МИНОБРНАУКИ РОССИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

Факультет *компьютерных наук*

Кафедра *программирования и информационных технологий*

*Наложение текстур на трёхмерное облако точек*

Курсовая работа

*230200 Информационные системы*

*Информационные системы и сетевые технологии*

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_*Н.А. Тюкачёв*

Студент \_\_\_\_\_\_\_\_\_\_\_ *А.Н. Клековкин,*

*3 курс, очное*

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_ *В.?. Тарасов*

Содержание

[Содержание 2](#_Toc420164433)

[Введение 3](#_Toc420164434)

[1. Постановка задач 6](#_Toc420164435)

[2. Средства разработки 7](#_Toc420164436)

[2.1. Язык программирования Python 7](#_Toc420164437)

[2.2. Пакет NumPy 10](#_Toc420164438)

[2.3. OpenCV 10](#_Toc420164439)

[3. Локальные дескрипторы 13](#_Toc420164440)

[3.1. Детектор Харриса 14](#_Toc420164441)

[3.2. Фильтр на основе Лапласиана 16](#_Toc420164442)

[4. Полигонная сетка 19](#_Toc420164443)

[5. Аффинные преобразования 26](#_Toc420164444)

[6. Реализация 29](#_Toc420164445)

[Заключение 32](#_Toc420164446)

[Список литературы 33](#_Toc420164447)

Введение

В настоящее время широкое распространение и развитие приобретает такая отрасль кибернетики, как компьютерное зрение.

Компьютерное зрение — теория и технология создания машин, которые могут производить обнаружение, слежение и классификацию объектов.

Как научная дисциплина, компьютерное зрение относится к теории и технологии создания искусственных систем, которые получают информацию из изображений. Видеоданные могут быть представлены множеством форм, таких как видеопоследовательность, изображения с различных камер или трехмерными данными, например с устройства [Kinect](https://ru.wikipedia.org/wiki/Kinect) или медицинского сканера.

Как технологическая дисциплина, компьютерное зрение стремится применить теории и модели компьютерного зрения к созданию [систем компьютерного зрения](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D1%80%D0%B5%D0%BD%D0%B8%D1%8F&action=edit&redlink=1).

Примерами применения таких систем могут быть:

* Системы управления процессами ([промышленные роботы](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BC%D1%8B%D1%88%D0%BB%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D1%80%D0%BE%D0%B1%D0%BE%D1%82), [автономные транспортные средства](https://ru.wikipedia.org/w/index.php?title=%D0%90%D0%B2%D1%82%D0%BE%D0%BD%D0%BE%D0%BC%D0%BD%D0%BE%D0%B5_%D1%82%D1%80%D0%B0%D0%BD%D1%81%D0%BF%D0%BE%D1%80%D1%82%D0%BD%D0%BE%D0%B5_%D1%81%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B2%D0%BE&action=edit&redlink=1))
* Системы видеонаблюдения
* Системы организации информации (например, для индексации баз данных изображений)
* Системы моделирования объектов или окружающей среды (анализ медицинских изображений, топографическое моделирование)
* Системы взаимодействия (например, устройства ввода для системы человеко-машинного взаимодействия)
* Системы [дополненной реальности](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C)
* [Вычислительная фотография](https://ru.wikipedia.org/w/index.php?title=%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F&action=edit&redlink=1), например для мобильных устройств с камерами

Область компьютерного зрения может быть охарактеризована как молодая, разнообразная и динамично развивающаяся. До сих пор не выведена стандартная формулировка того, как должна решаться проблема компьютерного зрения. Вместо этого, существует масса методов для решения различных строго определённых задач компьютерного зрения, где методы часто зависят от задач и редко могут быть обобщены для широкого круга применения. Многие из методов и приложений все ещё находятся в стадии фундаментальных исследований, но всё большее число методов находит применение в коммерческих продуктах, где они часто составляют часть большей системы, которая может решать сложные задачи (например, в области медицинских изображений или измерения и контроля качества в процессах изготовления). В большинстве практических применений компьютерного зрения компьютеры предварительно запрограммированны для решения отдельных задач, но методы, основанные на знаниях, становятся всё более общими.

Компьютерное зрение, [обработка изображений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D0%B7%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9), и [машинное зрение](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B7%D1%80%D0%B5%D0%BD%D0%B8%D0%B5) - тесно связанные области. Но до сих пор точно не определено, являются ли они разделами одной, более широкой. При детальном анализе может показаться, что это лишь разные названия одной и той же области. Чтобы не возникало путаницы, принято различать их как направления, сфокусированные на определённом предмете изучения.

[Обработка изображений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D0%B7%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) или анализ изображений, в основном сосредоточены на работе с двухмерными изображениями, то есть как преобразовать одно изображение в другое. Например, попиксельные операции увеличения контрастности, операции по выделению краёв, устранению шумов или геометрические преобразования, такие как [Аффинные преобразования](https://ru.wikipedia.org/wiki/%D0%90%D1%84%D1%84%D0%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Данные операции предполагают, что обработка/анализ изображения действуют независимо от содержания самих изображений.

Компьютерное зрение сосредотачивается на обработке трехмерных сцен, спроектированных на одно или несколько изображений. Например, восстановлением структуры или другой информации о трехмерной сцене по одному или нескольким изображениям. Компьютерное зрение часто зависит от более или менее сложных допущений относительно того, что представлено на изображениях.

[Машинное зрение](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B7%D1%80%D0%B5%D0%BD%D0%B8%D0%B5) сосредотачивается на применении, в основном промышленном, например, автономные роботы и системы визуальной проверки и измерений. Это значит, что технологии датчиков изображения и теории управления связаны с обработкой видеоданных для управления роботом и обработка данных в реальном времени осуществляется аппаратно или программно.

Также существует область, названная [Визуализация](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B7%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), которая первоначально была связана с процессом создания изображений, но иногда имела дело с обработкой и анализом. Например, [рентгенография](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BD%D1%82%D0%B3%D0%B5%D0%BD%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F) работает с анализом видеоданных медицинского применения.

Наконец, [распознавание образов](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D0%BE%D0%B7%D0%BD%D0%B0%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2) является областью, которая использует различные методы для получения информации из видеоданных, в основном, основанные на статистическом подходе. Значительная часть этой области посвящена практическому применению этих методов.

Одной из типичных задач компьютерного зрения является задача восстановления трёхмерной сцены. Наиболее типичные подзадачи восстановления трехмерных сцен:

* Восстановление земного рельефа по аэрокосмическим изображениям для нужд картографии и мониторинга промышленных объектов и чрезвычайных ситуаций.
* Восстановление плана помещений по изображениям, полученным с цифровых фотоаппаратов и кинокамер для нужд перепланировки и оборудования промышленных помещений и архитектуры.

1. Постановка задач

Целью данной работы является изучение средств и алгоритмов восстановления трёхмерной сцены по фотографии, в частности построение сцены по заданному облаку точек.

На начальном этапе имеем трёхмерное облако точек, соответствующее множеству углов некоторого объекта в пространстве и изображение (фотографию) этого объекта. Фотография сделана с определённого ракурса (известно расстояние до объекта и угол поворота объекта относительно наблюдателя/камеры).

В ходе исследования необходимо разработать систему для наложения текстур объекта с изображения на облако точек.

Использующиеся инструменты: язык программирования Python и библиотеки Numpy и OpenCV

На выходе мы должны получить трёхмерное текстурированное изображение объекта.

1. Средства разработки
   1. Язык программирования Python

Python - [высокоуровневый язык программирования](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. [Синтаксис](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) ядра Python минималистичен. В то же время [стандартная библиотека](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_Python) включает в себя большой объём полезных функций.

Python поддерживает несколько [парадигм программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), в том числе [структурное](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [объектно-ориентированное](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [функциональное](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [императивное](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [аспектно-ориентированное](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D0%BF%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Основные архитектурные черты — [динамическая типизация](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [автоматическое управление памятью](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29), полная [интроспекция](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D1%80%D0%BE%D1%81%D0%BF%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29), механизм [обработки исключений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9), поддержка [многопоточных вычислений](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D0%BE%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C) и удобные высокоуровневые [структуры данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Код в Python организовывается в функции и [классы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29), которые могут объединяться в [модули](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) (они в свою очередь могут быть объединены в пакеты).

Появившись сравнительно поздно, Python создавался под влиянием множества языков программирования:

* [ABC](https://ru.wikipedia.org/wiki/ABC_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29) — отступы для группировки операторов, высокоуровневые структуры данных (map) (Python фактически создавался как попытка исправить ошибки, допущенные при проектировании ABC);
* [Modula-3](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D0%B0-3) — пакеты, модули, использование else совместно с try и except, именованные аргументы функций (на это также повлиял [Common Lisp](https://ru.wikipedia.org/wiki/Common_Lisp));
* [С](https://ru.wikipedia.org/wiki/%D0%A1_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29), [C++](https://ru.wikipedia.org/wiki/C%2B%2B) — некоторые синтаксические конструкции (как пишет сам [Гвидо ван Россум](https://ru.wikipedia.org/wiki/%D0%93%D0%B2%D0%B8%D0%B4%D0%BE_%D0%B2%D0%B0%D0%BD_%D0%A0%D0%BE%D1%81%D1%81%D1%83%D0%BC) — он использовал наиболее непротиворечивые конструкции из С, чтобы не вызвать неприязнь у С-программистов к Python[[15]](https://ru.wikipedia.org/wiki/Python" \l "cite_note-foreword-15));
* [Smalltalk](https://ru.wikipedia.org/wiki/Smalltalk) — объектно-ориентированное программирование;
* [Lisp](https://ru.wikipedia.org/wiki/Lisp) — отдельные черты [функционального программирования](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) (lambda, map, reduce, filter и другие);
* [Fortran](https://ru.wikipedia.org/wiki/Fortran) — срезы массивов, комплексная арифметика;
* [Miranda](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D1%80%D0%B0%D0%BD%D0%B4%D0%B0_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29) — [списочные выражения](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BD%D0%B0_%D0%9F%D0%B8%D1%82%D0%BE%D0%BD%D0%B5#.D0.A1.D0.BF.D0.B8.D1.81.D0.BE.D1.87.D0.BD.D1.8B.D0.B5_.D0.B2.D1.8B.D1.80.D0.B0.D0.B6.D0.B5.D0.BD.D0.B8.D1.8F);
* [Java](https://ru.wikipedia.org/wiki/Java) — модули logging, unittest, threading (часть возможностей оригинального модуля не реализована), xml.sax стандартной библиотеки, совместное использование finally и except при обработке исключений, использование @ для [декораторов](https://ru.wikipedia.org/wiki/Python#.D0.94.D0.B5.D0.BA.D0.BE.D1.80.D0.B0.D1.82.D0.BE.D1.80.D1.8B);
* [Icon](https://ru.wikipedia.org/wiki/Icon_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29) — [генераторы](https://ru.wikipedia.org/wiki/Python#.D0.93.D0.B5.D0.BD.D0.B5.D1.80.D0.B0.D1.82.D0.BE.D1.80.D1.8B).

Большая часть других возможностей Python (например, байт-компиляция исходного кода) также была реализована ранее в других языках.

Одним из важнейших плюсов языка Python является портируемость. Он работает почти на всех известных платформах — от [КПК](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D0%BC%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) до [мейнфреймов](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B9%D0%BD%D1%84%D1%80%D0%B5%D0%B9%D0%BC). Существуют версии под [Microsoft Windows](https://ru.wikipedia.org/wiki/Microsoft_Windows), практически все варианты [UNIX](https://ru.wikipedia.org/wiki/UNIX) (включая [FreeBSD](https://ru.wikipedia.org/wiki/FreeBSD) и [Linux](https://ru.wikipedia.org/wiki/Linux)), [Plan 9](https://ru.wikipedia.org/wiki/Plan_9), [Mac OS](https://ru.wikipedia.org/wiki/Mac_OS) и [Mac OS X](https://ru.wikipedia.org/wiki/Mac_OS_X), [iPhone OS](https://ru.wikipedia.org/wiki/IPhone_OS) 2.0 и выше, [Palm OS](https://ru.wikipedia.org/wiki/Palm_OS), [OS/2](https://ru.wikipedia.org/wiki/OS/2), [Amiga](https://ru.wikipedia.org/wiki/Amiga), [HaikuOS](https://ru.wikipedia.org/wiki/HaikuOS), [AS/400](https://ru.wikipedia.org/wiki/AS/400) и даже [OS/390](https://ru.wikipedia.org/wiki/OS/390), [Windows Mobile](https://ru.wikipedia.org/wiki/Windows_Mobile), [Symbian](https://ru.wikipedia.org/wiki/Symbian) и [Android](https://ru.wikipedia.org/wiki/Android).

Однако, по мере устаревания платформы её поддержка в основной ветви языка прекращается. Например, с серии 2.6 прекращена поддержка [Windows 95](https://ru.wikipedia.org/wiki/Windows_95), [Windows 98](https://ru.wikipedia.org/wiki/Windows_98) и [Windows ME](https://ru.wikipedia.org/wiki/Windows_ME). Тем не менее на этих платформах можно использовать предыдущие версии Python — на данный момент сообщество активно поддерживает версии Python начиная от 2.3 (для них выходят исправления).

При этом, в отличие от многих портируемых систем, для всех основных платформ Python имеет поддержку характерных для данной платформы технологий (например, Microsoft [COM](https://ru.wikipedia.org/wiki/Microsoft_Component_Object_Model)/[DCOM](https://ru.wikipedia.org/wiki/Microsoft_Distributed_Component_Object_Model)). Более того, существует специальная версия Python для [виртуальной машины Java](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_Java) — [Jython](https://ru.wikipedia.org/wiki/Jython), что позволяет интерпретатору выполняться на любой системе, поддерживающей Java, при этом классы Java могут непосредственно использоваться из Python и даже быть написанными на Python. Также несколько проектов обеспечивают интеграцию с платформой [Microsoft .NET](https://ru.wikipedia.org/wiki/Microsoft_.NET), основные из которых — [IronPython](https://ru.wikipedia.org/wiki/IronPython) и [Python.Net](https://ru.wikipedia.org/w/index.php?title=Python.Net&action=edit&redlink=1).

Ещё одним достоинство Python выступает [динамическая типизацию](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), то есть тип переменной определяется только во время исполнения. Поэтому вместо «присваивания значения переменной» лучше говорить о «связывании значения с некоторым именем». В Python имеются встроенные типы: [булевый](https://ru.wikipedia.org/wiki/%D0%91%D1%83%D0%BB%D0%B5%D0%B2%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D0%B8%D0%BF), [строка](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%BE%D0%BA%D0%BE%D0%B2%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF), [Unicode](https://ru.wikipedia.org/wiki/Unicode)-строка, целое число произвольной точности, число [с плавающей запятой](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B2%D0%B0%D1%8E%D1%89%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D1%8F%D1%82%D0%B0%D1%8F), [комплексное число](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BB%D0%B5%D0%BA%D1%81%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) и некоторые другие. Из [коллекций](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) в Python встроены: [список](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29), [кортеж](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D1%80%D1%82%D0%B5%D0%B6_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29) (неизменяемый список), [словарь](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2), [множество](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE) и другие. Все значения являются объектами, в том числе функции, методы, модули, классы.

Добавить новый тип можно либо написав [класс](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) (class), либо определив новый тип в модуле расширения (например, написанном на языке C). Система классов поддерживает [наследование](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) (одиночное и [множественное](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BD%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и [метапрограммирование](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%B0%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Возможно наследование от большинства встроенных типов и типов расширений.

Все объекты делятся на ссылочные и атомарные. К атомарным относятся int, long (в версии 3 любое число int, т.к. в версии 3 нет ограничения на размер), complex и некоторые другие. При присваивании атомарных объектов копируется их значение, в то время как для ссылочных копируется только указатель на объект, таким образом, обе переменные после присваивания используют одно и то же значение. Ссылочные объекты бывают изменяемые и неизменяемые. Например, строки и кортежи являются неизменяемыми, а списки, словари и многие другие объекты — изменяемыми. Кортеж в Python является, по сути, неизменяемым списком. Во многих случаях кортежи работают быстрее списков, поэтому если вы не планируете изменять последовательность, то лучше использовать именно их.

Дизайн языка Python построен вокруг объектно-ориентированной модели программирования. Реализация [ООП](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) в Python является элегантной, мощной и хорошо продуманной, но вместе с тем достаточно специфической по сравнению с другими [объектно-ориентированными языками](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F).

Возможности и особенности:

1. Классы являются одновременно объектами со всеми ниже приведёнными возможностями.
2. Наследование, в том числе множественное.
3. Полиморфизм (все функции виртуальные).
4. Инкапсуляция (два уровня — общедоступные и скрытые методы и поля). Особенность — скрытые члены доступны для использования и помечены как скрытые лишь особыми именами.
5. Специальные методы, управляющие жизненным циклом объекта: конструкторы, деструкторы, распределители памяти.
6. Перегрузка операторов (всех, кроме is, '.', '=' и символьных логических).
7. Свойства (имитация поля с помощью функций).
8. Управление доступом к полям (эмуляция полей и методов, частичный доступ, и т. п.).
9. Методы для управления наиболее распространёнными операциями (истинностное значение, len(), глубокое копирование, [сериализация](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B8%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), итерация по объекту, …)
10. Метапрограммирование (управление созданием классов, триггеры на создание классов, и др.)
11. Полная [интроспекция](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D1%80%D0%BE%D1%81%D0%BF%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29).
12. Классовые и статические методы, классовые поля.
    1. Пакет NumPy

NumPy — это расширение языка [Python](https://ru.wikipedia.org/wiki/Python), добавляющее поддержку больших многомерных [массивов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2) и [матриц](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_%28%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29), вместе с большой библиотекой [высокоуровневых](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) математических функций для операций с этими массивами. Предшественник NumPy, Numeric, был изначально создан Jim Hugunin. NumPy — [открытое программное обеспечение](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), поучаствовать в разработке может любой желающий.

Поскольку [Python](https://ru.wikipedia.org/wiki/Python) — [интерпретируемый язык](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), математические алгоритмы часто работают в нём гораздо медленнее, чем в [компилируемых языках](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), таких как [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29) или [Java](https://ru.wikipedia.org/wiki/Java). NumPy пытается решить эту проблему для большого количества [вычислительных алгоритмов](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0), обеспечивая поддержку многомерных массивов и множество функций и операторов для работы с ними. Таким образом, любой алгоритм, который может быть выражен в основном как последовательность операций над массивами и матрицами, работает так же быстро, как эквивалентный код, выполняемый в MATLAB, а после специальной оптимизации скорость может достигнуть скорости компилируемых языков типа [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29).

NumPy можно рассматривать как хорошую свободную альтернативу [MATLAB](https://ru.wikipedia.org/wiki/MATLAB), поскольку язык программирования [MATLAB](https://ru.wikipedia.org/wiki/MATLAB) внешне напоминает NumPy: оба они интерпретируемые, и оба позволяют пользователям писать быстрые программы, пока большинство операций производятся над массивами или матрицами, а не над [скалярами](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D0%B0%D0%BB%D1%8F%D1%80). Преимущество [MATLAB](https://ru.wikipedia.org/wiki/MATLAB) в большом количестве доступных дополнительных тулбоксов, включая такие как пакет [Simulink](https://ru.wikipedia.org/w/index.php?title=Simulink&action=edit&redlink=1). Основные пакеты, дополняющие NumPy, это: [SciPy](https://ru.wikipedia.org/wiki/SciPy) — библиотека, добавляющая больше MATLAB-подобной функциональности; [Matplotlib](https://ru.wikipedia.org/wiki/Matplotlib) — пакет для создания графики в стиле [MATLAB](https://ru.wikipedia.org/wiki/MATLAB). Внутренне как [MATLAB](https://ru.wikipedia.org/wiki/MATLAB), так и NumPy основаны на библиотеке [LAPACK](https://ru.wikipedia.org/wiki/LAPACK), предназначенной для решения основных задач линейной алгебры.

* 1. OpenCV

OpenCV ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Open Source Computer Vision Library, библиотека компьютерного зрения с открытым исходным кодом) — библиотека алгоритмов [компьютерного зрения](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%BE%D0%B5_%D0%B7%D1%80%D0%B5%D0%BD%D0%B8%D0%B5), [обработки изображений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D0%B7%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) и численных алгоритмов общего назначения с [открытым кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Реализована на [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29)/[C++](https://ru.wikipedia.org/wiki/C%2B%2B), также разрабатывается для [Python](https://ru.wikipedia.org/wiki/Python), [Java](https://ru.wikipedia.org/wiki/Java), [Ruby](https://ru.wikipedia.org/wiki/Ruby), [Matlab](https://ru.wikipedia.org/wiki/Matlab), [Lua](https://ru.wikipedia.org/wiki/Lua) и других языков. Может свободно использоваться в академических и коммерческих целях — распространяется в условиях [лицензии BSD](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%86%D0%B5%D0%BD%D0%B7%D0%B8%D0%B8_BSD).

Плюсом библиотеки является её кроссплатформенность. OpenCV поддерживает популярные платформы, такие как Microsoft Windows, Unix, Android, iOS, MacOS.

В OpenCV, начиная с версии 2.2, используются несколько компактных модулей с узкой специализацией:

* opencv\_core — основная функциональность. Включает в себя базовые структуры, вычисления (математические функции, генераторы случайных чисел) и линейную алгебру, [DFT](https://ru.wikipedia.org/wiki/DFT), [DCT](https://ru.wikipedia.org/wiki/DCT), ввод/вывод для XML и YAWL и т. д.
* opencv\_imgproc — обработка изображений (фильтрация, геометрические преобразования, преобразование цветовых пространств и т. д.).
* opencv\_highgui — простой UI, ввод/вывод изображений и видео.
* opencv\_ml — модели машинного обучения (SVM, деревья решений, обучение со стимулированием и т. д.).
* opencv\_features2d — распознавание и описание плоских примитивов ([SURF](https://en.wikipedia.org/wiki/SURF) (англ.)[русск.](https://ru.wikipedia.org/w/index.php?title=SURF&action=edit&redlink=1), FAST и другие, включая специализированный фреймворк).
* opencv\_video — анализ движения и отслеживание объектов ([оптический поток](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D0%BE%D1%82%D0%BE%D0%BA), шаблоны движения, устранение фона).
* opencv\_objdetect — обнаружение объектов на изображении (нахождение лиц с помощью [алгоритма Виолы-Джонса](https://ru.wikipedia.org/w/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%92%D0%B8%D0%BE%D0%BB%D0%B0-%D0%94%D0%B6%D0%BE%D0%BD%D1%81%D0%B0&action=edit&redlink=1) ([англ.](https://en.wikipedia.org/wiki/Viola-Jones_object_detection_framework)), распознавание людей HOG и т. д.).
* opencv\_calib3d — калибровка камеры, поиск стерео-соответствия и элементы обработки трехмерных данных.
* opencv\_flann — библиотека быстрого поиска ближайших соседей (FLANN 1.5) и обертки OpenCV.
* opencv\_contrib — сопутствующий код, ещё не готовый для применения.
* opencv\_legacy — устаревший код, сохраненный ради обратной совместимости.
* opencv\_gpu — ускорение некоторых функций OpenCV за счет [CUDA](https://ru.wikipedia.org/wiki/CUDA), создан при поддержке [NVidia](https://ru.wikipedia.org/wiki/NVidia).

Данная библиотека часто используется для:

* Распознавание объектов.
* Распознавание текста.
* Устранение искажений.
* Выявление сходства и формы объектов.
* Слежение за перемещением объекта.
* Распознавание движений, жестов и многое другое.

1. Локальные дескрипторы

Особые дескрипторы (в разных источниках – **features/characteristic points/local feature points/interest point/локальные особенности**) –хорошо различимые фрагменты изображения. Это точки (пиксели) с характерной (особой) окрестностью – т.е. отличающиеся своей окрестностью от всех соседних точек.

Классический пример локальной особенности – вершина угла. Описываются вектором признаков вычисляемых на основе интенсивности/градиентов или других характеристик точек окрестности. Используя особые точки можно  анализировать как изображения целиком, так и объекты на них. Хорошие характерные точки позволяют справиться с изменением масштаба, ракурса и перекрытиями сцены или объекта.

Требования к локальным дескрипторам:

1. Повторимость (Особенность находится в одном и том же месте объекта, в независимости от масштаба, поворота, положения объекта и освещения)
2. Локальность (Особенность занимает малую область объекта, в следствии чего работа с ней не чувствительна к перекрытию)
3. Значимость (Каждая особенность имеет уникальное описание)
4. Компактность и эффективность (Количество особенностей должно быть много меньше количества пикселей в изображении)

Существует несколько методов поиска локальных дескрипторов:

1. Детектор Харриса ([Harris)](http://en.wikipedia.org/wiki/Corner_detection" \l "The_Harris_.26_Stephens_.2F_Plessey_corner_detection_algorithm" \t "_blank)
2. Детектор Харриса-Лапласа ([Harris-Laplace](http://en.wikipedia.org/wiki/Harris_affine_region_detector" \l "Harris.E2.80.93Laplace_detector_.28initial_region_points.29" \t "_blank))
3. детектор на основе лапласиана гауссианов ([LoG, Laplacian of Gaussian)](http://en.wikipedia.org/wiki/Blob_detection" \l "The_Laplacian_of_Gaussian" \t "_blank)
4. детектор на основе дифференциалов гауссианов ([DoG, Difference of Gaussian)](http://en.wikipedia.org/wiki/Difference_of_Gaussians)

**Лапласиан** – сумма вторых частных производных функции в точке. **Гауссиан** – это просто [гауссова функция](http://en.wikipedia.org/wiki/Gaussian_function) – которой обрабатывают изображение. Соответственно лапласиан гауссиана – представление функции-изображения после применения оператора Гаусса, через сумму частных производных (его трехмерный график, при некоторой доле воображения, напоминает мексиканскую шляпку или сомбреро).

* 1. Детектор Харриса

Одним из наиболее распространенных типов особых точек являются углы на изображении , т.к. в отличие от ребер углы на паре изображений можно однозначно сопоставить. Расположение углов можно определить, используя локальные детекторы. Входом локальных детекторов является черно-белое изображение. На выходе формируется матрица с элементами, значения которых определяют степень правдоподобности нахождения угла в соответствующих пикселях изображения. Далее выполняется отсечение пикселей со степенью правдоподобности, меньшей некоторого порога. Для оставшихся точек принимается, что они являются особыми.

Детектор Харриса – наиболее распространённый детектор локальных особенностей. Его часто называют детектором углов Харриса. Этот алгоритм основан на алгоритме обнаружения углов Моравека.

Алгоритм Моравека является одним из первых алгоритмов нахождения углов. Он проверяет каждый пиксель в изображении, чтобы определить является ли тот углом, рассматривая участки в области пикселя. Сходство определяется путем принятия суммы квадратов разностей между двумя участками. Меньшее число указывает на большее сходство.

Идея состоит в том, что если пиксель в области равномерной интенсивностью, то близлежащие участки будут выглядеть примерно одинаково. Если пиксель находится на краю, тогда соседние участки в направлении, перпендикулярном к краю будет выглядеть совершенно разными, но соседние участки в направлении, параллельном краю изменяются незначительно. Если пиксель на особенности с изменением во всех направлениях, то ни один из близлежащих участков не будет выглядеть примерно также.

Сила угла определяется как наименьшая сумма квадратов разностей между участком и его соседями (по горизонтали, вертикали и двум диагоналям). Если это число локально максимально, то особенность присутствует.

У этого алгоритма есть проблема связанная с тем, что он не изотропен: если угол не направлен в сторону соседей, то он не будет обнаружен, как точечная особенность.

Харрис и Стивенс улучшили детектор углов Моравека, рассматривая дифференциальную оценку угла по отношению к направлению непосредственно, вместо использования сдвинутых пятен. Эту оценку угла часто называют автокорреляционной, поскольку этот термин используется в том документе, в котором этот детектор описан. Однако с математической точки зрения используется метод суммы квадратов разностей.

Без потери общности будем считать, что используются полутоновые 2-мерные изображения. Пусть это изображение будет задано I. Рассмотрим вопрос о выделении области изображения (U, V) и перехода его по (х, у). Взвешенную сумму квадратов разностей между этими двумя областями, обозначим S, определяющуюся по формуле:

может быть аппроксимирована рядом Тейлора. Пусть *Ix* и *Iy* - будут частными производными от *I*, такими, что

Это приводит к следующему приближению:

Запишем в матричном виде:

где

Эта матрица - матрица Харриса, а угловые скобки означают усреднение (например, суммирование (U, V)). Если используется круглое окно (или округлые взвешенные окна, такие, как гауссовские), то ответ будет изотропным.

Угол (или, в общем, точечная особенность) характеризуется большим изменением S во всех направлениях вектора . На основе анализа собственных значений A, эта характеристика может быть выражена следующим образом: должно быть два "больших" собственных значения для точечных особенностей. На основании величины собственных значений , можно сделать следующие выводы на основе этих аргументов:

1. Если и то этот пиксель (х, у) не имеет особенности, представляющей интерес.
2. Если и λ2 имеет некоторое большое положительное значение, то обнаружен край.
3. Если λ1 и λ2 большие положительные значения, то угол найден.

Определение собственных значений вычислительно дорого, так как требует вычисления квадратного корня. Вместо этого используются следующие функции с M, где κ является настраиваемым параметром чувствительности:

Таким образом, алгоритм не имеет на самом деле вычисления собственного разложения матрицы, а вместо этого достаточно вычислить определитель и след от A чтобы найти углы, или, вернее, точки интереса в целом.

Детектор Харриса по сравнению с ранее рассмотренным детектором требует большего количества вычислений за счет необходимости построения сверток с Гауссовым ядром. При этом он достаточно восприимчив к шумам. Подавить шумы позволяет увеличение размера Гауссова окна, но это приводит к значительным вычислительным расходам, поэтому необходимо находить компромисс между качеством работы алгоритма и количеством выполняемых операций. Детектор Харриса обладает свойством анизотропии вдоль горизонтального и вертикального направлений, т.к. автокорреляционная матрица содержит первые производные только вдоль указанных направлений. По сравнению со своим предшественником данный детектор инвариантен относительно поворота, количество ошибок детектирования углов не велико за счет введения свертки с Гауссовыми весовыми коэффициентами. Результаты детектирования значительно меняются при масштабировании изображения. Впоследствии возникают модификации детектора Харриса, которые учитывают вторые производные функции интенсивности (например, детектор Харриса-Лапласа (Harris-Laplace) ).

* 1. Фильтр на основе Лапласиана

Иногда бывает недостаточно выделить углы объекта в изображении. Второй важной характеристикой являются грани. Методы поиска граней основаны на определении точек, в которых интенсивность резко меняется. Причиной этих изменений интенсивности может быть либо перепады в глубине, ориентации поверхности, изменении освещения и многих других факторов. В идеальном случае, результат применения детектора граней к изображению приводит к набору связанных линий, которые обозначают границы объектов.

Для выделения граней часто применяют фильтр, построенный с помощью оператора Лапласа.

Одним из ключевых элементов алгоритма является подавление немаксимумов. Идея основана на том, что граница должна проходить через максимум градиента на данном направлении. Эта идея становится интуитивно ясной при рассмотрении функции одной переменной: точка с экстремальным значением первой производной соответствует максимально быстрому перепаду значений.

Однако, как известно из математического анализа, необходимым и достаточным условием экстремального значения первой производной функции в некой точке является равенство нулю второй производной в этой точке, причем по разные стороны от точки вторая производная должна иметь разные знаки. Про такую точку говорят, что вторая производная в ней пересекает ноль.

В двумерном случае, который нас и интересует, аналогом первой производной является вектор градиента

Аналогом второй производной является скалярный оператор, называемый лапласианом.

**Оператор Лапласа или лапласиан** – это сумма вторых частных производных функции в точке:

Таким образом, для декартовых координат оператор Лапласа будет иметь вид:

В цифровой обработке изображений вторая частная производная по координате х в дискретном виде рассчитывается по формуле:

Аналогично для производной по у:

Дискретная формулировка двумерного лапласиана получается объединением этих двух составляющих:

Для реализации дискретного лапласиана строится маска, которая накладывается на изображение. Это приводит к получению изображения, содержащего сероватые линии на месте контуров и других разрывов, наложенные на темный фон.

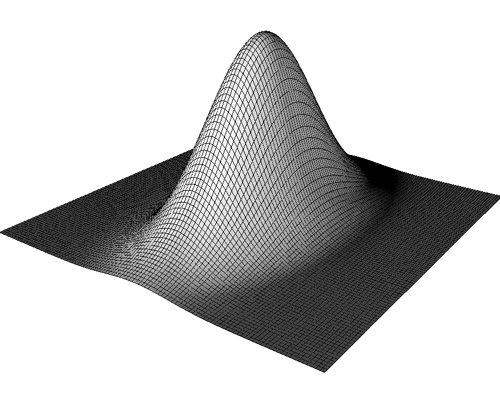
К дискретному лапласиану могут быть добавлены диагональные члены. Диагональные направления могут быть включены в формулу дискретного лапласиана добавлением еще двух членов — по одному для каждого из диагональных направлений. Вид каждого из них такой же, как в уравнении производной по координате *x* или по *y*, но указываются координаты точек, расположенных по диагоналям. Поскольку каждая диагональная добавка включает член *-2f(х, у),* то суммарный вычитаемый из суммы член составит *8f(x,у).*Такая маска является изотропной для поворотов на углы, кратные 45°.

Данный фильтр часто применяется в связке с функцией Гаусса или гауссианом. Идея поиска граней с помощью лапласиана гауссиана (LoG-детекторы) была предложена Джоном Канни в магистерской диссертации в 1983 году.

Стандартный вид гауссиана в общей форме представлен формулой:

В декартовых координатах гауссова функция имеет вид:

Графическое отображение этой функции представлено на рисунке 1.



1. График гауссиана

Если мы применим гауссиан к изображению, то мы получим размытие этого изображения.

Алгоритм Канни включает в себя следующие шаги:

1. Размытие исходного изображения с помощью функции Гаусса.
2. Поиск градиента. Границы намечаются там, где градиент принимает максимальное значение.
3. Подавление немаксимумов. Только локальные максимумы отмечаются как границы.
4. Итоговые границы определяются путем подавления всех краев, не связанных с определенными границами.
5. Полигонная сетка

Полигонная сетка - это совокупность вершин и граней, характеризующую форму объекта. Другими словами, это плоскости, являющиеся поверхностями объекта, т.е. те плоскости (полигоны), на которые необходимо наложить текстуры.

Гранями обычно являются [треугольники](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B5%D1%83%D0%B3%D0%BE%D0%BB%D1%8C%D0%BD%D0%B8%D0%BA), [четырехугольники](https://ru.wikipedia.org/wiki/%D0%A7%D0%B5%D1%82%D1%8B%D1%80%D0%B5%D1%85%D1%83%D0%B3%D0%BE%D0%BB%D1%8C%D0%BD%D0%B8%D0%BA) или другие простые [выпуклые многоугольники](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D0%BF%D1%83%D0%BA%D0%BB%D1%8B%D0%B9_%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%D1%83%D0%B3%D0%BE%D0%BB%D1%8C%D0%BD%D0%B8%D0%BA) (полигоны), так как это упрощает [рендеринг](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BD%D0%B4%D0%B5%D1%80%D0%B8%D0%BD%D0%B3), но сетки могут также состоять и из наиболее общих вогнутых многоугольников, или многоугольников с дырками.

Учение о полигональных сетках - это большой подраздел компьютерной графики и геометрического моделирования. Множество операций, проводимых над сетками, может включать [булеву алгебру](https://ru.wikipedia.org/wiki/%D0%91%D1%83%D0%BB%D0%B5%D0%B2%D0%B0_%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0), сглаживание, упрощение и многие другие. Разные представления полигональных сеток используются для разных целей и приложений. Для передачи полигональных сеток по сети используются сетевые представления, такие как «потоковые» и «прогрессивные» сетки. Объемные сетки отличаются от полигональных тем, что они явно представляют и поверхность и объём структуры, тогда как полигональные сетки явно представляют лишь поверхность, а не объём. Так как полигональные сетки широко используются в компьютерной графике, для них разработаны алгоритмы [трассировки лучей](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D1%81%D1%81%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%BB%D1%83%D1%87%D0%B5%D0%B9), [обнаружения столкновений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BD%D0%B0%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81%D1%82%D0%BE%D0%BB%D0%BA%D0%BD%D0%BE%D0%B2%D0%B5%D0%BD%D0%B8%D0%B9) и [динамики твердых тел](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D0%BA%D0%B0_%28%D1%84%D0%B8%D0%B7%D0%B8%D0%BA%D0%B0%29).

Математический эквивалент полигональных сеток - [неструктурированные сетки](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D0%BA%D0%B0) - изучаются методами [комбинаторной геометрии](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B1%D0%B8%D0%BD%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F).

В контексте данной задачи мы будем пользоваться треугольными гранями, так как треугольник является наиболее простой фигурой, а в любой поверхности объекта всегда можно найти три вершины (четырёх и более вершин может не быть).

Существует несколько способов представления полигонной сетки.

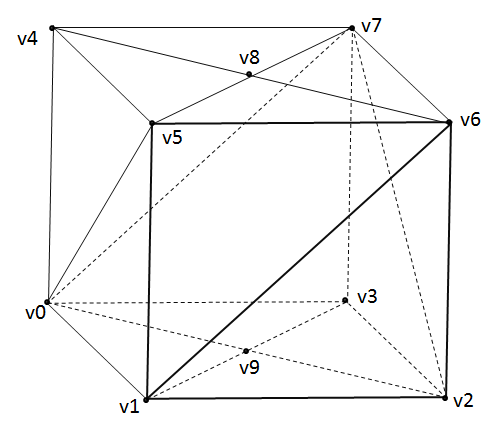
Самый простейший способ - вершинное представление - описывает объект как множество вершин, соединенных с другими вершинами. Это простейшее представление, но оно не широко используемое, так как информация о гранях и ребрах не выражена явно. Поэтому нужно обойти все данные чтобы сгенерировать список граней для рендеринга. Кроме того, нелегко выполняются операции на ребрах и гранях.

Однако, такие сетки извлекают выгоду из малого использования памяти и эффективной трансформации.

В таблице 1 приведено вершинное представление полигонной сетки параллелепипеда (Рис. 2). Первый столбец – имя вершины, второй – координаты данной вершины, третий - те вершины, с которыми соединена данная.

|  |  |  |
| --- | --- | --- |
| v0 | 0,0,0 | v1 v5 v4 v3 v9 |
| v1 | 1,0,0 | v2 v6 v5 v0 v9 |
| v2 | 1,1,0 | v3 v7 v6 v0 v9 |
| v3 | 0,1,0 | v2 v6 v7 v4 v9 |
| v4 | 0,0,1 | v5 v0 v3 v7 v8 |
| v5 | 1,0,1 | v6 v1 v0 v4 v8 |
| v6 | 1,1,1 | v7 v2 v1 v5 v8 |
| v7 | 0,1,1 | v4 v3 v2 v6 v8 |
| v8 | .5,.5,1 | v5 v6 v7 v8 |
| v9 | .5,.5,0 | v0 v1 v2 v3 |

1. Список вершин



1. Вершинное представление

Более сложным способом с использованием списка граней представляет объект как множество граней и множество вершин. Это самое широко используемое представление, будучи входными данными, типично принимаемыми современным графическим оборудованием.

Список граней лучше для моделирования, чем вершинное представление тем, что он позволяет явный поиск вершин грани, и граней окружающих вершину. Рисунок 2 показывает пример параллелепипеда в виде сетки с использованием списка граней. В этом примере у каждой грани обязательно 3 вершины. Однако это не означает что у каждой вершины одно и то же количество окружающих граней.

Для рендеринга грань обычно посылается в графический процессор как множество индексов вершин, и вершины посылаются как позиция/цвет/структуры нормалей (на рисунке дана лишь позиция). Поэтому изменения формы, но не геометрии, могут быть динамически обновлены через простую передачу данных вершины без обновления связанности граней.

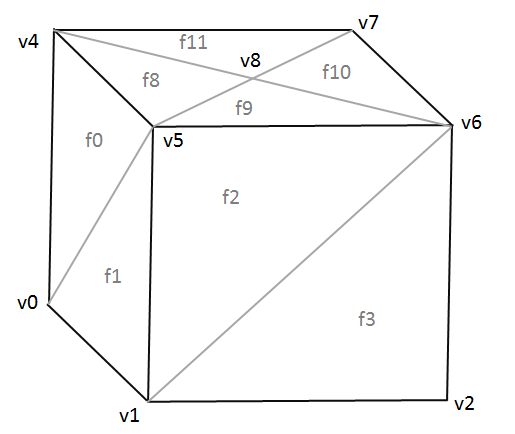
Моделирование требует легкого обхода всех структур. С сеткой использующей список граней очень легко найти вершины грани. Также, список вершин содержит список всех граней связанных с каждой вершиной. В отличие от вершинного представления, и грани и вершины явно представлены, так что нахождение соседних граней и вершин постоянно по времени. Однако, ребра не заданы явно, так что поиск все ещё нужен, чтобы найти все грани, окружающие заданную грань. Другие динамические операции, такие как разрыв или объединение грани, также сложны со списком граней.

|  |  |
| --- | --- |
| f0 | v0 v4 v5 |
| f1 | v0 v5 v1 |
| f2 | v1 v5 v6 |
| f3 | v1 v6 v2 |
| f4 | v2 v6 v7 |
| f5 | v2 v7 v3 |
| f6 | v3 v7 v4 |
| f7 | v3 v4 v0 |
| f8 | v8 v5 v4 |
| f9 | v8 v6 v5 |
| f10 | v8 v7 v6 |
| f11 | v8 v4 v7 |
| f12 | v9 v5 v4 |
| f13 | v9 v6 v5 |
| f14 | v9 v7 v6 |
| f15 | v9 v4 v7 |

1. Список граней

|  |  |  |
| --- | --- | --- |
| v0 | 0,0,0 | f0 f1 f12 f15 f7 |
| v1 | 1,0,0 | f2 f3 f13 f12 f1 |
| v2 | 1,1,0 | f4 f5 f14 f13 f3 |
| v3 | 0,1,0 | f6 f7 f15 f14 f5 |
| v4 | 0,0,1 | f6 f7 f0 f8 f11 |
| v5 | 1,0,1 | f0 f1 f2 f9 f8 |
| v6 | 1,1,1 | f2 f3 f4 f10 f9 |
| v7 | 0,1,1 | f4 f5 f6 f11 f10 |
| v8 | .5,.5,1 | f8 f9 f10 f11 |
| v9 | .5,.5,0 | f12 f13 f14 f15 |

1. Список вершин

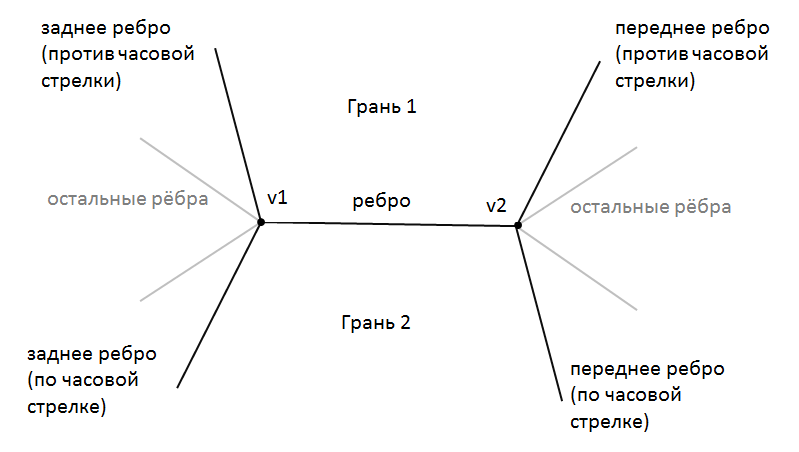


1. Графическое представление списка граней

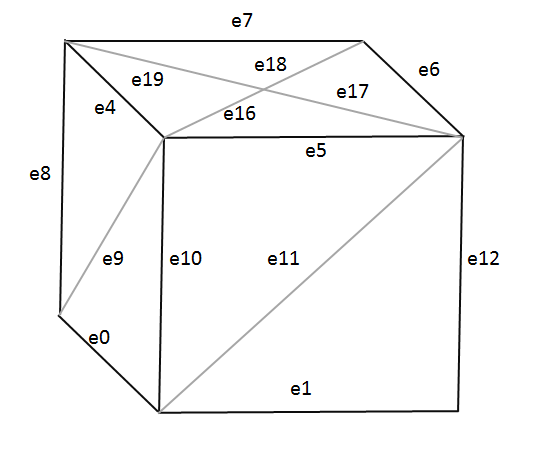
В 1975 Брюсом Баумгартом было выведено ещё одно представление полигонной сетки, которое получило название "Крылатое представление". Оно явно задаёт вершины, грани и ребра сетки. Это представление широко используется в программах для моделирования для предоставления высочайшей гибкости в динамическом изменении геометрии сетки, потому что могут быть быстро выполнены операции разрыва и объединения. Их основной недостаток - высокие требования памяти и увеличенная сложность из-за содержания множества индексов.

"Крылатое" представление решает проблему обхода от ребра к ребру и обеспечивает упорядоченное множество граней вокруг ребра. Для любого заданного ребра число исходящих ребер может быть произвольным. Чтобы упростить это, "крылатое" представление предоставляет лишь четыре, ближайшие ребра по часовой и против часовой стрелки на каждом конце ребра. Другие ребра можно обойти постепенно. Поэтому информация о каждом ребре напоминает бабочку (рис. 3), из-за чего представление называется "крылатым". Рисунок 4 показывает пример параллелепипеда в "крылатом" представлении. Полные данные по ребру состоят из двух вершин (конечные точки), двух граней (по каждую сторону), и четырех ребер ("крылья" ребра) (таблица ?).

Рендеринг "крылатого" представления графическим оборудованием требует генерирования списка индексов граней. Обычно это делается только когда изменяется геометрия. "Крылатое" представление идеально подходит для динамической геометрии, такой как подразделение поверхностей и интерактивное моделирование, так как изменения сетки могут происходить локально. Обход вокруг сетки, что может пригодиться для обнаружения столкновений, может быть эффективно выполнено.



1. “Крылья” ребра



1. “Крылатое” представление

В таблице 4 представлен пример списка рёбер “Крылатого” представления. Для каждого ребра указаны вершины начала и конца, две грани и “крылья”.

|  |  |  |  |
| --- | --- | --- | --- |
| e0 | v0 v1 | f1 f12 | e9 e23 e10 e20 |
| e1 | v1 v2 | f3 f13 | e11 e20 e12 e21 |
| e2 | v2 v3 | f5 f14 | e13 e21 e14 e22 |
| e3 | v3 v0 | f7 f15 | e15 e22 e8 e23 |
| e4 | v4 v5 | f0 f8 | e19 e8 e16 e9 |
| e5 | v5 v6 | f2 f9 | e16 e10 e17 e11 |
| e6 | v6 v7 | f4 f10 | e17 e12 e18 e13 |
| e7 | v7 v4 | f6 f11 | e18 e14 e19 e15 |
| e8 | v0 v4 | f7 f0 | e3 e9 e7 e4 |
| e9 | v0 v5 | f0 f1 | e8 e0 e4 e10 |
| e10 | v1 v5 | f1 f2 | e0 e11 e9 e5 |
| e11 | v1 v6 | f2 f3 | e10 e1 e5 e12 |
| e12 | v2 v6 | f3 f4 | e1 e13 e11 e6 |
| e13 | v2 v7 | f4 f5 | e12 e2 e6 e14 |
| e14 | v3 v7 | f5 f6 | e2 e15 e13 e7 |
| e15 | v3 v4 | f6 f7 | e14 e3 e7 e15 |
| e16 | v5 v8 | f8 f9 | e4 e5 e19 e17 |
| e17 | v6 v8 | f9 f10 | e5 e6 e16 e18 |
| e18 | v7 v8 | f10 f11 | e6 e7 e17 e19 |
| e19 | v4 v9 | f11 f8 | e7 e4 e18 e16 |
| e20 | v1 v9 | f12 f13 | e0 e1 e23 e21 |
| e21 | v2 v9 | f13 f14 | e1 e2 e20 e22 |
| e22 | v3 v9 | f14 f15 | e2 e3 e21 e23 |
| e23 | v0 v9 | f15 f12 | e3 e0 e22 e20 |

1. Список рёбер

В таблице 5 приведён список граней с отсекающими их рёбрами. Аналогичным образом строится список вершин.

|  |  |
| --- | --- |
| f0 | e4 e8 e9 |
| f1 | e0 e10 e9 |
| f2 | e5 e10 e11 |
| f3 | e1 e12 e11 |
| f4 | e6 e12 e13 |
| f5 | e2 e14 e13 |
| f6 | e7 e14 e15 |

1. Список граней

|  |  |
| --- | --- |
| f7 | e3 e8 e15 |
| f8 | e4 e16 e19 |
| f9 | e5 e17 e16 |
| f10 | e6 e18 e17 |
| f11 | e7 e19 e18 |
| f12 | e0 e23 e21 |
| f13 | e1 e20 e18 |
| f14 | e2 e21 e22 |
| f15 | e3 e22 e23 |

Таблица 5. Список граней (продолжение)

Помимо выше перечисленных есть и другие способы представления полигонной сетки.

Потоковые сетки хранят грани упорядочено, но независимо, так чтобы сетку можно было пересылать по частям. Порядок граней может быть пространственным, спектральным, или базированным на других свойствах сетки. Потоковые сетки позволяют рендерить очень большие сетки даже тогда, когда они ещё загружаются.

Прогрессивные сетки передают данные о вершинах и гранях с повышающимся уровнем детализации. В отличие от потоковых сеток, прогрессивные сетки дают общую форму целого объекта, но на низком уровне детализации. Дополнительные данные, новые ребра и грани, прогрессивно увеличивают детализацию сетки.

Нормальные сетки передают постепенные изменения сетки как множество смещений нормалей от базовой сетки. С помощью этой техники, ряд текстур отображает желаемые нарастающие изменения. Нормальные сетки компактны, так как для выражения смещения нужно лишь одно скалярное значение. Однако, техника требует ряд сложных трансформаций чтобы создать текстуры сдвига.

1. Аффинные преобразования

*Аффинное* преобразование (от [лат.](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D1%82%D0%B8%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA)*affinis* — соприкасающийся, близкий, смежный) — отображение плоскости или пространства в себя, при котором параллельные прямые переходят в параллельные прямые, пересекающиеся в пересекающиеся, скрещивающиеся в скрещивающиеся.

Аффинное преобразование *f: Rn →Rn* есть преобразование вида

где ~M — [обратимая матрица](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%B8%D0%BC%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0) (неособенный [аффинор](https://ru.wikipedia.org/wiki/%D0%90%D1%84%D1%84%D0%B8%D0%BD%D0%BE%D1%80)) и .

Иначе говоря, преобразование называется аффинным, если его можно получить следующим образом:

1. Выбрать «новый» [базис](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B8%D1%81) пространства с «новым» [началом координат](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%87%D0%B0%D0%BB%D0%BE_%D0%BA%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B0%D1%82) *v*
2. Каждой точке xпространства поставить в соответствие точку f(x), имеющую те же координаты относительно «новой» системы координат, что и xв «старой».

Свойства аффинного преобразования:

1. При аффинном преобразовании прямая переходит в прямую.
   1. Если размерность пространства *n* ≥ 2, то любое преобразование пространства (то есть [биекция](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B5%D0%BA%D1%86%D0%B8%D1%8F) пространства на себя), которое переводит прямые в прямые, является аффинным. Это определение используется в [аксиоматическом](https://ru.wikipedia.org/wiki/%D0%90%D0%BA%D1%81%D0%B8%D0%BE%D0%BC%D0%B0) построении [аффинной геометрии](https://ru.wikipedia.org/wiki/%D0%90%D1%84%D1%84%D0%B8%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F)
2. Аффинные преобразования образуют [группу](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D1%83%D0%BF%D0%BF%D0%B0_%28%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29) относительно [композиции](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%B7%D0%B8%D1%86%D0%B8%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B9).
3. Любые три точки, не лежащие на одной прямой и их образы соответственно (не лежащие на одной прямой) однозначно задают аффинное преобразование плоскости.

Аффинное преобразование можно представить как матрицу перехода в однородных координатах.

В декартовых координатах любое аффинное преобразование будет иметь вид:

На практике удобно задавать аффинное преобразование одной матрицей. При этом используются однородные координаты:

Заметим, что первые три значения последней строки равны в матрице преобразования 0. Это **необходимое** **условие** того, что преобразование будет аффинным. В общем случае произвольная матрица размера 4x4 задает проективное преобразование. Такие преобразования используются для проецирования трехмерной сцены.

Частным случаем аффинных преобразований являются просто движения (без какого-либо сжатия или растяжения). Движения — это такие преобразования, которые сохраняют расстояние между любыми двумя точками неизменным, а именно параллельные переносы, повороты, различные симметрии и их комбинации.

Параллельный перенос – преобразование, при котором все точки пространства перемещаются в одном и том же направлении на одно и то же расстояние. Матрица такого преобразования имеет вид:

Поворот – преобразование, при котором пространство точек поворачивается вокруг некоторой прямой на угол α.

Матрица поворота вокруг оси OX:

Матрица поворота вокруг оси OY:

Матрица поворота вокруг оси OZ:

Другой случай аффинных преобразований — это растяжения и сжатия относительно прямой.

Есть еще важный класс аффинных преобразований — это сжатия и растяжения относительно точки. Они называются преобразованиями подобия или гомотетиями. Подобное преобразование получается путём комбинирования преобразований растяжения относительно перпендикулярных прямых, пересекающихся в данной точке.

В общем виде матрица преобразования растяжения имеет вид:

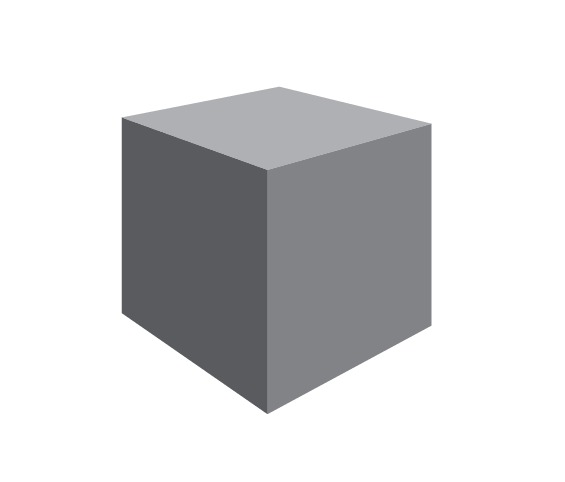
где sx, sy, sz – коэффициенты растяжения по осям OX, OY, OZ соответственно.

Перечисленные выше преобразования (параллельный перенос, поворот, растяжение) являются элементарными аффинными преобразованиями. Сложные аффинные преобразования всегда можно представить как комбинацию элементарных преобразований.

1. Реализация

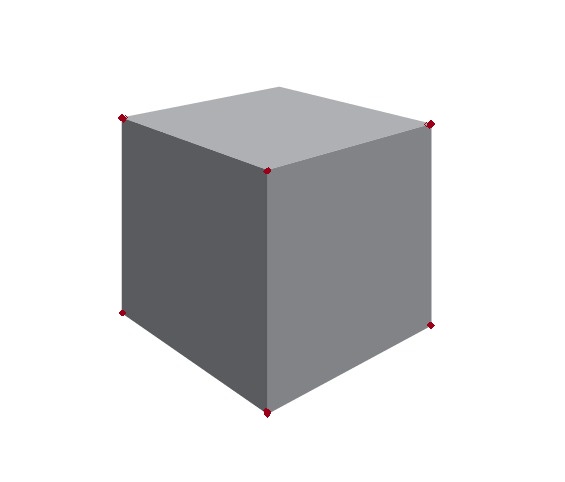
Как уже было сказано выше, мы будем использовать для разработки системы язык программирования Python с подключенными к нему библиотекой OpenCV и математическим модулем NumPy.

В качестве примера было взято изображение куба (рис 6). Координаты точек в трёхмерном пространстве мы зададим вручную.



1. Исходное изображение

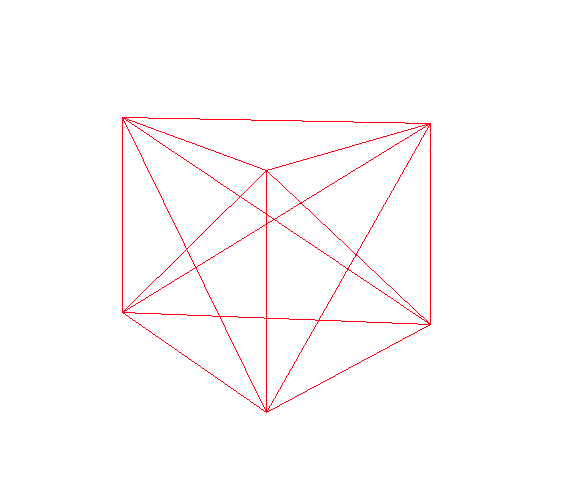
На первом шаге найдём углы объекта на изображении при помощи метода Харриса. Так как метод даёт по несколько точек для каждого угла (рис 7), усредним полученные результаты. Для этого у всех точек, лежащих на малом расстоянии друг от друга, будем брать среднее по координате *x* и по координате *y.* В результате этого для каждой группы точек будет найдена точка аналогичная центру масс. Набор этих точек будем считать углами объекта.



1. Резульат детектора Харриса

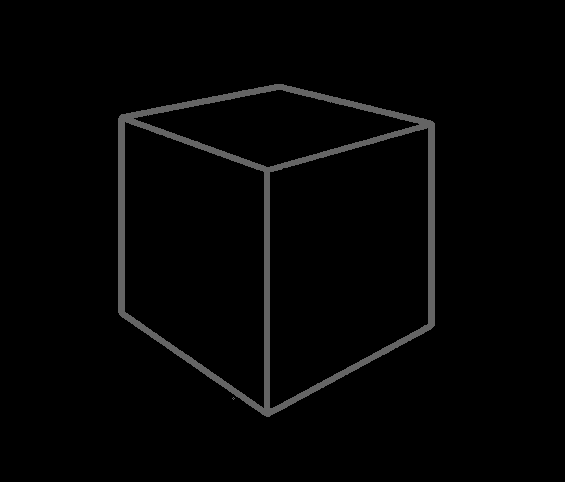
После этого необходимо сопоставить точки, найденные методом Харриса, и точки из трёхмерного облака. Для этого построим проекцию облака на плоскость изображения объекта. Координаты этой плоскости в системе координат облака не сложно найти (по условию нам известно расстояние до объекта и угол, под которым был сделан снимок). Применим аффинный поворот к облаку по каждой оси на соответствующую компоненту этого угла. Мы получили новую систему координат, в которой ось *Oy* перпендикулярна плоскости изображения, в результате чего мы можем воспользоваться ортогональной проекцией. Таким образом, координаты по осям *Ox* и *Oz* есть координаты проекции на плоскость изображения по осям абсцисс и ординат, соответственно. Теперь мы можем привести проекцию и изображение к единому размеру (применив аффинное растяжение) и сопоставить координаты точек.

На следующем шаге необходимо получить полигонную сетку. Мы будем рассматривать только те полигоны, которые видны наблюдателю (так как мы имеем текстуры только для этих полигонов). Для начала разобьём изображение на треугольники, соединив точки углов. (Рис. 8.)

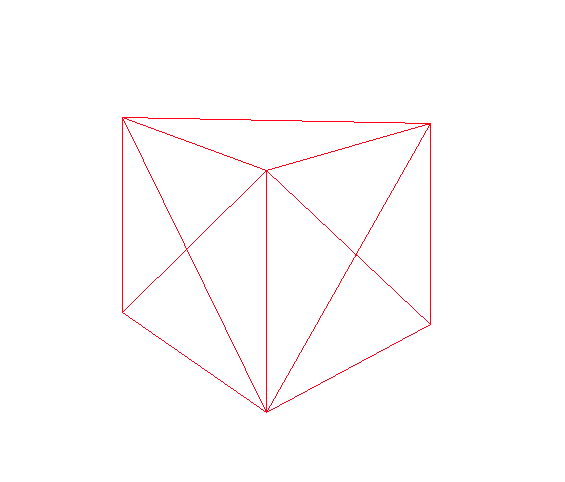


1. Разбиение на треугольники

Теперь необходимо понять, какие из этих треугольников являются гранями объекта. Для этого воспользуемся фильтром на основе лапласиана (Рис. 9.). Пройдя по точкам, отмеченным этим фильтром, мы можем отбросить те треугольники, по которым проходят рёбра объекта (Рис. 10.).



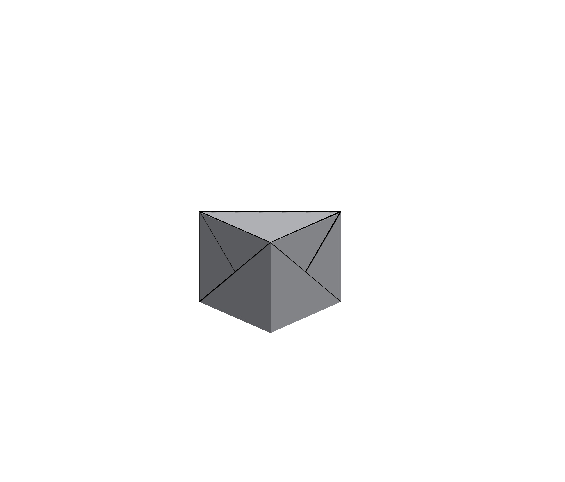
1. Результат применения фильтра



1. Грани объекта

Теперь мы можем перенести части изображения на проекцию трёхмерного облака. Стоит отметить, что это не обязательно проекция на плоскость изображения, о которой говорилось ранее, а любая проекция исходного облака.

Для того, чтобы скопировать грань, построим маску, в которой треугольник с координатами грани будет заполнен белым цветом, а пространство вокруг него – черным. В бинарном виде белый цвет задаётся единицами, а чёрный – нулями. Исходя из этого можно сделать вывод о том, что произведя операцию побитового И (bitwise and) над маской и изображением, мы получим изображение грани на чёрном фоне. Подобным образом наложим маску на проекцию, так, чтобы полигон, на который мы будем накладывать текстуру, стал чёрным. Теперь проведём аффинные преобразования с текстурой, совместив её с проекцией и применим операцию побитового ИЛИ (bitwise or) к проекции и полигону текстуры. Проведя подобные манипуляции с каждой гранью объекта, получим изображение этого объекта в той проекции, которую задали (Рис. 11.).



1. Результат работы программы

Заключение

В ходе работы, были изучены основные алгоритмы поиска локальных особенностей, такие как алгоритм Харриса и применение фильтра на основе лапласиана.

Был проведён анализ различных подходов к построению полигонной сетки и выявлены плюсы и минусы тех из них, которые наиболее часто применяются в настоящее время.

Список литературы