Министерство образования и науки РФ
ФГБОУ ВО «Тверской государственный университет»
Факультет прикладной математики и кибернетики
Кафедра информационных технологий
Направление 02.04.02 – «Фундаментальная информатика и информационные технологии»
Магистерская программа «Информационные технологии в управлении и принятии решений»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

«Лазерное сканирование поверхностей»

Автор:
Федотов Александр Александрович

Научный руководитель:
к.ф.-м.н.
Сорокин Сергей Владимирович

Допущен к защите:

Руководитель ООП:

_____/Язенин А.В./

(подпись, дата)

Заведующий кафедрой: информационных технологий

_____/Язенин А.В./

(подпись, дата)

Тверь 2018

Оглавление

Введение

На сегодняшний день создание трехмерных моделей реальных объектов и поверхностей является актуальной проблемой компьютерной графики и компьютерного зрения. Эти области быстро развиваются и постоянно появляются новые технологии. Трехмерное моделирование стало частью многих областей науки, техники и медицины. Например, один из подходов построения трёхмерных моделей – реконструкция формы предметов и поверхностей с помощью 3D-сканера.

3D-сканер представляет собой специальное устройство (рис. 1), которое анализирует физический объект и на основе полученных данных создает его трехмерную модель.

Рис. 1. 3D-сканнер

3D-сканеры можно разделить по методу сканирования на контактные и бесконтактные.

Контактные сканеры (рис. 2) построены по принципу обвода модели специальным высокочувствительным щупом, посредством которого в компьютер передаются трехмерные координаты сканируемой области. Недостаток такого вида сканеров заключается в непосредственном контакте с поверхностью, что не исключает деформацию предмета.

Рис. 2. Контактный 3D-сканнер

Бесконтактные сканеры используют определённые виды излучения или обычный свет, а изменения в излучении фиксируются камерой. Существуют сканеры на основе структурированного белого света, лазерные сканеры и сканеры, которые используют метод фотограмметрии.

Сканирование на основе структурированного белого света заключается в проецировании на объект световой сетки. Световая сетка покрывает некоторую область объекта, а камера фиксирует каждое изменение рисунка сетки (рис. 3).

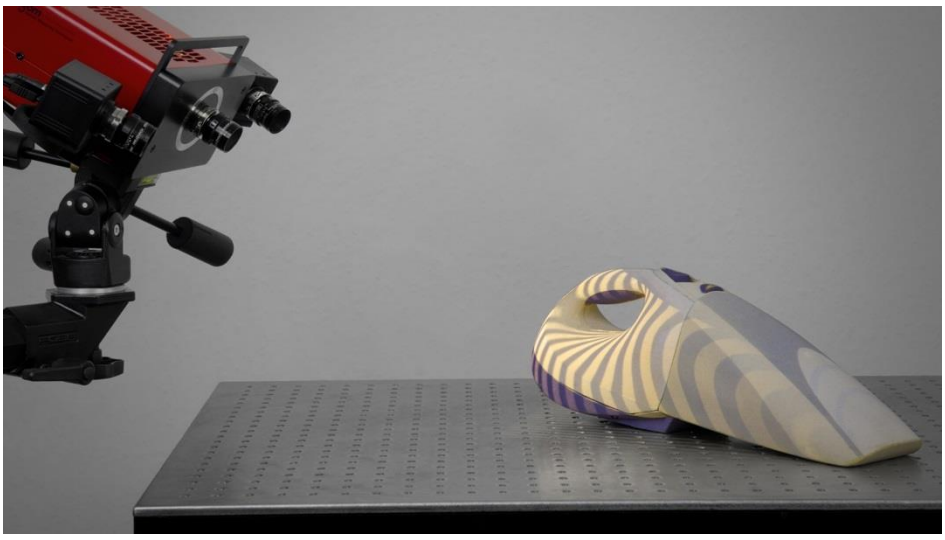

Рис. 3. 3D-сканнер на основе структурированного белого света

Сканеры с лазерной технологией основаны на проецировании лазерного луча. Все искажения лазера воспринимаются измерительной камерой, которая его отслеживает (рис. 4).

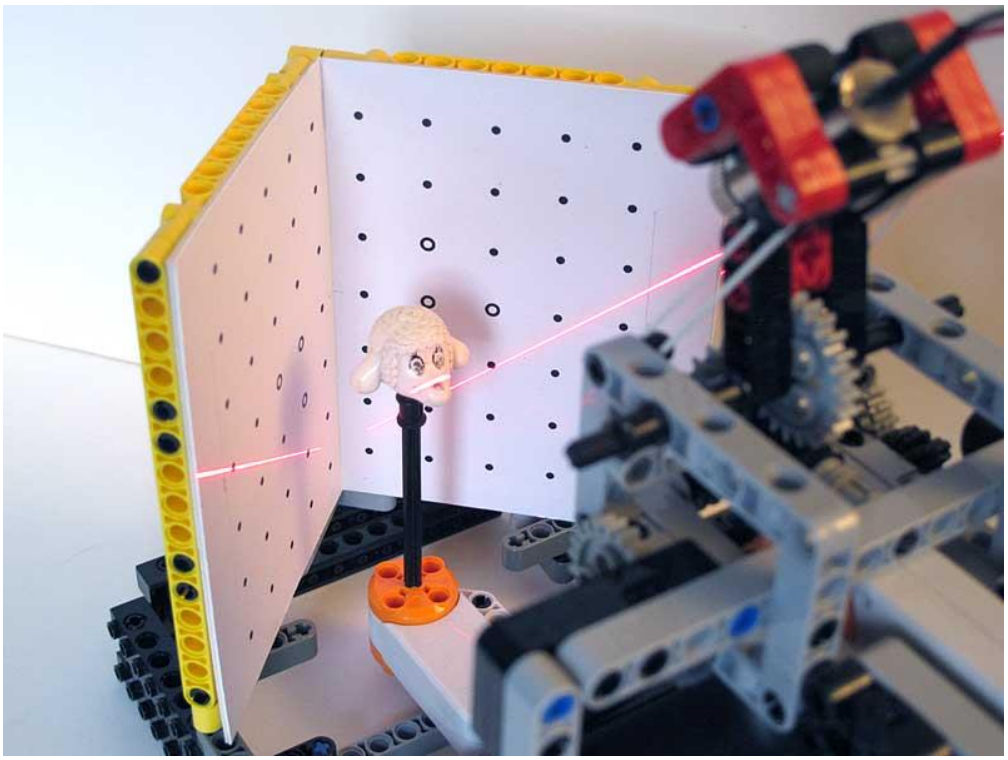

Рис. 4. 3D-сканнер на основе лазерной технологии

Также существуют бесконтактные сканеры, которые не излучают свет, а вместо этого используют отраженный свет. Так называемая технология на основе фотограмметрии представляет собой фотографирование объекта с разных сторон, а затем на основе полученных изображений воссоздается трехмерная модель.

В настоящий момент, кроме описанных выше технологий, существует также множество подходов и методов к созданию трехмерных моделей объектов без использования специальной техники. Например, один из таких подходов использует идею 3D-сканера с лазерной технологией. Идея заключается в использовании простых и доступных технических средств: видеокамера, устройство, которое может проецировать линию лазера, например, лазерный уровень, и специальный угол с отмеченными опорными точками.

Процесс сканирования (рис. 5) выглядит следующим образом: объект исследования помещается в специальный угол, далее устанавливается видеокамера и направляется на объект, и используя лазерное устройство, происходит проецирование лазерного луча на предмет.

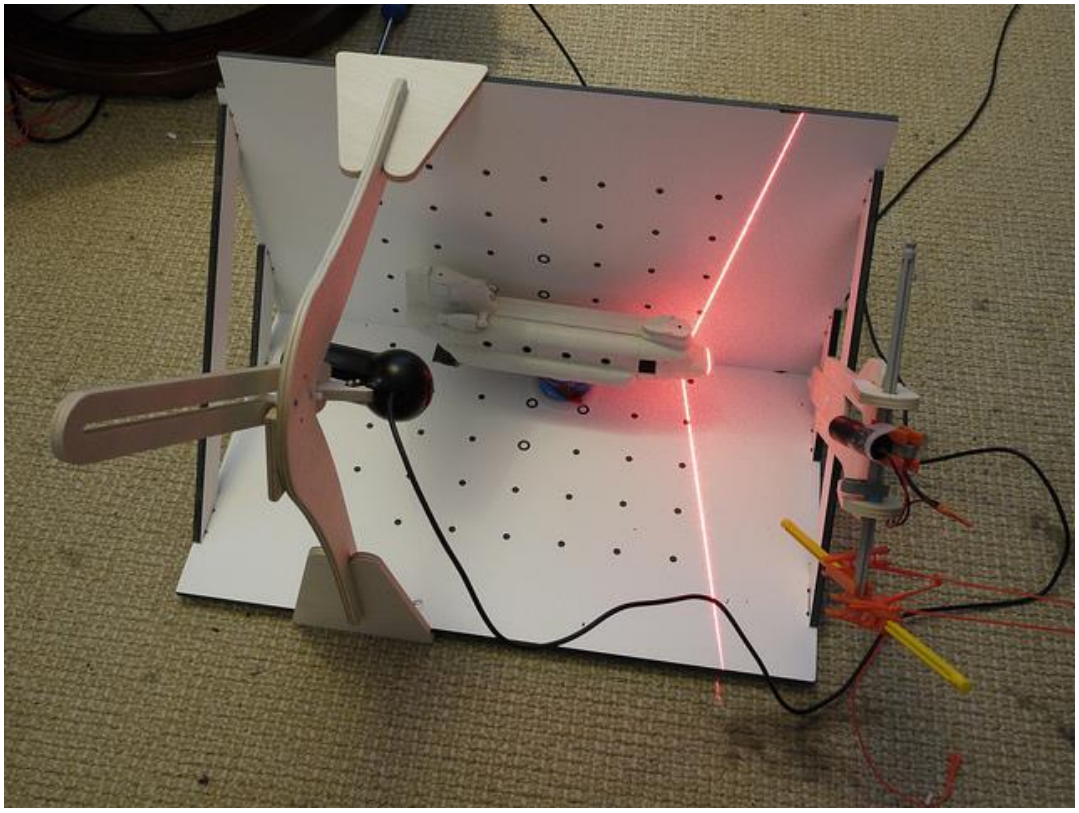

Рис. 5. Процесс сканирования

Все данные об искажении лазера записываются на видеокамеру, затем с помощью разработанных алгоритмов компьютер анализирует видеофайл с исследуемым объектом, и на выходе алгоритма получается трехмерная модель.

В нашей работе мы преследуем цель создать алгоритм построения трёхмерных моделей поверхностей. Наш подход к решению задачи опирается на вышеописанный метод сканирования без специальной техники. То есть необходимо реализовать алгоритмы для анализа и обработки видеофайлов, и алгоритм для восстановления трёхмерной формы.

Целью магистерской диссертации является разработка приложения для построения трёхмерных моделей поверхностей с помощью лазерного сканирования. Для достижения указанной цели необходимо решить следующие задачи:

1. изучить предметную область;
2. реализовать алгоритм обнаружения линии лазера на изображении;
3. реализовать алгоритм нахождения матрицы внутренних параметров камеры;
4. реализовать алгоритм нахождения уравнения плоскости положения лазера;
5. реализовать алгоритм нахождения трёхмерных координат точек, попавших в плоскость лазера.

1. Описание задачи

Для построения трёхмерной поверхности рассмотрим лазерную технологию бесконтактного 3D-сканера.

Бесконтактный трёхмерный лазерный сканер способен построить трехмерную модель, при этом, не требуя специально оснащённого помещения и профессионального освещения предмета, кроме того, не происходит физического контакта устройства непосредственно с поверхностью сканируемого предмета. Лазерная технология основана на проецировании лазерного луча на объект сканирования. Все искажения лазера фиксируются видеокамерой, после этого, все данные передаются на компьютер, который их обработает и построит трехмерную модель.

Как правило, для сканирования такой технологией необходимы некоторые опорные точки. Поэтому объект помещается в угол, где на фоне напечатаны опорные точки (рис. 6).

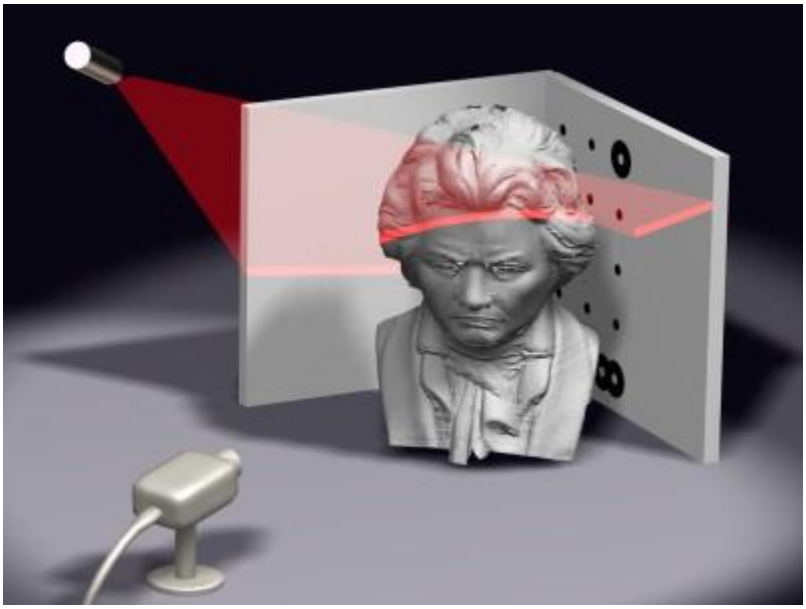

Рис. 6. Лазерное сканирование объекта

Так как нашей задачей является сканирование поверхностей, то становится очевидно, что поверхность в такой угол не поместить, и такой метод не подходит. Поэтому его необходимо модифицировать.

Пусть имеется некоторая неровная поверхность, например, поверхность пещеры, и нужно построить её трёхмерную модель. Требуется разработать алгоритм, которому на вход будет подаваться видеофайл со сканируемой поверхностью, а на выходе – её трёхмерная форма. Для подготовки входных данных понадобятся лазер, опорная рамка и видеокамера, которая будет выступать в качестве регистрирующего устройства. После того как видеосъёмка произведена, данные с искажениями лазера на поверхности передаются на компьютер на вход алгоритму.

1.1. Опорная рамка

В пункте 1 был сделан вывод, что метод сканирования объектов не подходит для сканирования поверхности из-за того, что объект нужно помещать в угол с опорными точками. И поэтому для модификации этого метода предлагается вместо угла использовать рамку с опорными точками (рис. 7).

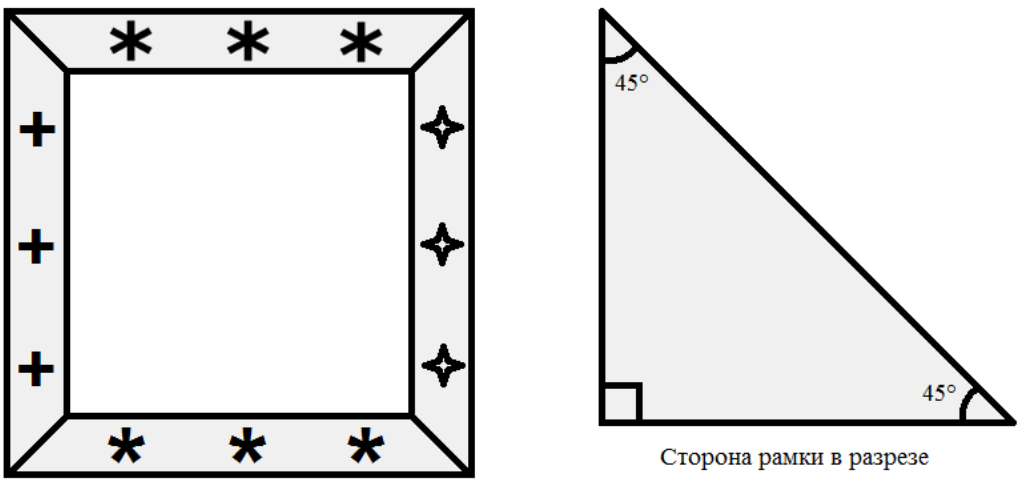

Рис. 7. Схема опорной рамки

Рамка имеет квадратную форму с нанесёнными символами на её сторонах. Символы необходимы для выделения опорных областей, которые помогают в реконструкции поверхности. Стороны рамки в разрезе представляют собой прямоугольный треугольник с углами в 45 градусов.

2. Теоретическая часть

2.1. Модель камеры

Строго говоря, различные точки пространства предметов отображаются оптической системой камеры в пространстве изображений на различных расстояниях от фокальной плоскости. Однако, если расстояние между камерой и наблюдаемой сценой значительно превышает фокусное расстояние оптической системы, можно считать, что изображение строится в ее фокальной плоскости. В этом случае можно воспользоваться проективной моделью камеры, в которой изображение трехмерного объекта получается проецированием его в фокальную плоскость (плоскость изображения) через единственную точку, называемую оптическим центром. Прямая линия, перпендикулярная плоскости изображения и проходящая через эту точку, называется оптической осью камеры, а точка пересечения оптической оси с плоскостью изображения – главной точкой.

Определим в трехмерном пространстве ортогональную правую систему координат *OXYZ*, начало которой совпадает с оптическим центром (рис. 8).

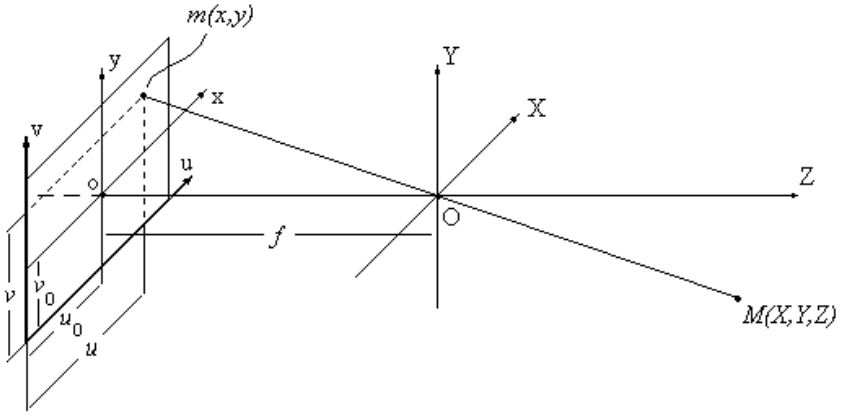

Рис. 8. Проективная модель камеры

Ось *OZ* – с оптической осью камеры. Такая система называется стандартной системой координат камеры. Пусть плоскость изображения находится на расстоянии *f* от оптического центра. В этой плоскости зададим систему координат *oxy* с началом в главной точке и осями *ox* и *oy*, параллельными осям *OX* и *OY* соответственно. Легко убедиться, что в стандартной системе координат проекцией точки трехмерного пространства *M* с координатами *(X,Y,Z)* является точка *m* в плоскости изображения с координатами *(x,y)*, причем $x = f\frac{X}{Z}, y = f\frac{Y}{Z}$.

2.2. Связь между различными системами координат

Рассмотрим подробнее как получаются соотношения $x = f\frac{X}{Z}, y = f\frac{Y}{Z}$. Пусть нас интересует некоторый объект, который располагается в глобальной системе координат (рис. 9).

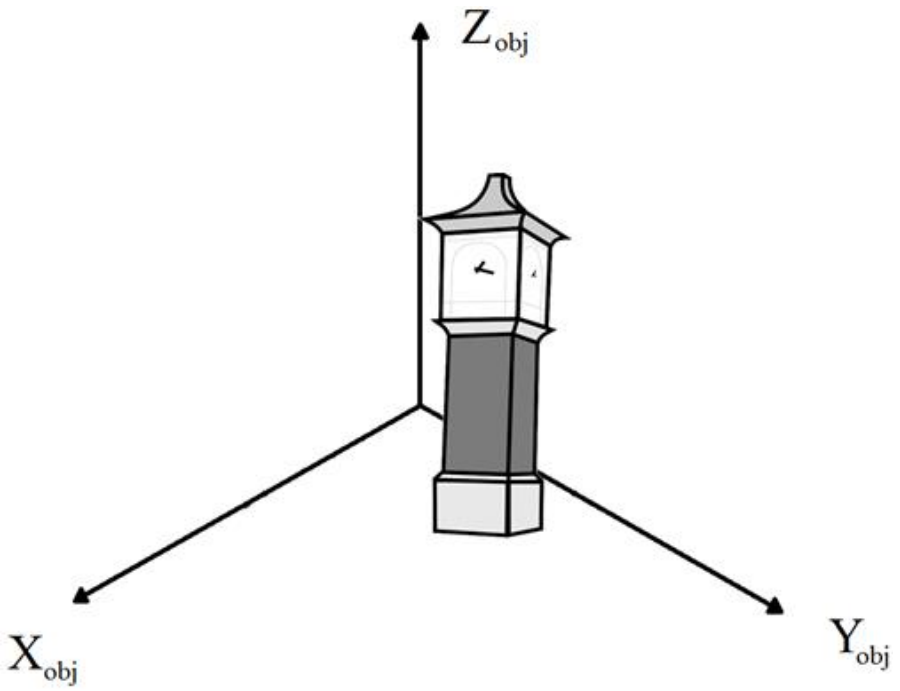

Рис. 9. Расположение объекта в глобальной системе координат

Есть также камера, находится в глобальных координатах и при этом имеет свои координаты, и f – фокусное расстояние до плоскости с изображением объекта (рис. 10).

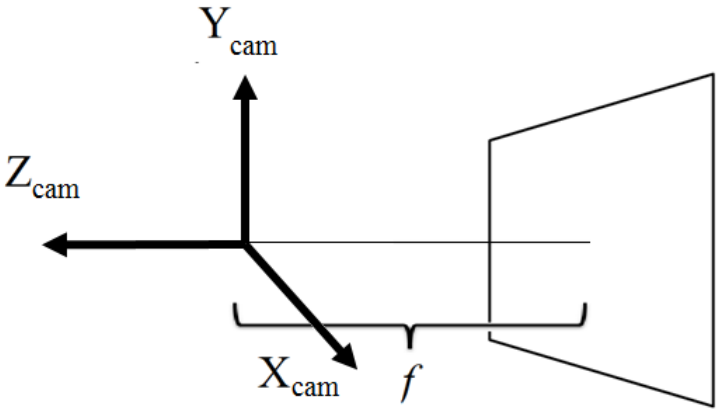

Рис. 10. Глобальные координаты камеры

При проецировании на плоскость объекта получается следующая картинка (рис. 11). Каждая трехмерная точка реального объекта отображается в одну двумерную точку на плоскости. В этой плоскости система координат располагается так: x || X_{cam}, y || Y_{cam}.

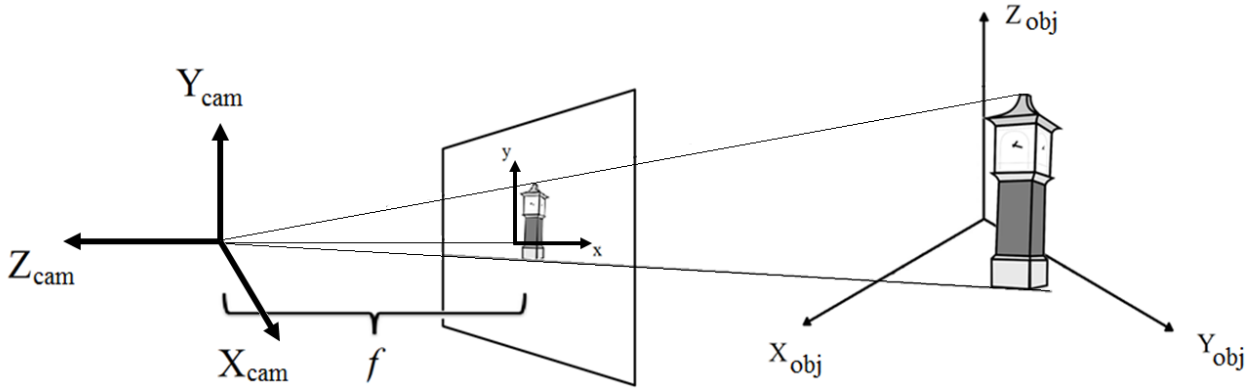

Рис. 11. Проекция трёхмерного объекта на плоскость

А на экране монитора система координат будет как на рис. 12.

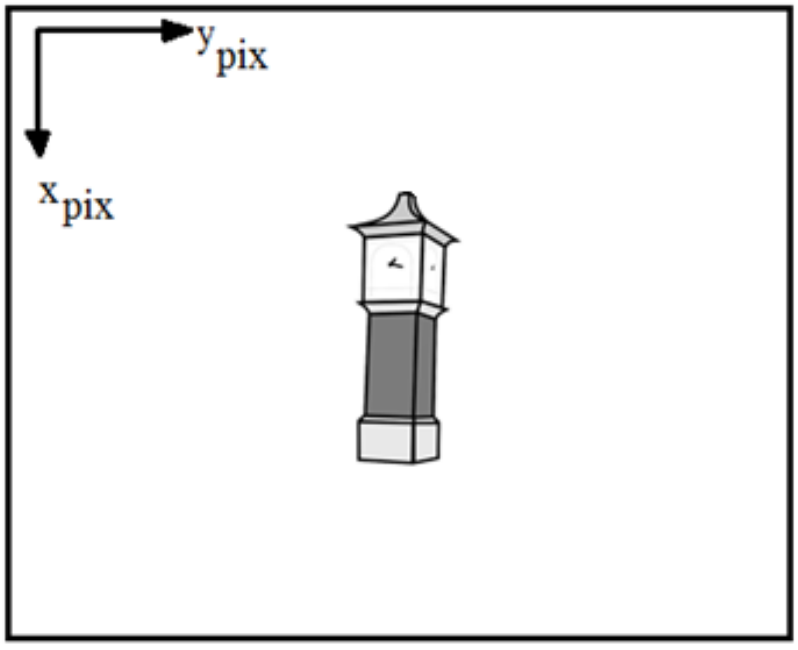

Рис. 12. Изображение объекта на экране монитора

Таким образом, имеем четыре различные системы координат, которые связаны между собой. И можно получать двумерные координаты точек по трехмерным координатам точек и наоборот.

Связь между системами координат можно отобразить в следующем виде:

Таблица 1

Глобальные координаты		Координаты камеры		Координаты плоскости проецирования трёхмерного объекта		Координаты на экране
X_{obj} Y_{obj} Z_{obj}	$\longleftrightarrow$	X_{cam} Y_{cam} Z_{cam}	$\longleftrightarrow$	x y	$\longleftrightarrow$	x_{pix} y_{pix}

Переход от глобальных координат к координатам камеры происходит с помощью формул сдвига и поворота. Чтобы это сделать нужно каждую из осей OX_{obj}, OY_{obj} и OZ_{obj} повернуть на угол α, β и γ соответственно, так, чтобы направления осей глобальной системы координат совпадали с направлением осей в системе координат камеры. Затем необходимо сдвинуть глобальную систему координат на вектор $t = (t_x, t_y, t_z)$ (рис. 13).

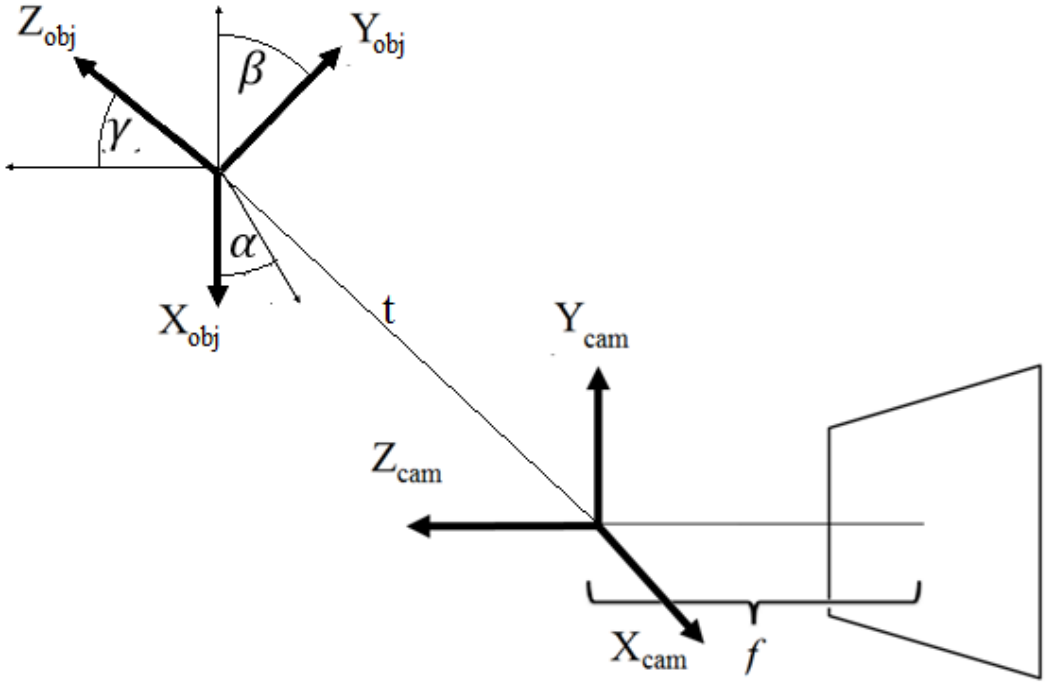

Рис. 13. Операция сдвига и поворота

Используя однородные координаты, данные соотношения можно записать следующим образом:

1. для сдвига:

$$\begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix} \quad (10)$$

2. для поворота:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) & 0 \\ 0 & sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (11)$$

$$R_y(\beta) = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\beta) & 0 & cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (12)$$

$$R_z(\gamma) = \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 & 0 \\ sin(\gamma) & cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (13)$$

То есть преобразование из глобальных координат в координаты камеры можно записать так:

$$P_{cam} = T * R * P_{obj} =$$

$$= \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} *$$

$$* \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ 1 \end{bmatrix}, \quad (14)$$

где r_{ij} – коэффициенты поворота вокруг осей OX_{obj}, OY_{obj}, OZ_{obj},

t_x, t_y, t_z – координаты смещения по осям OX_{obj}, OY_{obj}, OZ_{obj}.

Формулы перехода от координат камеры к координатам плоскости проецирования трехмерного объекта получаются из рассмотрения отношения подобных треугольников. На рис. 14 рассмотрим треугольники ABC и AB_1C_1 для координаты *x*, и треугольники DKL и DK_1L_1 для координаты *y*:

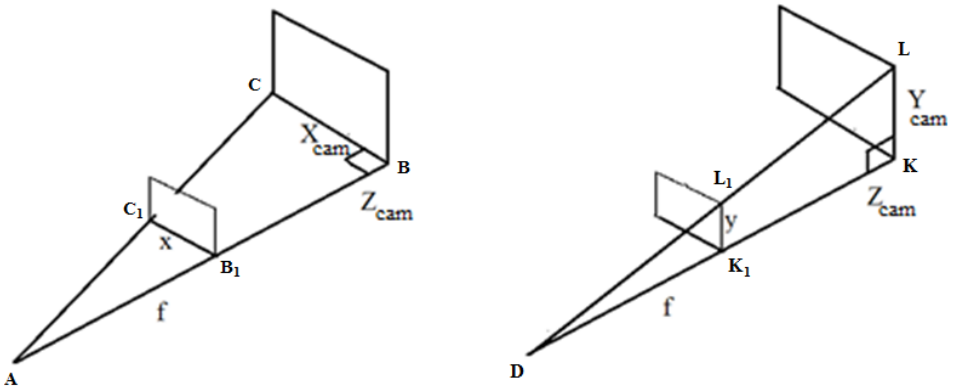

Рис. 14. Соотношение подобных треугольников

Треугольники ABC и AB_1C_1 подобны, а значит получаем соотношение: $\frac{AB_1}{AB} = \frac{B_1C_1}{BC}$ или $\frac{f}{Z_{cam}} = \frac{x}{X_{cam}}$, откуда следует $x = f\frac{X_{cam}}{Z_{cam}}$. Проведя те же рассуждения для треугольников DKL и DK_1L_1, получим $y = f\frac{Y_{cam}}{Z_{cam}}$.

Или в матричном виде:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{bmatrix}, \text{откуда } x = \frac{x'}{w} \text{ и } y = \frac{y'}{w}. \quad (15)$$

Преобразование системы координат плоскости проецирования трехмерного объекта в экранные координаты происходит по формулам сдвига:

$$\begin{bmatrix} x_{pix} \\ y_{pix} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (16)$$

Теперь для преобразования из координат камеры в координаты плоскости изображения можно записать окончательную формулу:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & o_x & 0 \\ 0 & f & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} P_{cam}, \quad (17)$$

где (o_x, o_y) – точка пересечения оптической оси камеры с плоскостью изображения, она же главная точка; P_{cam} – точка в системе координат камеры; f – фокусное расстояние.

Все преобразования были рассмотрены только в одну сторону, т.е. получение двумерных координат точки по трехмерным координатам. Но они применимы и в обратную сторону, собственно этим и будем пользоваться для восстановления трехмерной формы поверхности по последовательности изображений.

2.3. Матрица внутренних параметров камеры

Описанный выше метод преобразования координат подходит, если известны f фокусное расстояние, вектор перемещения t = (t_x, t_y, t_z) и углы поворота α, β, γ. Но в большинстве случаев эти параметры оказываются неизвестны. Тогда можно воспользоваться алгоритмом нахождения матрицы камеры.

Нахождение матрицы внутренних параметров камеры осуществляется затем, чтобы связать трёхмерную систему координат с двумерной. В этом нам поможет рамка с опорными точками. Поскольку известны размеры рамки, то легко можно вычислить трехмерные координаты опорных точек, а также координаты их проекций в плоскости изображения.

Формула для проецирования трёхмерных глобальных координат в координаты плоскости изображения выглядит следующим образом:

$$P_{cam} = \begin{bmatrix} x & 0 & o_x & 0 \\ 0 & y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} * P_w, \qquad (18)$$

где P_{cam} – точка в системе координат камеры, A – матрица внутренних параметров камеры, P_w – точка в глобальной системе координат.

Или по-другому, преобразование (18) выглядит следующим образом:

$$\begin{bmatrix} kx \\ ky \\ k \end{bmatrix} = A * P_w = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \qquad (19)$$

Задача состоит в определении элементов матрицы A. Матрица A имеет размер 3x4 и содержит 12 неизвестных.

Перепишем соотношение из матричного вида в виде линейных уравнений:

$$\begin{cases} kx = a_{11} * X + a_{12} * Y + a_{13} * Z + a_{14}, \\ ky = a_{21} * X + a_{22} * Y + a_{23} * Z + a_{24}, \\ k = a_{31} * X + a_{32} * Y + a_{33} * Z + a_{34}. \end{cases} \quad (20)$$

Подставляя k в первые два, получим:

$$\begin{cases} a_{11} * X + a_{12} * Y + a_{13} * Z + a_{14} - a_{41} * X * x - a_{42} * Y * x - a_{43} * Z * x - a_{44} * x = 0, \\ a_{21} * X + a_{22} * Y + a_{23} * Z + a_{24} - a_{41} * X * y - a_{42} * Y * y - a_{43} * Z * y - a_{44} * y = 0. \end{cases} \quad (21)$$

Полученные уравнения содержат 12 неизвестных ($a_{11}, \dots, a_{34}$), и 5 известных (X, Y, Z, x, y). Поскольку одной точки недостаточно для решения этой системы линейных уравнений, построим систему линейных уравнений, где известны N точек. В качестве таких точек будут выступать опорные точки на рамке, координаты которых мы знаем. Для увеличения точности оценок внутренних параметров следует использовать большее количество опорных точек, например, можно взять N = 16 (по четыре точки с каждой стороны рамки).

То есть получаем систему из 2*N линейных уравнений и 12 неизвестных:

$$\begin{cases} a_{11} * X_1 + a_{12} * Y_1 + a_{13} * Z_1 + a_{14} - a_{31} * X_1 * x_1 - a_{32} * Y_1 * x_1 - a_{33} * Z_1 * x_1 - a_{34} * x_1 = 0, \\ a_{11} * X_2 + a_{12} * Y_2 + a_{13} * Z_2 + a_{14} - a_{31} * X_2 * x_2 - a_{32} * Y_2 * x_2 - a_{33} * Z_2 * x_2 - a_{34} * x_2 = 0, \\ \vdots \\ a_{11} * X_N + a_{12} * Y_N + a_{13} * Z_N + a_{14} - a_{31} * X_N * x_N - a_{32} * Y_N * x_N - a_{33} * Z_N * x_N - a_{34} * x_N = 0, \\ a_{21} * X_1 + a_{22} * Y_1 + a_{23} * Z_1 + a_{24} - a_{31} * X_1 * y_1 - a_{32} * Y_1 * y_1 - a_{33} * Z_1 * y_1 - a_{34} * y_1 = 0, \\ a_{21} * X_2 + a_{22} * Y_2 + a_{23} * Z_2 + a_{24} - a_{31} * X_2 * y_2 - a_{32} * Y_2 * y_2 - a_{33} * Z_2 * y_2 - a_{34} * y_2 = 0, \\ \vdots \\ a_{21} * X_N + a_{22} * Y_N + a_{23} * Z_N + a_{24} - a_{31} * X_N * y_N - a_{32} * Y_N * y_N - a_{33} * Z_N * y_N - a_{34} * y_N = 0. \end{cases} \quad (22)$$

Перепишем систему линейных уравнений (22) в матричном виде:

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 & -x_1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -x_2X_2 & -x_2Y_2 & -x_2Z_2 & -x_2 \\ & & & & & & & \vdots & & & & \\ X_N & Y_N & Z_N & 1 & 0 & 0 & 0 & 0 & -x_NX_N & -x_NY_N & -x_NZ_N & -x_N \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 & -y_1 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 & -y_2X_2 & -y_2Y_2 & -y_2Z_2 & -y_2 \\ & & & & & & & \vdots & & & & \\ 0 & 0 & 0 & 0 & X_N & Y_N & Z_N & 1 & -y_NX_N & -y_NY_N & -y_NZ_N & -y_N \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{24} \\ a_{31} \\ a_{32} \\ a_{33} \\ a_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (23)$$

Или по-другому:

$$G * A' = \bar{0} \qquad (24)$$

где G – матрица размером 2Nx12,

A' – вектор размерности 12x1, вектор $\bar{0}$ – размерности 2Nx1.

Рассмотрим некоторые особенности системы (24). Очевидно, что одним из тривиальных решений будет $A' = \bar{0}$, которое не имеет физического смысла. Известно, что если однородная линейная система имеет хотя бы одно ненулевое решение, то она имеет бесконечное множество решений. Здесь необходимо различать два случая.

Первый – когда ранг матрицы G на единицу меньше размера вектора A'. Тогда существует только одно, с точностью до произвольного скалярного множителя, решение. Именно этот случай и представляет практический интерес. Для реализации этого условия необходимо, чтобы количество уравнений в (23) было не менее 11, следовательно, количество опорных точек должно быть не менее шести, в нашем случае используется 16 опорных точек.

Второй случай реализуется, если ранг G меньше размера вектора A'на два и более. Здесь может существовать множество различных решений системы, среди которых осуществить правильный выбор без привлечения дополнительных данных невозможно. Такая ситуация возникает, если все опорные точки лежат в одной плоскости.

Чтобы избежать этой ситуации, в качестве опорного объекта часто используют опорный угол (рис. 15), на котором нанесены точки, не лежащие в одной плоскости.

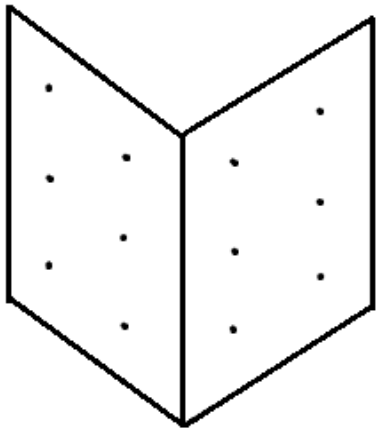

Рис. 15. Опорный угол

В нашем случае мы используем опорную рамку с четырьмя плоскостями (рис. 16). Каждая сторона рамки имеет наклон 45 градусов, на каждой из них нанесены опорные точки.

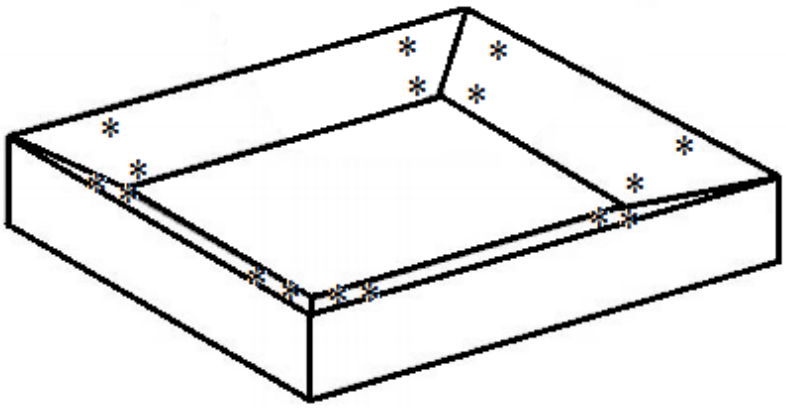

Рис. 16. Опорная рамка

Как уже было сказано, мы хотим получать точные оценки внутренних параметров камеры, для этого необходимо брать больше, чем шесть опорных точек. Но тогда системы будут получаться переопределенными. И для того, чтобы решить такую систему воспользуемся методом наименьших квадратов. Метод наименьших квадратов позволяет минимизировать ошибки измерения координат.

Положим $a_{34} = 1$, тогда система примет вид:

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -x_2X_2 & -x_2Y_2 & -x_2Z_2 \\ & & & & & & & \vdots & & & \\ X_N & Y_N & Z_N & 1 & 0 & 0 & 0 & 0 & -x_NX_N & -x_NY_N & -x_NZ_N \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 & -y_2X_2 & -y_2Y_2 & -y_2Z_2 \\ & & & & & & & \vdots & & & \\ 0 & 0 & 0 & 0 & X_N & Y_N & Z_N & 1 & -y_NX_N & -y_NY_N & -y_NZ_N \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{24} \\ a_{31} \\ a_{32} \\ a_{33} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \quad (25)$$

Получили систему с 11 неизвестными, решить которую можно следующим образом: так как матрица G не квадратная умножим обе части уравнения на G^T, получим квадратную матрицу $G^T * G$:

$$G^T * G * A'' = G^T * R' \qquad (26)$$

Затем умножим на обратную матрицу к $G^T * G$:

$$E * A'' = (G^T * G)^{-1} * G^T * R', \qquad (27)$$

где $E -$ единичная матрица.

Таким образом, оценивается матрица внутренних параметров камеры.

2.4. Описание алгоритма

Вход: видеофайл с исследуемой поверхностью и изменениями лазера.

Алгоритм:

1. Произвести разметку опорных точек на кадре;
2. Найти матрицу внутренних параметров камеры;
3. Для каждого кадра из видеоряда:
4. Определить положение линии лазера;
5. Найти параметры плоскости, которую образует лазер;
6. Найти трёхмерные координаты точек, освещённых лазером.

На выходе имеем облако точек, по которым восстанавливается трехмерная форма поверхности.

Более подробное описание алгоритма приведено на рис. 17 в виде диаграммы потоков данных.

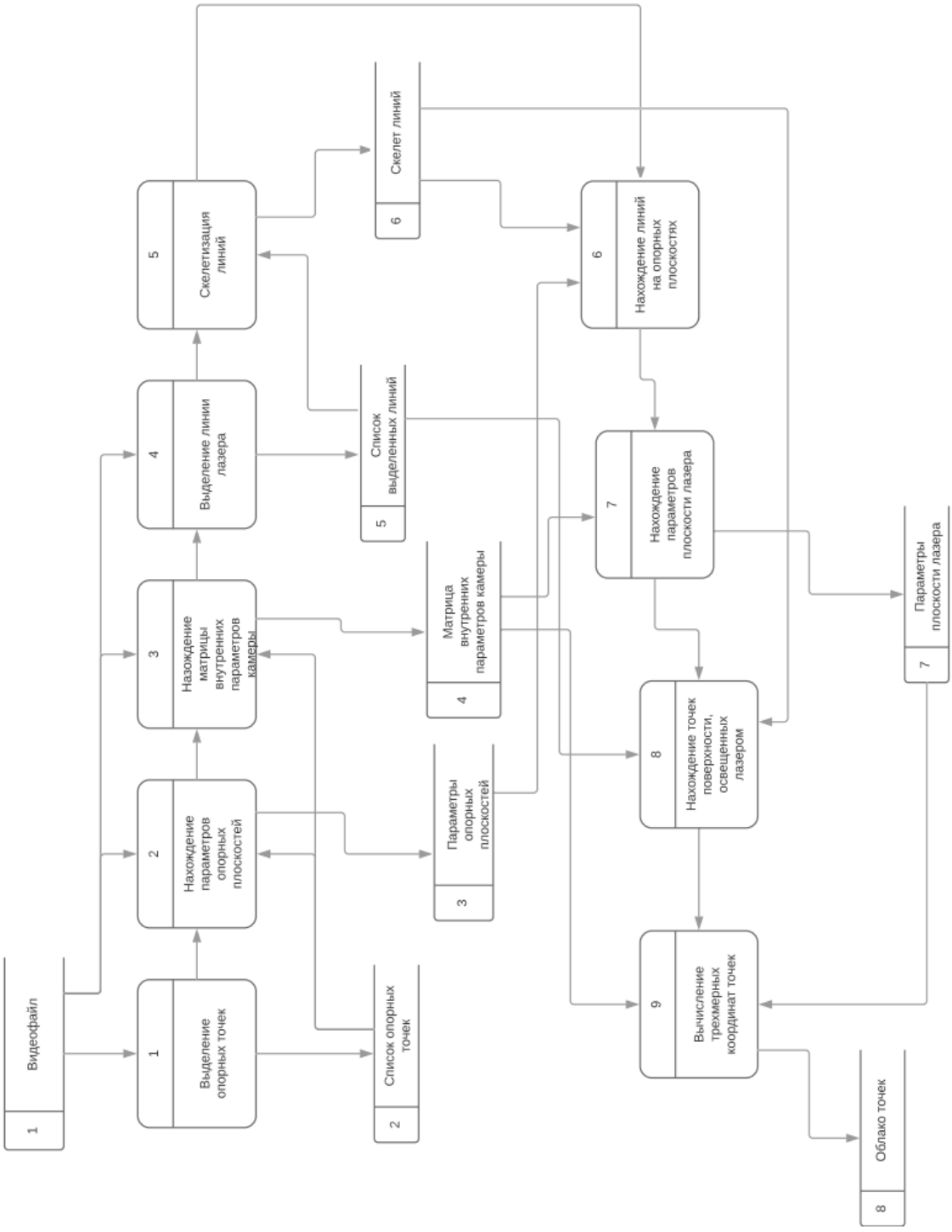

Рис. 17. Диаграмма потоков данных

1. Выделение опорных точек

Для выделения опорных точек на изображении нам поможет опорная рамка (рис. 7, рис. 16). На самой рамке отмечены некоторые точки. Пользователь выполняет этот подготовительный этап и выделяет точки на изображении.

2. Нахождение параметров опорных плоскостей.

Так как мы знаем размеры рамки, то нам не составит труда найти трехмерные координаты опорных точек. А затем по известным координатам определить коэффициенты опорных плоскостей. Как уже ранее было сказано, на каждой из сторон рамки приходится по четыре опорные точки. А для определения уравнения плоскости достаточно знать три точки. Более того, гарантируется, что по этим трём точкам можно найти уравнение плоскости, т.к. точки попарно различные и не лежат на одной прямой. Для нахождения коэффициентов плоскости составляется определитель:

$$\begin{vmatrix} x - x_0 & x_1 - x_0 & x_2 - x_0 \\ y - y_0 & y_1 - y_0 & y_2 - y_0 \\ z - z_0 & z_1 - z_0 & z_2 - z_0 \end{vmatrix} = 0, \qquad (28)$$

раскрывая который получим искомые коэффициенты.

Можно воспользоваться другим методом, менее вычислительно затратным. Заметим, что второй и третий столбец определителя являются координатами двух векторов, исходящие из одной точки (рис. 18). Т.е. если имеются точки $M_0(x_0, y_0, z_0)$, $M_1(x_1, y_1, z_1)$ и $M_2(x_2, y_2, z_2)$, то координаты векторов будут такими:

$\overline{M_0M_1}(x_1 - x_0,\ y_1 - y_0,\ z_1 - z_0)$ и $\overline{M_0M_2}(x_2 - x_0,\ y_2 - y_0,\ z_2 - z_0)$.

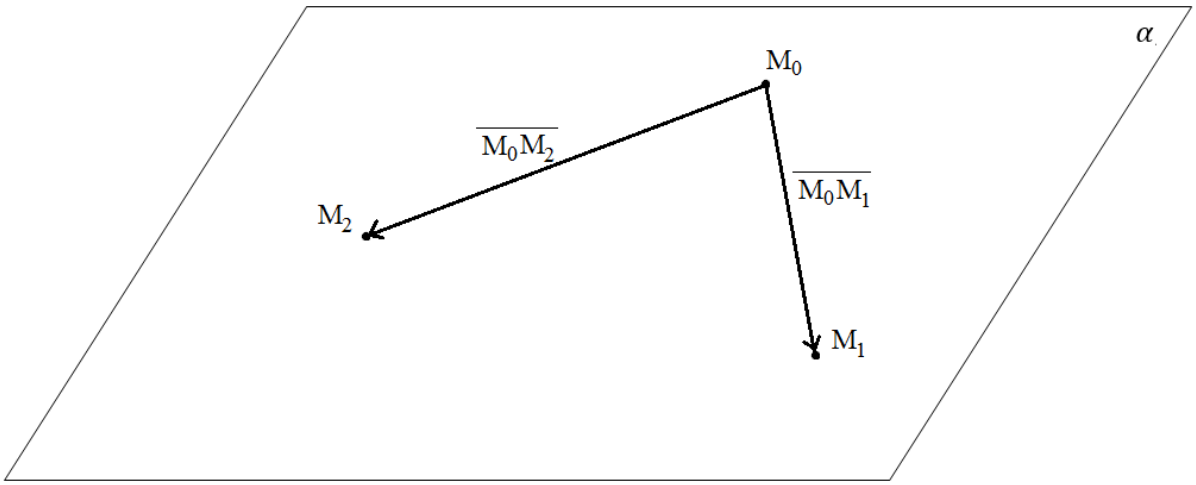

Рис. 18. Расположение векторов на плоскости

Для того чтобы найти коэффициенты плоскости, достаточно найти вектор нормали к этой плоскости. Будем искать вектор нормали по двум векторам $\bar{p} = \overline{M_0M_1}$ и $\bar{q} = \overline{M_0M_2}$, используя векторное произведение:

$$\bar{n} = \bar{p} * \bar{q} = \begin{vmatrix} i & j & k \\ p_x & p_y & p_z \\ q_x & q_y & q_z \end{vmatrix}, \qquad (29)$$

где i = (1, 0, 0), j = (0, 1, 0), k = (0, 0, 1).

То есть формулу можно переписать следующим образом:

$$\bar{n} = \bar{p} * \bar{q} = (p_y * q_z - p_z * q_y,\ p_z * q_x - p_x * q_z,\ p_x * q_y - p_y * q_x). \quad (30)$$

Собственно, для плоскости, выражаемой уравнением

$$Ax + By + Cz + D = 0. \qquad (31)$$

Коэффициенты A, B, C вычисляются следующим образом:

$$A = p_y * q_z - p_z * q_y, \qquad (32)$$

$$B = p_z * q_x - p_x * q_z, \qquad (33)$$

$$C = p_x * q_y - p_y * q_x. \qquad (34)$$

Коэффициент D вычисляется как сумма произведения коэффициентов вектора нормали с любой из рассматриваемых опорных точек:

$$D = -\sum_{i=1}^{3} \overline{n_i} * M_i. \qquad (35)$$

3. Нахождение матрицы внутренних параметров камеры

Определение внутренних параметров камеры было описано в пункте 3.6. На данном этапе алгоритма необходимо решить систему линейных уравнений:

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -x_2X_2 & -x_2Y_2 & -x_2Z_2 \\ & & & & & & & \vdots & & & \\ X_N & Y_N & Z_N & 1 & 0 & 0 & 0 & 0 & -x_NX_N & -x_NY_N & -x_NZ_N \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 & -y_2X_2 & -y_2Y_2 & -y_2Z_2 \\ & & & & & & & \vdots & & & \\ 0 & 0 & 0 & 0 & X_N & Y_N & Z_N & 1 & -y_NX_N & -y_NY_N & -y_NZ_N \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{24} \\ a_{41} \\ a_{42} \\ a_{43} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}. \qquad (36)$$

В которой неизвестным является вектор $a = (a_{11}, \dots, a_{43})^T$.

Система решается умножением на псевдообратную матрицу к матрице

$$G = \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -x_2X_2 & -x_2Y_2 & -x_2Z_2 \\ & & & & & & & \vdots & & & \\ X_N & Y_N & Z_N & 1 & 0 & 0 & 0 & 0 & -x_NX_N & -x_NY_N & -x_NZ_N \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 & -y_2X_2 & -y_2Y_2 & -y_2Z_2 \\ & & & & & & & \vdots & & & \\ 0 & 0 & 0 & 0 & X_N & Y_N & Z_N & 1 & -y_NX_N & -y_NY_N & -y_NZ_N \end{bmatrix}. \qquad (37)$$

Сначала находим G^T – транспонированную, а затем находим обратную матрицу к $(G^T * G)$. И тем самым домножив обе части на $(G^T * G)^{-1}$, найдем искомый вектор $a = (a_{11}, \dots, a_{43})^T$.

4. Выделение линии лазера.

Поскольку нас интересует искажения линии лазера, первым этапом необходимо уметь обнаруживать их. Для обнаружения линии лазера можно воспользоваться выделением по цвету. Будем считать, что лазер имеет красный цвет и будем обнаруживать точки, которые имеют самую большую красную компоненту в цветовой модели RGB. Задаём некоторый порог и для каждой точки изображения проверяем, превосходит ли красная компонента этот порог. Если да, то это и будет одна из точек линии лазера. Порог задается вручную и выбирается на усмотрение пользователя, если задать слишком низкий порог, то будут обнаружены точки, которые не относятся к линии лазера, если задать слишком высокий порог, некоторые из точек лазера могут быть не обнаружены. Этап выделения точек лазера повторяется для всех кадров видеоряда.

5. Скелетизация линии лазера.

После обнаружения точек лазера на каждом кадре мы знаем их точное расположение на изображении. Но поскольку сама линия лазера имеет ширину в несколько пикселей, могут возникнуть трудности с дальнейшими вычислениями. Поэтому воспользуемся алгоритмом скелетизации и выделим некий центр у линии лазера. Данная линия будет иметь ширину 1 пиксель, что значительно упростит расчёты.

Сам алгоритм скелетизации заключается в нахождении некоторых равноудаленных от границы фигуры точек. В нашем случае скелет линии – это и есть сама линия только тоньше.

В работе применяется алгоритм Зонга-Суня (Zhang-Suen) [11]. Данный метод состоит в последовательном удалении пикселей, начиная с границы. Алгоритм проверяет каждую точку по отдельности и принимает решение удалить её или оставить в зависимости от того будет ли удаление нарушать топологию фигуры или нет.

6. Нахождение линии лазера на опорных плоскостях

На данном этапе определяем координаты точек лазера на опорных плоскостях. Здесь нас интересуют коэффициенты прямой линии, которая проходит через найденные точки. Но точки могут не лежать на одной линии и быть разбросаны. Тогда можно воспользоваться методом наименьших квадратов.

Например, рассмотрим один кадр из видеоряда. На рис. 19 видно, что линия лазера располагается на левой и верхней опорных областях, и требуется узнать параметры этих линий.

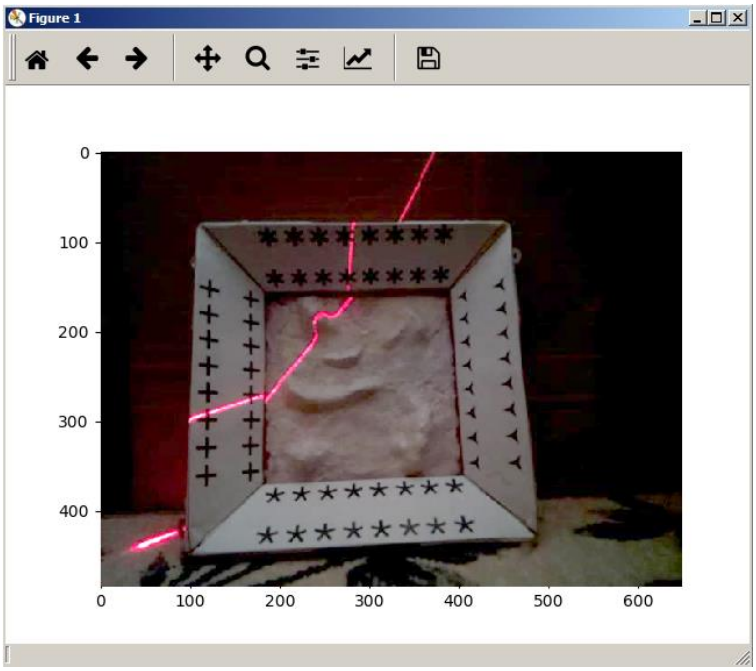

Рис. 19. Кадр из видеоряда

На этапе скелетизации линии лазера получается следующая картинка (рис. 20). Справа на рис. 20 желтым цветом выделены точки, по которым будет производиться линейная аппроксимация методом наименьших квадратов.

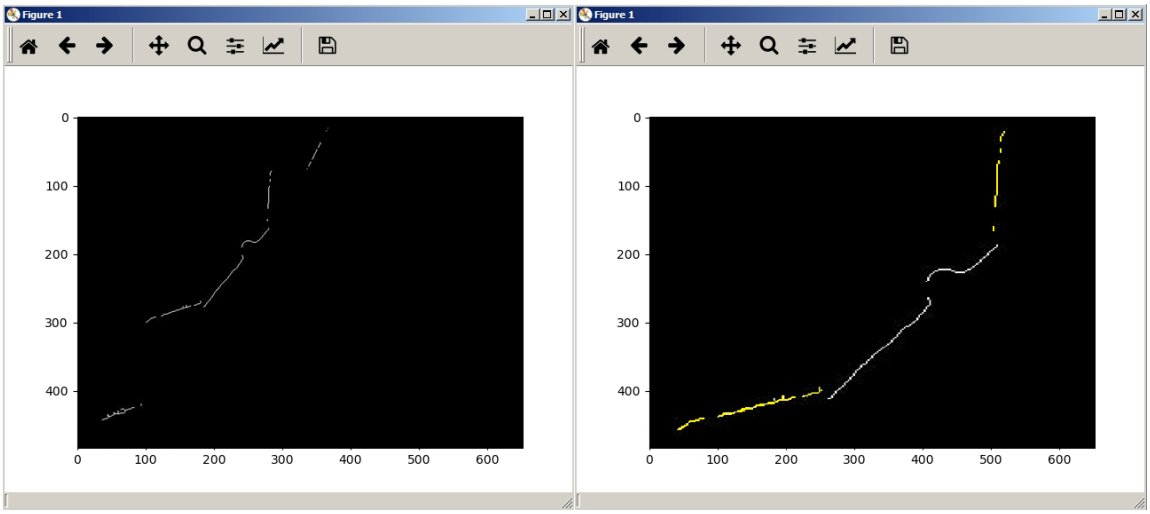

Рис. 20. Скелет линии лазера

Применяя МНК для набора точек на левой опорной области, получим следующие параметры прямой: a = 0.35, b = 22.89, для точек на верхней опорной области: a = 12.73, b = 711.65.

7. Нахождение параметров плоскости лазера

Теперь зная уравнение прямой линии лазера, уравнения опорных прямых, составленные по двум точкам, можем определить в каком месте линия лазера пересекает опорные прямые. После нахождения точек в двумерных координатах плоскости изображения находим трехмерные координаты. Составляется матрица из матрицы внутренних параметров камеры и коэффициентов опорной плоскости, затем составляем следующее соотношение:

$$\begin{bmatrix} a_{11} a_{12} a_{13} a_{14} \\ a_{21} a_{22} a_{23} a_{24} \\ a_{41} a_{42} a_{43} a_{44} \\ A_s \;\; B_s \;\; C_s \;\; D_s \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \\ 0 \end{bmatrix}, \qquad (38)$$

где $\begin{bmatrix} a_{11} a_{12} a_{13} a_{14} \\ a_{21} a_{22} a_{23} a_{24} \\ a_{41} a_{42} a_{43} a_{44} \end{bmatrix}$ – матрица внутренних параметров камеры;

A_s, B_s, C_s, D_s – коэффициенты опорной плоскости.

Откуда получаем:

$$X_{3D} = \frac{X}{w}, Y_{3D} = \frac{Y}{w}, Z_{3D} = \frac{Z}{w}. \qquad (39)$$

Для нахождения параметров плоскости лазера необходимо, чтобы линия лазера пересекала две любые опорные плоскости. Таким образом, будут известны четыре трёхмерные точки, которые являются пересечением линии лазера и опорных прямых на опорных плоскостях. Для получения коэффициентов плоскости, которую образует лазер достаточно взять три точки и провести те же вычисления, что и на этапе 2 алгоритма. Т.е. найти коэффициенты для уравнения $Ax + By + Cz + D = 0$:

$$A = p_y * q_z - p_z * q_y, \qquad (40)$$

$$B = p_z * q_x - p_x * q_z, \qquad (41)$$

$$C = p_x * q_y - p_y * q_x, \qquad (42)$$

$$D = -\sum_{i=1}^{3} \overline{n_i} * M_i. \qquad (43)$$

8. Нахождение точек, освещённых лазером на исследуемой области

На данном этапе происходит поиск точек лазера на сканируемой поверхности. Алгоритм для каждого кадра в видеоряде определяет двумерные координаты точек лазера.

9. Нахождение трёхмерных координат точек, освещённых лазером

Завершающий этап алгоритма по вычислению трёхмерных координат. Все результаты вычислений, происходившие на предыдущих шагах, потребуются здесь для вычисления 3D координат точек исследуемой поверхности. На этапе 7 алгоритма были вычислены коэффициенты плоскости, которую образует лазер, и также мы знаем двумерные координаты точек на сканируемой поверхности. Значит можем провести те же вычисления, что и в пункте 7 алгоритма. Формируем матрицу из матрицы внутренних параметров камеры и коэффициентов плоскости лазера:

$$\begin{bmatrix} a_{11}a_{12}a_{13}a_{14} \\ a_{21}a_{22}a_{23}a_{24} \\ a_{41}a_{42}a_{43}a_{44} \\ A_L\ B_L\ C_L\ D_L \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \\ 0 \end{bmatrix}. \qquad (44)$$

Откуда получаем:

$$X_L^{3D} = \frac{X}{w}, Y_L^{3D} = \frac{Y}{w}, Z_L^{3D} = \frac{Z}{w}. \qquad (45)$$

Так повторяется для каждого кадра видеоряда и формируется облако точек в 3D координатах, по которым можно восстановить трёхмерную форму.

3. Разработка приложения

3.1. Описание программной реализации

Программа реализована на языке программирования Python с использованием библиотеки обработки изображений skimage, библиотеки научных вычислений numpy и библиотеки matplotlib для отрисовки изображений.

Реализация шагов алгоритма выполнена в классе VideoAnalyzer в соответствующих методах (рис. 21).

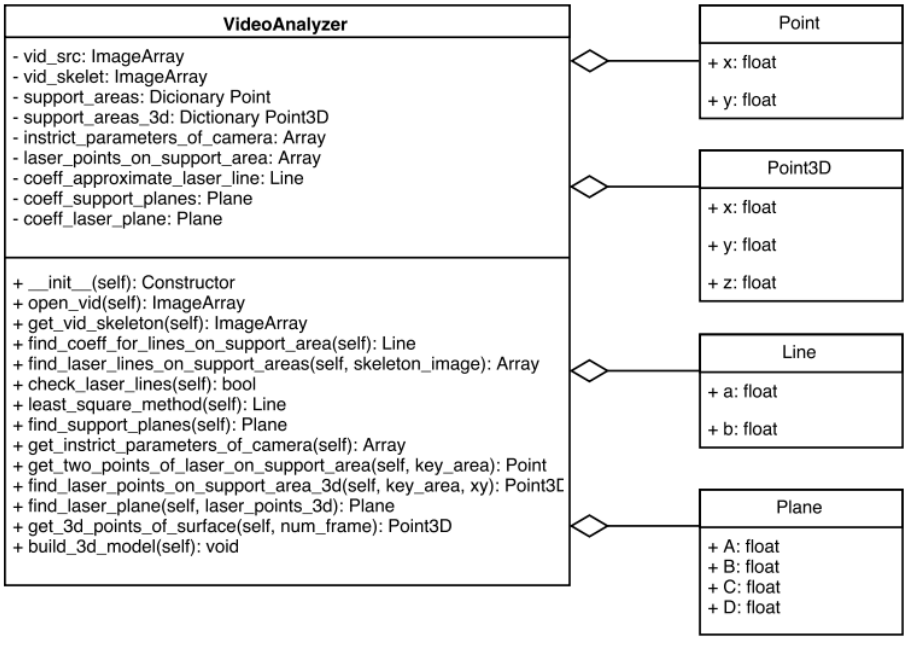

Рис. 21. Схема классов

Для хранения координат в программе реализованы следующие структуры: Point – точка в двумерном пространстве, Point3D – точка в трехмерном пространстве. Линии и плоскости представляются с помощью структур Line и Plane, которые хранят коэффициенты уравнения прямой и плоскости соответственно.

В таблицах 2 и 3 приведено краткое описание полей и методов класса.

Таблица 2

Поле класса	Описание
vid_src	Исходный видеофайл
vid_skelet	Скелет лазера на каждом кадре
support_areas	Точки опорных плоскостей в 2D
support_areas_3d	Точки опорных плоскостей в 3D
instrict_parameters_of_camera	Матрица внутренних параметров камеры
laser_points_on_support_area	Список точек лазера на опорных плоскостях
coeff_approximate_laser_line	Параметры линии лазера на опорных плоскостях
coeff_support_planes	Параметры опорных плоскостей
coeff_laser_plane	Параметры плоскости лазера

Таблица 3

Метод класса	Описание
__init__()	Конструктор, начальная инициализация.
open_vid()	Производит чтение видеофайла
get_vid_skeleton()	Получает скелет линии лазера на каждом кадре
find_coeff_for_lines_on_support_area()	Находит параметры линий опорных областей
find_laser_lines_on_support_areas()	Находит линию лазера на опорных областях
check_laser_lines()	Проверяет есть ли в кадре две линии лазер на опорных областях
least_square_method()	Аппроксимирует линейной функцией точки лазера на опорных областях
find_support_planes()	Находит параметры опорных плоскостей
get_instrict_parameters_of_camera()	Находит матрицу внутренних параметров камеры
get_two_points_of_laser_on_support_area()	Выделяет две точки на опорной области
find_laser_points_on_support_area_3d()	Находит 3D координаты точек на опорной области

find_laser_plane()	Находит параметры плоскости лазера
get_3d_points_of_surface()	Получает 3D координаты точек исследуемой поверхности
build_3d_model()	Строит 3D модель поверхности

3.2. Результаты работы программы

В этом разделе представлены результаты работы программы. Программа анализирует видеофайл кадр за кадром. На рис. 22 представлен один кадр из видеоряда. На данном рисунке показано как программа определяет линию лазера на опорных областях рамки.

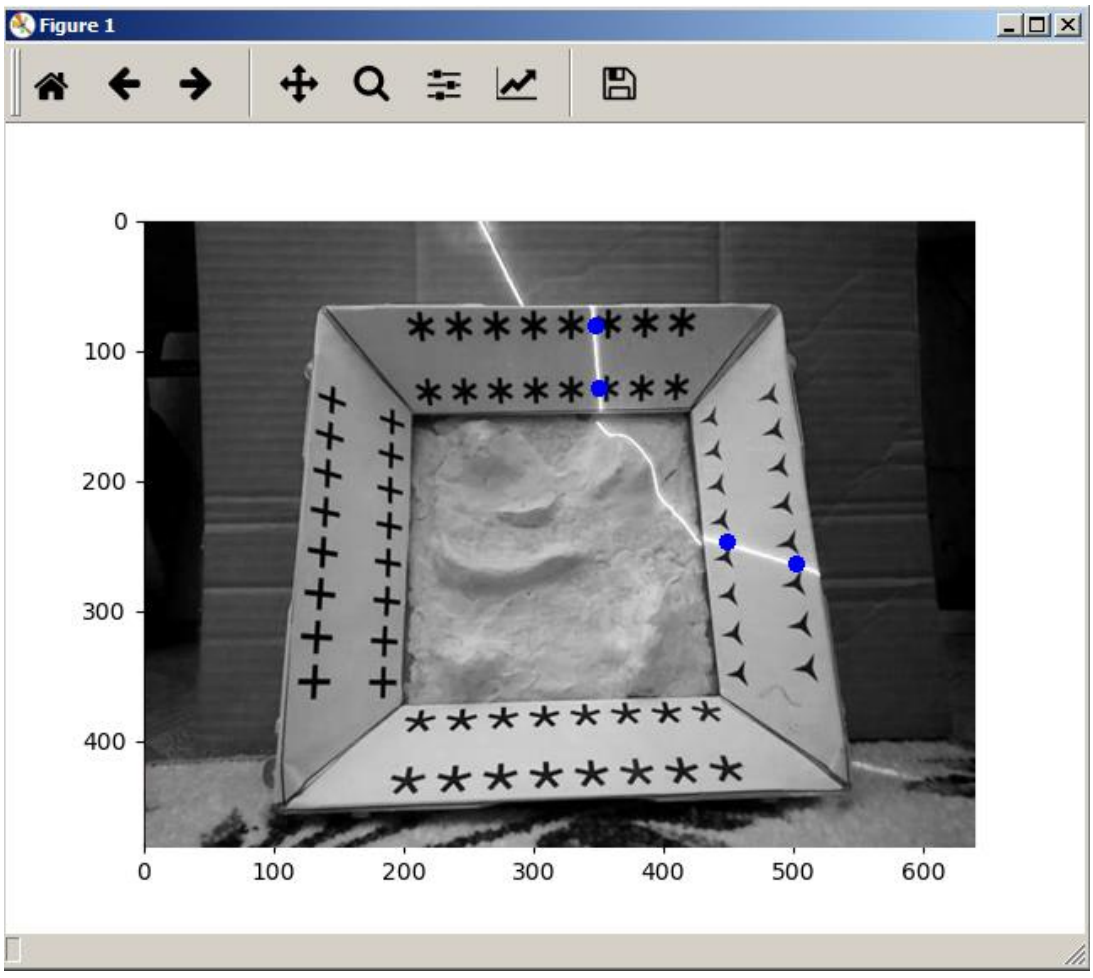

Рис. 22. Определение линии лазера на опорных областях

Затем, алгоритм для каждого кадра выделяет скелет линии лазера (рис. 23).

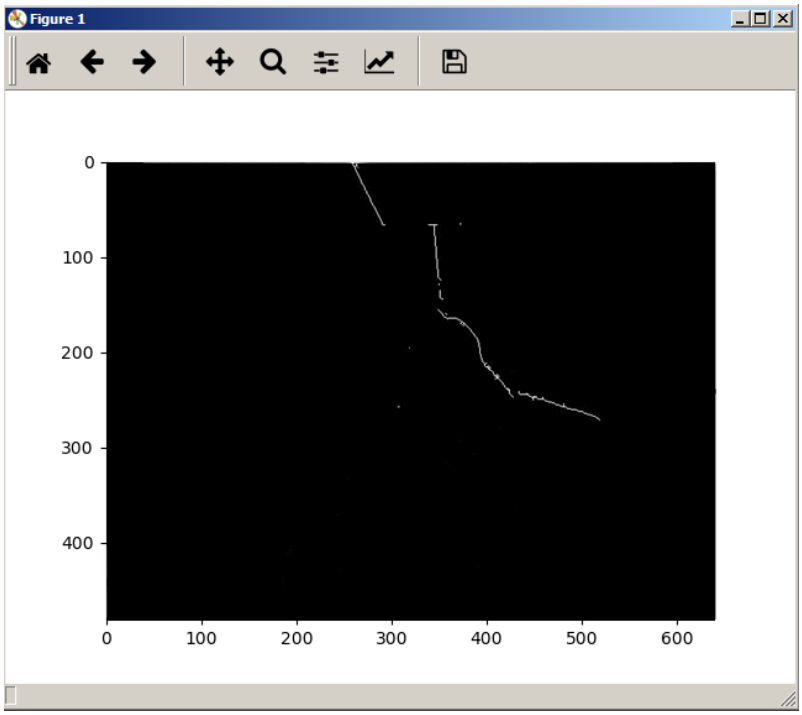

Рис. 23. Выделение скелета линии лазера

После всех расчётов программа строит трехмерную модель поверхности (рис. 24).

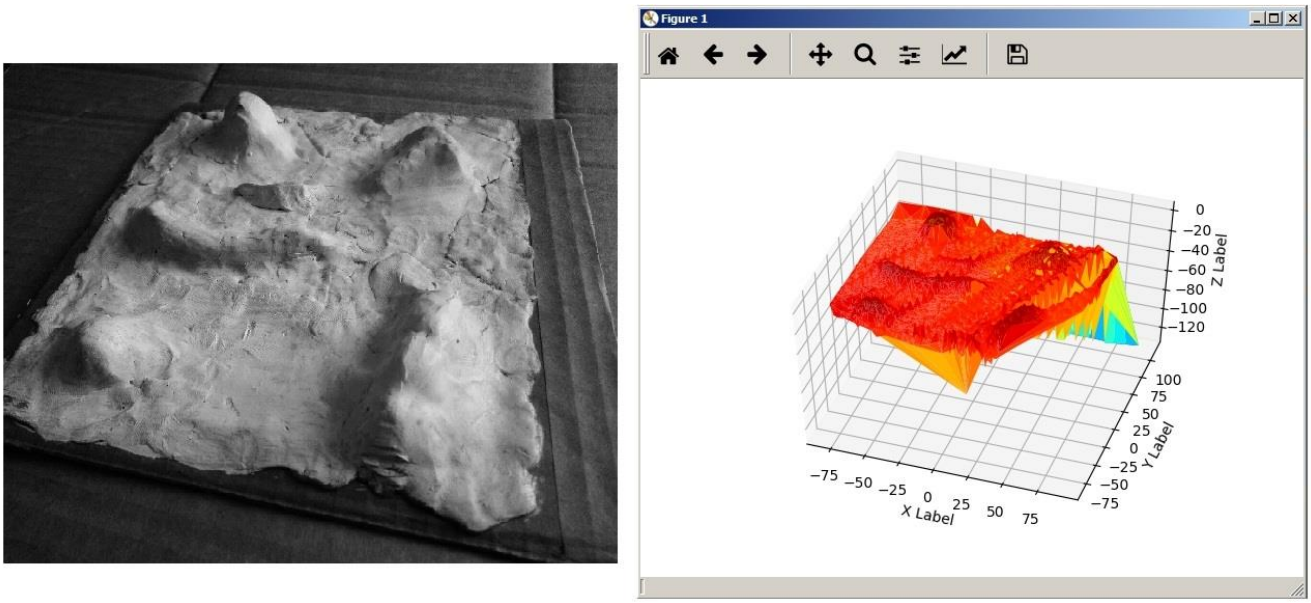

Рис. 24. Сравнение настоящей поверхности с построенной моделью

Заключение

В ходе исследования была изучена предметная область, поставлена и сформулирована задача, намечены пути решения поставленных задач. Были изучены инструменты для работы с видеорядом. Реализован алгоритм выделения линии лазера. Были изучены теория о связи различных систем координат, их преобразованиях из одной системы координат в другую. Был реализован алгоритм нахождения матрицы внутренних параметров камеры, алгоритм нахождения уравнения плоскости положения лазера, алгоритм определения координат точек в трехмерном пространстве по двумерным координатам точек, освещенных лазером. Разработано приложение, которое на вход получает видеофайл, обрабатывает его, и, используя геометрические данные рамки, строит трехмерную модель исследуемой поверхности.

Список использованной литературы

[1] Все о 3D-сканнерах: от разновидностей до применения // Персональный сайт [Электронный ресурс]. – Электрон. дан. – Режим доступа: http://can-touch.ru/blog/vse-o-3d-skanerax/

[2] Исакина Т. В. Магистерская диссертация: «Алгоритмы реконструкции трехмерных поверхностей по изображению», Тверь, 2014.

[3] Трехмерная реконструкция по одному изображению. / Антон Конушин, Вадим Конушин, Ольга Баринова и др. // Курс – «Введение в компьютерное зрение», МГУ ВМК, Graphics & Media Lab. – 2009. – 89 с.

[4] Цифровая обработка изображений в информационных системах: учебное пособие / И.С. Грузман, В.С. Киричук, В.П. Косых и др. – Новосибирск: НГТУ. – 2002. – 168 с.

[5] Ajmal Mian, Mehdi Ravanbakhsh, Lecture 08 – Camera Calibration. – CITS 4402 Computer Vision, The University Of Western Australia [Электронный ресурс]. – Электрон. дан. – Режим доступа: http://teaching.csse.uwa.edu.au/units/CITS4402/lectures/Lecture08-CameraCalibration.pdf.

[6] Henrik Aanæs, Lecture Notes on Computer Vision. – DTU informatics – 2009.

[7] Mubarak Shah, Fundamental of computer vision. – Computer Science Department University of Central Florida, Orlando, December 7, 1997. – 133c.

[8] Peter Sturm, Some lecture notes on Geometric Computer Vision. – INRIA Grenoble - Rhône-Alpes. – 2015.

[9] Robert Collins, Lecture 12: Camera Projection. – CSE486, Penn State [Электронный ресурс]. – Электрон. дан. – Режим доступа: http://www.cse.psu.edu/~rtc12/CSE486/lecture12.pdf

[10] Robert Collins, Lecture 13: Camera Projection II. – CSE486, Penn State [Электронный ресурс]. – Электрон. дан. – Режим доступа: http://www.cse.psu.edu/~rtc12/CSE486/lecture13.pdf

[11] T. Y. Zhang and C. Y. Suen. A fast parallel algorithm for thinning digital patterns, Communications of the ACM, March 1984, Volume 27, Number 3.

[12] Wilhelm Burger, Zhang's Camera Calibration Algorithm: In-Depth Tutorial and Implementation. – Department of Digital Media, University of Applied Sciences Upper Austria, School of Informatics. – 2016.

Приложение

```
# -*- coding: utf-8 -*-

import sys
import imageio      # Для чтения видео
import numpy as np
from numpy import linalg    # Для решение СЛУ
from skimage.morphology import skeletonize      # Для скелетизации линии лазера
from PIL import Image, ImageDraw                # Для рисования на изображении
from PIL.ImageQt import ImageQt
from PyQt5.QtWidgets import QApplication, QMainWindow, QFileDialog
from PyQt5.QtGui import QPixmap, QImage, QMouseEvent
from PyQt5 import uic

class VideoAnalyzer(QMainWindow):

    VID_SIZE = (640, 480)       # Работает с видео размерами 640x480
    BREAK_FRAME = 1000    # Ограничение по кадрам
    THRESHOLD = 150           # Порог бинаризации

    RED = 0     # Красная компонента в изображении,
            # представленном массивом: image[row][pix][color]
            # или
            # [[[RED, G, B], [RED, G, B] ... ], ..., [[RED, G, B], [RED, G, B] ...]]

    # Изображение представляется массивом image[row][pix][color]
    # Последовательность изображений: vid = [image_1, ..., image_N]
    vid_src = []          # Исходная последовательность изображений
    vid_skelet = []       # Последовательность изображений после скелетизации

    # Файл с 3D точками
    file_with_3d_points = open('3d_points.txt', 'w')

    # Опорные области представляются в виде 4-х точек (двумерных)
    # support_area:
    #             key: 'top_left_point' - верхняя левая точка
    #                  'top_right_point' - верхняя правая точка
    #                  'bottom_left_point' - нижняя левая точка
    #                  'bottom_right_point' - нижняя правая точка
    #             value: {x, y}
    # Координаты {x, y} записаны в системе координат изображения.
    support_area_top = {
                 'top_left_point':     {'x': 0, 'y': 0},
                 'top_right_point':    {'x': 0, 'y': 0},
                 'bottom_left_point':  {'x': 0, 'y': 0},
                 'bottom_right_point': {'x': 0, 'y': 0},
                }

    support_area_bottom = {
                 'top_left_point':     {'x': 0, 'y': 0},
                 'top_right_point':    {'x': 0, 'y': 0},
```

```
            'bottom_left_point':  {'x': 0, 'y': 0},
            'bottom_right_point': {'x': 0, 'y': 0},
            }

support_area_left = {
            'top_left_point':     {'x': 0, 'y': 0},
            'top_right_point':    {'x': 0, 'y': 0},
            'bottom_left_point':  {'x': 0, 'y': 0},
            'bottom_right_point': {'x': 0, 'y': 0},
            }

support_area_right = {
            'top_left_point':     {'x': 0, 'y': 0},
            'top_right_point':    {'x': 0, 'y': 0},
            'bottom_left_point':  {'x': 0, 'y': 0},
            'bottom_right_point': {'x': 0, 'y': 0},
            }

support_areas = {
            'top_area':    support_area_top,
            'bottom_area': support_area_bottom,
            'left_area':   support_area_left,
            'right_area':  support_area_right,
            }

# Коэффициенты прямых линий, которые образуют опорную область
#       key: 'top_area' - верхняя область
#            'bottom_area' - нижняя область
#            'left_area' - левая область
#            'right_area' - правая область
#       value:
#           key:    'top_line' - верхняя линия
#                   'bottom_line' - нижняя линия
#                   'left_line' - левая линия
#                   'right_line' - правая линия
#           value: {a, b}
coeff_lines_support_area = {
                    'top_area': {
                        'top_line':    {'a': 0, 'b': 0},
                        'bottom_line': {'a': 0, 'b': 0},
                        'left_line':   {'a': 0, 'b': 0},
                        'right_line':  {'a': 0, 'b': 0},
                        },
                    'bottom_area': {
                        'top_line':    {'a': 0, 'b': 0},
                        'bottom_line': {'a': 0, 'b': 0},
                        'left_line':   {'a': 0, 'b': 0},
                        'right_line':  {'a': 0, 'b': 0},
                        },
                    'left_area': {
                        'top_line':    {'a': 0, 'b': 0},
                        'bottom_line': {'a': 0, 'b': 0},
```

```
                    'left_line':   {'a': 0, 'b': 0},
                    'right_line':  {'a': 0, 'b': 0},
                    },
                'right_area': {
                    'top_line':    {'a': 0, 'b': 0},
                    'bottom_line': {'a': 0, 'b': 0},
                    'left_line':   {'a': 0, 'b': 0},
                    'right_line':  {'a': 0, 'b': 0},
                    },
                }

# Координаты опорных областей в трехмерных координатах
# support_area_3d:
#           key: 'top_left_point' - верхняя левая точка
#                'top_right_point' - верхняя правая точка
#                'bottom_left_point' - нижняя левая точка
#                'bottom_right_point' - нижняя правая точка
#           value: {x, y, z}
# Координаты {x, y, z} записаны в системе координат рамки (не изображения).
# Начало координат находится в центре рамки.
support_area_top_3d = {
            'top_left_point':     {'x': -69.7, 'y': 113.541, 'z': 25.1},
            'top_right_point':    {'x': 69.7,  'y': 114.248, 'z': 25.1},
            'bottom_left_point':  {'x': -69.7, 'y': 88.085,  'z': 0},
            'bottom_right_point': {'x': 69.7,  'y': 88.578,  'z': 0},
            }

support_area_bottom_3d = {
            'top_left_point':     {'x': -69.7, 'y': -88.085,  'z': 0},
            'top_right_point':    {'x': 69.7,  'y': -87.378,  'z': 0},
            'bottom_left_point':  {'x': -69.7, 'y': -113.541, 'z': 25.1},
            'bottom_right_point': {'x': 69.7,  'y': -112.48, 'z': 25.1},
            }

support_area_left_3d = {
            'top_left_point':     {'x': -112.48,  'y': 69.7,  'z': 25.1},
            'top_right_point':    {'x': -87.378,  'y': 69.7,  'z': 0},
            'bottom_left_point':  {'x': -113.541, 'y': -69.7, 'z': 25.1},
            'bottom_right_point': {'x': -88.085,  'y': -69.7, 'z': 0},
            }

support_area_right_3d = {
            'top_left_point':     {'x': 88.58,   'y': 69.7,  'z': 0},
            'top_right_point':    {'x': 114.248, 'y': 69.7,  'z': 25.1},
            'bottom_left_point':  {'x': 87.378,  'y': -69.7, 'z': 0},
            'bottom_right_point': {'x': 112.48,  'y': -69.7, 'z': 25.1},
            }

support_areas_3d = {
            'top_area':    support_area_top_3d,
            'bottom_area': support_area_bottom_3d,
            'left_area':   support_area_left_3d,
```

```
                'right_area':  support_area_right_3d,
               }

# Точки линии лазера на опорных областях
# key: 'top_area'
#      'bottom_area'
#      'left_area'
#      'right_area'
# value: [[x0, y0], ..., [xN, yN]]
laser_points_on_support_area = {
                            'top_area':    [],
                            'bottom_area': [],
                            'left_area':   [],
                            'right_area':  [],
                           }

# Коэффициенты линии лазера.
# Аппроксимированных по МНК.
# key: 'top_area'
#      'bottom_area'
#      'left_area'
#      'right_area'
# value: {a, b}
coeff_approximate_laser_line = {
                            'top_area':    {'a': 0, 'b': 0},
                            'bottom_area': {'a': 0, 'b': 0},
                            'left_area':   {'a': 0, 'b': 0},
                            'right_area':  {'a': 0, 'b': 0},
                            }


# Коэффициенты опорных плоскостей
coeff_support_planes = {
                    'top_area':    {'A': 0, 'B': 0, 'C': 0, 'D': 0},
                    'bottom_area': {'A': 0, 'B': 0, 'C': 0, 'D': 0},
                    'left_area':   {'A': 0, 'B': 0, 'C': 0, 'D': 0},
                    'right_area':  {'A': 0, 'B': 0, 'C': 0, 'D': 0},
                    }

# Матрица внутренних параметров камеры
instrict_parameters_of_camera = []


# Коэффициенты плоскости лазера
coeff_laser_plane = {'A': 0, 'B': 0, 'C': 0, 'D': 0}

#----------------Методы----------------------

def __init__(self):
    '''
    Конструктор.
    '''
```

```
        super().__init__()
        uic.loadUi('ui/mainwindow.ui', self)

        self.move(70, 70)
        self.setFixedSize(1200, 680)

        self.COUNT_POINTS = 16
        self.curr_count_points = 0

        # Вешаем события на мышь
        self.canvas_lbl.mousePressEvent = self.get_coord_support_points

        # Соединяем элементы UI с функциями
        self.start_btn.clicked.connect(self.run)
        self.open_vid_file_act.triggered.connect(self.open_vid)

    #-----------------------------------------------------------------------------------
    def get_coord_support_points(self, event):
        '''
        Получает позицию курсора на канвасе и записывает координаты.

        Последовательность выбора точек жестко зашита:
            1. Верхняя область
                1.1. Верхняя левая точка
                1.2. Верхняя правая точка
                1.3. Нижняя левая точка
                1.4. Нижняя правая точка
            2. Нижняя область
                2.1-2.4. аналогично
            3. Левая область
                3.1-3.4. аналогично
            4. Правая область
                4.1-4.4. аналогично
        '''
        if self.curr_count_points < self.COUNT_POINTS:
            # Просто константы
            sa = ['top_area', 'bottom_area', 'left_area', 'right_area']
            sp = ['top_left_point', 'top_right_point', 'bottom_left_point', 'bottom_right_point']

            x, y = event.pos().x(), event.pos().y()

            self.teOutput.append('x: ' + str(x) + ' y: ' + str(y))
            if self.curr_count_points != self.COUNT_POINTS - 1:
                self.teOutput.append(sa[(self.curr_count_points + 1) // 4] + ' ' +
sp[(self.curr_count_points + 1) % 4] + ':')

            x, y = self.to_other_coordinates_2d(x, y)

            self.support_areas[sa[self.curr_count_points // 4]][sp[self.curr_count_points % 4]]['x'] =
x
            self.support_areas[sa[self.curr_count_points // 4]][sp[self.curr_count_points % 4]]['y'] =
y
```

```
            self.curr_count_points += 1

    #-----------------------------------------------------------------------------
    def open_vid(self):
        '''
        Открывает видеофайл.

        Работает в Anaconda3 ver.4.3.0.1 для win-x86
        А также дополнительно установленным ffmpeg:
            conda install ffmpeg -c conda-forge
            imageio.plugins.ffmpeg.download() - хотя работает без этого
                                        возможно из-за того, что вместе с этим файлом лежит
ffmpeg.exe
        '''
        filename, ok = QFileDialog.getOpenFileName(self, "Open Video File", "", "Avi files
(*.avi);;Mp4 files (*.mp4);;All files (*.*)")

        if ok:
            self.vid_src = imageio.get_reader(filename,  'ffmpeg')

            temp_img = self.vid_src.get_data(50)
            imageio.imwrite('__temp.jpg', temp_img)
            self.canvas_lbl.setPixmap(QPixmap('__temp.jpg'))

            self.teOutput.append('Select points:')
            self.teOutput.append('top_area top_left_point:')

    def get_vid_skeleton(self):
        '''
        Скелетизация каждого кадра.
        '''
        self.progressBar.reset()
        self.progressBar.setMaximum(self.BREAK_FRAME)
        for i, frame in enumerate(self.vid_src):
            if i == self.BREAK_FRAME:
                break
            frame_red = frame[:, :, self.RED]       # Оставляет только красную компоненту на
изображении
            frame_bin = frame_red > self.THRESHOLD  # Бинаризация изображения
            frame_skelet = skeletonize(frame_bin)   # Скелетизация изображения

            self.vid_skelet.append(frame_skelet)
            self.progressBar.setValue(i)

    #-----------------------------------------------------------------------------

    #----------------Методы для нахождения коэффициентов линий опорных областей----------
--
    def get_coeff_line(self, x1, y1, x2, y2):
        '''
        Получает коэффициенты прямой по двум точкам.
```

```
    ax + by + c = 0 => a = y1 - y2, b = x2 - x1, c = x1 * y2 - x2 * y1;
    y = px + q      => p = -a / b, q = -c / b;
    '''
    a = y1 - y2
    b = x2 - x1
    c = x1 * y2 - x2 * y1

    try:
        p = -a / b
        q = -c / b
    except:
        p = -a / (b + 1)
        q = -c / (b + 1)

    return {'a': p, 'b': q}

def find_coeff_for_lines_on_support_area(self):
    '''
    Находит коэффициенты прямых линий, которые образуют опорную область.
    '''
    for key_area, supp_area in self.support_areas.items():
        x1 = supp_area['top_left_point']['x']
        y1 = supp_area['top_left_point']['y']

        x2 = supp_area['top_right_point']['x']
        y2 = supp_area['top_right_point']['y']

        x3 = supp_area['bottom_left_point']['x']
        y3 = supp_area['bottom_left_point']['y']

        x4 = supp_area['bottom_right_point']['x']
        y4 = supp_area['bottom_right_point']['y']

        top_line    = self.get_coeff_line(x1, y1, x2, y2)
        bottom_line = self.get_coeff_line(x3, y3, x4, y4)
        left_line   = self.get_coeff_line(x1, y1, x3, y3)
        right_line  = self.get_coeff_line(x2, y2, x4, y4)

        self.coeff_lines_support_area[key_area]['top_line']['a'] = top_line['a']
        self.coeff_lines_support_area[key_area]['top_line']['b'] = top_line['b']

        self.coeff_lines_support_area[key_area]['bottom_line']['a'] = bottom_line['a']
        self.coeff_lines_support_area[key_area]['bottom_line']['b'] = bottom_line['b']

        self.coeff_lines_support_area[key_area]['left_line']['a'] = left_line['a']
        self.coeff_lines_support_area[key_area]['left_line']['b'] = left_line['b']

        self.coeff_lines_support_area[key_area]['right_line']['a'] = right_line['a']
        self.coeff_lines_support_area[key_area]['right_line']['b'] = right_line['b']

        self.teOutput.append(key_area)
```

```
        self.teOutput.append('top_line: ' + 'a: ' + str(round(top_line['a'], 3)) + ' b: ' +
str(round(top_line['b'], 3)))
        self.teOutput.append('bottom_line: ' + 'a: ' + str(round(bottom_line['a'], 3)) + ' b: ' +
str(round(bottom_line['b'], 3)))
        self.teOutput.append('left_line: ' + 'a: ' + str(round(left_line['a'], 3)) + ' b: ' +
str(round(left_line['b'], 3)))
        self.teOutput.append('right_line: ' + 'a: ' + str(round(right_line['a'], 3)) + ' b: ' +
str(round(right_line['b'], 3)))

    #-----------------------------------------------------------------------------

    #-----------------Методы для обнаружения линии лазера на опорных областях----------
    def find_min_max_points(self, support_area):
        '''
        Находит минимальные и максимальные координаты точек опорной области.
        '''
        x_min = 0xffff
        y_min = 0xffff
        x_max = 0
        y_max = 0

        for _, xy in support_area.items():
            x_, y_ = self.to_default_coordinates_2d(xy['x'], xy['y'])
            if x_ < x_min:
                x_min = x_

            if x_ >= x_max:
                x_max = x_

            if y_ < y_min:
                y_min = y_

            if y_ >= y_max:
                y_max = y_

        return {'x_min': x_min, 'y_min': y_min,
                'x_max': x_max, 'y_max': y_max,}

    def find_laser_lines_on_support_areas(self, skeleton_image):
        '''
        Находит на опорных областях линию лазера.
        '''
        self.laser_points_on_support_area = {
                                        'top_area': [],
                                        'bottom_area': [],
                                        'left_area': [],
                                        'right_area': [],
                                        }

        for key_area, supp_area in self.support_areas.items():
            min_max = self.find_min_max_points(supp_area)
```

```
        for y in range(min_max['y_min'], min_max['y_max']):
            for x in range(min_max['x_min'], min_max['x_max']):
                try:
                    if skeleton_image[y][x] == True:
                        x_, y_ = self.to_other_coordinates_2d(x, y)
                        self.laser_points_on_support_area[key_area].append([x_, y_])
                except:
                    continue

#-------------------------------------------------------------------------------

#------------Метод наименьших квадратов для линий на опорных областях------------

def check_laser_lines(self):
    '''
    Проверяет есть ли в кадре две линии лазера, которые находятся на опорных областях.
    '''
    count_lines = 0
    for key_area, arr in self.laser_points_on_support_area.items():
        # Считаем, что линия состоит более, чем из 10 точек
        if len(arr) > 10:
            count_lines += 1

    if count_lines < 2:
        return None

    else:
        # Выбрать две линии с наибольшим количеством точек
        result = {}
        for _ in range(2):
            k_max = ''
            v_max = []
            for k, v in self.laser_points_on_support_area.items():
                if len(v) > len(v_max) and k not in result:
                    k_max = k
                    v_max = v

            result[k_max] = v_max

        return result

def least_square_method(self):
    '''
    Метод наименьших квадратов.
    '''
    points = self.check_laser_lines()
    if points is None:
        return None

    for key_area, v in points.items():
        n = len(v)
        x_sum = 0
```

```
        y_sum = 0
        xy_sum = 0
        x2_sum = 0

        for x, y in v:
            x_sum += x
            y_sum += y
            xy_sum += (x*y)
            x2_sum += (x**2)

        try:
            a = (n * xy_sum - x_sum * y_sum) / (n * x2_sum - x_sum**2)
            b = (y_sum - a * x_sum) / n
        except:
            a = 0
            b = 0

        self.coeff_approximate_laser_line[key_area]['a'] = a
        self.coeff_approximate_laser_line[key_area]['b'] = b

    return points

#------------------------------------------------------------------------

def find_support_planes(self):
    '''
    Находит коэффициенты A, B, C, D опорных плоскостей.
    '''
    for key_area, val in self.coeff_support_planes.items():
        x1 = self.support_areas_3d[key_area]['top_left_point']['x']
        y1 = self.support_areas_3d[key_area]['top_left_point']['y']
        z1 = self.support_areas_3d[key_area]['top_left_point']['z']

        x2 = self.support_areas_3d[key_area]['top_right_point']['x']
        y2 = self.support_areas_3d[key_area]['top_right_point']['y']
        z2 = self.support_areas_3d[key_area]['top_right_point']['z']

        x3 = self.support_areas_3d[key_area]['bottom_left_point']['x']
        y3 = self.support_areas_3d[key_area]['bottom_left_point']['y']
        z3 = self.support_areas_3d[key_area]['bottom_left_point']['z']

        p1 = np.array([x1, y1, z1])
        p2 = np.array([x2, y2, z2])
        p3 = np.array([x3, y3, z3])

        # Вектора на плоскости
        v1 = p1 - p2
        v2 = p1 - p3

        # Вектор нормали плоскости
        nv = np.cross(v1, v2)
        a, b, c = nv
```

```
        d = -np.dot(nv, p3)

        self.coeff_support_planes[key_area]['A'] = a
        self.coeff_support_planes[key_area]['B'] = b
        self.coeff_support_planes[key_area]['C'] = c
        self.coeff_support_planes[key_area]['D'] = d

        self.teOutput.append(key_area + ': A: ' + str(round(a, 3)) + ' B: ' + str(round(b, 3)) + ' C: '
+ str(round(c, 3)) + ' D: ' + str(round(d, 3)))

    def get_instrict_parameters_of_camera(self):
        '''
        Находит внутренние параметры камеры.

        [x0_3d, y0_3d, z0_3d, 1, 0, 0, 0, 0, -x0_2d * x0_3d, -x0_2d * y0_3d, -x0_2d * z0_3d] [r0]
[x0_2d]
        [0, 0, 0, 0, x0_3d, y0_3d, z0_3d, 1, -y0_2d * x0_3d, -y0_2d * y0_3d, -y0_2d * z0_3d] [r1]
[y0_2d]
        .                       .                       .       .
        .                       .                       .   =   .
        .                       .                       .       .
        [xN_3d, yN_3d, zN_3d, 1, 0, 0, 0, 0, -xN_2d * xN_3d, -xN_2d * yN_3d, -xN_2d * zN_3d]
[rN-1]    [xN_2d]
        [0, 0, 0, 0, xN_3d, yN_3d, zN_3d, 1, -yN_2d * xN_3d, -yN_2d * yN_3d, -yN_2d * zN_3d]
[rN]      [yN_2d]

        Вектор r - внутренние параметры камеры имеет размерность (N+1), но чтобы решить
данную систему
                нужно взять N элементов, а N+1 элемент принять за единицу.

        Система решается следующим образом:
        D * R = B <=> D^T * D * R = D^T * B <=> R = (D^T * D)^-1 * D^T * B, где D^T -
транспонированная матрица, D^-1 - обратная матрица.

        '''
        matrix_a = []
        matrix_b = []

        # Подготовка и заполнение данными
        for key_area, sa in self.support_areas_3d.items():
            for key_point, arr in sa.items():
                x_3d = arr['x']
                y_3d = arr['y']
                z_3d = arr['z']

                x_2d = self.support_areas[key_area][key_point]['x']
                y_2d = self.support_areas[key_area][key_point]['y']

                matrix_a.append([x_3d, y_3d, z_3d, 1, 0, 0, 0, 0, -x_2d * x_3d, -x_2d * y_3d, -x_2d *
z_3d])
```

```
            matrix_a.append([0, 0, 0, 0, x_3d, y_3d, z_3d, 1, -y_2d * x_3d, -y_2d * y_3d, -y_2d *
z_3d])

            matrix_b.append(x_2d)
            matrix_b.append(y_2d)

        a = np.array(matrix_a)
        b = np.array(matrix_b)

        # Решение системы линейных уравнений
        a_tr = a.transpose()
        a = np.matmul(a_tr, a)
        a_inv = linalg.inv(a)

        r = np.matmul(a_inv, np.matmul(a_tr, b))

        self.instrict_parameters_of_camera = r.tolist()
        self.instrict_parameters_of_camera.append(1.0)

    def get_two_points_of_laser_on_support_area(self, key_area):
        '''
        Получает две точки на опорной плоскости.
        '''
        laser_line_a = self.coeff_approximate_laser_line[key_area]['a']
        laser_line_b = self.coeff_approximate_laser_line[key_area]['b']

        if key_area == 'top_area' or key_area == 'bottom_area':
            top_line_a = self.coeff_lines_support_area[key_area]['top_line']['a']
            top_line_b = self.coeff_lines_support_area[key_area]['top_line']['b']

            bottom_line_a = self.coeff_lines_support_area[key_area]['bottom_line']['a']
            bottom_line_b = self.coeff_lines_support_area[key_area]['bottom_line']['b']

            try:
                top_x = (top_line_b - laser_line_b) / (laser_line_a - top_line_a)
            except:
                top_x = (top_line_b - laser_line_b) / (laser_line_a - top_line_a + 1)
            top_y = laser_line_a * top_x + laser_line_b


            try:
                bottom_x = (bottom_line_b - laser_line_b) / (laser_line_a - bottom_line_a)
            except:
                bottom_x = (bottom_line_b - laser_line_b) / (laser_line_a - bottom_line_a + 1)
            bottom_y = laser_line_a * bottom_x + laser_line_b

            return {'x1': top_x, 'y1': top_y, 'x2': bottom_x, 'y2': bottom_y}

        else:
            left_line_a = self.coeff_lines_support_area[key_area]['left_line']['a']
            left_line_b = self.coeff_lines_support_area[key_area]['left_line']['b']
```

```
        right_line_a = self.coeff_lines_support_area[key_area]['right_line']['a']
        right_line_b = self.coeff_lines_support_area[key_area]['right_line']['b']

        try:
            left_x = (left_line_b - laser_line_b) / (laser_line_a - left_line_a)
        except:
            left_x = (left_line_b - laser_line_b) / (laser_line_a - left_line_a + 1)
        left_y = laser_line_a * left_x + laser_line_b

        try:
            right_x = (right_line_b - laser_line_b) / (laser_line_a - right_line_a)
        except:
            right_x = (right_line_b - laser_line_b) / (laser_line_a - right_line_a + 1)
        right_y = laser_line_a * right_x + laser_line_b

        return {'x1': left_x, 'y1':left_y, 'x2': right_x, 'y2': right_y}

def find_laser_points_on_support_area_3d(self, key_area, xy):
    '''
    Находит 3D координаты точек лазера на опорных плоскостях.
    '''
    matrix_a = self.instrict_parameters_of_camera[:]
    matrix_a.append(self.coeff_support_planes[key_area]['A'])
    matrix_a.append(self.coeff_support_planes[key_area]['B'])
    matrix_a.append(self.coeff_support_planes[key_area]['C'])
    matrix_a.append(self.coeff_support_planes[key_area]['D'])
    matrix_a = [matrix_a[0:4], matrix_a[4:8], matrix_a[8:12], matrix_a[12:]]
    matrix_a = np.array(matrix_a)

    matrix_b1 = []
    matrix_b1.append(xy['x1'])
    matrix_b1.append(xy['y1'])
    matrix_b1.append(1)
    matrix_b1.append(0)
    matrix_b1 = np.array(matrix_b1)

    matrix_b2 = []
    matrix_b2.append(xy['x2'])
    matrix_b2.append(xy['y2'])
    matrix_b2.append(1)
    matrix_b2.append(0)
    matrix_b2 = np.array(matrix_b2)

    point_1_3d = linalg.solve(matrix_a, matrix_b1).tolist()
    point_2_3d = linalg.solve(matrix_a, matrix_b2).tolist()

    for i in range(len(point_1_3d) - 1):
        point_1_3d[i] /= point_1_3d[-1]
        point_2_3d[i] /= point_2_3d[-1]

    del(point_1_3d[-1])
    del(point_2_3d[-1])
```

```
        point_1_3d = {'x': point_1_3d[0], 'y': point_1_3d[1], 'z': point_1_3d[2]}
        point_2_3d = {'x': point_2_3d[0], 'y': point_2_3d[1], 'z': point_2_3d[2]}

        return [point_1_3d, point_2_3d]

    def find_laser_plane(self, laser_points_3d):
        '''
        Находит параметры плоскости лазера по точкам на опорной области.
        '''
        p1 = np.array([laser_points_3d[1]['x'], laser_points_3d[1]['y'], laser_points_3d[1]['z']])
        p2 = np.array([laser_points_3d[2]['x'], laser_points_3d[2]['y'], laser_points_3d[2]['z']])
        p3 = np.array([laser_points_3d[3]['x'], laser_points_3d[3]['y'], laser_points_3d[3]['z']])

        # Вектора на плоскости
        v1 = p1 - p2
        v2 = p1 - p3

        # Вектор нормали плоскости
        nv = np.cross(v1, v2)
        a, b, c = nv

        d = -np.dot(nv, p3)

        self.coeff_laser_plane['A'] = a
        self.coeff_laser_plane['B'] = b
        self.coeff_laser_plane['C'] = c
        self.coeff_laser_plane['D'] = d

    def get_3d_points_of_surface(self, num_frame):
        '''
        Получает 3D координаты поверхности
        '''
        x_min_surface = self.support_area_top['bottom_left_point']['x'] - 10
        y_min_surface = self.support_area_top['bottom_left_point']['y'] - 10
        x_min_surface, y_min_surface = self.to_default_coordinates_2d(x_min_surface,
y_min_surface)

        x_max_surface = self.support_area_bottom['top_right_point']['x'] + 5
        y_max_surface = self.support_area_bottom['top_right_point']['y'] + 5
        x_max_surface, y_max_surface = self.to_default_coordinates_2d(x_max_surface,
y_max_surface)

        for y in range(y_min_surface, y_max_surface):
            for x in range(x_min_surface, x_max_surface):
                if self.vid_skelet[num_frame][y][x] == True:
                    x_, y_ = self.to_other_coordinates_2d(x, y)

                    matrix_a = self.instrict_parameters_of_camera[:]
                    matrix_a.append(self.coeff_laser_plane['A'])
                    matrix_a.append(self.coeff_laser_plane['B'])
                    matrix_a.append(self.coeff_laser_plane['C'])
```

```
            matrix_a.append(self.coeff_laser_plane['D'])
            matrix_a = [matrix_a[0:4], matrix_a[4:8], matrix_a[8:12], matrix_a[12:]]
            matrix_a = np.array(matrix_a)

            matrix_b = []
            matrix_b.append(x_)
            matrix_b.append(y_)
            matrix_b.append(1)
            matrix_b.append(0)
            matrix_b = np.array(matrix_b)

            if linalg.det(matrix_a) != 0.0:
                point_3d = linalg.solve(matrix_a, matrix_b).tolist()

                for i in range(len(point_3d) - 1):
                    point_3d[i] /= point_3d[-1]

                del(point_3d[-1])

                self.file_with_3d_points.write(str(point_3d[0]) + ' ' +
                                               str(point_3d[1]) + ' ' +
                                               str(point_3d[2]) + '\n')

def to_other_coordinates_2d(self, x, y):
    '''
    Переводит экранные координаты в систему координат плоскости изображения.
    '''
    return (x - self.VID_SIZE[0] // 2, self.VID_SIZE[1] // 2 - y)

def to_default_coordinates_2d(self, x, y):
    '''
    Переводит координаты плоскости изображения в экранные координаты.
    '''
    return (self.VID_SIZE[0] // 2 + x, self.VID_SIZE[1] // 2 - y)

def paint_surface(self):
    '''
    По облаку точек рисует 3D повреxность.
    '''
    import matplotlib.pyplot as plt
    from mpl_toolkits.mplot3d import Axes3D
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    file = open('3d_points.txt', 'r')
    x = []
    y = []
    z = []

    for i, line in enumerate(file):
        x_, y_, z_ = line.split(' ')
        if float(z_) > 20:
```

```
                continue
            x.append(float(x_))
            y.append(float(y_))
            z.append(float(z_))

        file.close()

        ax.plot_trisurf(x, y, z, cmap='jet')
        ax.set_xlabel('X Label')
        ax.set_ylabel('Y Label')
        ax.set_zlabel('Z Label')
        plt.show()

#---------------------------------
    def analyze_frame(self, points, im, num_frame):
        '''
        Анализирует кадр из видеоряда.
        '''
        if points is None:
            return
        else:
            draw = ImageDraw.Draw(im)

            laser_points_3d = []
            for key_area, value in points.items():
                a = self.coeff_approximate_laser_line[key_area]['a']
                b = self.coeff_approximate_laser_line[key_area]['b']

                xy = self.get_two_points_of_laser_on_support_area(key_area)
                laser_points_3d.append(self.find_laser_points_on_support_area_3d(key_area, xy)[0])
                laser_points_3d.append(self.find_laser_points_on_support_area_3d(key_area, xy)[1])

                r = 6
                x1, y1 = self.to_default_coordinates_2d(xy['x1'], xy['y1'])
                x2, y2 = self.to_default_coordinates_2d(xy['x2'], xy['y2'])

                draw.ellipse((x1 - r, y1 - r, x1 + r, y1 + r), fill='blue')
                draw.ellipse((x2 - r, y2 - r, x2 + r, y2 + r), fill='blue')

            if not num_frame % 5:
                imageio.imwrite('__temp.jpg', np.asarray(im))    # Рисуется на draw, im тоже
изменяется
                self.canvas_lbl.setPixmap(QPixmap('__temp.jpg'))

            self.find_laser_plane(laser_points_3d)
            self.get_3d_points_of_surface(num_frame)

    def run(self):
        '''
        Запускает алгоритм нахождения 3D точек.
        '''
        self.teOutput.append('Find lines coeff on support area...')
```

```
        self.find_coeff_for_lines_on_support_area()
        self.teOutput.append('----Done----')

        self.teOutput.append('Find instrict parameters of camera...')
        self.get_instrict_parameters_of_camera()

        s_out = ''
        for k in range(3):
            for x in self.instrict_parameters_of_camera[(4*k):(4*(k+1))]:
                s_out += str(round(x, 3)) + ', '
            self.teOutput.append(s_out)
            s_out = ''

        self.teOutput.append('----Done----')

        self.teOutput.append('Find support planes...')
        self.find_support_planes()
        self.teOutput.append('----Done----')

        self.teOutput.append('Skeletonize laser...')
        self.get_vid_skeleton()
        self.teOutput.append('----Done----')

        self.teOutput.append('Find 3D points...')
        self.progressBar.reset()

        for i, frame in enumerate(self.vid_src):
            self.progressBar.setValue(i)
            if i == self.BREAK_FRAME:
                break

            self.find_laser_lines_on_support_areas(self.vid_skelet[i])
            points = self.least_square_method()

            im = Image.fromarray(frame)
            self.analyze_frame(points, im, i)

        self.teOutput.append('----Done----')

        self.vid_src.close()
        self.file_with_3d_points.close()

        self.paint_surface()

app = QApplication(sys.argv)
w = VideoAnalyzer()
w.show()
sys.exit(app.exec_())
```