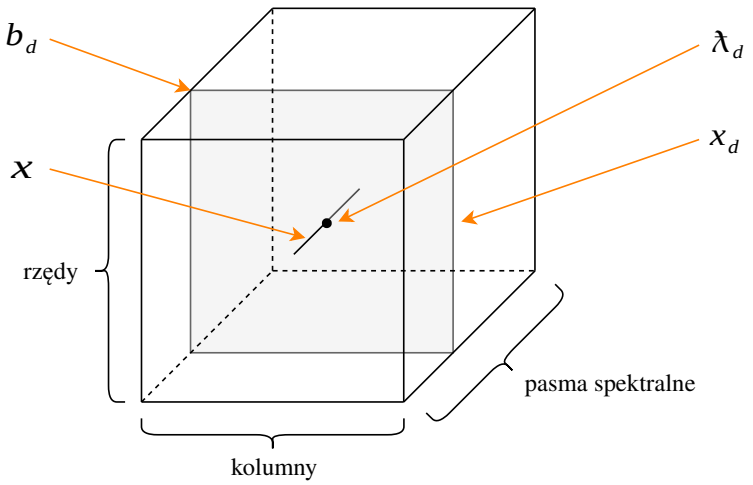
Rozdział 2

Obrazowanie wielospektralne

Obrazowanie wielospektralne jest techniką rejestracji obrazu za pomocą fal elektromagnetycznych o wybranej częstotliwości spośród widma spektroskopowego. Podczas gdy ludzkie oko widzi w głównie w trzech zakresach spektralnych (czerwonym, niebieskim oraz żółtym), obraz wielospektralny jest rejestrowany w znacznie większej ilości zakresów (przykładowo 31).

2.1. Format danych

Dane wielospektralne są często nazywane kostką wielospektralną.



Rys. 2.1: Schemat kostki wielospektralnego

Na rysunku 2.1 zilustrowano układ danych w kostce wielospektralnej. Kostka taka składa się z n_x pikseli x. Każdy piksel jest wektorem współczynników spektralnych o długości n_D, gdzie n_D jest liczbą obrazów spektralnych, na które składa się dana wielospektralna. Każdy współczynnik x_d jest wartością reakcji sensorycznej dla odpowiadającego pasma spektralnego b_d skoncentrowanego wokół fali λ_d. W skrócie obraz wielospektralny jest zbiorem obrazów rejestrowanych przy użyciu fal elektromagnetycznych o zadanych długościach.

2.1.1. Konsekwencje formatu danych

Ze względu na swoją charakterystykę obrazy wielospektralne mogą bezproblemowo osiągać rozmiary setek megabajtów, lub nawet gigabajtów. Większość danych pochodnych, które są efektem analizy tego obrazu posiadają podobne rozmiary. Informacja ta jest kluczowa podczas projektowania mechanizmu zarządzania danymi w takim systemie. Biorąc pod uwagę rozmiar danych mechanizm taki powinien:

- unikać tworzenia zbędnych kopi danych,
- dokonywać obliczeń danych wyłącznie na żądanie,
- zwalniać z pamięci dane, które nie są już wizualizowane przez aplikację.

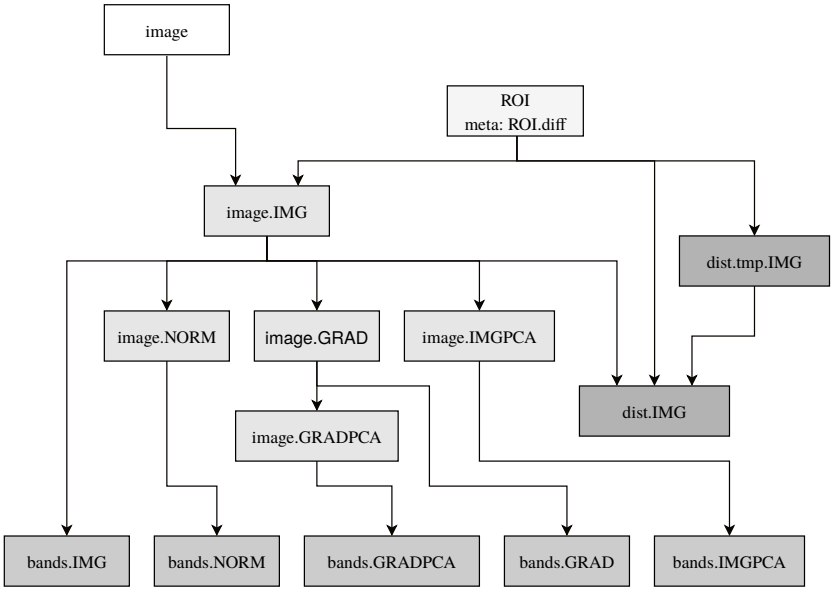
2.2. Dane w systemie Gerbil

Oryginalny obraz wielospektralny jest traktowany jako dana wejściowa w systemie. Na jego podstawie powstają dane pochodne. Są to głównie kolejne obrazy oraz histogramy wielospektralne. Do stworzenia prototypu mechanizmu zarządzania danymi oraz procesem przetworzenia użyte zostały poniższe dane:

- **image** – oryginalny obraz wielospektralny. Dana ta jest obliczana podczas inicjalizacji aplikacji. Użytkownik może wejść w interakcję z systemem dopiero gdy image zostanie przetworzone.
- **ROI (Region of Interest)** – wyselekcjonowany podzbiór danych, w tym przypadku wybrane prostokątne zaznaczenie obrazu,
- **image.IMG** – fragment obrazu oryginalnego zdeterminowany przez ROI,
- **image.NORM** – image.IMG po normalizacji wektora agregującego piksele na przestrzeni pasm spektralnych,
- **image.GRAD** – gradient obrazu image.IMG,
- **image.PCA** – image.IMG po zastosowaniu metody PCA (analizy głównych składowych),
- **image.GRADPCA** – image.GRAD po zastosowaniu metody PCA,
- **bands.*.N** – pojedynczy N-ty obraz spektralny danej reprezentacji (przykładowo bands.NORM.6),
- **dist.IMG** - histogram wielospektralny obrazu image.IMG,
- **dist.tmp.IMG** - dana pomocnicza używana do uzyskania danej dist.IMG.

Z racji, że jedne dane produkują inne, łatwo jest zdefiniować hierarchię danych w tym systemie.

Na rysunku 2.2 przedstawiono diagram zależności danych. Dane jednego koloru są do siebie semantycznie zbliżone. Przykładowo, image.NORM, image.GRAD, image.GRADPCA itp. są reprezentacjami obrazu oryginalnego. Dane posiadają również swoje metadane. Przykładowo metadaną ROI jest ROI.diff, które określa różnicę pomiędzy aktualnym a poprzednim ROI.



Rys. 2.2: Graf zależności danych w systemie Gerbil

2.2.1. Wpływ hierarchii danych na proces wykonania

Analizując rysunek 2.2 można dojść do wniosku, że proces przetworzenia danych jest dyktowany poprzez ich hierarchię. Przykładowo, do obliczenia image.GRADPCA wymagane jest aby dane image, ROI, image.IMG oraz image.GRAD były już przetworzone. Dodatkowo można określić porządek, w którym te dane powinny zostać obliczone:

1. image (podczas inicjalizacji systemu),
2. ROI,
3. image.IMG,
4. image.GRAD,
5. image.GRADPCA.

Rozdział 3

Technologie wykorzystane w systemie Gerbil

3.1. C++

System Gerbil jest rozwijany w języku C++.

3.1.1. STL

3.2. Qt

Qt jest platformą deweloperską wyposażoną w narzędzia pozwalające usprawnić proces wytwarzania oprogramowania oraz interfejsów użytkownika dla aplikacji desktopowych, wbudowanych bądź mobilnych. [1]

Platforma Qt posiada szerokie spektrum funkcjonalności. Między innymi są to:

- system meta-obiektów,
- mechanizm sygnałów i slotów służący do komunikacji pomiędzy obiektami,
- wbudowany system przynależności obiektów,
- wieloplatformowe wsparcie modułu wielowątkowości.

W systemie Gerbil jest wykorzystywane Qt w wersji 5.7.

3.2.1. Sygnały i sloty

Spośród rozrzerzeń języka C++, jakie oferuje Qt na szczególną uwagę zasługuje mechanizm sygnałów i slotów. Dzięki niemu możliwe jest skomunikowanie dwóch dowolnych obiektów w sposób alternatywny do użycia wywołań zwrotnych.

Sygnał jest wysyłany, gdy nastąpi jakieś zdarzenie (np. naciśnięcie przycisku przez użytkownika), natomiast slot jest odpowiedzią na ten sygnał. Sygnatury sygnału i slotu muszą być

[1] Qt 5.7 `http://doc.qt.io/qt-5/index.html`

zgodne. Mechanizm ten jest luźno powiązany (ang. loosely coupled). Oznacza to, że klasa emitująca sygnał nie musi być swiadoma klasy odbierającej. Sygnały i sloty pozwalają na przekazanie dowolnej liczby argumentów dowolnego typu. Sygnały muszą zostać zadeklarowane po słowie kluczowym `signals`. Z jednym slotem można połączyć dowolną ilość sygnałów, i odwrotnie – z jednym sygnałem można skojarzyć dowolną ilość slotów.

Wszystkie klasy korzystające z tego mechanizmu muszą w swojej deklaracji zawierać makro `Q_OBJECT` oraz dziedziczyć (bezpośrednio bądź pośrednio) po klasie `QObject`. [2]

Składnia

Sposób tworzenia połączeń zostanie zilustrowany na przykładzie. Za punkt wyjścia posłużą dwie klasy: `Sender` (Listing 3.1) oraz `Receiver` (Listing 3.2).

Listing 3.1: Klasa Sender

```
class Sender : public QObject
{
    Q_OBJECT
public:
    explicit Sender(QObject *parent = 0) : QObject(parent) {}

signals:
    void sendMessage(QString msg);

};
```

Listing 3.2: Klasa Receiver

```
class Receiver : public QObject
{
    Q_OBJECT
public:
    explicit Receiver(QObject *parent = 0) : QObject(parent) {}

    void receiveMessageMethod(QString msg) {
        std::cout << "got message in method: " << msg;
    }

public slots:
    void receiveMessageSlot(QString msg) {
        std::cout << "got message: " << msg;
    }
};
```

[2] Signals & Slots | Qt Core 5.7 `http://doc.qt.io/qt-5/signalsandslots.html`

Z analizy listingów 3.1 oraz 3.2 wynika, że klasa `Sender` zawiera sygnał `sendMessage`, natomiast klasa `Receiver` zawiera publiczną metodę `receiveMessageMethod` oraz publiczny slot `receiveMessageSlot`.

W Qt występują dwa rodzaje składni pozwalające na ustanowienie połączenia. Jedna z nich (starsza) pozwala na ustanowienie połączenia jedynie pomiędzy sygnałem a sygnałem, bądź sygnałem a slotem. Drugi rodzaj składni, wprowadzony w Qt5 pozwala dodatkowo na nawiązanie połączenia pomiędzy sygnałem a metodą klasy. Na listingu 3.3 przedstawiony jest zarówno stary jak i nowy zapis.

Listing 3.3: Składnia tworzenia połączeń między obiektami

```
Sender sender;
Receiver receiver;

//stara skladnia
//poprawne
QObject::connect(&sender, SIGNAL(sendMessage(QString)), &receiver,
    ↳ SLOT(receiveMessageSlot(QString)));
//niepoprawne
QObject::connect(&sender, SIGNAL(sendMessage(QString)), &receiver,
    ↳ SLOT(receiveMessageMethod(QString)));

//nowa skladnia
QObject::connect(&sender, &Sender::sendMessage, &receiver,
    ↳ &Receiver::receiveMessageMethod);
QObject::connect(&sender, &Sender::sendMessage, &receiver,
    ↳ &Receiver::receiveMessageSlot);
```

3.2.2. Wątek GUI oraz wątki robocze

3.3. boost

3.4. TBB

3.5. OpenCV

Rozdział 4

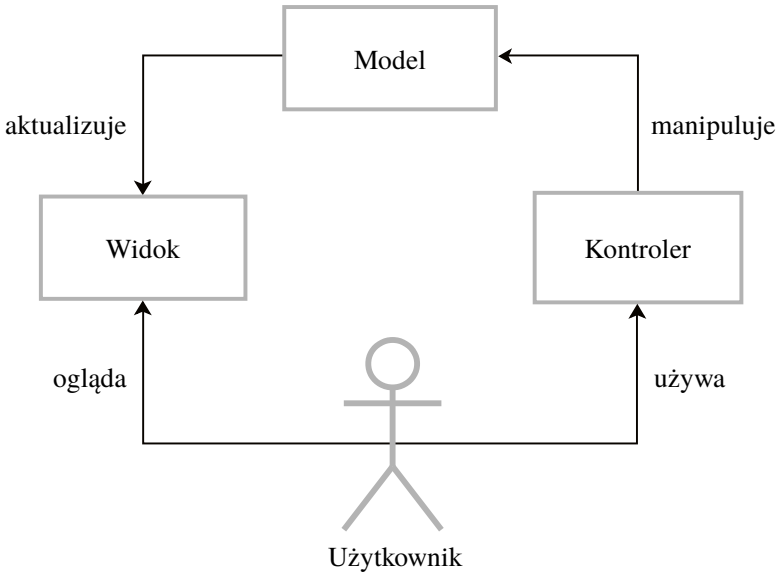
Aktualny stan projektu Gerbil

4.1. Wzorzec MVC

Aplikacja Gerbil jest zaprojektowana według wzorca MVC (Model-View-Controller) z wykorzystaniem platformy Qt. MVC jest wzorcem architektonicznym używanym często do tworzenia interfejsów użytkownika. Podstawą MVC są trzy obiekty:

- model - komponent odpowiedzialny za serwowanie danych,
- widok - komponent odpowiedzialny za wizualizację danych,
- kontroler - komponent definiujący logikę, za pomocą której interfejs użytkownika odpowiada na żądania.

Podział tych ról można zaobserwować na rysunku 4.1.



Rys. 4.1: Podział ról we wzorcu architektonicznym MVC

Dzięki wykorzystaniu tego wzorca sposób, w jakim przechowywane są dane nie ma wpływu na to jak są one przedstawione użytkownikowi. [1]

4.2. Architektura aplikacji Gerbil

W aplikacji Gerbil wzorzec MVC zastosowano w sposób klasyczny:

- modele są odpowiedzialne za obliczenia danych oraz sygnalizowanie pojawienia się ich nowej wersji,
- widoki wyświetlają dane,
- kontrolery zajmują się kojarzeniem akcji użytkownika z konkretną funkcjonalnością modelu.

Dodatkowo w aplikacji występuje wątek roboczy. W nim uruchomiona jest kolejka zadań. Zadanie (Task)jest komponentem realizującym wykonanie czasochłonnego algorytmu analizy danych. Modele tworzą zadania i przekazują je do kolejki. Kolejka przyjmuje zadania i wykonuje je po kolei. W ten sposób skomplikowane obliczenia nie blokują wątku GUI, które pozostaje przez cały czas responsywne.

W takiej architekturze pojawia się problem dostępu do danych, ponieważ dwa wątki (wątek GUI, w którym znajdują się komponenty MVC oraz wątek roboczy, w którym wykonywane są zadania) próbują uzyskać dostęp do tych samych danych. Zadania wykonywane w tle powinny w bezpieczny sposób dokonywać zapisu danych. Widoki zaś powinny być w stanie bezawaryjnie wizualizować dane oraz zadbać o aktualność prezentowanych danych.

4.2.1. Wady architektury

System ten jest mocno zdecentralizowany. Na barkach kontrolerów spoczywa odpowiedzialność odpowiedniej propagacji sygnałów informujących o nowej wersji danych, inwalidacji danych, jak również zapytań o dokonanie nowych obliczeń. Prowadzi to do:

- zaciemnienia kodu źródłowego zbędnymi instrukcjami warunkowymi,
- zignorowania pewnych sygnałów,
- podjęcia niewłaściwej decyzji.

Zarządzanie zadaniami również jest wadliwe. W razie gdy użytkownik poprzez interakcję z systemem zleci wykonanie kilku zadań na raz, które dokonują obliczeń na tych samych danych, system może zachować się w sposób nieoczekiwany. Prowadzi to do zakończenia aplikacji z powodu naruszenia pamięci. W najgorszym wypadku powinien zakolejkować te zadania i wykonać jedno po drugim.

Wiele komponentów interfejsu użytkownika przechowuje własne uchwyty do danych oraz ewentualnie muteks. Wobec tego same dokonują synchronizacji lub nie robią tego wcale. Nieprzemyślany model doprowadził do wielu patologi. Przykład stanowi używanie współdzielonych wskaźników do przekazywania danych, które z założenia już powinny być współdzielone.

[1] Model/View Programming | Qt Widgets 5.7 `http://doc.qt.io/qt-5/model-view-programming.html`

Konkluzja

Aktualny model współdzielonych danych, w powiązaniu z modelem zarządzania nimi nie gwarantuje bezpiecznego dostępu do danych ani prawidłowego przebiegu procesu wykonania zadań. Biorąc pod uwagę wyżej wymienione problemy nowy mechanizm zarządzania danymi oraz procesem wykonania powinien:

- posiadać wewnętrzny mechanizm synchronizacji dostępu do danych,
- gwarantować bezpieczny dostęp do współdzielonych danych,
- gwarantować bezpieczne wykonanie zadań w tle,
- gwarantować prawidłową kolejność procesu przetwarzania danych,
- posiadać scentralizowany mechanizm propagacji sygnałów,
- prawidłowo propagować informację o dostępności nowej wersji danych,
- prawidłowo propagować informację o żądaniu obliczeń nowych danych.