

Министерство образования и науки Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования  
«Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой \_\_\_\_\_

Преподаватель \_\_\_\_\_  
(подпись)

«\_\_\_\_\_» \_\_\_\_\_ 2018 г.

Отчет  
по лабораторной работе № 2

по дисциплине «Интеллектуальные технологии обработки изображений»

**ЛР 09.04.04.20.000 О**

Студент группы 8ПИ-61 А.Г. Шевелёва  
(И. О., Фамилия)

Преподаватель старший преподаватель М.Г. Казаков  
должность, ученое звание (И. О., Фамилия)

Барнаул 2018

### Постановка задачи:

- Из заданного изображения построить гауссову пирамиду
- Реализовать отображение результатов

### Решение:

#### Фильтр Гаусса.

Фильтр Гаусса убирает высокочастотную — информацию из изображения (работает низкочастотным фильтром). Применение данного фильтра означает свёртку изображения ядром Гаусса.

$$G_{\sigma} = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Рисунок 1 — Формула для вычисления ядра Гаусса

$\sigma$  (сигма) — радиус ядра свёртки.

Фильтр Гаусса обладает следующими свойствами:

- Свёртка фильтра самого с собой дает так же фильтр Гаусса:



- Свёртка два раза с Гауссовым ядром радиусом  $\sigma$  = свертке с ядром радиусом  $\sigma\sqrt{2}$

- Свёртки с фильтрами  $\sigma_1$  и  $\sigma_2$  = свертке с фильтром  $\sigma_3^2 = \sigma_1^2 * \sigma_2^2$

### Результат работы:



Рисунок 2 — Исходное изображение



Рисунок 3 — Исходное свёрнутое Гауссовым ядром

## Пирамиды.

### Свойства пирамид:

- Представление изображения на разных уровнях детализованности
- От наиболее детализированного до наименее детализированного
- Каждый следующий уровень в два раза меньше предыдущего

### Scale space (пространство масштаба)

- $L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$

$$G_{\sigma} = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

$\sigma$  – параметр масштаба

- (альтернативное определение через  $t = \sigma^2$ )
- $\sigma=0$ :

$$L(x, y, 0) = I(x, y)$$

### Пирамиды из октав.

- Плавное изменение  $\sigma$
- Производим свертки до  $2\sigma_0$ , затем уменьшаем в два раза
- max число октав зависит от размера изображения
- число масштабов внутри октавы любое фиксированное



Рис 4 — Первые уровни каждой октавы

## Исходный код программы:

### Pyramid.h

```
struct Layer{
    Image *image;
    double sigmaCurrent;
    double sigmaGlobal;
};

class Pyramid
{
private:
    const double startSigma = 0.5;
    const double zeroSigma = 1.6;
    int countLayers;    //количество уровней в октаве - любое фиксированное
    int countOctaves;
    vector<vector<Layer>> vectorOctaves;
public:
    Pyramid(Image *image, int countLayersInOctave, Border border);
    int getCountLayers();
    int getCountOctaves();
    Image* getImageFromPyramid(int octaveNumber, int layerNumber);
private:
    int calculateCountOctaves(int width, int height);
    void calculatePyramidOctaves(Image *image, Border border);
    double accountDeltaSigma(double sigmaPrevious, double sigmaFollowing);
};
```

### Pyramid.cpp

```
Pyramid::Pyramid(Image *image, int countLayersInOctave, Border border)
{
    countLayers = countLayersInOctave;
    countOctaves = calculateCountOctaves(image->getWidth(), image->getHeight());
    calculatePyramidOctaves(image, border);
}

int Pyramid::getCountLayers(){
    return countLayers;
}

int Pyramid::getCountOctaves(){
    return countOctaves;
}

int Pyramid::calculateCountOctaves(int width, int height){
    int minSide;
```

```

    if(width < height)
        minSide = width;
    else
        minSide = height;

    return log(minSide);
}

void Pyramid::calculatePyramidOctaves(Image *image, Border border){
    double k = pow(2, 1.0/countLayers);

    for(int i = 0; i < countOctaves; i++){
        vector<Layer> layerVector;
        if(i == 0){
            Layer layer;
            layer.sigmaCurrent = zeroSigma;
            layer.sigmaGlobal = zeroSigma;
            double deltaSigma = accountDeltaSigma(startSigma, layer.sigmaCurrent);

            ImageModification *imageModifGaussX = new ImageModification(Modification::GaussX, deltaSigma);
            Image *gaussXImg = image->modificateImage(imageModifGaussX, border);
            ImageModification *imageModifGaussY = new ImageModification(Modification::GaussY, deltaSigma);
            layer.image = gaussXImg->modificateImage(imageModifGaussY, border);

            layerVector.push_back(layer);
        }
        else{
            vector<Layer> octavePrevious = vectorOctaves[i - 1];

            Layer layer;
            layer.sigmaCurrent = octavePrevious[countLayers - 1].sigmaCurrent / 2;
            layer.sigmaGlobal = octavePrevious[countLayers - 1].sigmaGlobal * 2;

            layer.image = (octavePrevious[countLayers - 1].image->reduceImage(border);
            layerVector.push_back(layer);
        }
        for(int j = 1; j < countLayers; j++){
            Layer layer;
            layer.sigmaCurrent = layerVector[j - 1].sigmaCurrent * k;
            layer.sigmaGlobal = layerVector[j - 1].sigmaGlobal * k;
            double deltaSigma = accountDeltaSigma(layerVector[j - 1].sigmaCurrent, layer.sigmaCurrent);

            ImageModification *imageModifGaussX = new ImageModification(Modification::GaussX, deltaSigma);
            Image *gaussXImg = layerVector[j - 1].image->modificateImage(imageModifGaussX, border);
            ImageModification *imageModifGaussY = new ImageModification(Modification::GaussY, deltaSigma);
            layer.image = gaussXImg->modificateImage(imageModifGaussY, border);

            layerVector.push_back(layer);
        }
        vectorOctaves.push_back(layerVector);
    }
}

double Pyramid::accountDeltaSigma(double sigmaPrevious, double sigmaFollowing){
    return sqrt(pow(sigmaFollowing, 2) - pow(sigmaPrevious, 2));
}

Image* Pyramid::getImageFromPyramid(int octaveNumber, int layerNumber){
    if(octaveNumber >= countOctaves)
        octaveNumber = countOctaves - 1;
    if(octaveNumber < 0)
        octaveNumber = 0;

```

```

if(layerNumber >= countLayers)
    layerNumber = countLayers - 1;
if(layerNumber < 0)
    layerNumber = 0;

return vectorOctaves[octaveNumber][layerNumber].image;
}

```

## Main.cpp

```

void MainWindow::on_pushButton_Gaus_clicked()
{
    //Taycc
    ImageModification *imageModifGaussX = new ImageModification(Modification::GaussX, 6);
    Image *gaussXImg = image->modificateImage(imageModifGaussX, Border::Black);

    ImageModification *imageModifGaussY = new ImageModification(Modification::GaussY, 6);
    Image *gaussXYImg = gaussXImg->modificateImage(imageModifGaussY, Border::Black);
    showImageAtSecondScreen(gaussXYImg->getImage());

    gaussXYImg->saveImage(filePath + "gaussXYImg");
}

void MainWindow::on_pushButton_PyramidShow_clicked()
{
    Pyramid *pyramid = new Pyramid(image, 4, Border::Black);
    int countOctaves = pyramid->getCountOctaves();
    int countLayers = pyramid->getCountLayers();

    for(int i = 0; i < countOctaves; i++)
        for(int j = 0; j < countLayers; j++){
            Image *imagePyramid = pyramid->getImageFromPyramid(i, j);
            imagePyramid->saveImage(filePath + "pyramidImg" + QString::number(i) + QString::number(j));
        }

    showImageAtSecondScreen(pyramid->getImageFromPyramid(ui->spinBox_Octave->value(), ui->spinBox_Layer->value())->getImage());
}

```