

## Лабораторная работа №2

Форматы представления цифровых изображений и видео.

### Графические файловые форматы

Как уже говорилось ранее, при хранении растровых изображений, как правило, приходится иметь дело с файлами большого размера. В этой связи важной задачей является выбор соответствующего формата файла.

Форматов графических файлов существует великое множество и выбор приемлемого отнюдь не является тривиальной задачей. Для облегчения выбора воспользуемся классификациями. **По типу хранимой графической информации:**

- растровые (TIFF, GIF, BMP, JPEG);
- векторные (AI, CDR, FH7, DXF);
- смешанные/универсальные (EPS, PDF).

Следует учитывать, что файлы практически любого векторного формата позволяют хранить в себе и растровую графику. Однако часто это приводит к

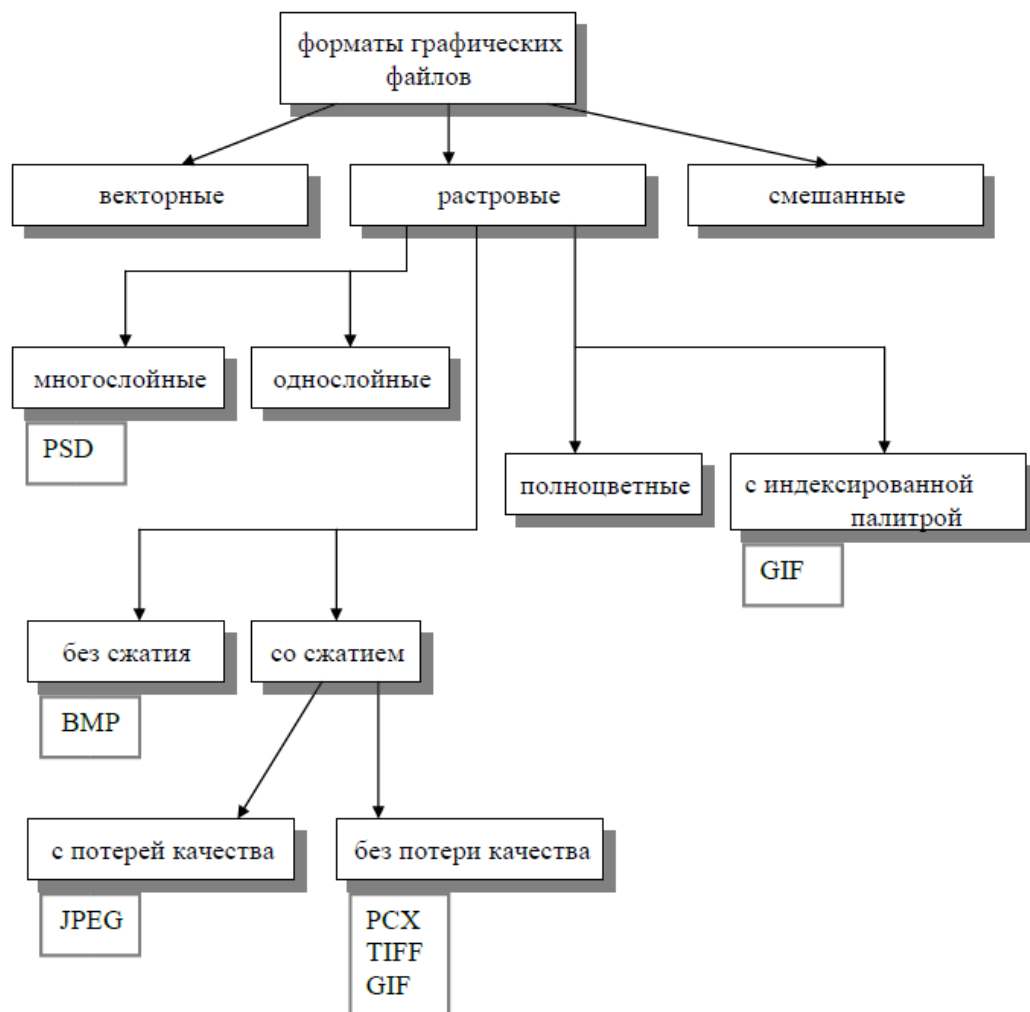


Рис. 3.1. Графические файловые форматы

Скажем несколько слов о наиболее популярных графических форматах.

## 1. BMP

**BMP** (от англ. BitMap Picture) — формат хранения растровых изображений. BMP был создан компанией Microsoft и широко используется в операционных системах семейства Windows.

Глубина цвета в данном формате может быть 1, 2, 4, 8, 16, 24, 32, 48 бит на пиксель, максимальные размеры изображения 65535×65535 пикселей. Однако, глубина 2 бит официально не поддерживается.

Формат BMP является примером хранения *полноцветных изображений*. В этом случае, цвета пикселей можно определять, явно задавая несколько параметров цвета. Например, в RGB-модели конечный цвет каждого пикселя определяется тремя слагаемыми для трех основных цветов.

В формате BMP есть поддержка сжатия по *алгоритму RLE*. Алгоритм RLE или *алгоритм кодирования повторов* оперирует сериями данных, то есть последовательностями, в которых один и тот же символ встречается несколько раз подряд. При кодировании строка одинаковых символов, составляющих серию, заменяется строкой, которая содержит сам повторяющийся символ и количество его повторов.

Рассмотрим изображение, состоящее из строки белых и черных пикселей. Опишем эту строку как:

WWWWBWWWWBBBWWWWBWWWW

Здесь B представляет чёрный пиксель, а W обозначает белый. Если мы применим RLE кодирование к этой строке, то получим следующее:

4W1B4W3B4W1B4W

Файлы формата BMP могут иметь расширения .bmp, .dib и .rle. DIB означает *аппаратно-независимый растр* (Device Independent Bitmap). При использовании этого формата программист может получить доступ ко всем элементам структур, описывающих изображение, при помощи обычного указателя. Но эти данные не используются для непосредственного управления экраном, так как они всегда хранятся в системной памяти, а не в специализированной видеопамяти. Формат пикселя в оперативной памяти может отличаться от того формата, который должен заноситься в видеопамять для индикации точки такого же цвета. Например, в DIB-формате может использоваться 24 бита для задания пикселя, а графический адаптер в этот момент может работать в режиме HighColor с цветовой глубиной 16 бит. При этом ярко-красная точка в аппаратно-независимом формате будет задаваться тремя байтами 0x0000ff, а в видеопамяти — словом 0xF800.

DDB означает *аппаратно-зависимый растр* (Device Dependent Bitmap, DDB). Этот формат всегда содержит цветовые коды, совпадающие с кодами видеобуфера, но храниться он может как в системной, так и в видеопамяти. В обоих случаях он содержит только коды цвета в том формате, который обеспечит пересылку изображения из ОЗУ в видеопамять при помощи простого копирования.

Таким образом, достоинством формата BMP является простота обращения к отдельным пикселям на изображении, что может быть использовано при написании демонстрационных программ по компьютерной графике. К недостаткам нужно отнести сравнительно большие размеры файлов, хранящих изображения в формате BMP, вследствие несовершенства алгоритма RLE.

Кроме поддержки полноцветных изображений формат BMP может обеспечивать хранения изображений с использованием *индексированной палитры*. Второй подход заключается в том, что в первой части файла, хранящего изображение, хранится «*палитра*», в которой с помощью одной из цветовых моделей кодируются цвета, присутствующие на изображении. А вторая часть, которая непосредственно описывает пиксели изображения, фактически состоит из индексов в палитре.

Формат BMP может использовать режим индексирования цветов при следующих значениях глубины цвета: 1 бит (2 цвета), 2 бита (4 цвета), 4 бита (16 цветов), 8 бит (256 цветов).

Благодаря использованию палитры имеется возможность адаптировать изображение к цветам, присутствующим на изображении. В таком случае изображение ограничено не заданными цветами, а максимальным количеством одновременно используемых цветов.

Достоинством палитры является возможность существенно сократить размер файла с изображением. Недостатком является возможность потери цветов при ограниченном размере палитры.

## 2. TIFF

**TIFF** (англ. Tagged Image File Format) — формат хранения растровых графических изображений. Изначально был разработан компанией Aldus в сотрудничестве с Microsoft. TIFF был выбран в качестве основного графического формата операционной системы Mac OS X. В настоящее время авторские права на спецификации формата принадлежат компании Adobe.

Принцип хранения данных в TIFF основан на использовании специальных маркеров (тэгов) в сочетании с битовыми последовательностями кусков растра.

Формат TIFF поддерживает большую глубину цвета: 8, 16, 32 и 64 бит на канал при целочисленном кодировании, а также 32 и 64 бит на канал при представлении значения пикселя числами с плавающей запятой.

Структура формата гибкая и позволяет сохранять изображения в режиме цветов с палитрой, а также в различных цветовых пространствах: бинарном (двухцветном), полутоновом, с индексированной палитрой, RGB, CMYK, YCbCr, CIE Lab. Помимо традиционных цветов CMY формат поддерживает цветоделение с большим числом красок, в частности систему Hexachrome компании Pantone. В систему Hexachrome, известную как CMYKOG, добавлены оранжевые и зеленые краски для лучшего представления цветов при печати.

Помимо прочих достоинств формат TIFF позволяет сохранять растровые изображения с компрессией без потери качества. Этот формат поддерживает сжатие без потери качества по **алгоритму LZW компрессии**.



Алгоритм Лемпеля — Зива — Велча (Lempel-Ziv-Welch, LZW) — это универсальный алгоритм сжатия данных без потерь данных. Алгоритм на удивление прост. Если в двух словах, то LZW-сжатие заменяет строки символов некоторыми кодами. Это делается без какого-либо анализа входного текста. Вместо этого при добавлении каждой новой строки символов просматривается уже существующая таблица строк. Сжатие происходит, когда код заменяет строку символов. Коды, генерируемые LZW-алгоритмом, могут быть любой длины, но они должны содержать больше бит, чем единичный символ. Первые 256 кодов (когда используются 8-битные символы) по умолчанию соответствуют стандартному набору символов. Остальные коды соответствуют обрабатываемым алгоритмом строкам. Простой метод может работать с 12-битными кодами. Значения кодов 0 - 255 соответствуют отдельным байтам, а коды 256 - 4095 соответствуют подстрокам.

Приведем пример LZW кодирования применительно к изображению. Так, если в изображении имеются наборы из розового, оранжевого и зелёного пикселей, повторяющиеся 50 раз, LZW выявляет это, присваивает данному набору отдельное число (например, 7) и затем сохраняет эти данные 50 раз в виде числа 7. Метод LZW, так же, как и RLE, лучше действует на участках однородных, свободных от шума цветов, он действует гораздо лучше, чем RLE, при сжатии произвольных графических данных, но процесс кодирования и распаковки происходит медленнее.

Кроме LZW-компрессии формат TIFF поддерживает следующие алгоритмы сжатия:

- RLE;
- LZ77;
- ZIP;
- JBIG;
- JPEG;
- CCITT Group 3, CCITT Group 4.

При этом JPEG является просто инкапсуляцией формата JPEG в формат TIFF. Формат TIFF позволяет хранить изображения, сжатые по стандарту JPEG, без потерь данных (JPEG-LS).

### 3. GIF

Первая версия формата GIF (Graphics Interchange Format, «Формат для обмена графической информацией») была разработана в 1987 г. специалистами компьютерной сети CompuServe. Этот формат сочетает в себе редкий набор достоинств, неоценимых при той роли, которую он играет в WWW. Сам по себе формат содержит уже достаточно хорошо упакованные графические данные.

Формат GIF использует для хранения изображений индексированную палитру, ограничивающую количество цветов 256 значениями. Размер палитры может быть и меньше. Если в изображении используется, скажем, 64 цвета (26), то для хранения каждого пикселя будет использовано ровно шесть бит и ни битом больше.

Поскольку GIF использует для сжатия LZW алгоритм, то степень сжатия графической информации сильно зависит от уровня ее повторяемости и предсказуемости, а иногда еще и от ориентации картинка. Так как LZW метод сканирует изображение по строкам, то, к примеру, плавный переход цветов (градиент), направленный сверху вниз, сожмется куда лучше, чем тех же размеров градиент, ориентированный слева направо, а последний – лучше, чем градиент по диагонали.

Изменив порядок следования данных в файле, создатели GIFa заставили картинку рисоваться не только сверху вниз, но и, если можно так выразиться, «с глубины к поверхности», – то есть становиться все четче и детальнее по мере подхода из сети новых данных.

Для этого файл с изображением тасуется при записи так, чтобы сначала шли все строки пикселей с номерами, кратными восьми (первый проход), затем четырем (второй проход), потом двум и, наконец, последний проход – все оставшиеся строки с нечетными номерами. Во время приема и декодирования такого файла каждый следующий проход заполняет «дыры» в предыдущих, постепенно приближая изображение к исходному состоянию. Поэтому такие изображения были названы *чересстрочными (interlaced)*.

Другой полезной возможностью формата является использование прозрачности и поддержка анимационных изображений. Для создания анимации используется нескольких статичных кадров, а также информация о том, сколько времени каждый кадр должен быть показан на экране.

## 4. PNG

**PNG** (portable network graphics) - растровый формат хранения графической информации, использующий сжатие без потерь по алгоритму Deflate.

PNG поддерживает три основных типа растровых изображений:

- Полутоновое изображение (с глубиной цвета 16 бит);
- Цветное индексированное изображение (палитра 8 бит для цвета глубиной 24 бит);
- Полноцветное изображение (с глубиной цвета 48 бит).

Формат PNG спроектирован для замены устаревшего и более простого формата GIF, а также, в некоторой степени, для замены значительно более сложного формата TIFF. Алгоритм сжатия Deflate является свободным в отличие от защищенного патентами и соответственно платного LZW, используемого в GIF. В отличие от LZW кодирования, которое позволяет эффективно сжимать горизонтальные одноцветные области, с Deflate сжатием можно забыть про эти ограничения.

В формате GIF один из цветов в палитре может быть объявлен «прозрачным». В этом случае в программах, которые поддерживают прозрачность GIF (например, большинство современных браузеров) сквозь пиксели, окрашенные «прозрачным» цветом будет виден фон. Однако создатели PNG пошли еще дальше, разработав технологию *альфа-канала*. Альфа-канал позволяет добиться эффекта частичной прозрачности пикселей. Также, в PNG поддерживается гамма-коррекция.

**Гамма-коррекция** - коррекция функции яркости в зависимости от характеристик устройства вывода. Повышение показателя гамма-коррекции позволяет повысить контрастность, разборчивость тёмных участков изображения, не делая при этом чрезмерно контрастными или яркими светлые детали снимка.

Кроме этого PNG формат имеет следующие преимущества перед GIF:

- практически неограниченное количество цветов в изображении (GIF использует в лучшем случае 8-битный цвет);
- двумерная чересстрочная развёртка;
- возможность расширения формата пользовательскими блоками.

К недостаткам формата можно отнести, что PNG изначально был предназначен лишь для хранения одного изображения в одном файле. Поэтому данный формат не поддерживает анимацию. Для решения этой проблемы создана модификация формата APNG.

## 5. JPEG

**JPEG** (Joint Photographic Experts Group, по названию организации разработчика) - один из популярных графических форматов, основанный на алгоритме сжатия JPEG и применяемый для хранения фотоизображений и подобных им изображений.

Алгоритм JPEG разработан группой экспертов из Международной организации по стандартизации (ISO) специально для сжатия полноцветных 24-битовых изображений.

Процесс сжатия по схеме JPEG состоит из нескольких шагов. На первом шаге производится преобразование изображения из цветовой модели RGB в модель YUV, основанной на характеристиках яркости и цветности.

В модели YUV, Y-компонента – светлота (яркость). Компоненты U и V содержат информацию о цвете.

На следующем после преобразования шаге изображение разделяется на квадратные участки размером 8x8 пикселей. После этого над каждым участком производится дискретное косинус-преобразование (ДКП). При этом выполняется анализ каждого блока, разложение его на составляющие цвета и подсчет частоты появления каждого цвета.



## Формат GIF и фрактальное сжатие

Фрактальная архивация основана на том, что изображение представляется в более компактной форме — с помощью коэффициентов системы *итеративных функций* (Iterated Function System — IFS). IFS представляет собой набор трехмерных аффинных преобразований, в нашем случае переводящих одно изображение в другое. Преобразованию подвергаются точки в трехмерном пространстве ( $x$ -координата,  $y$ -координата, яркость).

Рассмотрим гипотетическую Фотокопировальную Машину, состоящую из экрана, на котором изображена исходная картинка, и системы линз, проецирующих изображение на другой экран. Линзы могут проецировать часть изображения произвольной формы в любое другое место нового изображения. При этом:

- а) Области, в которые проецируются изображения, не пересекаются;
- б) Линза может менять яркость и уменьшать контрастность;
- в) Линза может зеркально отражать и поворачивать свой фрагмент изображения;
- г) Линза должна масштабировать (уменьшать) свой фрагмент изображения.

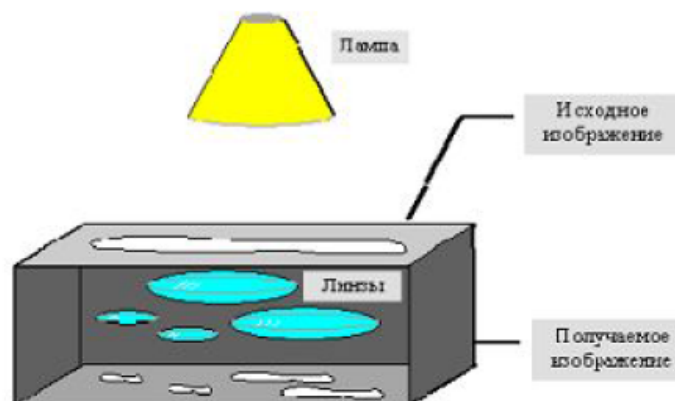


Рисунок 81 – Гипотетическая фотокопировальная машина.

Расставляя линзы и меняя их характеристики, мы можем управлять получаемым изображением. Одна итерация работы Машины заключается в том, что по исходному изображению с помощью проектирования строится новое, после чего новое берется в качестве исходного. Утверждается, что в процессе итераций мы получим изображение, которое перестанет изменяться. Оно будет зависеть только от расположения и характеристик линз, и не будет зависеть от исходной картинки. Это изображение называется “неподвижной точкой” или *аттрактором* данной IFS. Соответствующая теория гарантирует наличие ровно одной неподвижной точки для каждой IFS.

Поскольку отображение линз является сжимающим, каждая линза в явном виде задает самоподобные области в нашем изображении. Благодаря самоподобию мы получаем сложную структуру изображения при любом увеличении. Таким образом, интуитивно понятно, что система итеративных функций задает фрактал.

Для фрактального алгоритма компрессии, как и для других алгоритмов сжатия с потерями, очень важны механизмы, с помощью которых можно будет регулировать степень сжатия и степень потерь. К настоящему времени разработан достаточно большой набор таких методов. Во-первых, можно ограничить количество аффинных преобразований, заведомо обеспечив степень сжатия не ниже фиксированной величины. Во-вторых, можно потребовать, чтобы в ситуации, когда разница между обрабатываемым фрагментом и наилучшим его приближением будет выше определенного порогового значения, этот фрагмент дробился обязательно (для него обязательно заводится несколько “линз”). В-третьих, можно запретить дробить фрагменты размером меньше, допустим, четырех точек. Изменяя пороговые значения и приоритет этих условий, можно гибко управлять коэффициентом компрессии изображения в диапазоне от побитового соответствия до любой степени сжатия.

Формат графических файлов назван FIF (Fractal Image Format) и запатентован фирмой Iterated Systems. Например, закодировав какое-то изображение двумя аффинными преобразованиями, мы однозначно определяем его с помощью 12-ти коэффициентов. Если теперь задаться какой-либо начальной точкой (например,  $X=0$   $Y=0$ ) и запустить итерационный процесс, то мы после первой итерации получим две точки, после второй - четыре, после третьей - восемь и т.д. Через несколько десятков итераций совокупность полученных точек будет описывать закодированное изображение. Но проблема состоит в том, что очень трудно найти коэффициенты IFS, которые кодировали бы произвольное изображение.



*Характеристики фрактального алгоритма:*

*Коэффициенты компрессии:* 2-2000 (Задается пользователем).

*Класс изображений:* Полноцветные 24 битные изображения или изображения в градациях серого без резких переходов цветов (фотографии). Желательно, чтобы области большей значимости (для восприятия) были более контрастными и резкими, а области меньшей значимости — неконтрастными и размытыми.

*Симметричность:* 100-100000

*Характерные особенности:* Может свободно масштабировать изображение при разархивации, увеличивая его в 2-4 раза без появления “лестничного эффекта”. При увеличении степени компрессии появляется “блочный” эффект на границах блоков в изображении.

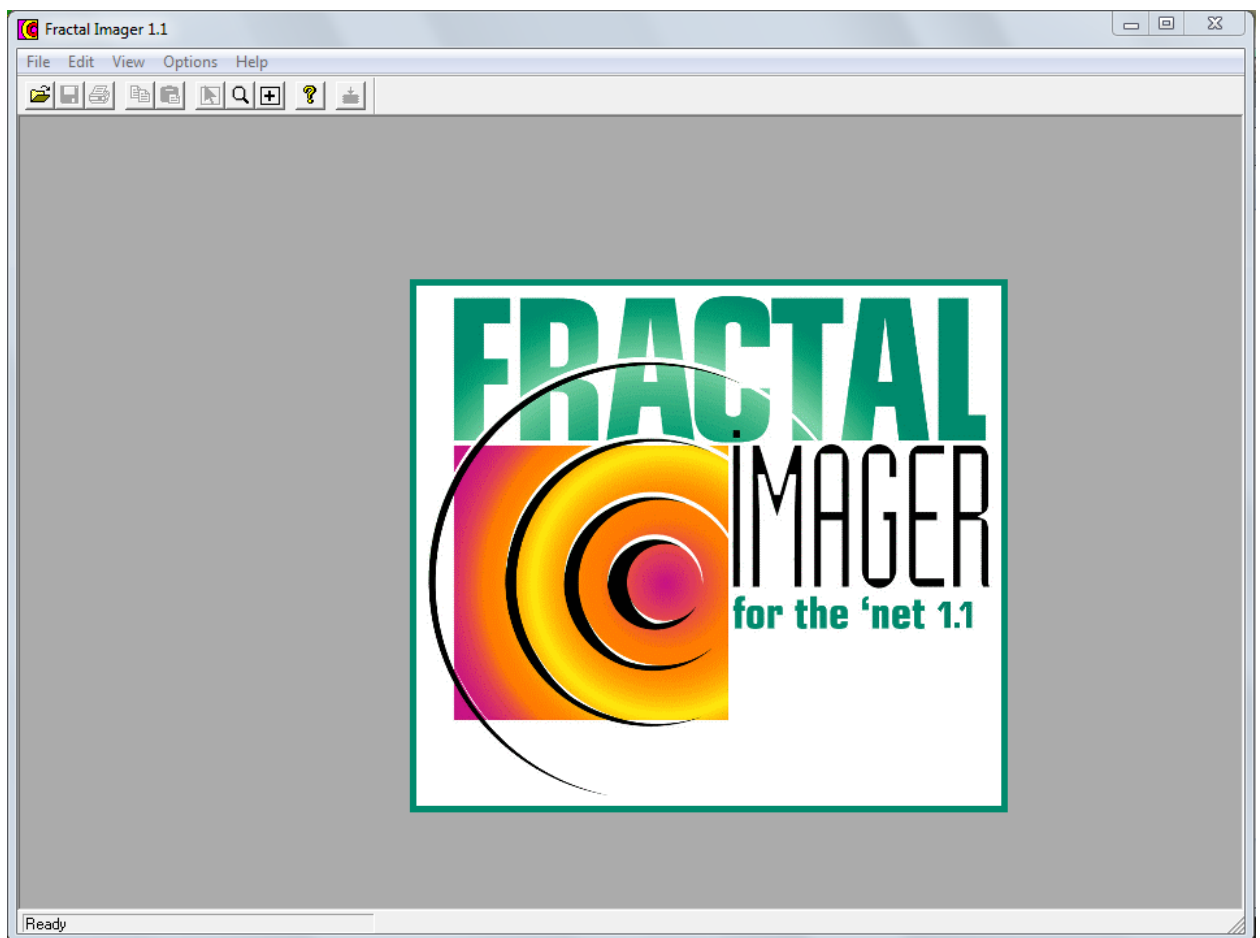


Рис. 3. Приложение фрактального сжатия изображений Fractal imager от Iterated Systems.

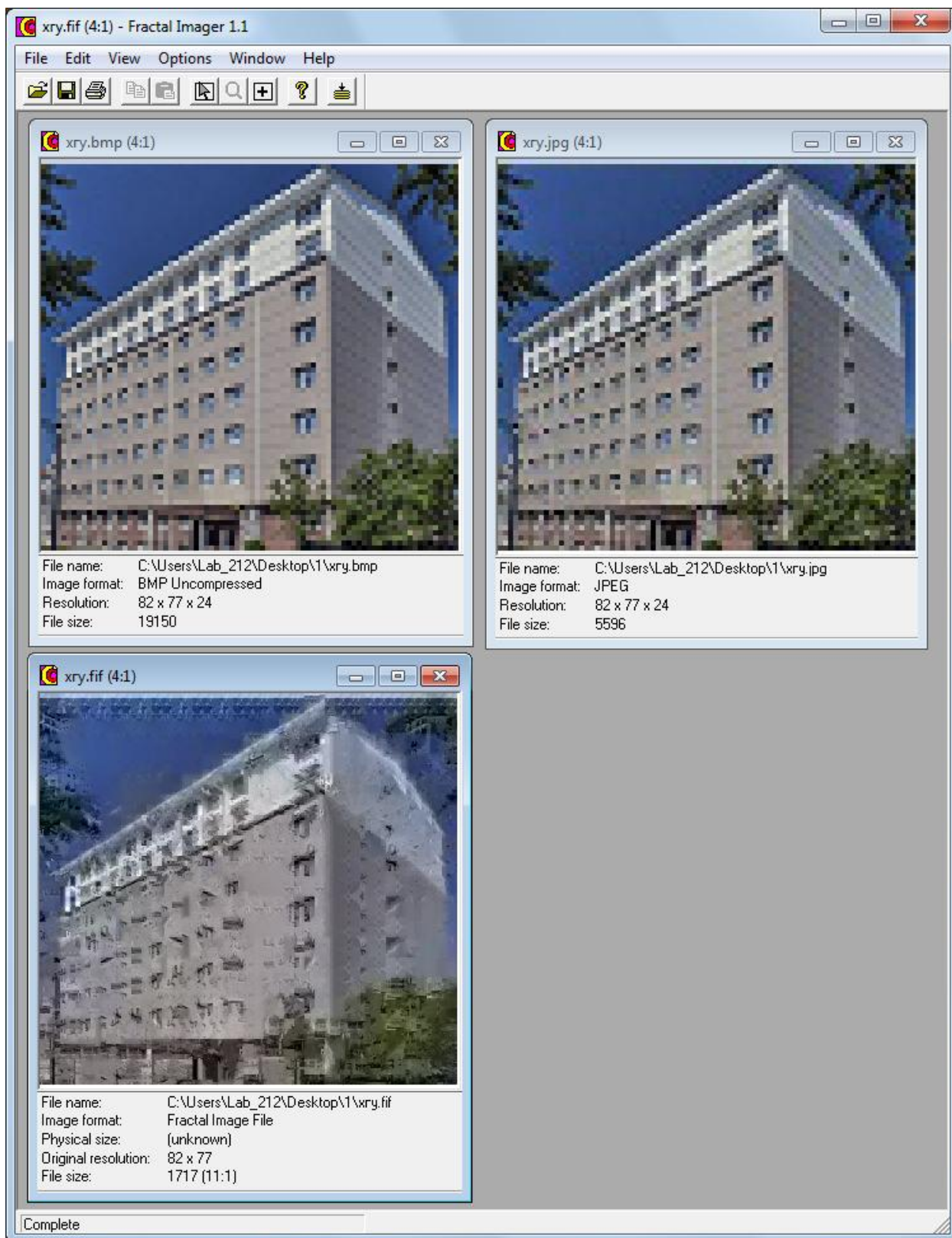


Рис. 4. Сравнение изображений, представленных в различных форматах

# 1. Структура bmp-файла

Если упрощенно, в начале файла идут заголовки, потом уже сами бинарные данные:

## Общая структура

Данные в формате BMP состоят из трёх основных блоков различного размера:

1. Заголовок из структуры BITMAPFILEHEADER и блока BITMAPINFO. Последний содержит:
  - Информационные поля.
  - Битовые маски для извлечения значений цветовых каналов (опциональные).
  - Таблица цветов (опциональная).
2. Цветовой профиль (опциональный).
3. Пиксельные данные.

По сути, чтобы добраться до описания пиксельных данных, нужно пропустить все заголовки.

Описание структуры BITMAPFILEHEADER (которая, собственно, идет с начала файла BMP):

- 2 байта - `bftype` - сигнатура формата. По сути - две буквы **BM**
- 4 байта - `bfsizе` - размер файла в байтах
- 4 байта - поля `bfReserved1`, `bfReserved2` - зарезервировано (не используется, в 99% случаев будет заполнено нулями)
- 4 байта - `bfoffbits` - Положение пиксельных данных относительно начала данной структуры (в байтах).



	bfType (сигнатура)	btSize (размер файла)	btReserved1, btReserved2 (не используется)	bfOffBits (смещение начала пиксельных данных)	
00000000	42 4D	36 93 D5 00	00 00 00 00	36 00 00 00	28 00 00
00000018	00 00 01 00	18 00	00 00 00 00	00 00 93 D5	00 00 00 00
00000030	00 00 00 00	00 00 00 00	FF FF FF FF	FF FF FF FF	FF FF FF FF
00000048	FF FF FF FF	FF FF FF FF	FF FF FF FF	FF FF FF FF	FF FF FF FF
00000060	FF FF FF FF	FF FF FF FF	FF FF FF FF	FF FF FF FF	FF FF FF FF

В поле `bfOffBits` в файле записано число 0x36, и по смещению 0x36 от начала файла подряд идут значения FF. Тройка байт FF FF FF (8 бит \* 3 = 24 бита на пиксель) как раз кодируют один пиксель белого цвета, т.е. это как раз и есть искомые пиксельные данные.

Что еще желательно знать? Во-первых при чтении нужно проверить, что картинка действительно имеет формат RGB с 24 битами на пиксель. Во-вторых нужно знать размер картинки.

Для этого нужно смотреть что идет следом за структурой. А следом идет структура `BITMAPINFO`, но сложность в том, что она может быть разных версий. Версия задается в первых четырех байтах. Самая старая версия имела размер 12 байт, более новые от 40 байт и выше. Будем для простоты считать, что поддерживаем только новые форматы BMP, тем более что начало структуры `BITMAPINFO` у новых форматов совпадает.

Итак, в структурах `BITMAPINFO` новых форматах по порядку идут:

- 4 байта - `biSize` - размер структуры, нужен для определения формата структуры
- 4 байта - `biWidth` - ширина изображения
- 4 байта - `biHeight` - высота изображения
- 2 байта - `biPlanes` - количество "плоскостей" (в BMP не используется, а используется в курсорах и значках)
- 2 байта - `biBitCount` - количество бит на пиксель
- 4 байта - `biCompression` - использование сжатия (может использоваться, например, [RLE](#) или JPEG сжатие)
- 4 байта - `biSizeImage` - размер пиксельных данных. В простейшем случае (без сжатия, например) будет равняться `biWidth*biHeight*biBitCount/8`.
- прочая информация...

Снова смотрим на скриншот:



Отсюда получаем:

- Размер структуры `BITMAPINFO` - `0x28` (40 байт) - соответствует версии структуры `BITMAPINFOHEADER`
- Ширина изображения - `40 0B 00 00` = `0x0B << 8 | 0x40` = 2880 пикселей
- Высота изображения - `54 06 00 00` = `0x06 << 8 | 0x54` = 1620 пикселей
- `biPlanes` = 1
- `biBitCount` - `0x18` = 24 бит на пиксель
- `biCompression` - 0 (без сжатия)
- `biSizeImage` - `00 93 D5 00` = `0xD5 << 16 | 0x93 << 8 | 0` = 13996800, то же самое если посчитать `2880*1620*24/8 = 13996800`

**Задание 1.** Открыть `bmp`-файл в любом шестнадцатеричном редакторе. Определить размер, высоту и ширину изображения. Определить размер пиксельных данных и смещение начала пиксельных данных.

## 2.

### **Представление изображения в виде массива**

Массивы в NumPy многомерные и могут использоваться для представления векторов, матриц и изображений. Массив очень похож на список (или список списков), но может содержать только элементы одного типа. Если тип не указан при создании массива, то он автоматически выводится из данных.

Ниже показано, как это делается для изображений:

```
im = array(Image.open('empire.jpg'))
print im.shape, im.dtype

im = array(Image.open('empire.jpg').convert('L'), 'f')
print im.shape, im.dtype
```

На консоли будет напечатано:

```
(800, 569, 3) uint8
(800, 569) float32
```

Первый кортеж описывает форму массива изображения (количество строк, столбцов и цветовых каналов), а следующая за ним строка – тип данных, хранящихся в элементах массива. Изображения обычно кодируются 8-разрядными целыми без знака (uint8), поэтому в первом случае после загрузки изображения и преобразования его в массив печатается тип «uint8». Во втором случае производится преобразование в полутоновое изображение, и при создании массива указан дополнительный аргумент «f». Эта короткая команда означает, что нужно использовать тип с плавающей точкой.

Обратите внимание, что для полутоновых изображений кортеж, описывающий форму, содержит только два элемента; понятно, что информация о цвете не нужна.

К элементам массива можно обращаться по индексам. Чтобы получить значение с координатами  $i, j$  и цветовым каналом  $k$ , нужно написать:

```
value = im[i,j,k]
```



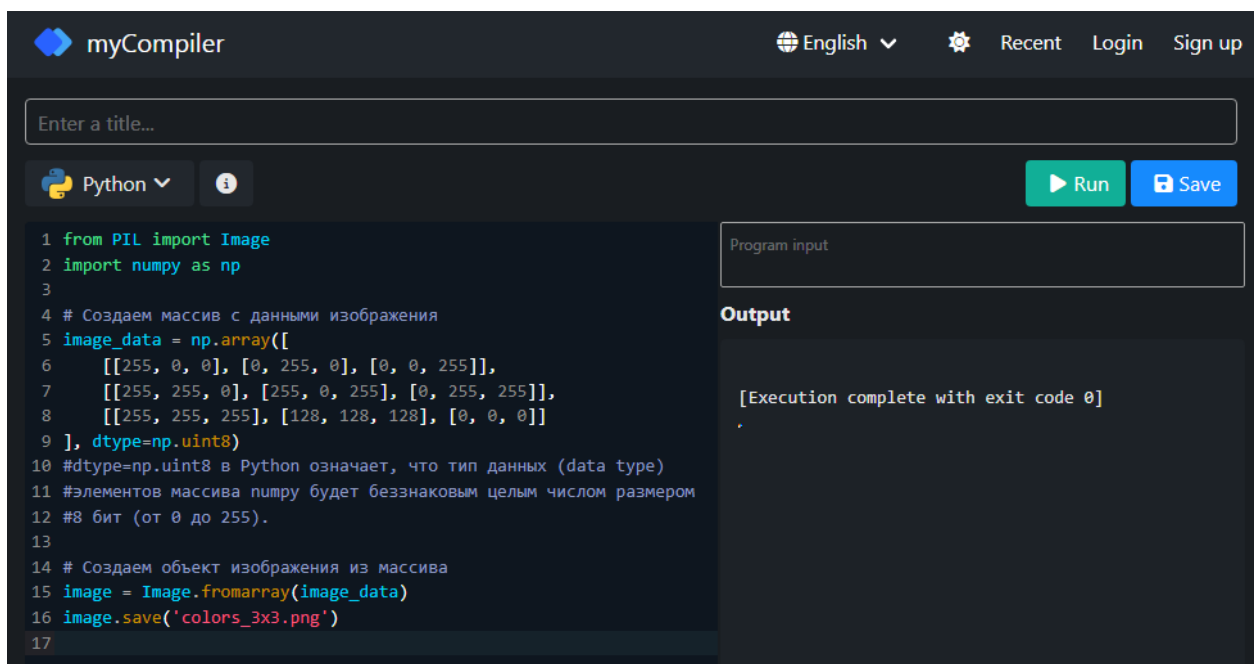
Операция срезки массива позволяет обращаться сразу к нескольким элементам. Она возвращает представление части массива, определяемое интервалами индексов. Вот несколько примеров для полутонового изображения:

```
im[i, :] = im[j, :]    # скопировать значения из строки j в строку i
im[:, i] = 100         # присвоить всем элементам в столбце i значение 100
im[:100, :50].sum()   # просуммировать элементы в прямоугольнике,
                        # образованном первыми 100 строками
                        # и первыми 50 столбцами
im[50:100, 50:100]    # строки 50-100, столбцы 50-100 (сотые не
                        # включаются)
im[i].mean()           # среднее значение в строке i
im[:, -1]              # последний столбец
im[-2, :] (или im[-2]) # предпоследняя строка
```

Обратите внимание на пример, где указан только один индекс. В таком случае он интерпретируется как индекс строки. Также обратите внимание на последние два примера. Отрицательный индекс отсчитывается от последнего элемента в обратном направлении. Мы часто будем использовать срежку для доступа к значениям пикселей, поэтому важно понимать, как она работает.

Есть много способов работы с массивами. Мы будем знакомиться с ними по ходу изложения.

**Задание 2.** Привести изображение, полученное в результате выполнения следующего кода:




The screenshot shows the myCompiler web interface. At the top, there's a header with the logo, language selection (English), settings, and user options (Recent, Login, Sign up). Below the header is a text input field for a title. The main area is divided into two sections: a code editor on the left and an output panel on the right. The code editor contains Python code that creates a 3x3 numpy array with specific values and saves it as a 3x3 PNG image. The output panel shows the message "[Execution complete with exit code 0]".



```
1 from PIL import Image
2 import numpy as np
3
4 # Создаем массив с данными изображения
5 image_data = np.array([
6     [[255, 0, 0], [0, 255, 0], [0, 0, 255]],
7     [[255, 255, 0], [255, 0, 255], [0, 255, 255]],
8     [[255, 255, 255], [128, 128, 128], [0, 0, 0]]
9 ], dtype=np.uint8)
10 #dtype=np.uint8 в Python означает, что тип данных (data type)
11 #элементов массива numpy будет беззнаковым целым числом размером
12 #8 бит (от 0 до 255).
13
14 # Создаем объект изображения из массива
15 image = Image.fromarray(image_data)
16 image.save('colors_3x3.png')
17
```

Output




[Execution complete with exit code 0]


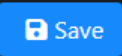


 myCompiler

English   Recent Login Sign up

Enter a title...

 Python  


 

```
1 from PIL import Image
2 import binascii
3
4 # Данные в формате hex с пробелами
5 hex_data = "FF D8 FF E0 00 10 4A 46 49 46 00 01
6
7 # Удаляем все пробелы из строки
8 hex_data = hex_data.replace(" ", "")
9
10 # Преобразуем hex строку в байты
11 binary_data = binascii.unhexlify(hex_data)
12
13 # Сохраняем байты в файл
14 with open("cveta.jpg", "wb") as file:
15     file.write(binary_data)
16
17 pil_im=Image.open("cveta.jpg")
18 pil_im.save('output_image.png')
19
```

Program input

**Output**

[Execution complete with exit code 0]



**Задание 5\*. Имеется произвольный поврежденный bmp-файл (сохранена только область пиксельных данных). Написать код, способный правильно вывести данное изображение.**