|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **ИУК «Информатика и управление»** |
| **КАФЕДРА** | **ИУК4 «Программное обеспечение ЭВМ,** |
| **информационные технологии»** | |

**Лабораторная работа №1**

**«Системы мультимедиа, прототипирование и разработка интерфейса веб-проекта»**

**ДИСЦИПЛИНА: «Проектирование программного обеспечения»**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил: студент гр. ИУК4-11М | |  |  | ( | Сафронов Н.С. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |
| Проверил: | |  |  | ( | Белов Ю.С. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |

|  |  |
| --- | --- |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | |
|  | - Балльная оценка:  - Оценка: |

Калуга, 2024

**Цель работы**: формирование и закрепление практических навыков по обработки изображений и работы с различными цветовыми системами.

**Постановка задачи**

**Вариант 7**

Создать форму с растровым изображением с возможностью регулировки яркости по всем каналам и по C, M, Y в отдельности.

**Листинг программы**

**main.py**

import sys

from PyQt6.QtWidgets import QApplication

from ui import MainWindow

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

window = MainWindow()

window.show()

app.exec()

**ui.py**

import numpy as np

from PyQt6.QtCore import Qt

from PyQt6.QtGui import QPixmap, QImage

from PyQt6.QtWidgets import (

QMainWindow, QVBoxLayout, QWidget, QLayout,

QGroupBox, QSlider, QHBoxLayout, QLabel, QPushButton, QFileDialog,

)

from models.images import BrightnessAdjustment

from services.images import ImageService

class MainWindow(QMainWindow):

"""."""

def \_\_init\_\_(self):

super(MainWindow, self).\_\_init\_\_()

self.setWindowTitle('Brightness Adjustment Tool')

widget = QWidget()

layout = self.\_construct\_layout()

widget.setLayout(layout)

self.setCentralWidget(widget)

self.\_images: ImageService | None = None

def \_construct\_image\_layout(self):

self.\_image\_layout = QVBoxLayout()

self.\_image\_view = QLabel()

self.\_image\_view.setFixedSize(600, 600)

self.\_image\_view.setAlignment(Qt.AlignmentFlag.AlignCenter)

self.\_image\_layout.addWidget(self.\_image\_view)

self.adjustSize()

return self.\_image\_layout

def \_construct\_control\_layout(self):

brightness\_control\_layout = QVBoxLayout()

cyan\_label = QLabel('Cyan')

brightness\_control\_layout.addWidget(cyan\_label)

self.\_cyan\_slider = QSlider(Qt.Orientation.Horizontal)

self.\_cyan\_slider.setValue(50)

self.\_cyan\_slider.valueChanged.connect(self.\_adjust\_channel)

brightness\_control\_layout.addWidget(self.\_cyan\_slider)

magenta\_label = QLabel('Magenta')

brightness\_control\_layout.addWidget(magenta\_label)

self.\_magenta\_slider = QSlider(Qt.Orientation.Horizontal)

self.\_magenta\_slider.setValue(50)

self.\_magenta\_slider.valueChanged.connect(self.\_adjust\_channel)

brightness\_control\_layout.addWidget(self.\_magenta\_slider)

yellow\_label = QLabel('Yellow')

brightness\_control\_layout.addWidget(yellow\_label)

self.\_yellow\_slider = QSlider(Qt.Orientation.Horizontal)

self.\_yellow\_slider.setValue(50)

self.\_yellow\_slider.valueChanged.connect(self.\_adjust\_channel)

brightness\_control\_layout.addWidget(self.\_yellow\_slider)

overall\_label = QLabel('All Channels')

brightness\_control\_layout.addWidget(overall\_label)

self.\_overall\_slider = QSlider(Qt.Orientation.Horizontal)

self.\_overall\_slider.setValue(50)

self.\_overall\_slider.valueChanged.connect(self.\_adjust\_overall)

brightness\_control\_layout.addWidget(self.\_overall\_slider)

brightness\_control\_group = QGroupBox('Brightness Control')

brightness\_control\_group.setLayout(brightness\_control\_layout)

image\_control\_group = QGroupBox('Image Loading')

image\_control\_layout = QHBoxLayout()

button = QPushButton('Load Image')

button.clicked.connect(self.\_load\_image)

image\_control\_layout.addWidget(button)

image\_control\_group.setLayout(image\_control\_layout)

layout = QVBoxLayout()

layout.addWidget(brightness\_control\_group)

layout.addWidget(image\_control\_group)

layout.addStretch(1)

return layout

def \_construct\_layout(self) -> QLayout:

layout = QHBoxLayout()

layout.addLayout(self.\_construct\_image\_layout())

layout.addLayout(self.\_construct\_control\_layout())

return layout

def \_load\_image(self):

image = QFileDialog.getOpenFileName(

None, 'OpenFile', '', "Image file(\*.jpg)"

)

image\_path = image[0]

pixmap = QPixmap(image\_path)

self.\_initial\_image = pixmap

self.\_images = ImageService.from\_pixmap(self.\_initial\_image)

self.\_set\_image(self.\_images.to\_pixmap())

def \_set\_image(self, pixmap: QPixmap):

ratio = pixmap.width() / pixmap.height()

if pixmap.width() < pixmap.height():

scaled = pixmap.scaled(

int(600 \* ratio),

600,

Qt.AspectRatioMode.KeepAspectRatio,

)

else:

scaled = pixmap.scaled(

600,

int(600 / ratio),

Qt.AspectRatioMode.KeepAspectRatio,

)

self.\_image\_view.setPixmap(scaled)

self.\_image\_layout.update()

self.adjustSize()

def \_get\_brightness\_adjustment(self) -> BrightnessAdjustment:

return BrightnessAdjustment(

cyan=self.\_cyan\_slider.value() - 50,

yellow=self.\_yellow\_slider.value() - 50,

magenta=self.\_magenta\_slider.value() - 50,

)

def \_adjust\_channel(self, \_: int):

self.\_images.adjust\_brightness(self.\_get\_brightness\_adjustment())

self.\_set\_image(self.\_images.adjusted\_to\_pixmap())

def \_adjust\_overall(self, value: int):

adjustment = BrightnessAdjustment(

cyan=value - 50,

yellow=value - 50,

magenta=value - 50,

)

self.\_images.adjust\_brightness(adjustment)

self.\_set\_image(self.\_images.adjusted\_to\_pixmap())

**services/images.py**

import typing as t

import numpy as np

from PyQt6.QtGui import QPixmap, QImage

from models.images import CmyChannels, BrightnessAdjustment

class ImageService:

def \_\_init\_\_(self, rgb\_array: np.array):

self.\_image: np.ndarray = rgb\_array

self.\_adjusted: np.ndarray | None = None

self.\_channels = None

@classmethod

def from\_pixmap(cls, pixmap: QPixmap) -> t.Self:

image = pixmap.toImage()

image = image.convertToFormat(QImage.Format.Format\_RGB888)

width = image.width()

height = image.height()

ptr = image.bits().asstring(width \* height \* 3)

image = np.frombuffer(ptr, np.uint8).reshape((height, width, 3))

return cls(image)

def adjusted\_to\_pixmap(self) -> QPixmap:

return self.\_to\_pixmap(self.\_adjusted)

def to\_pixmap(self) -> QPixmap:

return self.\_to\_pixmap(self.\_image)

def adjust\_brightness(self, adjustment: BrightnessAdjustment) -> np.array:

channels = self.\_channels or self.\_parse\_image\_channels(self.\_image)

for channel, percent in adjustment.items():

channels[channel] = self.\_adjust\_channel\_brightness(

channel=channels[channel],

percent=percent,

)

self.\_adjusted = self.\_convert\_cmy\_to\_rgb(channels)

return self.\_adjusted

@classmethod

def \_to\_pixmap(cls, image: np.array) -> QPixmap:

channels = cls.\_parse\_image\_channels(image)

image = cls.\_convert\_cmy\_to\_rgb(channels)

height, width, channels = image.shape

bytes\_per\_line = channels \* width

return QPixmap.fromImage(

QImage(

image.data, width, height,

bytes\_per\_line, QImage.Format.Format\_RGB888,

)

)

@classmethod

def \_parse\_image\_channels(cls, rgb\_array: np.array) -> CmyChannels:

cyan = 255 - rgb\_array[:, :, 0].astype(np.float64)

cyan /= cyan.max() or 1

magenta = 255 - rgb\_array[:, :, 1].astype(np.float64)

magenta /= magenta.max() or 1

yellow = 255 - rgb\_array[:, :, 2].astype(np.float64)

yellow /= yellow.max() or 1

return CmyChannels(

cyan=cyan,

magenta=magenta,

yellow=yellow,

)

@classmethod

def \_convert\_cmy\_to\_rgb(cls, channels: CmyChannels) -> np.array:

red = 255 - (channels['cyan'] \* 255).astype(np.uint8)

green = 255 - (channels['magenta'] \* 255).astype(np.uint8)

blue = 255 - (channels['yellow'] \* 255).astype(np.uint8)

return np.dstack([red, green, blue])

@classmethod

def \_adjust\_channel\_brightness(

cls,

channel: np.ndarray[np.float64],

percent: float,

) -> np.ndarray[np.uint8]:

channel -= percent / 100

channel[channel > 1] = 1

channel[channel < 0] = 0

return channel

**models/images.py**

import typing as t

import numpy as np

class CmyChannels(t.TypedDict):

cyan: np.ndarray[np.float64]

magenta: np.ndarray[np.float64]

yellow: np.ndarray[np.float64]

class BrightnessAdjustment(t.TypedDict):

cyan: float

magenta: float

yellow: float

**Результаты выполнения лабораторной работы**

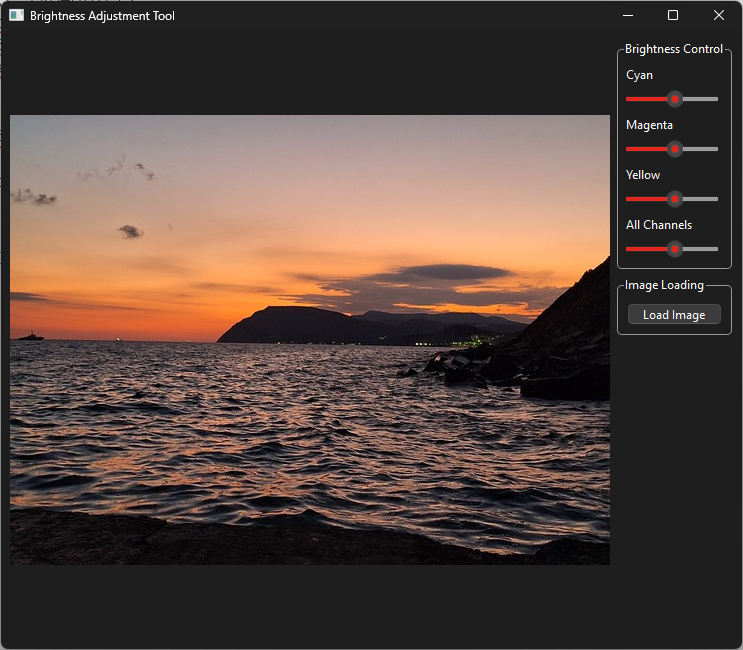


Рисунок 1 – Главное окно приложения

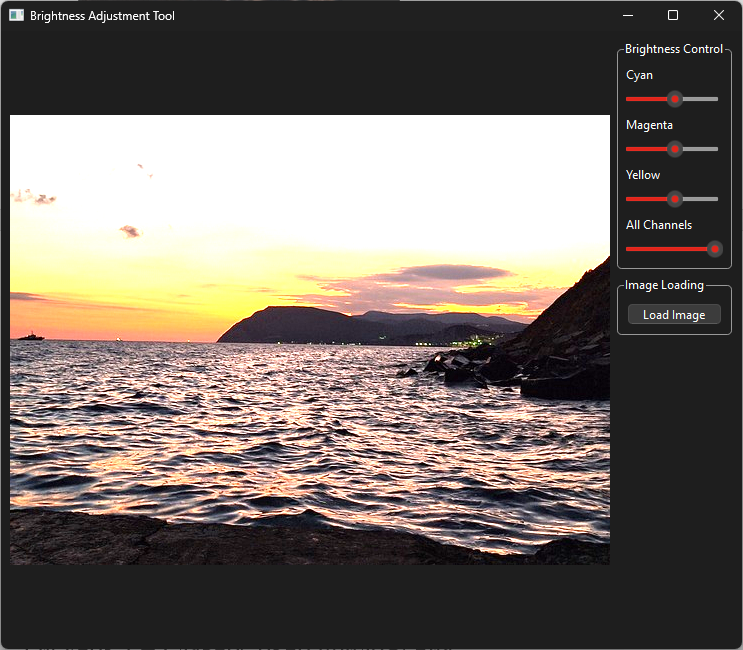


Рисунок 2 – Изображение с отредактированной яркостью

**Вывод**: в ходе выполнения лабораторной работы были сформированы и закреплены практические навыки по обработки изображений и работы с различными цветовыми системами.