

Politechnika Śląska
Wydział Matematyki Stosowanej
Kierunek Informatyka
Studia stacjonarne I stopnia

Projekt inżynierski

FilmUpper

Kierujący projektem:
dr inż. Adam Zielonka

Autorzy:
Kamil Rutkowski
Jakub Rup

Gliwice 2017

*To jest
dedykacja*

Projekt inżynierski:

FilmUpper

kierujący projektem: dr inż. Adam Zielonka

1. **Kamil Rutkowski** – (50%)

Struktura aplikacji, Algorytmika

2. **Jakub Rup** – (50%)

Kodowanie i dekodowanie plików, Interfejs użytkownika, Algorytmika

Podpisy autorów projektu

1.
2.

Podpis kierującego projektem

.....

Oświadczenie kierującego projektem inżynierskim

Potwierdzam, że niniejszy projekt został przygotowany pod moim kierunkiem i kwalifikuje się do przedstawienia go w postępowaniu o nadanie tytułu zawodowego: inżynier.

Data

Podpis kierującego projektem

Oświadczenie autorów

Świadomy/a odpowiedzialności karnej oświadczam, że przedkładany projekt inżynierski na temat:

FilmUpper

został napisany przez autorów samodzielnie.

Jednocześnie oświadczam, że ww. projekt:

- nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2000 r. Nr 80, poz. 904, z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym, a także nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,
- nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów wyższej uczelni lub tytułów zawodowych.
- nie zawiera fragmentów dokumentów kopiowanych z innych źródeł bez wyraźnego zaznaczenia i podania źródła.

Podpisy autorów projektu

1. Kamil Rutkowski,
2. Jakub Rup,

nr albumu: 112233, (podpis:)

nr albumu: 112233, (podpis:)

Gliwice, dnia

Spis treści

Wstęp	7
1. Technologia	9
1.1. Język programowania	9
1.1.1. C++	9
1.1.2. C#	10
1.1.3. Rust	10
1.1.4. Wybór języka programowania	10
2. Dekodowanie oraz enkodowanie filmów	11
3. Interfejs użytkownika	13
4. Struktura programu	15
4.1. Komunikacja z użytkownikiem	15
4.2. Organizacja algorytmów i sposobu przekształcania plików wideo	16
4.2.1. IFrameReader - odczytywanie informacji z pliku wideo	17
4.2.2. FrameEnhancerBase i IFrameEnhancerHeader - ulepszanie pojedynczej klatki	18
5. Algorytmy	21
Rysunki	23
Programy	25
Literatura	27

Wstęp

FilmUpper jest aplikacją pozwalającą na poprawianie jakości obrazu w filmie, poprzez zwiększanie rozdzielczości oraz zwiększanie ilości klatek na sekundę w nim występujących. Dzięki takim zabiegom jakość oglądanego przez nas obrazu znacząco się poprawia, jednakże nigdy nie będzie ona tak dobra, jak jakość obrazu nagrywanego z ustawieniami na które chcemy dany film skonwertować.

1. Technologia

Przetwarzanie plików filmowych nawet w przypadku zastosowania najłatwiejszych algorytmów sposób jest zadaniem bardzo wymagającym pod względem wydajnościowym, dlatego wybór technologii ma kluczowe znaczenie, gdyż wpływa na wydajność całej aplikacji. Należy zwrócić uwagę na wiele czynników które mogą wpłynąć na działanie programu jak również na sposób jego implementacji.

1.1. Język programowania

Wybór właściwego języka programowania jest dla nas zadaniem kluczowym, ze względu na to, że użycie odpowiedniego języka znacząco wpływa na prędkość działania programu oraz na dostępność narzędzi i bibliotek wspomagających pracę nad przetwarzanymi danymi. Znajomość danego języka także była dla nas jednym z kluczowych czynników przy jego wyborze. Naszym rozważaniom poddaliśmy następujące języki programowania.

1.1.1. C++

Język C++ jest jednym z najczęściej używanych języków niskopoziomowych. Jego popularność jest skutkiem bardzo długiego czasu na rynku oraz pewnej prostoty użycia. Kolejne wersje tego wciąż rozwijającego się języka dodają nowe, sprawdzone i ułatwiające tworzenie programów rozwiązania z innych języków programowania. Bardzo duża wydajność oraz mnogość dostępnych bibliotek związanych z dekodowaniem i enkodowaniem plików filmowych jest bardzo ważnym aspektem tego wyboru. Wybór ten wiąże się także z kilkoma negatywnymi cechami tego języka, wymóg ręcznego zarządzania pamięcią, mało przejrzysta składnia przy tworzeniu rozwiązań o dużym stopniu skomplikowania, mało czytelne komunikaty odnośnie błędów podczas kompilacji i działania programu oraz brak ułatwień które poznaliśmy w językach wysokopoziomowych są jednymi z nich.

1.1.2. C#

Wysokopoziomowe rozwinięcie języka z rodziny C. Mimo posiadania składni podobnej do C++, język ten porzuca wiele z nieprzyjemnych jego aspektów. Poprzez automatyczne zarządzanie pamięcią, usunięcie składni charakterystycznej dla wskaźników oraz dodanie wielu nowych mechanizmów, wygląd kodu oraz prędkość tworzenia programów znacząco wzrasta. Bardzo ważnym składnikiem C# jest także LINQ które umożliwia bardzo kompaktowe i przejrzyste działanie na kolekcjach co jest dużym plusem przy przetwarzaniu plików wideo, jako że są one kolekcjami pojedynczych klatek złożonych z kolekcji pikseli. Wszystkie z tych udogodnień mają jednak cenę w postaci mniejszej wydajności w stosunku do C++ oraz brak wystarczającego wsparcia dla zarządzania plikami wideo jako że język ten jest stworzony z myślą o szybkim tworzeniu aplikacji biurowych.

1.1.3. Rust

Prędkość działania porównywalna z językiem C++, duże bezpieczeństwo pod względem zarządzania pamięcią, łatwość w konwersji programu jednowątkowego na wielowątkowy, składnia języka oraz wiele udogodnień zaciągniętych z języków wysokiego poziomu. To jedne z wielu punktów które zachęcały do wyboru tego języka. Niestety, z uwagi na to, że jest to język stosunkowo młody zauważalny jest brak lub wczesna wersja bibliotek umożliwiających przyjemną pracę na plikach filmowych. Jest to także język z którym nie mamy dużego doświadczenia.

1.1.4. Wybór języka programowania

Rozważając cechy każdego z języków, nasze umiejętności w posługiwaniu się nimi oraz specyfikę programu który chcemy napisać wybraliśmy język C++. Kluczowymi cechami które przekonały nas do wyboru tego języka były: mnogość bibliotek związanych z tematem projektu lub uniwersalnie wspomagających często napotykanne problemy, prędkość działania oraz duża znajomość samego języka.

2. Dekodowanie oraz enkodowanie filmów

3. Interfejs użytkownika

4. Struktura programu

Struktura w jakiej organizowalibyśmy kod programu jest bardzo istotnym zagadnieniem z punktu widzenia przejrzystości systemu oraz ergonomii pracy z nim. Chcieliśmy zachować także możliwość łatwej rozbudowy funkcjonalności polegającej na dodawaniu nowych algorytmów które mogłyby z łatwością przetwarzać obraz wideo. Struktura naszego programu dzieli się na 2 główne części: część odpowiadającą za komunikację z użytkownikiem oraz wybór odpowiednich algorytmów, oraz część polegającą na przetwarzaniu pliku wideo przez wybrane przez użytkownika algorytmy. Tworzyliśmy architekturę naszej aplikacji zgodnie z paradygmatem odwrócenia sterowania (IoC), polegającym na przekazywaniu gotowych obiektów do klas ich potrzebujących. Znacząco ułatwiło to projektowanie całej aplikacji a zwłaszcza jej testowanie.

4.1. Komunikacja z użytkownikiem

Do komunikacji z użytkownikiem stosujemy klasy FilmUpperView (dalej View) i FilmUpperController (dalej controller). Jest to uproszczona wersja wzorca projektowego MVC(Model, View, Controller) Początkowo chcieliśmy stworzyć infrastrukturę opartą na bardzo podobnym w założeniach wzorcu MVVM (Model, View Model, Controller) oraz zastosować rozwiązania reaktywne w kontakcie z użytkownikiem. Mimo, że głównym celem systemów reaktywnych jest asynchroniczne zarządzanie zdarzeniami poprzez stosowanie strumieni zamiast tradycyjnego podejścia do danych, to reaktywne podejście do zdarzeń bardzo upraszcza ich obsługę oraz jest bardzo przejrzyste pod względem przepływu sterowania. Niestety z uwagi na to, że rozwiązania te stosowaliśmy tylko w aplikacjach opartych na języku C#, nie byliśmy zaznajomieni z tą technologią w językach niższego poziomu. Nasza aplikacja nie jest także na tyle rozbudowana pod względem złożoności możliwych stanów ani jej elementy nie reagują w sposób znaczny na zmiany które użytkownik wykonuje w interfejsie, więc zauważyliśmy, że lepszym rozwiązaniem będzie zastosowanie prostszego modelu. Komunikacja między użytkownikiem a programem odbywa się pomiędzy klasami View a Controller. W klasie View zarządzamy interfejsem użyt-

kownika, wyświetlającym możliwe do wyboru algorytmy i przekazujemy jego decyzje do Controllera. W klasie Controller następuje ustalenie jakie algorytmy są dostępne do wykorzystania przez użytkownika, obsługiwany jest proces ulepszania jakości obrazu oraz zapisywania wyniku tej operacji. Obecnie algorytmy które są dostępne do wykorzystania są wpisane na stałe w kodzie programu, przez co aby dodać kolejny algorytm należy go dopisać do odpowiedniego wektora algorytmów. Jedną z możliwych i podstawowych opcji, które rozważaliśmy do rozbudowy funkcjonalności programu jest możliwość automatycznego dodawania dostępnych algorytmów poprzez walidację zawartości odpowiednich folderów. Umożliwiało by to rozbudowę funkcjonalności programu przez następnych użytkowników, a mianowicie poprzez tworzenie bibliotek dynamicznych o odpowiedniej budowie (opartej o odpowiednie klasy bazowe) i ich automatyczne wczytywanie przy starcie programu. Dało by to bardzo dużą możliwość rozbudowy bazowej funkcjonalności, jednakże w obecnej fazie, funkcjonalność ta nie jest uważana przez nas za podstawową.

4.2. Organizacja algorytmów i sposobu przekształcania plików wideo

W naszym projekcie z założenia stosujemy dwa typy algorytmów. Są to algorytmy:

- wpływające na jakość pojedynczej klatki wideo,
- wpływające na ilość klatek wideo na sekundę.

Algorytmy te korzystają z klas które przechowują ważne informacje odnośnie przetwarzanego wideo. Klasami tymi są klasy VideoFrame oraz FilmQualityInfo.

Klasa VideoFrame przechowuje informację na temat pojedynczej klatki wideo.

Przy ich użyciu oraz przy użyciu klas bazowych IFrameReader i IFrameWriter (i stworzonych na ich podstawie implementacjach) przetwarzamy plik wideo w następujący sposób:

1. klasa IFrameReader czyta kolejne klatki z pliku wejściowego
2. FrameEnhancerBase przetwarza przeczytane klatki, zwiększając ich rozdzielczość zgodnie z zastosowanym algorytmem

3. FpsEnhancerBase przetwarza ulepszone przez FrameEnhancerBase klatki, tworząc nowe klatki zgodnie z wybranym algorytmem
4. IFrameWriter zapisuje otrzymane przez FpsEnhancerBase klatki

4.2.1. IFrameReader - odczytywanie informacji z pliku wideo

Ta klasa bazowa ma za zadanie odczytywać informację o klatkach wideo oraz dźwięku. Utworzyliśmy ją jako klasę bazową ze względu na dwa aspekty:

1. możliwość stworzenia klasy testowej o ustalonych właściwościach - daje to bardzo dużą swobodę w prowadzeniu testów, bez względu na stan pracy nad właściwą implementacją
2. umożliwia stosunkowo bezbolesną zmianę biblioteki dekodującej pliki wideo bez zmiany sposobu korzystania z niej, daje nam to też swobodę na przyszłość, jeśli chcielibyśmy porównać działanie 2 różnych bibliotek umożliwiających odczyt plików wideo.

Klasa ta dostarcza następującą funkcjonalność:

```
class IFrameReader
{
public:
    virtual VideoFrame* ReadNextFrame() {};
    virtual FilmQualityInfo* GetVideoFormatInfo() {};
    virtual bool AreFramesLeft() { return true; };
};
```

Klasy dziedziczące po niej implementują następującą funkcjonalność:

```
* virtual VideoFrame* ReadNextFrame() {};
```

- metoda ta pozwala na odczytanie następnej klatki razem z odpowiadającymi jej próbkami dźwiękowymi,

```
* FilmQualityInfo* GetVideoFormatInfo() {};
```

- metoda ta przekazuje informację o formacie danych przechowywanych w pliku wideo,

```
* bool AreFramesLeft() { return true; };
```

- metoda ta informuje, czy pozostały jeszcze klatki do odczytu.

4.2.2. FrameEnhancerBase i IFrameEnhancerHeader - ulepszanie pojedynczej klatki

Zadaniem klasy bazowej FrameEnhancerBase jest przetwarzanie otrzymanej klatki w taki sposób, aby przy użyciu wybranego algorytmu zwiększyć rozdzielczość klatki wideo. Klasa bazowa IFrameEnhancerHeader jest odpowiedzialna za podstawowe informacje na temat danego FrameEnhancerBase oraz za utworzenie obiektu tej klasy. Daje nam to możliwość posiadania wszystkich wymaganych do wyświetlenia informacji bez tworzenia samych obiektów (tworzymy je dopiero w momencie w którym są potrzebne).

Możliwości FrameEnhancerBase to:

```
class FrameEnhancerBase
{
protected:
    IFrameReader* _inputFrameStream;
    FilmQualityInfo* _targetQualityInfo;
    FilmQualityInfo* _sourceQualityInfo;

public:
    FrameEnhancerBase(IFrameReader* inputFrameReader, FilmQualityInfo* targetQualityInfo) {
        _inputFrameStream = inputFrameReader;
        _targetQualityInfo = targetQualityInfo;
        _sourceQualityInfo = _inputFrameStream->GetVideoFormatInfo();
    }

    virtual VideoFrame* ReadNextEnhancedFrame() { return nullptr; };
    FilmQualityInfo* GetSourceQuality() { return _sourceQualityInfo; };
    virtual bool AreFramesLeft() { return _inputFrameStream->AreFramesLeft(); };
};
```

Klasa ta dostarcza następującą funkcjonalność:

```
* IFrameReader* _inputFrameStream;
```

- `_inputFrameStream` zawiera obiekt implementacji `IFrameReader` dzięki któremu klasa ta może czytać klatki do przetworzenia,

```
* FilmQualityInfo* _targetQualityInfo;
```

- `_targetQualityInfo` jest obiektem przechowującym informację na temat docelowego formatu wideo. Dzięki temu obiektowi jesteśmy w stanie ustalić docelowy rozmiar klatki wideo,

```
* FilmQualityInfo* _sourceQualityInfo;
```

- `_sourceQualityInfo` jest obiektem przechowującym informację na temat źródłowego formatu wideo. Mimo tego, że mamy ciągły dostęp do tej informacji przy pomocy `_inputFrameStream`, to przechowywanie tej informacji w tej postaci jest o wiele wygodniejsze,

```
* FrameEnhancerBase(IFrameReader* inputFrameReader,  
FilmQualityInfo* targetQualityInfo) {  
    _inputFrameStream = inputFrameReader;  
    _targetQualityInfo = targetQualityInfo;  
    _sourceQualityInfo = _inputFrameStream->GetVideoFormatInfo();  
}
```

- konstruktor klasy bazowej przypisuje wartości zmiennych oraz uzyskuje dane na temat formatu pliku źródłowego,

```
* virtual VideoFrame* ReadNextEnhancedFrame() { return nullptr; };
```

- zwraca przetworzoną klatkę wideo wraz z dźwiękiem,

```
* FilmQualityInfo* GetSourceQuality() { return _sourceQualityInfo; };
```

- zwraca informacje na temat jakości pliku źródłowego,

```
* virtual bool AreFramesLeft() { return _inputFrameStream->AreFramesLeft(); };
```

- zwraca informacje na temat tego, czy pozostały jeszcze jakieś klatki do odczytu.

Możliwości klasy `IFrameEnhancerHeader` są następujące:

```
class IFrameEnhancerHeader
{
public:
IFrameEnhancerHeader(std::string name, std::string description)
{
Name = name;
Description = description;
}
std::string Name;
std::string Description;
virtual FrameEnhancerBase* Enhancer(IFrameReader* inputFrameReader,
    FilmQualityInfo* targetQualityInfo) {
    return new FrameEnhancerBase(inputFrameReader, targetQualityInfo); };
};
```

Dostarcza ona następującą funkcjonalność:

* std::string Name;

- jest nazwą algorytmu przetwarzającego plik wideo,

* std::string Description;

- jest opisem algorytmu dla użytkownika,

```
* virtual FrameEnhancerBase* Enhancer(IFrameReader* inputFrameReader,
    FilmQualityInfo* targetQualityInfo) {
    return new FrameEnhancerBase(inputFrameReader, targetQualityInfo); };
};
```

- zwraca nowo utworzony obiekt klasy ulepszającej jakość klatki wideo.

5. Algorytmy

Rysunki

Tu rysunki

Programy

Tu programy

```
#include <stdio.h>

int main()
{
    printf("Hello world\n");
}
```

Oraz

```
<?php
    echo "test=$test";
?>
```

Twierdzenie 1. *Twierdzenie Twierdzenie Twierdzenie Twierdzenie Twierdzenie*

Literatura

[1] Jakaś pozycja literatury

[2] Jakaś pozycja literatury