

УТВЕРЖДАЮ

Должность

_____ ФИО

“ ” _____ 2018 г.

Пояснительная записка № 004

Этап 4. Программная реализация и тестирование

НИР

**Реконструкция 3D модели поверхности микроскопического объекта по
серии изображений**

(«Get3DModel»)

Н.Новгород

2018

Реферат

Пояснительная записка 004, страниц 27.

КЛЮЧЕВЫЕ СЛОВА: математическая модель, структура решения, ограничения на решение, эталон, оценка решения, алгоритм решения, схема оценки работы алгоритма, Get3DModel, тестовый базис, эталон.

В пояснительной записке в рамках проекта НИР «Get3DModel» представлены:

- Программная реализация расчетного блока кода;
- Тестовая инфраструктура;
- Тестирование кода;
- тривиальные примеры;
- комбинированные (сложные) примеры;
- эталоны для тривиальных примеров и метрики для сложных примеров;
- код нарезки фрагментов исходных данных (картинок);
- файлы формата .camera;

А также параметры оценки выходных данных:

- суммарная ошибка;
- максимальная ошибка;
- средняя ошибка;
- глубина уровня;
- заполняемость;
- равномерность распределения точек;
- время выполнения;

Оглавление

Термины и определения.....	4
1 Программная реализация расчетного блока кода	5
2 Тестовая инфраструктура	7
3 Тестирование кода.....	20
4 Список используемых источников	27

Термины и определения

Get3DModel – разрабатываемый в рамках текущей НИР ([1]) программный модуль реконструкции 3D модели поверхности микроскопического объекта по серии изображений

3D изображение – изображение, полученное путем моделирования объемных объектов в трехмерном пространстве.

Градиент - вектор, своим направлением указывающий направление наибольшего возрастания некоторой величины, значение которой меняется от одной точки пространства к другой (скалярного поля), а по величине (модулю) равный скорости роста этой величины в этом направлении.

Тривиальные примеры – примеры входных данных; набор изображений одинакового размера (формат.png размером не больше 1К), полученный микросъемкой одного и того же объекта с разной высоты.

Комбинированные (сложные) примеры - примеры входных данных; набор изображений одинакового размера (формат.png размером не больше 4К), полученный микросъемкой одного и того же объекта с разной высоты, содержащие в себе комбинации сложных для анализа фрагментов (блики, размытость, затемнения и тд.);

Ошибка – модуль разности высот соответствующих точек из эталонной и полученной моделей.

1 Программная реализация расчетного блока кода

Рассматриваемый блок состоит из 5 логических составляющих:

1. Класс для динамического подбора ядер.

Интерфейс: `IAalysis`

Класс реализующий интерфейс: `Analysis`

Основной функционал:

- а. Добавить изображение для анализа
`void addImageAnalysis (Data.Imageimage);`
`image` : изображение анализа
- б. Получить ядра для каждой точки
`List<IMathematical>getCore();`

2. Класс для обработки изображения.

Интерфейс: `IChangeImage`

Класс реализующий интерфейс: `ChangeImage`

Основной функционал:

- а. Перевод изображения в монохром
`Bitmap translateToMonochrome(Bitmap image);`
`image` : изображение для перевода

3. Класс для отсева точек не несущих достоверную информацию

Интерфейс: `IElimination`

Класс реализующий интерфейс: `Elimination`

Основной функционал:

- а. Рассчитать градиенты на изображении
`void calculateGradientImage(Image image);`
`image` : изображение для расчёта
- б. Получить достоверные точки
`List<Data.Point>getSolution();`

4. Класс для математических подсчетов

Интерфейс: `IMathematical`

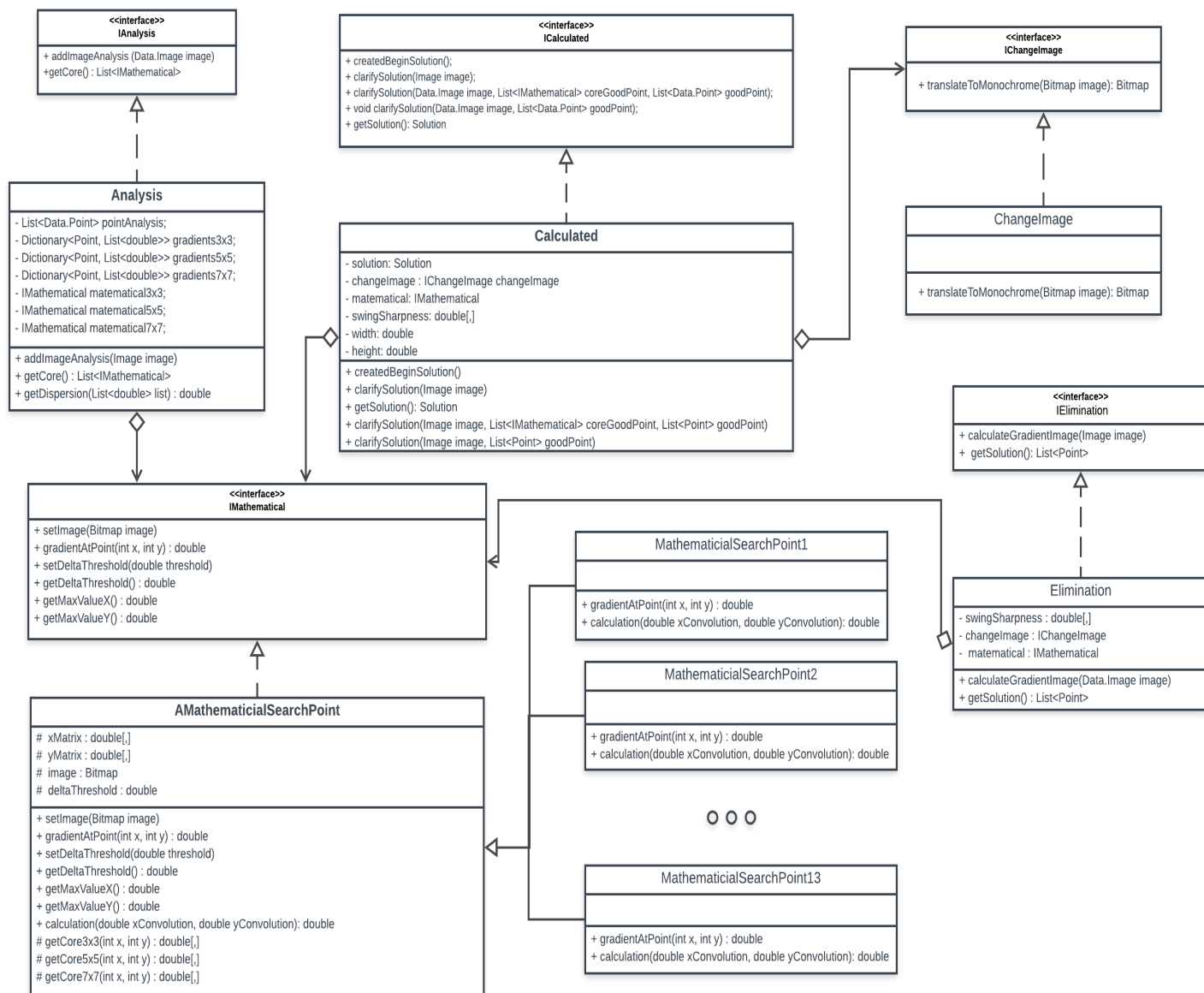
Абстрактный класс реализующий интерфейс: `MathematicalSearchPoint`

Список классов наследуемых от абстрактного, реализующие разные варианты ядер для подсчета градиента: `MathematicalSearchPoint1` - `MathematicalSearchPoint13`

Основной функционал:

- а. Установка изображения для расчетов
`void setImage(Bitmap image);`
`image` : изображение для расчёта

- b. Подсчет градиента в точке
doublegradientAtPoint(intx, inty);
[x,y] : координаты точки
 - c. Установить пороговое значение
voidsetDeltaThreshold(doublethreshold);
threshold : пороговое значение
 - d. Получить пороговое значение
doublegetDeltaThreshold();
 - e. Установить пороговое значение
voidsetDeltaThreshold(doublethreshold);
threshold : пороговое значение
5. Основной класс для подсчета решения задачи
Интерфейс: ICalculated
Класс реализующий интерфейс: Calculated
Основной функционал:
- a. Создать начальное решение
voidcreatedBeginSolution();
 - b. Уточнить решение
voidclarifySolution(Imageimage);
image: изображение для уточнения решения
 - c. Уточнить решение
voidclarifySolution(Data.Imageimage, List<IMathematical>coreGoodPoint, List<Data.Point>goodPoint);
image: изображение для уточнения решения
coreGoodPoint: список ядер для каждой достоверной точки
goodPoint: список достоверных точек
 - d. Уточнить решение
voidclarifySolution(Data.Imageimage, List<Data.Point>goodPoint);
image: изображение для уточнения решения
goodPoint: список достоверных точек
 - e. Получить решение
Solution getSolution();



2 Тестовая инфраструктура

Для анализа эффективности того или иного алгоритма, необходимо решение следующих задач:

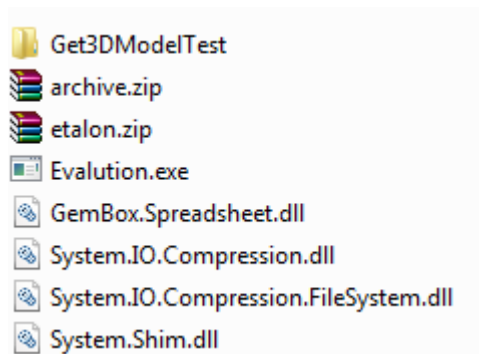
1. Программная реализация тестовой инфраструктуры для оценки алгоритмов;
2. Подготовка входных примеров разных размеров для тестирования алгоритмов;
3. Сравнение полученных выходных данных с эталонами (для тривиальных примеров) по параметрам, обговорённым с Заказчиком;

4. Оформление данных, полученных с помощью тестов, в виде таблицы(образец таблицы представлен ниже – таблица 1).

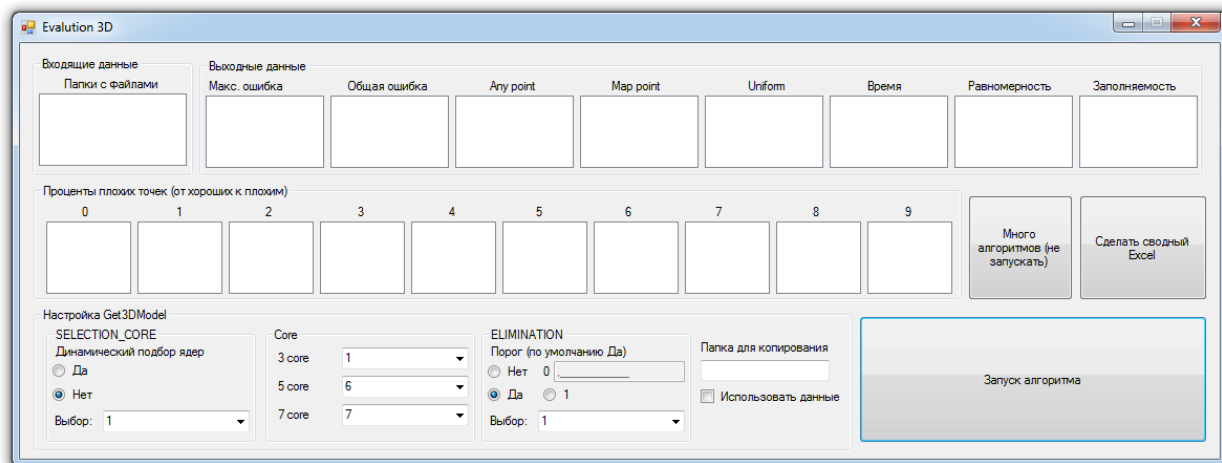
Описание программной реализации тестовой инфраструктуры

1) Для корректной работы программы необходимо:

- Программа Evaluation.exe;
- 4 DLL-файла: GemBox.Spreadsheet.dll, System.IO.Compression.dll, System.IO.Compression.FileSystem.dll и System.Shim.dll;
- Zip-архив archive.zip, в котором находятся папки с картинками и файлом sample.camera в каждой из них;
- Zip-архив etalon.zip, в котором находятся папки с теми же названиями, что и в предыдущем файле, а в этих папках файл etalon.dat – эталонный dat-файл;
- Папка с программой Get3DModelTest, в которой находится соответствующая программа со всем содержимым.



2) Запуск программы:



3) В нижней части программы в группе Настройка Get3DModel устанавливаем настройки в соответствии с файлом «Настройка для тестового подпроекта с помощью файла настроек *setting.ini.docx*». Так же, как и в данном файле, программа настройки программы разделены на 3 части: SelectionCore (динамический подбор ядер), Core (настройка ядер для процедуры отсева) и Elimination (настройка отсева).

- В блоке SelectionCore есть 2 кнопки radiobutton, отвечающие за выбор использовать или нет динамический подбор ядер. В случае нединамического подбора, существует возможность выбора одной из 11 стратегий (MathematicalSearchPoint1-...- MathematicalSearchPoint11).
- Блок Core имеет 3 настройки, с выбором номеров стратегий для 3х, 5ти и 7ми ядер в соответствии с ТЗ.
- Блок Elimination имеет 3 radiobutton, с помощью которых можно выбрать порог по умолчанию (Да), любой свой в диапазоне от $0 \leq x < 1$ (Нет) и порог, равный единице (1). Кроме того, есть выбор стратегии для подсчёта градиента (также номера от 1 до 11).

4) Необходимо также указать название папки для копирования (см. далее).

5) После запуска алгоритма (кнопка «Запуск алгоритма») последовательно будут проделаны следующие действия:

- Создастся папка из текстового поля;
- В неё будет разархивировано содержимое архива *archive.zip*;
- В неё же скопирован файл настроек *setting.ini*, который будет сгенерирован программой;
- Запущена программа Get3DModelTest из папки с программой.
- После окончания работы той программы (около 5-10 минут) будет создана папка «Results_НазваниеПапкиДляКопирования» и разархивирован файл *etalon.zip*
- В программе будут заполнены все оставшиеся поля (группы «Выходные данные» и «Проценты плохих точек (от хороших к плохим)» (см скриншот из пункта 2).

6) Также в папке «Results_НазваниеПапкиДляКопирования» будут созданы Excel-файлы: один файл будет называться «Данные», а остальные будут названы также, как и названия папок.

7) В файле «Данные.xls» есть несколько листов:

- Лист «Данные» содержит сравнительную информацию по всем экспериментам из верхней части программы (Макс ошибка, Общая ошибка, Map point, Any point, Эффективность, Время генерации, Равномерность, Заполняемость).

Название	Максимальная ошибка	Общая ошибка	Map point	Any point	Эффективность	Время генерации файлов	Равномерность	Заполняемость
test_3x3	9	190894,53	640000	141319	99,67	214546	100	22
test_conus	9,00	121229,05	640000	169475	99,97	185124	100	26
test_QS	27,5	531420,25	640000	87565	98,27	405456	100	13
test_torus	21	413816,21	640000	126688	97,93	394767	100	19

- Лист «Сравнение» содержит сравнительную информацию о процентах плохих точек

	1	2	3	4	5	6	7	8	9	10	11
1	Название	0	1	2	3	4	5	6	7	8	9
2	test_3x3	2,37%	0,00%	1,89%	0,00%	0,00%	0%	0,00%	0,00%	0,00%	0,00%
3	test_conu	3,39%	0,75%	0,61%	0,46%	0,29%	0,15%	0,06%	0,01%	0,00%	0,00%
4	test_QS	0,82%	0,42%	0,86%	0,39%	0,41%	0,21%	0,00%	0,01%	0,02%	0,00%
5	test_torus	2,23%	0,03%	0,08%	0,16%	0,04%	0,03%	0,03%	0,02%	0,02%	0,02%

- Остальные листы названы также, как и папки. Там находятся те же данные, что и в предыдущем листе, только в более привычном виде (транспонированном).

	1	2	3
1	Начало	Конец	Процент
2	0	0,80	2,37%
3	0,80	1,60	0,00%
4	1,60	2,40	1,89%
5	2,40	3,20	0,00%
6	3,20	4,00	0,00%
7	4,00	4,80	0%
8	4,80	5,60	0,00%
9	5,60	6,40	0,00%
10	6,40	7,20	0,00%
11	7,20	8,00	0,00%

8) Также в той же папке создадутся файлы с названием папок. В каждом из этих файлов будут:


- Лист «Данные» это данные по процентам точек (как в пункте 7В)

	1	2	3
1	Начало	Конец	Процент
2	0	0,80	2,37%
3	0,80	1,60	0,00%
4	1,60	2,40	1,89%
5	2,40	3,20	0,00%
6	3,20	4,00	0,00%
7	4,00	4,80	0%
8	4,80	5,60	0,00%
9	5,60	6,40	0,00%
10	6,40	7,20	0,00%
11	7,20	8,00	0,00%

- Лист «Статистика» это одна строка из файла Данные, листа Данные (пункт 7А)

1	2	3	4	5	6	7
Название	Максимальная ошибка	Общая ошибка	Map point	Any point	Эффективность	Время генерации файлов
test_3x3	8	24550,15	640000	27408	98,78	147

- Листы «0»... «9» это листы с координатами точек. «0» - это лист с наилучшим совпадением точек, а «9», соответственно, с наихудшим. Первая координата это X, вторая - Y

	1	2
1	33	
2	33	221
3	293	85
4	293	230
5	323	25
6	412	25
7		

- 9) В программе есть ещё способ использовать ранее полученные данные для получения Excel-файлов. Для этого перед запуском алгоритма необходимо поставить галочку checkbox в поле «Использовать данные» и в текстовое поле «Папка для копирования» написать нужное название папки (например, для папки «*Resulst_0000*» необходимо написать «0000»).
- 10) Для получения сводной таблицы по всем экспериментам можно воспользоваться кнопкой «Сделать сводный Excel». После нажатия этой кнопки будет создан файл "*Сводная.xls*" в папке с программой.
- 11) Кнопка «Много алгоритмов (не запускать)» работает пока в тестовом режиме, запуск её нежелателен. Она должна простым перебором создать все возможные *setting.ini*-файлы и запустить алгоритмы со всеми этими настройками.

Тестирование проводилось на разных типах картинок, предоставленных заказчиком:

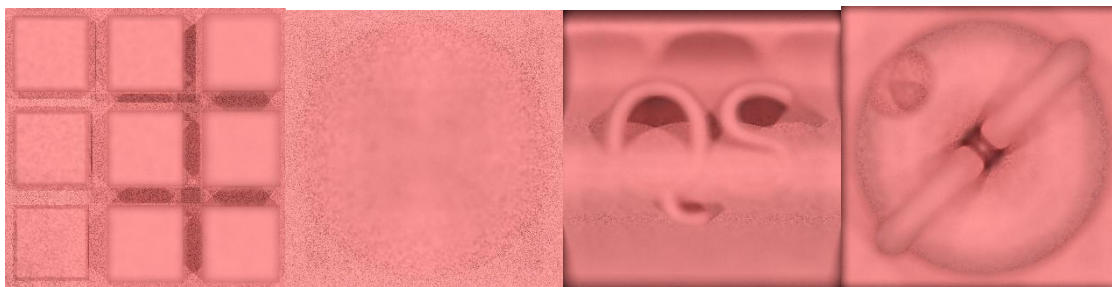
Тривиальные примеры

Картинки формата .png размером не более 1К.

Образцы:

test3x3

test_conustest_QStest_torus



Эталоны:

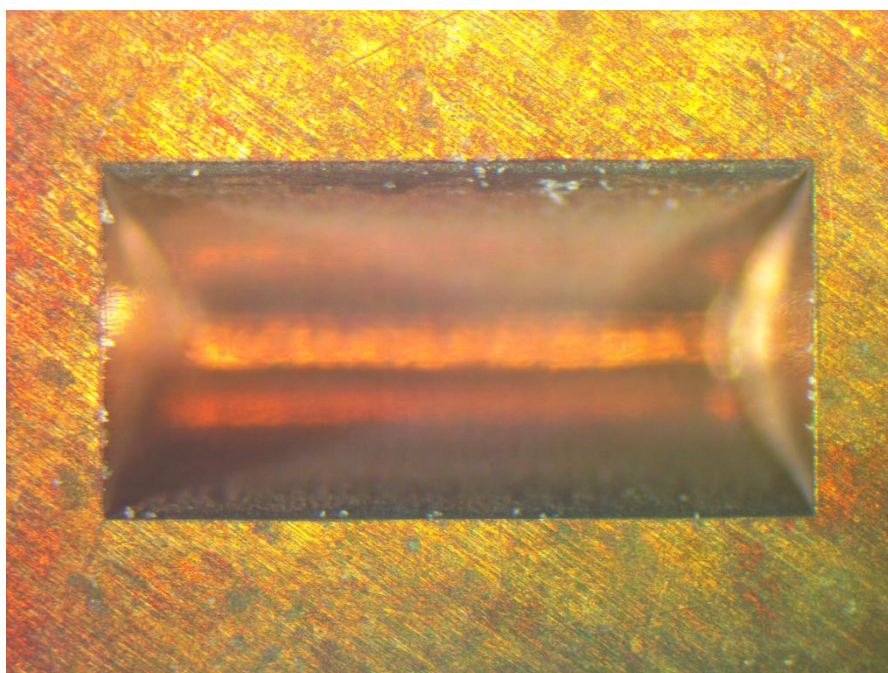
Файлы формата .dat, содержащие матрицу размера, соответствующего входной картинке, где номера строк/столбцов матрицы – координаты точек, ячейки матрицы – высоты соответствующих точек.

Комбинированные (сложные) примеры

Картинки формата .png размером не более 4К

Образец:

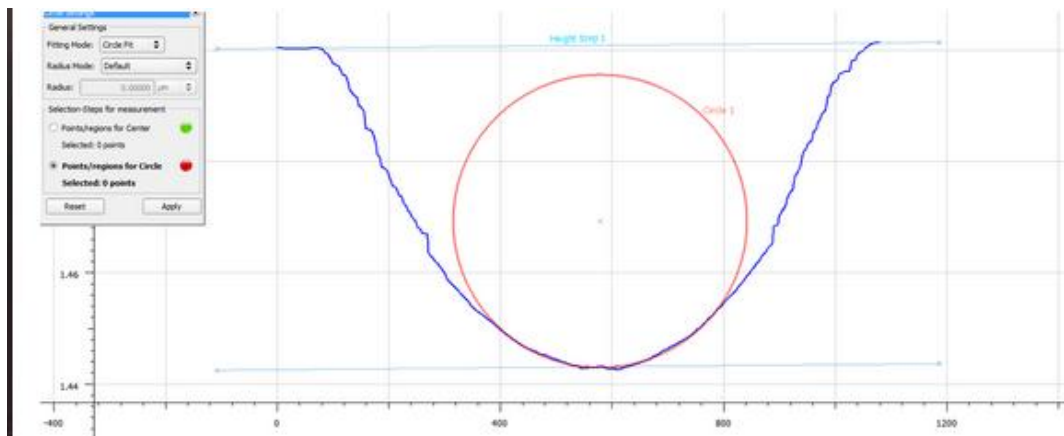
trench.png



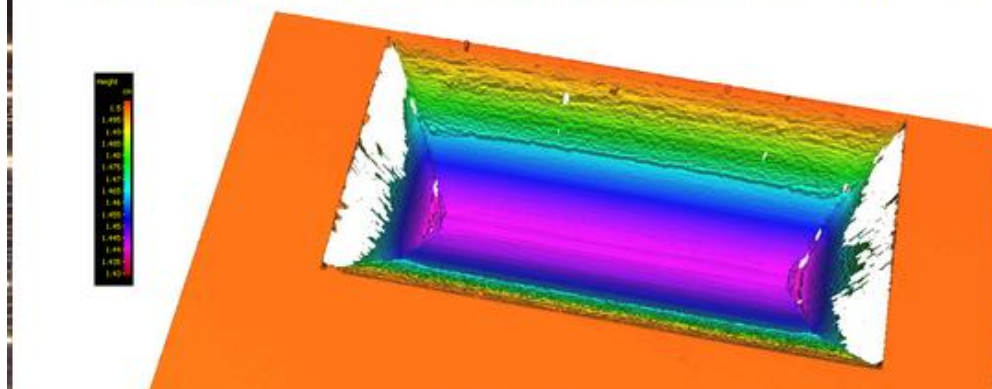
Эталон:

В данном примере требований к точности нет, однако, результаты оценки выходной модели по этому тесту должны быть отражены в отчёте в отдельной таблице.

Метрики, предоставленные Заказчиком



А это результаты обработки данного объекта коммерческой системой (Alicona):



Файлы формата .camera

параметры оптической системы, при помощи которой были получены изображения поверхности микроскопического объекта (фокусное расстояние, наблюдаемая ширина в фокусе, коэффициент для вычисления абсолютной высоты фокуса);

Схема оценки выходных данных

При условии, что ограничения по времени и по заполнению точек выполняются, оценка производится по следующим параметрам (по убыванию значимости для Заказчика):

- **Суммарная ошибка** (Ошибка – модуль разности высот соответствующих точек из эталонной и полученной моделей);

Математическое описание параметра:

Пусть $R_{S_t \times N_t}$ и $M_{S_t \times N_t}$ – матрицы t – ого тестового набора, содержащие высоты точек эталонной и полученной моделей соответственно, тогда r_{ij_t} и m_{ij_t} – элементы эталонной и полученной матриц соответственно, где r и m – высоты точек с координатами i, j . ($r, m \in \mathbb{Z}$; $(i, j) \in \{G_t\}$, где G_t – множество точек t -ого тестового набора, для которых $r_{ij_t} > 0$ (высота точки положительна), $t = \overline{1, p}$, где p – количество тестовых наборов, $S_t \times N_t$ – размеры картинки t – ого тестового набора).

Индивидуальная оценка теста:

$$\sum_{(i,j) \in G_t} |r_{ij_t} - m_{ij_t}|$$

Средняя оценка тестового набора:

$$\frac{\sum_{t=1}^p (\sum_{(i,j) \in G_t} |r_{ij_t} - m_{ij_t}|)}{p}$$

- **Максимальная ошибка**

Индивидуальная оценка теста:

$$\max_{(i,j) \in G_t} |r_{ij_t} - m_{ij_t}|$$

Нормированная оценка:

$$\frac{\max_{(i,j) \in G_t} |r_{ij_t} - m_{ij_t}|}{\max r_{sn_t} - \min r_{sn_t}},$$

где $s = \overline{1, S}$, $n = \overline{1, N}$, знаменатель дроби - разность между максимальной и минимальной высотами эталона соответствующего t - ого теста.

Максимальная ошибка тестового набора:

$$\frac{\sum_{t=1}^p \frac{\max_{(i,j) \in G_t} |r_{ij_t} - m_{ij_t}|}{\max r_{sn_t} - \min r_{sn_t}}}{p},$$

где $s=\overline{1, S}$, $n=\overline{1, N}$, и p – количество тестовых наборов.

- **Средняя ошибка**

Описание параметра:

Равна отношению суммарной ошибки теста к количеству точек, с высотой $r_{ij_t} > 0$

Индивидуальная оценка теста:

$$\frac{\sum_{(i,j) \in G_t} |r_{ij_t} - m_{ij_t}|}{G_t},$$

где G_t – множество точек t -ого тестового набора, для которых $r_{ij_t} > 0$

(высота точки положительна), r_{ij_t} и m_{ij_t} – элементы эталонной и полученной матриц соответственно, где r и m – высоты точек с координатами i, j .

Нормированная оценка:

$$\frac{\sum_{t=1}^p \frac{\sum_{(i,j) \in G_t} |r_{ij_t} - m_{ij_t}|}{G_t}}{p}, \text{ где}$$

p – количество тестовых наборов.

- **Глубина уровня**

Описание параметра:

Равна отношению средней ошибки теста к количеству его (теста) картинок.

Индивидуальная оценка теста:

$$\frac{\frac{\sum_{(i,j) \in G_t} |r_{ij_t} - m_{ij_t}|}{G_t}}{W_t},$$

где числитель дроби – средняя ошибка, W_t – количество картинок t – ого, $t = \overline{1, p}$ теста.

Нормированная глубина уровня:

$$\frac{\sum_{t=1}^p \sum_{(i,j) \in G_t} \frac{|r_{ij_t} - m_{ij_t}|}{\frac{G_t}{W_t}}}{p},$$

p – количество тестовых наборов.

- **Заполняемость**

Математическое описание параметра:

Пусть D – множество точек полученной модели, для которых соответствующие высоты больше нуля ($m_{ij} > 0$), $D \in \{S \times N\}$.

Тогда заполняемость вычисляется по формуле:

$$\frac{D}{S \cdot N}$$

- **Равномерность распределения точек**

Математическое описание параметра:

Точки восстанавливаемых объектов, высоты которых найдены алгоритмом, должны быть равномерно распределены по исследуемой области. Для этого необходимо вычислить вектор:

$$V = (v_{1_t}, v_{2_t}, \dots, v_{l_t}), \text{ где}$$

l – количество уровней равномерного распределения;

t – количество тестовых наборов;

Координата вектора:

$$v_{i_t} = \frac{fact_{i_t}}{real_{i_t}}, i = \overline{1, l}, \text{ где}$$

$fact_{i_t}$ – количество областей i -ого уровня, содержащих хотя бы одну точку с найденной высотой.

$real_{i_t}$ – количество областей i -ого уровня, на которые делим изображение.

Поэтому исходя из параметра равномерности необходимо выполнение следующего условия:

Индивидуальная оценка теста:

$$\frac{\sum_{i=1}^l v_{i_t}}{l} * 100 \geq R$$

Среднее распределение для всех тестовых наборов высчитывается по формуле:

$$\frac{\sum_{t=1}^p \frac{\sum_{i=1}^l v_{i_t}}{l} * 100}{p}$$

- **Время выполнения**

Высчитывается время выполнения одного теста. Для оценки алгоритма необходима общая оценка времени выполнения всех тестов. В связи с этим, применяется нормировка времени.

Математическое описание параметра:

Пусть t_i – время выполнения i -го теста ($i = \overline{1, p}$, где p – количество тестовых наборов);

n_i – число картинок i -го теста;

$m_i * s_i$ – количество точек картинки i – го теста.

Индивидуальная оценка теста:

$$\frac{t_i}{m_i \cdot s_i \cdot n_i}$$

Средняя оценка тестового набора:

$$\frac{\sum_{i=1}^p \frac{t_i}{m_i \cdot s_i \cdot n_i}}{p}$$

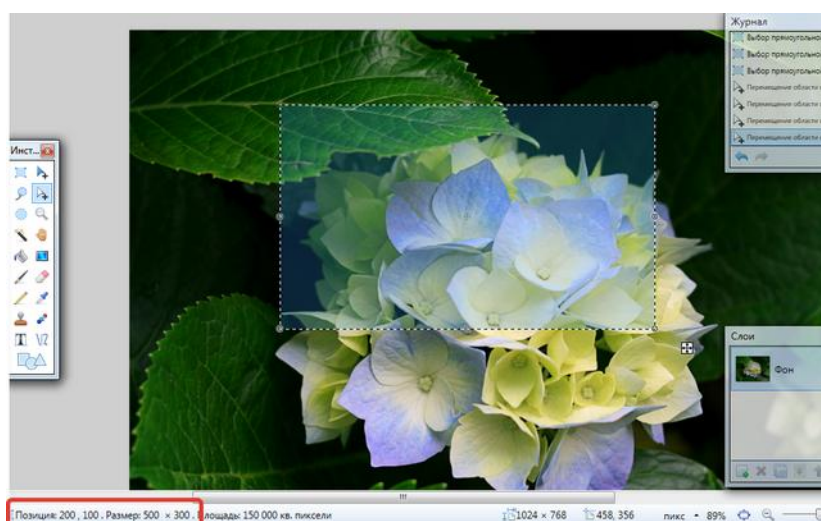
Результаты будут представлены в виде итоговой таблицы, в которой будут отражены сведения по каждому тесту отдельно, а также нормированные сведения по всем тестам.

Нарезка фрагментов

Запуск программы происходит в файле формата .bat, файл содержит:

- название программы;
- формат файлов;
- координаты левого верхнего угла (сначала горизонтальная, затем вертикальная);
- размер вырезаемого изображения;

Пример: cutpng.exe png 200 100 500 300



Оформление результатов тестирования

Таблица 1.

Core 1.10.11

Название	максимальная ошибка	Суммарная ошибка	Общее число точек	Число точек с положительной высотой	Время генерации файлов	Равномерность	Заполняемость	Средняя ошибка	Глубина уровня	Время нормированное
test_3x3	6	10301,24	640000	14441	87878	100	2,25	0,71	0,07	0,01373
test_conus	3	765,76	640000	9517	85706	83,33	1,48	0,08	0,00	0,01339
test_QS	27,5	29611,90	640000	5009	198338	66,66	0,78	5,91	0,26	0,01408
test_torus	14,35	5407,93	640000	8482	197547	83,33	1,32	0,63	0,02	0,01403
Σ	12,71	11521,70	640000	9362,25	142367,25	83,33	1,45	1,83	0,08	0,01380

3 Тестирование кода

Задачей являлось:

1. Создание тестового набора для каждой нетривиальной функции или метода консольного приложения Get3DModel.exe.
2. Прогон программы на тестах с получением отчета по результатам тестов.
3. Оценка результаты выполнения программы на наборе тестов с целью принятия решения о продолжении или остановке тестирования.
4. После очередного изменения кода - регрессионное тестирование (обнаружение ошибок в уже протестированных участках исходного кода).

Целью являлось:

1. Выявление локализованных ошибок в реализации алгоритмов.
2. Выявление ошибок кодирования (ошибки работы с условиями, счетчиками циклов, использования локальных переменных и ресурсов)

Обзор:

1. Расположение тестов:

Все тесты находятся в проекте Get3DModel.UnitTests

2. Тестовые классы.

Именованые: [Имя класса]Tests

Каждый тестовый класс тестирует только одну сущность. Класс проекта соответствует одному тестовому классу.

3. Тестовые методы:

Именованые: [Название тестируемого методаTest].

4. Оформление теста: «arrange-act-assert» (организация – действие – утверждение).

В модульном тесте чётко определены предусловия (инициализация тестовых данных, предварительные установки), действия (то, что тестируется) и постусловия (что должно быть в результате выполнения действия).

5. Результаты тестирования представлены в таблице:

Тестируемый блок	Тестируемый класс	Приоритет	Тестируемый метод	Название тестового метода/Тест-кейс	Результат (True\False)
Calculate dBlock	EliminationTests	1	void calculateGradientImage(Data.Image image)	calculateGradientImageTest Проверка подсчета градиентов картинки	True
			List<Data.Point>getSolution()	getSolutionTest Проверка получения списка хороших точек (отсева)	True
	MathematicalSearchPoint1 (2 ядра свертки: 3x3 вычисление градиента: по среднему арифметическому)	1	MathematicalSearchPoint1()	MathematicalSearchPoint_1Test Проверка ядер свертки	True
			double gradientAtPoint(int x, int y)	gradientAtPoint1_CornerPoint_Test Проверка градиента угловой точки	True
			double gradientAtPoint(int x, int y)	gradientAtPoint1_BoundaryPoint_Test Проверка градиента граничной точки	True

		double gradientAtPoint(int x, int y)	gradientAtPoint1_IntPoint_Tes t Проверка градиента внутренней точки	True
MathematicalSearchPoint2 (2 ядра свертки: 3x3 вычисление градиента: по сумме квадратов)	1	MathematicalSearchPoint2()	MathematicalSearchPoint_2Test Проверка ядер свертки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint2_CornerPoint _Test Проверка градиента угловой точки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint2_BoundaryPoint _Test Проверка градиента границной точки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint2_IntPoint_Tes t Проверка градиента внутренней точки	True
MathematicalSearchPoint3 (1 ядро свертки: 3x3 вычисление градиента: по ядру свертки)	1	MathematicalSearchPoint3()	MathematicalSearchPoint_3Test Проверка ядра свертки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint3_CornerPoint _Test Проверка градиента угловой точки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint3_BoundaryPoint _Test Проверка градиента границной точки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint3_IntPoint_Tes t Проверка градиента внутренней точки	True
MathematicalSearchPoint4 (1 ядро свертки: 3x3 вычисление градиента: по ядру свертки)	1	MathematicalSearchPoint4()	MathematicalSearchPoint_4Test Проверка ядра свертки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint4_CornerPoint _Test Проверка градиента угловой точки	True

		double gradientAtPoint(int x, int y)	gradientAtPoint4_BoundaryPoint_Test Проверка градиента границной точки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint4_IntPoint_Test Проверка градиента внутренней точки	True
MathematicalSearchPoint5 (2 ядра свертки: 3x3 вычисление градиента: максимум из градиентов)	1	MathematicalSearchPoint5() double gradientAtPoint(int x, int y)	MathematicalSearchPoint_5Test Проверка ядра свертки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint5_CornerPoint_Test Проверка градиента угловой точки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint5_BoundaryPoint_Test Проверка градиента границной точки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint5_IntPoint_Test Проверка градиента внутренней точки	True
MathematicalSearchPoint6 (2 ядра свертки: 5x5 вычисление градиента: по сумме квадратов)	1	MathematicalSearchPoint6() double gradientAtPoint(int x, int y)	MathematicalSearchPoint_6Test Проверка ядер свертки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint6_CornerPoint_Test Проверка градиента угловой точки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint6_BoundaryPoint_Test Проверка градиента границной точки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint6_IntPoint_Test Проверка градиента внутренней точки	True
MathematicalSearchPoint7 (2 ядра свертки: 7x7)	1	MathematicalSearchPoint7() double gradientAtPoint(int x, int y)	MathematicalSearchPoint_7Test Проверка ядер свертки	True

	вычисление градиента: по сумме квадратов)		double gradientAtPoint(int x, int y)	gradientAtPoint7_CornerPoint_Test Проверка градиента угловой точки	True
			double gradientAtPoint(int x, int y)	gradientAtPoint7_BoundaryPoint_Test Проверка градиента граничной точки	True
			double gradientAtPoint(int x, int y)	gradientAtPoint7_IntPoint_Test Проверка градиента внутренней точки	True
	MathematicalSearchPoint8 (2 ядра свертки: 5x5 вычисление градиента: по среднему арифметическому)	1	MathematicalSearchPoint8()	MathematicalSearchPoint_8Test Проверка ядра свертки	True
			double gradientAtPoint(int x, int y)	gradientAtPoint8_CornerPoint_Test Проверка градиента угловой точки	True
			double gradientAtPoint(int x, int y)	gradientAtPoint8_BoundaryPoint_Test Проверка градиента граничной точки	True
			double gradientAtPoint(int x, int y)	gradientAtPoint8_IntPoint_Test Проверка градиента внутренней точки	True
	MathematicalSearchPoint9 (2 ядра свертки: 7x7 вычисление градиента: по среднему арифметическому)	1	MathematicalSearchPoint9()	MathematicalSearchPoint_9Test Проверка ядра свертки	True
			double gradientAtPoint(int x, int y)	gradientAtPoint9_CornerPoint_Test Проверка градиента угловой точки	True
			double gradientAtPoint(int x, int y)	gradientAtPoint9_BoundaryPoint_Test Проверка градиента граничной точки	True
			double gradientAtPoint(int x, int y)	gradientAtPoint9_IntPoint_Test Проверка градиента внутренней точки	True

MathematicalSearchPoint10 (2 ядра свертки: 5x5 вычисление градиента: по среднему арифметическому)	1	MathematicalSearchPoint10()	MathematicalSearchPoint_10 Test Проверка ядра свертки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint10_CornerPoint_Test Проверка градиента угловой точки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint10_BoundaryPoint_Test Проверка градиента граничной точки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint10_InteriorPoint_Test Проверка градиента внутренней точки	True
MathematicalSearchPoint11 (2 ядра свертки: 7x7 вычисление градиента: по среднему арифметическому)	1	MathematicalSearchPoint11()	MathematicalSearchPoint_11 Test Проверка ядра свертки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint11_CornerPoint_Test Проверка градиента угловой точки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint11_BoundaryPoint_Test Проверка градиента граничной точки	True
		double gradientAtPoint(int x, int y)	gradientAtPoint11_InteriorPoint_Test Проверка градиента внутренней точки	True
Analysis	1	private double getDispersion(List<double> list)	getDispersionTest Проверка вычисления дисперсии для значений, которые лежат в list вероятность каждого значения = 1/n где n - количество элементов в list	True
	1	public List<IMathematical> getCore()	getCoreTest Проверка возвращаемого ядра для точки, (при котором дисперсия минимальна)	True

		1	public void addImageAnalysis(Image image)	addImageAnalysisTest Проверка получения градиентов для трех разных ядер	True
Data	Image	1	Intwidth()	widthTest Проверка получения ширины изображения в пикселях	True
			Intheight()	heightTest() Проверка получения высоты изображения в пикселях	True
			Color GetPixel(int x, int y)	GetPixelTest() Проверка получения цвета пикселя изображения	True
			Doubletall, Image(path)	tallTest Проверка считывания высоты изображения при создании объекта конструктором Image(path)	True
	Solution	2	void createdBeginSolution(int width, int height)	createdBeginSolution_image10x50_SizeBitmap10x50 Начальное решение представляет Bitmap заданного размера	True
			void createdBeginSolution(int width, int height)	createdBeginSolution_image10x20_WhitePix Начальное решение представляет Bitmap из белых пикселей	True
			void createdBeginSolution(int width, int height)	createdBeginSolution_image10x20_minus1 Начальное решение представляет Bitmap с матрицей высот ==-1	True
ParsingInputData	Parser	1	Dictionary<string, double>readConfig(string path)	readConfigTest_CorrectFile Считывания корректного файла настроек по указанному пути	True
			Dictionary<string, double>readConfig(string path)	readConfigTest_IncorrectBigFile Считывание некорректного файла настроек - количество строк больше 2-ух	True

			Dictionary<string, double>readConfig(string path)	readConfigTest_IncorrectSmall File Считывание некорректного файла настроек - количество строк меньше 2-ух	True
			Dictionary<string, double>readConfig(string path)	readConfigTest_IncorrectWayFile Считывание несуществующего файла	True
			System.Drawing.Bitmap readPNG(string path)	readPNGTest_BlackImage Считывание изображения	True
Preserve	PreserveOBJ	1	void saveOBJ(Data.Solution solution, Data.Setting setting, string path)	savePNGTest Проверка сохранения OBJ файла	True
	PreservePNG	1	VoidsavePNGTest()	savePNGTest Проверка сохранения изображения в png файл	True

4 Список используемых источников

1. Техническое задание на научно-исследовательскую работу «Реконструкция 3D модели поверхности микроскопического объекта по серии изображений), Нижний Новгород, 2018.
2. Пояснительная записка № 001. Тестовый базис для тестирования ПО «Get3DModel». НИР «Get3DModel» (Тестовый базис). Н.Новгород, 2018