IaMP_Lab2

2

Створено системою Doxygen 1.9.1

1 Звіт з лабораторних робіт №2 та №3

за дисципліною "Обробка зображень та мультимедіа"
студента групи ПА-17-2
Панасенка Єгора Сергійовича
Кафедра комп'ютерних технологій
ФПМ, ДНУ, 2017-2018 навч.р.
Тема: "Точкові методи обробки зображень. Методи фільтрації зображень."
Варіант 17

Звіт доступний за посиланням
https://gaurapanasenko.github.io/unilab_opt/IaMP_Lab2/html/index.html.
Вихідний код доступний за посиланням
https://github.com/gaurapanasenko/unilab/tree/master/08/IaMP_Lab2

1.0.1 Постановка задачі лабораторної роботи 2

1.0.1.1 Завдання I.

Вибрати одну із фотографій. Перетворити її на напівтонове зображення (програмно або за допомогою існуючих додатків). Згенерувати два зображення: затемнене та висвітлене.

Розробити програму, в якій будуть виконуватись такі дії із завантаженим напівтоновим зображенням:

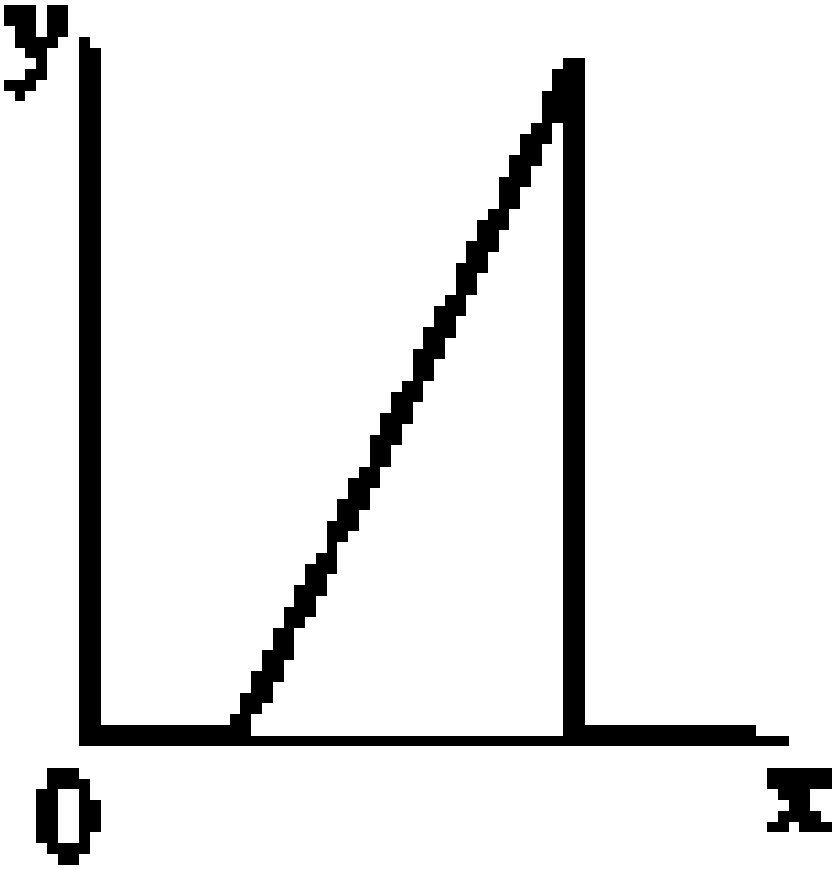
розрахунок та побудування гістограми; вирівнювання гістограми зображення, метод – за варіантами:

Варіант	Метод
1, 5, 9, 13, 17, 21, 25	Еквалізація гістограми

Показати, як працює розроблена програма на затемненому та на висвітленому зображеннях. Зробити висновки щодо реалізованого методу.

1.0.1.2 Завдання II.

Реалізувати програмно один з способів препарування напівтонового зображення (за варіантами):



Продемонструвати результати препарування напівтонового зображення для різних параметрів (на прикладі свого зображення).

1.0.2 Лабораторна робота 3

Розробити програму, в якій до зображення застосовується фільтр за варіантами:

Варіант	Метод фільтрації
4	Max-фільтр довільного розміру (з можливістю обрати розмір фільтру)

Показати, як працює розроблена програма на декількох фото. Зробити висновки щодо реалізованого методу.

1.0.3 Лабораторна робота 1

1.0.3.1 Завдання I.

Було розроблену програму на мові C++ та графіному інтерфейсі ImGui. Ця програма відкриває стандартні формати зображень перетворює їх у напівтонові зображення и еквалізує зображення. Також програма будує гістограми до зображень і здатна препарувати зображення. Продемонстуємо роботу програми.

2 Алфавітний покажчик класів

2.1 Класи

Класи, структури, об'єднання та інтерфейси з коротким описом.

3 Покажчик файлв

3.1 Файли

Повний список файлів.

4 Класи

4.1 Клас Image

Stores pixels, width and height.

#include <image.h>

Загальнодоступні елементи

- Image (shared_ptr< const pixel_t[]> data, const int width, const int height)

  Image construcor. Stores reference to pixels, height and width of an image.
- shared_ptr< const int[256]> calcHistogram () const

  Calculates histogram of an image. Calculates histogram by red channel so Image must be grayscaled before.
- shared_ptr< const Image > toGray () const

  Convert image to grayscale. Still uses 24 bytes per pixel.
- shared_ptr< const Image > dissect (channel_t dissection[256]) const

  Dissect image. Dissecting image of every color channel except of alpha. For best result Image must be grayscaled before.
- shared_ptr< const Image > dilate (int params[2]) const

  Dilate image. Applies max filter. For this filter used mask with size (2∗params[0]+1)x(2∗params[1]+1)
- shared_ptr< const Image > erode (int params[2]) const

  Erode image. Applies max filter. For this filter used mask with size (2∗params[0]+1)x(2∗params[1]+1)

Загальнодоступні статичні елементи

- static Image fromFile (const char ∗path)

  Read image from file.

Загальнодоступні атрибути

- shared_ptr< const pixel_t[]> data

  Constant pixels array. Stores smart pointer to height ∗ width pixels of an image.
- const int width

  Width of an image.
- const int height

  Height of an image.

4.1.1 Детальний опис

Stores pixels, width and height.

Див. визначення в файлі image.h, рядок 32

4.1.2 Конструктор(и)

4.1.2.1 Image() Image::Image (

shared_ptr< const pixel_t[]> data,
const int width,
const int height)

Image construcor. Stores reference to pixels, height and width of an image.

Аргументи

data	Smart reference to array of pixels, size must be height ∗ width.
width	Width of an image.
height	Height of an image.

Див. визначення в файлі image.cpp, рядок 9
```
00011     : data(data), width(width), height(height)
00012 {
00013 }
```

4.1.3 Опис методів компонент

4.1.3.1 calcHistogram() std::shared_ptr< const int[256]> Image::calcHistogram ( ) const

Calculates histogram of an image. Calculates histogram by red channel so Image must be grayscaled before.

Повертає

Smart pointer to histogram.

Див. визначення в файлі image.cpp, рядок 24
```
00025 {
00026     std::shared_ptr<int[256]> histogram(new int[256]{0});
00027     const pixel_t *in_data = data.get();
00028
00029     int size = height * width;
00030     for (int i = 0; i < size; i++)
00031         histogram[in_data[i][0]]++;
00032
00033     return histogram;
00034 }
```

4.1.3.2 dilate() shared_ptr< const Image > Image::dilate (int params[2]) const

Dilate image. Applies max filter. For this filter used mask with size (2∗params[0]+1)x(2∗params[1]+1)

Аргументи

params	size of mask.

Повертає

Smart pointer to dilated image.

Див. визначення в файлі image.cpp, рядок 73
```
00074 {
```

```
00075     std::shared_ptr<pixel_t[]> out_data(new pixel_t[width * height]);
00076     const pixel_t *in_data = data.get();
00077     int i, j, k, l, kbegin, kend, lbegin, lend;
00078
00079     for (i = 0; i < height; i++) {
00080         for (j = 0; j < width; j++) {
00081             channel_t mx = in_data[i * width + j][0];
00082             kbegin = max(i - params[1], 0);
00083             kend = min(i + params[1] + 1, height);
00084             lbegin = max(j - params[0], 0);
00085             lend = min(j + params[0], width - 1);
00086             for (k = kbegin; k != kend; k++) {
00087                 for (l = lbegin; l != lend; l++) {
00088                     mx = max(mx, in_data[k * width + l][0]);
00089                 }
00090             }
00091             for (k = 0; k < 3; k++) {
00092                 out_data[i * width + j][k] = mx;
00093             }
00094             out_data[i * width + j][3] = 255;
00095         }
00096     }
00097     return std::make_shared<Image>(out_data, width, height);
00098 }
```

4.1.3.3 dissect() shared_ptr< const Image > Image::dissect (channel_t dissection[256]) const

Dissect image. Dissecting image of every color channel except of alpha. For best result Image must be grayscaled before.

Аргументи

dissection	New values for all possible channel values.

Повертає

Smart pointer to dissected image.

Див. визначення в файлі image.cpp, рядок 57

```
00058 {
00059     const int size = width * height;
00060     std::shared_ptr<pixel_t[]> out_data(new pixel_t[size]);
00061     const pixel_t *in_data = data.get();
00062
00063     for (int i = 0; i < size; i++) {
00064         for (int j = 0; j < 3; j++) {
00065             out_data[i][j] = dissection[in_data[i][j]];
00066         }
00067         out_data[i][3] = 255;
00068     }
00069
00070     return std::make_shared<Image>(out_data, width, height);
00071 }
```

4.1.3.4 erode() shared_ptr< const Image > Image::erode (int params[2]) const

Erode image. Applies max filter. For this filter used mask with size (2∗params[0]+1)x(2∗params[1]+1)

Аргументи

params	size of mask.

Повертає

Smart pointer to eroded image.

Див. визначення в файлі image.cpp, рядок 100

```
00101 {
00102     std::shared_ptr<pixel_t[]> out_data(new pixel_t[width * height]);
00103     const pixel_t *in_data = data.get();
00104     int i, j, k, l, kbegin, kend, lbegin, lend;
00105 
00106     for (i = 0; i < height; i++) {
00107         for (j = 0; j < width; j++) {
00108             channel_t mx = in_data[i * width + j][0];
00109             kbegin = max(i - params[1], 0);
00110             kend = min(i + params[1] + 1, height);
00111             lbegin = max(j - params[0], 0);
00112             lend = min(j + params[0], width - 1);
00113             for (k = kbegin; k != kend; k++) {
00114                 for (l = lbegin; l != lend; l++) {
00115                     mx = min(mx, in_data[k * width + l][0]);
00116                 }
00117             }
00118             for (k = 0; k < 3; k++) {
00119                 out_data[i * width + j][k] = mx;
00120             }
00121             out_data[i * width + j][3] = 255;
00122         }
00123     }
00124     return std::make_shared<Image>(out_data, width, height);
00125 }
```

4.1.3.5 fromFile() Image Image::fromFile (const char * path) [static]

Read image from file.

Аргументи

path	Path of image file.

Повертає

Image class, that also references to pixels.

Див. визначення в файлі image.cpp, рядок 15

```
00016 {
00017     int width = 0, height = 0;
00018     pixel_t *data = (pixel_t *)stbi_load(path, &width, &height, NULL, COMP);
00019     assert(data != NULL);
00020     std::shared_ptr<pixel_t[]> ptr(data, stbi_image_free);
00021     return {ptr, width, height};
00022 }
```

4.1.3.6 toGray() std::shared_ptr< const Image > Image::toGray ( ) const

Convert image to grayscale. Still uses 24 bytes per pixel.

Повертає

Smart pointer to grayscaled image.

Див. визначення в файлі image.cpp, рядок 36

```
00037 {
00038     int size = width * height;
00039     std::shared_ptr<pixel_t[]> out_data(new pixel_t[size]);
00040     const pixel_t *in_data = data.get(), *pixel;
00041     unsigned char avg;
00042 
00043     for (int i = 0; i < size; i++) {
00044         pixel = &in_data[i];
00045         avg = (*pixel[0] + *pixel[1] + *pixel[2]) / 3;
00046         //avg = fmax((int)pixel[0], (int)pixel[1]);
00047         //avg = fmax((int)avg, (int)pixel[2]);
00048         for (int k = 0; k < 3; k++) {
00049             out_data[i][k] = avg;
00050         }
00051         out_data[i][3] = 255;
00052     }
00053 
00054     return std::make_shared<Image>(out_data, width, height);
00055 }
```

4.1.4 Компонентні дані

4.1.4.1 data shared_ptr<const pixel_t[]> Image::data

Constant pixels array. Stores smart pointer to height ∗ width pixels of an image.

Див. визначення в файлі image.h, рядок 39

4.1.4.2 height const int Image::height

Height of an image.

Див. визначення в файлі image.h, рядок 47

4.1.4.3 width const int Image::width

Width of an image.

Див. визначення в файлі image.h, рядок 43

Документація цих класів була створена з файлів:

- image.h
- image.cpp

4.2 Клас ImageData

Image metadata class. Stores pointer to image, and histograms. Also stores maximum value of histogram.

#include <imagedata.h>

Загальнодоступні елементи

- ImageData (shared_ptr< const Image > image)
  ImageData constructor. Calulates and stores histogram based on image.
- shared_ptr< const Image > equalize () const
  Equalize image using histogram.

Загальнодоступні статичні елементи

- static shared_ptr< const float[256]> copyHistogram (shared_ptr< const int[256]> histogramI)
  Converts histogram from integer type to float.

Загальнодоступні атрибути

- shared_ptr< const Image > image
  Smart pointer to image.
- shared_ptr< const int[256]> histogramI
  Smart pointer to histogram with integer type.
- int maxHistogramI
  Maximum value of histogramI.
- shared_ptr< const float[256]> histogramF
  Smart pointer to histogram with float type.
- float maxHistogramF
  Maximum value of histogramF.

4.2.1 Детальний опис

Image metadata class. Stores pointer to image, and histograms. Also stores maximum value of histogram.

Див. визначення в файлі imagedata.h, рядок 10

4.2.2 Конструктор(и)

4.2.2.1 ImageData() ImageData::ImageData (shared_ptr< const Image > image)

ImageData constructor. Calulates and stores histogram based on image.

Аргументи

image	Smart pointer to image.

Див. визначення в файлі imagedata.cpp, рядок 4

```
00005     : image(image), histogramI(image->calcHistogram()),
00006       maxHistogramI(*std::max_element(histogramI.get(), histogramI.get() + 256)),
00007       histogramF(copyHistogram(histogramI)),
00008       maxHistogramF(maxHistogramI)
00009 {
00010 }
```

4.2.3 Опис методів компонент

4.2.3.1 copyHistogram() std::shared_ptr< const float[256]> ImageData::copyHistogram (shared_ptr< const int[256]> histogramI) [static]

Converts histogram from integer type to float.

Аргументи

histogramI	Integer type histogram.

Повертає

Float type histogram.

Див. визначення в файлі imagedata.cpp, рядок 13

```
00014 {
00015     std::shared_ptr<float[256]> histogram(new float[256]{0});
00016     std::copy(histogramI.get(), histogramI.get() + 256, histogram.get());
00017     return histogram;
00018 }
```

4.2.3.2 equalize() std::shared_ptr< const Image > ImageData::equalize ( ) const

Equalize image using histogram.

Повертає

New equalized image.

Див. визначення в файлі imagedata.cpp, рядок 20

```
00021 {
00022     int width = image->width, height = image->height;
00023     int size = width * height;
00024     std::shared_ptr<pixel_t[]> data(new pixel_t[width * height]);
00025     const pixel_t *in_data = image->data.get();
00026     const int *histogram = histogramI.get();
00027     int accum = 0;
00028     int s[256];
00029
```

```
00030     for (int i = 0; i < 256; i++) {
00031         accum += histogram[i];
00032         s[i] = 255 * accum / size;
00033     }
00034
00035     for (int i = 0; i < size; i++) {
00036         channel_t cur = s[in_data[i][0]];
00037         data[i][0] = cur;
00038         data[i][1] = cur;
00039         data[i][2] = cur;
00040         data[i][3] = 255;
00041     }
00042
00043     return std::make_shared<Image>(data, width, height);
00044 }
```

4.2.4 Компонентні дані

4.2.4.1 histogramF shared_ptr<const float[256]> ImageData::histogramF

Smart pointer to histogram with float type.

Див. визначення в файлі imagedata.h, рядок 27

4.2.4.2 histogramI shared_ptr<const int[256]> ImageData::histogramI

Smart pointer to histogram with integer type.

Див. визначення в файлі imagedata.h, рядок 19

4.2.4.3 image shared_ptr<const Image> ImageData::image

Smart pointer to image.

Див. визначення в файлі imagedata.h, рядок 15

4.2.4.4 maxHistogramF float ImageData::maxHistogramF

Maximum value of histogramF.

Див. визначення в файлі imagedata.h, рядок 31

4.2.4.5 maxHistogramI int ImageData::maxHistogramI

Maximum value of histogramI.

Див. визначення в файлі imagedata.h, рядок 23

Документація цих класів була створена з файлів:

- imagedata.h
- imagedata.cpp

4.3 Клас Processor

Processing image. Calculates histogram, applies dissection or max filter and stores result image to texture.

#include <processor.h>

Загальнодоступні елементи

- Processor (shared_ptr< const ImageData > input)
  Processor constuctor.
- void updateDissection ()
  Calculates new image and stores it to texture.
- bool process_image (const char *name)
  Implements interface to manipulate fields.

Приватні дані

- shared_ptr< const ImageData > orig
  Original image with calculated histogram.
- shared_ptr< const ImageData > data
  Currently shown image on display.
- const Texture texture
  Texture used to draw with OpenGL.
- int dissection_x [2]
  Range of channel value where apply dissection.
- float dissection_y [2]
  Range of proportional coefficients between dissection values. Coefficients must be in range [0, 1] to make sure that new channel values will not out of [0, 255] range.
- channel_t dissection [256]
  New channel values for dissection for image.
- float dissectionF [256]
  New channel values for dissection for image in float type.
- int dilate_params [2]
  Size of max filter mask.
- int erode_params [2]
  Size of min filter mask.
- bool dissected
  Flag to apply dissection to the image.
- bool dilate
  Flag to apply dilation to the image.
- bool erode
  Flag to apply erosion to the image.

4.3.1 Детальний опис

Processing image. Calculates histogram, applies dissection or max filter and stores result image to texture.

Also have implements interface to manipulate parameters in this class with imgui.

Див. визначення в файлі processor.h, рядок 15

4.3.2 Конструктор(и)

4.3.2.1 Processor() Processor::Processor (shared_ptr< const ImageData > input)

Processor constuctor.

Аргументи

input	Input image that will be processed by this class.

Див. визначення в файлі processor.cpp, рядок 7

```
00008     : orig(input),
00009       data(input),
00010       dissection_x{100, 200},
00011       dissection_y{0, 1},
00012       dissection{0},
00013       dilate_params{1, 1},
00014       erode_params{1, 1},
00015       dissected(false),
00016       dilate(false),
00017       erode(false)
00018 {
00019     updateDissection();
00020 }
```

4.3.3 Опис методів компонент

4.3.3.1 process_image() bool Processor::process_image (const char * name)

Implements interface to manipulate fields.

Аргументи

name	name of window that will be shown.

Повертає

boolean value that shows that it is needed to update image.

Див. визначення в файлі processor.cpp, рядок 47

```
00047                                   {
00048     bool opened = true;
00049     auto img = data->image;
00050     auto tex_id = (void*)(intptr_t)(texture.id);
00051     ImGui::PushID(name);
00052 
00053     ImVec2 size(img->width, img->height);
00054     ImGui::SetNextWindowSize(size, ImGuiCond_FirstUseEver);
00055     ImGui::PushStyleVar(ImGuiStyleVar_WindowPadding, ImVec2(0,0));
00056     ImGui::Begin(name, &opened, ImGuiWindowFlags_NoSavedSettings);
00057     ImGui::Image(tex_id, ImGui::GetContentRegionAvail());
00058     ImGui::End();
00059     ImGui::PopStyleVar();
00060 
00061     char info_name[128];
00062     snprintf(info_name, 128, "Info %s", name);
00063     ImGui::Begin(info_name, NULL, ImGuiWindowFlags_NoSavedSettings);
00064     ImGui::Text("pointer = %ld", (intptr_t)tex_id);
00065     ImGui::Text("size = %d x %d", img->width, img->height);
00066     ImGui::PlotHistogram(
00067             "##", data->histogramF.get(), 256, 0, "Histogram", 0.0f,
00068             data->maxHistogramF, ImVec2(0, 100.0f));
00069     bool changed = false;
00070     ImGui::Spacing();
00071     changed |= ImGui::Checkbox("Dissected", &dissected);
00072     if (dissected) {
00073         {
00074             auto val = dissection_x;
00075             changed |= ImGui::DragIntRange2("dissection x", val, val + 1, 1, 0,
00076                                             256, "Min: %d", "Max: %d");
00077         }
00078         {
00079             auto val = dissection_y;
00080             changed |= ImGui::DragFloatRange2("dissection y", val, val + 1, 0.01,
00081                                               0, 1, "Min: %.2f", "Max: %.2f");
00082         }
00083         ImGui::PlotLines("Lines", dissectionF, 256, 0, NULL, 0, 1.0f,
00084                          ImVec2(0, 80.0f));
00085     }
00086     ImGui::Spacing();
00087     changed |= ImGui::Checkbox("Dilate", &dilate);
00088     if (dilate) {
00089         changed |= ImGui::SliderInt2("dilate params", dilate_params, 0, 16);
00090     }
00091     changed |= ImGui::Checkbox("Erode", &erode);
00092     if (erode) {
00093         changed |= ImGui::SliderInt2("erode params", erode_params, 0, 16);
00094     }
00095     if (changed) updateDissection();
00096     ImGui::End();
00097 
00098     ImGui::PopID();
00099     return opened;
00100 }
```

4.3.3.2 updateDissection() void Processor::updateDissection ( )

Calculates new image and stores it to texture.

Див. визначення в файлі processor.cpp, рядок 22

```
00022                                   {
00023     data = orig;
00024     if (dissected) {
00025         float diffY = dissection_y[1] - dissection_y[0];
00026         int diffX = dissection_x[1] - dissection_x[0];
00027         float diff = diffY / diffX;
00028         float acc = dissection_y[0];
00029         memset(dissection, 0, 256 * sizeof(channel_t));
00030         memset(dissectionF, 0, 256 * sizeof(float));
00031         for (int i = dissection_x[0]; i < dissection_x[1]; i++) {
00032             acc += diff;
00033             dissection[i] = acc * 255;
00034             dissectionF[i] = acc;
00035         }
00036         data = make_shared<ImageData>(data->image->dissect(dissection));
00037     }
00038     if (dilate) {
00039         data = make_shared<ImageData>(data->image->dilate(dilate_params));
00040     }
```

```
00041     if (erode) {
00042         data = make_shared<ImageData>(data->image->erode(erode_params));
00043     }
00044     texture.update(*data->image);
00045 }
```

4.3.4 Компонентні дані

4.3.4.1 data shared_ptr<const ImageData> Processor::data [private]

Currently shown image on display.

Див. визначення в файлі processor.h, рядок 23

4.3.4.2 dilate bool Processor::dilate [private]

Flag to apply dilation to the image.

Див. визначення в файлі processor.h, рядок 61

4.3.4.3 dilate_params int Processor::dilate_params[2] [private]

Size of max filter mask.

Див. визначення в файлі processor.h, рядок 49

4.3.4.4 dissected bool Processor::dissected [private]

Flag to apply dissection to the image.

Див. визначення в файлі processor.h, рядок 57

4.3.4.5 dissection channel_t Processor::dissection[256] [private]

New channel values for dissection for image.

Див. визначення в файлі processor.h, рядок 41

4.3.4.6 dissection_x int Processor::dissection_x[2] [private]

Range of channel value where apply dissection.

Див. визначення в файлі processor.h, рядок 31

4.3.4.7 dissection_y float Processor::dissection_y[2] [private]

Range of proportional coefficients between dissection values. Coefficients must be in range [0, 1] to make sure that new channel values will not out of [0, 255] range.

Див. визначення в файлі processor.h, рядок 37

4.3.4.8 dissectionF float Processor::dissectionF[256] [private]

New channel values for dissection for image in float type.

Див. визначення в файлі processor.h, рядок 45

4.3.4.9 erode bool Processor::erode [private]

Flag to apply erosion to the image.

Див. визначення в файлі processor.h, рядок 65

4.3.4.10 erode_params int Processor::erode_params[2] [private]

Size of min filter mask.

Див. визначення в файлі processor.h, рядок 53

4.3.4.11 orig shared_ptr<const ImageData> Processor::orig [private]

Original image with calculated histogram.

Див. визначення в файлі processor.h, рядок 19

4.3.4.12 texture const Texture Processor::texture [private]

Texture used to draw with OpenGL.

Див. визначення в файлі processor.h, рядок 27

Документація цих класів була створена з файлів:

- processor.h
- processor.cpp

5 Файли

5.1 Файл image.cpp

```
#include <stb_image.h>
#include <algorithm>
#include "image.h"
```

Макровизначення

- #define STB_IMAGE_IMPLEMENTATION

5.1.1 Опис макровизначень

5.1.1.1 STB_IMAGE_IMPLEMENTATION #define STB_IMAGE_IMPLEMENTATION

Див. визначення в файлі image.cpp, рядок 1

5.2 image.cpp

```
00001 #define STB_IMAGE_IMPLEMENTATION
00002 #include <stb_image.h>
00003 #include <algorithm>
00004 #include "image.h"
00005
00006 using std::min;
00007 using std::max;
00008
00009 Image::Image(std::shared_ptr<const pixel_t[]> data,
00010              const int width, const int height)
00011     : data(data), width(width), height(height)
00012 {
00013 }
00014
00015 Image Image::fromFile(const char *path)
00016 {
00017     int width = 0, height = 0;
00018     pixel_t *data = (pixel_t *)stbi_load(path, &width, &height, NULL, COMP);
00019     assert(data != NULL);
00020     std::shared_ptr<pixel_t[]> ptr(data, stbi_image_free);
00021     return {ptr, width, height};
00022 }
00023
00024 std::shared_ptr<const int[256]> Image::calcHistogram() const
```

```
00025 {
00026     std::shared_ptr<int[256]> histogram(new int[256]{0});
00027     const pixel_t *in_data = data.get();
00028 
00029     int size = height * width;
00030     for (int i = 0; i < size; i++)
00031         histogram[in_data[i][0]]++;
00032 
00033     return histogram;
00034 }
00035 
00036 std::shared_ptr<const Image> Image::toGray() const
00037 {
00038     int size = width * height;
00039     std::shared_ptr<pixel_t[]> out_data(new pixel_t[size]);
00040     const pixel_t *in_data = data.get(), *pixel;
00041     unsigned char avg;
00042 
00043     for (int i = 0; i < size; i++) {
00044         pixel = &in_data[i];
00045         avg = (*pixel[0] + *pixel[1] + *pixel[2]) / 3;
00046         //avg = fmax((int)pixel[0], (int)pixel[1]);
00047         //avg = fmax((int)avg, (int)pixel[2]);
00048         for (int k = 0; k < 3; k++) {
00049             out_data[i][k] = avg;
00050         }
00051         out_data[i][3] = 255;
00052     }
00053 
00054     return std::make_shared<Image>(out_data, width, height);
00055 }
00056 
00057 shared_ptr<const Image> Image::dissect(channel_t dissection[]) const
00058 {
00059     const int size = width * height;
00060     std::shared_ptr<pixel_t[]> out_data(new pixel_t[size]);
00061     const pixel_t *in_data = data.get();
00062 
00063     for (int i = 0; i < size; i++) {
00064         for (int j = 0; j < 3; j++) {
00065             out_data[i][j] = dissection[in_data[i][j]];
00066         }
00067         out_data[i][3] = 255;
00068     }
00069 
00070     return std::make_shared<Image>(out_data, width, height);
00071 }
00072 
00073 shared_ptr<const Image> Image::dilate(int params[2]) const
00074 {
00075     std::shared_ptr<pixel_t[]> out_data(new pixel_t[width * height]);
00076     const pixel_t *in_data = data.get();
00077     int i, j, k, l, kbegin, kend, lbegin, lend;
00078 
00079     for (i = 0; i < height; i++) {
00080         for (j = 0; j < width; j++) {
00081             channel_t mx = in_data[i * width + j][0];
00082             kbegin = max(i - params[1], 0);
00083             kend = min(i + params[1] + 1, height);
00084             lbegin = max(j - params[0], 0);
00085             lend = min(j + params[0], width - 1);
00086             for (k = kbegin; k != kend; k++) {
00087                 for (l = lbegin; l != lend; l++) {
00088                     mx = max(mx, in_data[k * width + l][0]);
00089                 }
00090             }
00091             for (k = 0; k < 3; k++) {
00092                 out_data[i * width + j][k] = mx;
00093             }
00094             out_data[i * width + j][3] = 255;
00095         }
00096     }
00097     return std::make_shared<Image>(out_data, width, height);
00098 }
00099 
00100 shared_ptr<const Image> Image::erode(int params[]) const
00101 {
00102     std::shared_ptr<pixel_t[]> out_data(new pixel_t[width * height]);
00103     const pixel_t *in_data = data.get();
00104     int i, j, k, l, kbegin, kend, lbegin, lend;
00105 
00106     for (i = 0; i < height; i++) {
00107         for (j = 0; j < width; j++) {
00108             channel_t mx = in_data[i * width + j][0];
00109             kbegin = max(i - params[1], 0);
00110             kend = min(i + params[1] + 1, height);
00111             lbegin = max(j - params[0], 0);
```

```
00112             lend = min(j + params[0], width - 1);
00113             for (k = kbegin; k != kend; k++) {
00114                 for (l = lbegin; l != lend; l++) {
00115                     mx = min(mx, in_data[k * width + l][0]);
00116                 }
00117             }
00118             for (k = 0; k < 3; k++) {
00119                 out_data[i * width + j][k] = mx;
00120             }
00121             out_data[i * width + j][3] = 255;
00122         }
00123     }
00124     return std::make_shared<Image>(out_data, width, height);
00125 }
```

5.3 Файл image.h

#include <memory>

Класи

- class Image

  Stores pixels, width and height.

Макровизначення

- #define COMP 4

Визначення типів

- typedef unsigned char channel_t

  Channel type for image. Used unsigned char because our channel values are in range [0, 255] so has 8-bit per channel.
- typedef channel_t pixel_t[COMP]

  Pixel type. One pixel has COMP channels and channel defined by channel_t type. Used 8-bit chanel with values that in range [0, 255]. Pixel has 4 channels (red, green, blue and alpha) so takes 4 bytes.

5.3.1 Опис макровизначень

5.3.1.1 COMP #define COMP 4

Size of pixel in bypes.

Name means "components" and was taken from stb_image.h header file library.

Див. визначення в файлі image.h, рядок 10

5.3.2 Опис визначень типів

5.3.2.1 channel_t typedef unsigned char channel_t

Channel type for image. Used unsigned char because our channel values are in range [0, 255] so has 8-bit per channel.

Див. визначення в файлі image.h, рядок 19

5.3.2.2 pixel_t typedef channel_t pixel_t[COMP]

Pixel type. One pixel has COMP channels and channel defined by channel_t type. Used 8-bit chanel with values that in range [0, 255]. Pixel has 4 channels (red, green, blue and alpha) so takes 4 bytes.

Див. визначення в файлі image.h, рядок 27

5.4 image.h

```
00001 #ifndef IMAGE_H
00002 #define IMAGE_H
00003 #include <memory>
00004
00010 #define COMP 4
00011
00012 using std::shared_ptr;
00013
00019 typedef unsigned char channel_t;
00020
00027 typedef channel_t pixel_t[COMP];
00028
00032 class Image
00033 {
00034 public:
00039     shared_ptr<const pixel_t[]> data;
00043     const int width;
00047     const int height;
00048
00057     Image(shared_ptr<const pixel_t[]> data,
00058           const int width, const int height);
00064     static Image fromFile(const char *path);
00065
00071     shared_ptr<const int[256]> calcHistogram() const;
00072
00078     shared_ptr<const Image> toGray() const;
00079
00087     shared_ptr<const Image> dissect(channel_t dissection[256]) const;
00088
00096     shared_ptr<const Image> dilate(int params[2]) const;
00097
00105     shared_ptr<const Image> erode(int params[2]) const;
00106 };
00107
00108
00109 #endif // IMAGE_H
```

5.5 Файл imagedata.cpp

```
#include <algorithm>
#include "imagedata.h"
```

5.6 imagedata.cpp

```
00001 #include <algorithm>
00002 #include "imagedata.h"
00003
00004 ImageData::ImageData(std::shared_ptr<const Image> image)
00005     : image(image), histogramI(image->calcHistogram()),
00006       maxHistogramI(*std::max_element(histogramI.get(), histogramI.get() + 256)),
00007       histogramF(copyHistogram(histogramI)),
00008       maxHistogramF(maxHistogramI)
00009 {
00010 }
00011
00012 std::shared_ptr<const float[256]>
00013 ImageData::copyHistogram(std::shared_ptr<const int[256]> histogramI)
00014 {
00015     std::shared_ptr<float[256]> histogram(new float[256]{0});
00016     std::copy(histogramI.get(), histogramI.get() + 256, histogram.get());
00017     return histogram;
00018 }
00019
00020 std::shared_ptr<const Image> ImageData::equalize() const
00021 {
00022     int width = image->width, height = image->height;
00023     int size = width * height;
00024     std::shared_ptr<pixel_t[]> data(new pixel_t[width * height]);
00025     const pixel_t *in_data = image->data.get();
00026     const int *histogram = histogramI.get();
00027     int accum = 0;
00028     int s[256];
00029
00030     for (int i = 0; i < 256; i++) {
00031         accum += histogram[i];
00032         s[i] = 255 * accum / size;
00033     }
00034
00035     for (int i = 0; i < size; i++) {
00036         channel_t cur = s[in_data[i][0]];
00037         data[i][0] = cur;
00038         data[i][1] = cur;
00039         data[i][2] = cur;
00040         data[i][3] = 255;
00041     }
00042
00043     return std::make_shared<Image>(data, width, height);
00044 }
```

5.7 Файл imagedata.h

#include "image.h"

Класи

- class ImageData

  Image metadata class. Stores pointer to image, and histograms. Also stores maximum value of histogram.

5.8 imagedata.h

```
00001 #ifndef IMAGEDATA_H
00002 #define IMAGEDATA_H
00003 #include "image.h"
00004
00010 class ImageData {
00011 public:
00015     shared_ptr<const Image> image;
00019     shared_ptr<const int[256]> histogramI;
00023     int maxHistogramI;
00027     shared_ptr<const float[256]> histogramF;
00031     float maxHistogramF;
00032
00038     ImageData(shared_ptr<const Image> image);
00039
00045     static shared_ptr<const float[256]>
00046     copyHistogram(shared_ptr<const int[256]> histogramI);
00047
00052     shared_ptr<const Image> equalize() const;
00053 };
00054
00055 #endif // IMAGEDATA_H
```

5.9 Файл main.cpp

```
#include <string>
#include <memory>
#include <sstream>
#include <map>
#include "gui/app.h"
#include "processor.h"
```

Макровизначення

- #define PROJECT_NAME "IaMP_Lab2"

Функції

- int main (int, char **)

5.9.1 Опис макровизначень

5.9.1.1 PROJECT_NAME #define PROJECT_NAME "IaMP_Lab2"

Див. визначення в файлі main.cpp, рядок 9

5.9.2 Опис функцій

5.9.2.1 main() int main (int , char **)

Див. визначення в файлі main.cpp, рядок 13

```
00014 {
00015     App app(PROJECT_NAME);
00016     map<string, shared_ptr<Processor>> images;
00017 
00018     imgui_addons::ImGuiFileBrowser *file_dialog = app.getFile_dialog();
00019     // Main loop
00020     while (!app.should_closed())
00021     {
00022         app.begin_loop();
00023         auto mode = imgui_addons::ImGuiFileBrowser::DialogMode::OPEN;
00024 
00025         if (file_dialog->showFileDialog(
00026                 "Open File", mode, ImVec2(700, 310), ".png,.jpg,.bmp"))
00027         {
00028             string path(file_dialog->selected_path);
00029             auto img = Image::fromFile(path.c_str());
00030             auto pathImage = make_shared<Image>(img);
00031             auto gray = make_shared<ImageData>(pathImage->toGray());
00032             {
00033                 stringstream ss;
00034                 ss << "Original Image: " << file_dialog->selected_fn
00035                    << " " << file_dialog->selected_path;
00036                 auto data = make_shared<ImageData>(pathImage);
```

```
00037                 auto proc = make_shared<Processor>(data);
00038                 images.insert(make_pair(ss.str(), proc));
00039             }
00040             {
00041                 stringstream ss;
00042                 ss « "Gray Image: " « file_dialog->selected_fn
00043                    « " " « file_dialog->selected_path;
00044                 auto proc = make_shared<Processor>(gray);
00045                 images.insert(make_pair(ss.str(), proc));
00046             }
00047             {
00048                 stringstream ss;
00049                 ss « "Equalization: " « file_dialog->selected_fn
00050                    « " " « file_dialog->selected_path;
00051                 auto data = make_shared<ImageData>(gray->equalize());
00052                 auto proc = make_shared<Processor>(data);
00053                 images.insert(make_pair(ss.str(), proc));
00054             }
00055         }
00056 
00057         for (auto image = images.begin(); image != images.end();) {
00058             if (!image->second->process_image(image->first.c_str())) {
00059                 image = images.erase(image);
00060             } else {
00061                 ++image;
00062             }
00063         }
00064         app.end_loop();
00065     }
00066 
00067     return 0;
00068 }
```

5.10 main.cpp

```
00001 #include <string>
00002 #include <memory>
00003 #include <sstream>
00004 #include <map>
00005 
00006 #include "gui/app.h"
00007 #include "processor.h"
00008 
00009 #define PROJECT_NAME "IaMP_Lab2"
00010 
00011 using namespace std;
00012 
00013 int main(int, char**)
00014 {
00015     App app(PROJECT_NAME);
00016     map<string, shared_ptr<Processor» images;
00017 
00018     imgui_addons::ImGuiFileBrowser *file_dialog = app.getFile_dialog();
00019     // Main loop
00020     while (!app.should_closed())
00021     {
00022         app.begin_loop();
00023         auto mode = imgui_addons::ImGuiFileBrowser::DialogMode::OPEN;
00024 
00025         if (file_dialog->showFileDialog(
00026                 "Open File", mode, ImVec2(700, 310), ".png,.jpg,.bmp"))
00027         {
00028             string path(file_dialog->selected_path);
00029             auto img = Image::fromFile(path.c_str());
00030             auto pathImage = make_shared<Image>(img);
00031             auto gray = make_shared<ImageData>(pathImage->toGray());
00032             {
00033                 stringstream ss;
00034                 ss « "Original Image: " « file_dialog->selected_fn
00035                    « " " « file_dialog->selected_path;
00036                 auto data = make_shared<ImageData>(pathImage);
00037                 auto proc = make_shared<Processor>(data);
00038                 images.insert(make_pair(ss.str(), proc));
00039             }
00040             {
00041                 stringstream ss;
00042                 ss « "Gray Image: " « file_dialog->selected_fn
00043                    « " " « file_dialog->selected_path;
00044                 auto proc = make_shared<Processor>(gray);
00045                 images.insert(make_pair(ss.str(), proc));
00046             }
00047             {
00048                 stringstream ss;
00049                 ss « "Equalization: " « file_dialog->selected_fn
00050                    « " " « file_dialog->selected_path;
```

```
00051                 auto data = make_shared<ImageData>(gray->equalize());
00052                 auto proc = make_shared<Processor>(data);
00053                 images.insert(make_pair(ss.str(), proc));
00054             }
00055         }
00056 
00057         for (auto image = images.begin(); image != images.end();) {
00058             if (!image->second->process_image(image->first.c_str())) {
00059                 image = images.erase(image);
00060             } else {
00061                 ++image;
00062             }
00063         }
00064         app.end_loop();
00065     }
00066 
00067     return 0;
00068 }
```

5.11 Файл mainpage.dox

5.12 Файл processor.cpp

```
#include <stdio.h>
#include "processor.h"
#include "imgui.h"
```

5.13 processor.cpp

```
00001 #include <stdio.h>
00002 #include "processor.h"
00003 #include "imgui.h"
00004 
00005 using std::make_shared;
00006 
00007 Processor::Processor(shared_ptr<const ImageData> input)
00008     : orig(input),
00009       data(input),
00010       dissection_x{100, 200},
00011       dissection_y{0, 1},
00012       dissection{0},
00013       dilate_params{1, 1},
00014       erode_params{1, 1},
00015       dissected(false),
00016       dilate(false),
00017       erode(false)
00018 {
00019     updateDissection();
00020 }
00021 
00022 void Processor::updateDissection() {
00023     data = orig;
00024     if (dissected) {
00025         float diffY = dissection_y[1] - dissection_y[0];
00026         int diffX = dissection_x[1] - dissection_x[0];
00027         float diff = diffY / diffX;
00028         float acc = dissection_y[0];
00029         memset(dissection, 0, 256 * sizeof(channel_t));
00030         memset(dissectionF, 0, 256 * sizeof(float));
00031         for (int i = dissection_x[0]; i < dissection_x[1]; i++) {
00032             acc += diff;
00033             dissection[i] = acc * 255;
00034             dissectionF[i] = acc;
00035         }
00036         data = make_shared<ImageData>(data->image->dissect(dissection));
00037     }
00038     if (dilate) {
00039         data = make_shared<ImageData>(data->image->dilate(dilate_params));
00040     }
00041     if (erode) {
00042         data = make_shared<ImageData>(data->image->erode(erode_params));
00043     }
00044     texture.update(*data->image);
00045 }
00046 
```

```
00047 bool Processor::process_image(const char *name) {
00048     bool opened = true;
00049     auto img = data->image;
00050     auto tex_id = (void*)(intptr_t)(texture.id);
00051     ImGui::PushID(name);
00052
00053     ImVec2 size(img->width, img->height);
00054     ImGui::SetNextWindowSize(size, ImGuiCond_FirstUseEver);
00055     ImGui::PushStyleVar(ImGuiStyleVar_WindowPadding, ImVec2(0,0));
00056     ImGui::Begin(name, &opened, ImGuiWindowFlags_NoSavedSettings);
00057     ImGui::Image(tex_id, ImGui::GetContentRegionAvail());
00058     ImGui::End();
00059     ImGui::PopStyleVar();
00060
00061     char info_name[128];
00062     snprintf(info_name, 128, "Info %s", name);
00063     ImGui::Begin(info_name, NULL, ImGuiWindowFlags_NoSavedSettings);
00064     ImGui::Text("pointer = %ld", (intptr_t)tex_id);
00065     ImGui::Text("size = %d x %d", img->width, img->height);
00066     ImGui::PlotHistogram(
00067             "##", data->histogramF.get(), 256, 0, "Histogram", 0.0f,
00068             data->maxHistogramF, ImVec2(0, 100.0f));
00069     bool changed = false;
00070     ImGui::Spacing();
00071     changed |= ImGui::Checkbox("Dissected", &dissected);
00072     if (dissected) {
00073         {
00074             auto val = dissection_x;
00075             changed |= ImGui::DragIntRange2("dissection x", val, val + 1, 1, 0,
00076                                             256, "Min: %d", "Max: %d");
00077         }
00078         {
00079             auto val = dissection_y;
00080             changed |= ImGui::DragFloatRange2("dissection y", val, val + 1, 0.01,
00081                                               0, 1, "Min: %.2f", "Max: %.2f");
00082         }
00083         ImGui::PlotLines("Lines", dissectionF, 256, 0, NULL, 0, 1.0f,
00084                          ImVec2(0, 80.0f));
00085     }
00086     ImGui::Spacing();
00087     changed |= ImGui::Checkbox("Dilate", &dilate);
00088     if (dilate) {
00089         changed |= ImGui::SliderInt2("dilate params", dilate_params, 0, 16);
00090     }
00091     changed |= ImGui::Checkbox("Erode", &erode);
00092     if (erode) {
00093         changed |= ImGui::SliderInt2("erode params", erode_params, 0, 16);
00094     }
00095     if (changed) updateDissection();
00096     ImGui::End();
00097
00098     ImGui::PopID();
00099     return opened;
00100 }
```

5.14 Файл processor.h

```
#include <cstring>
#include "imagedata.h"
#include "gui/texture.h"
```

Класи

- class Processor

  Processing image. Calculates histogram, applies dissection or max filter and stores result image to texture.

5.15 processor.h

```
00001 #ifndef PROCESSOR_H
00002 #define PROCESSOR_H
00003 #include <cstring>
```

```
00004 #include "imagedata.h"
00005 #include "gui/texture.h"
00006
00015 class Processor {
00019     shared_ptr<const ImageData> orig;
00023     shared_ptr<const ImageData> data;
00027     const Texture texture;
00031     int dissection_x[2];
00037     float dissection_y[2];
00041     channel_t dissection[256];
00045     float dissectionF[256];
00049     int dilate_params[2];
00053     int erode_params[2];
00057     bool dissected;
00061     bool dilate;
00065     bool erode;
00066 public:
00071     Processor(shared_ptr<const ImageData> input);
00075     void updateDissection();
00076
00082     bool process_image(const char *name);
00083 };
00084
00085 #endif // PROCESSOR_H
```