

Лабораторная работа №1

Изображение. Кодирование и типы цифровых изображений.

ВВЕДЕНИЕ

Если заглянуть в историю, то можно проследить, как с момента появления первых ЭВМ люди стремятся разнообразить способы общения человека и машины, приблизившись к уровню общения человека с человеком. Это общение было бы гораздо более ограниченным, если бы не использовало один из наиболее простых способов — язык изображений, образов. Сегодня графические изображения на экране монитора современного персонального компьютера стали для нас нормой, совершенно неотъемлемым атрибутом интерфейса. Спектр применения компьютерной графики, помимо средства интерфейса «человек-машина», чрезвычайно широк: от создания рекламных роликов, компьютерных мультфильмов и игр, кроя одежды, малых и монументальных форм дизайна, компьютерной живописи до визуализации результатов научных изысканий. Можно с уверенностью сказать, что популярность Internet, и в частности WWW, во многом объясняется широким применением графики.

Рынок программного и аппаратного обеспечения компьютерной графики — один из самых динамичных. Об этом можно судить по объему литературы и числу сервисов Internet, посвященных так или иначе компьютерной графике.

Предметом данной работы является обширная область компьютерных наук, посвященная представлению данных в памяти ЭВМ в графической форме. Это самое общее определение, так как под данными можно понимать как непосредственно хранящееся в виде файла изображение в одном из графических форматов, так и протокол обмена командами между пользователем и ЭВМ (то, что мы называем графическим интерфейсом), и битовую последовательность, сформированную для вывода на экран или печатающее устройство.

Методы и способы представления и манипуляции этим видом данных относятся к компетенции компьютерной графики.

В работе рассматриваются различные способы представления изображений в памяти ЭВМ, методы и алгоритмы растеризации и обработки растровых изображений, матричные преобразования на плоскости и в пространстве.

1. Способы представления изображений в ЭВМ

Компьютерная (машинная) графика – область деятельности, изучающая создание, способы хранения и обработки изображений с помощью ЭВМ. Под *интерактивной* компьютерной графикой понимают раздел компьютерной графики, изучающий вопросы динамического управления со стороны пользователя содержанием изображения, его формой, размерами и цветом на экране с помощью интерактивных устройств взаимодействия. Кроме интерактивной в компьютерной графике выделяют разделы, изучающие методы работы с изображением на плоскости, так называемую *2D графику*, и *трехмерную (3D) графику*.

Трехмерное изображение отличается от двухмерного, тем, что строится исходя из математического описания некоторой трехмерной сцены. Математическое описание сцены чаще всего является моделью физических объектов в трехмерном пространстве. Таким образом, для получения трехмерного изображения требуется построить математическую модель сцены и объектов на ней, а далее визуализировать путем получения проекции с учетом освещения материалов и пр. В результате визуализации мы получим изображение на плоскости экрана или на выходе принтера.

Вопросы 2D, 3D графики, общего геометрического моделирования, связанные с визуализацией геометрических моделей входят в компетенцию *компьютерной геометрии*.

На специализацию в отдельных областях указывают названия некоторых разделов: инженерная графика, научная графика, Web-графика, компьютерная полиграфия и прочие.

Кроме этого, по способу представления изображения в памяти ЭВМ, компьютерную графику разделяют на *векторную, растровую и фрактальную*. Рассмотрим подробнее эти способы представления изображений, выделим их основные параметры и определим их достоинства и недостатки.

1.1. Растровое представление изображений

Что такое растровое изображение?

Возьмём фотографию (например, см. рис. 1.1). Конечно, она тоже состоит из маленьких элементов, но будем считать, что отдельные элементы мы рассмотреть не можем. Она представляется для нас, как реальная картина природы.

Теперь наложим на изображение прямоугольную сетку. Таким образом, разобьем изображение на прямоугольные элементы. Каждый прямоугольник закрасим цветом, преобладающим в нём (на самом деле программы при оцифровке генерируют некий «средний» цвет, т. е. если у нас была одна чёрная точка и одна белая, то прямоугольник будет иметь серый цвет).

Как мы видим, изображение стало состоять из конечного числа прямоугольников определённого цвета. Эти прямоугольники называют pixel (от PIX ELeмент) – *пиксел* или *пиксель*.



Рис. 1.1. Исходное изображение

Теперь каким-либо методом занумеруем цвета. Конкретная реализация этих методов нас пока не интересует. Для нас сейчас важно то, что каждый пиксель на рисунке стал иметь определённый цвет, обозначенный числом (рис. 1.2).

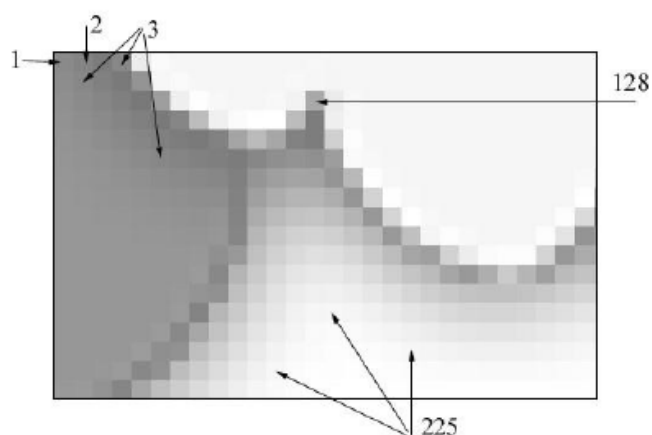


Рис. 1.2. Фрагмент оцифрованного изображения и номера цветов

Теперь пойдём по порядку (слева направо и сверху вниз) и будем в строчку выписывать номера цветов встречающихся пикселей. Получится строка примерно следующего вида:

1 2 8 3 212 45 67 45 127 4 78 225 34 ...

Вот эта строка и есть наши оцифрованные данные. Теперь мы можем сжать их (так как несжатые графические данные обычно имеют достаточно большой размер) и сохранить в файл.

Итак, под *растровым* (bitmap, raster) понимают способ представления изображения в виде совокупности отдельных точек (пикселей) различных цветов

или оттенков. Это наиболее простой способ представления изображения, ибо таким образом видит наш глаз.

Достоинством такого способа является возможность получения фотореалистичного изображения высокого качества в различном цветовом диапазоне. Высокая точность и широкий цветовой диапазон требуют увеличения объема файла для хранения изображения и оперативной памяти для его обработки, что можно отнести к недостаткам. Вторым существенным недостатком является потеря качества изображения при его масштабировании.

1.2. Векторное представление изображений

Для *векторной* графики характерно разбиение изображения на ряд графических примитивов – точки, прямые, ломаные, дуги, полигоны. Таким образом, появляется возможность хранить не все точки изображения, а координаты узлов примитивов и их свойства (цвет, связь с другими узлами и т. д.).

Вернемся к изображению на рис. 1.1. Взглянем на него по-другому. На изображении легко можно выделить множество простых объектов — отрезки прямых, ломанные, эллипс, замкнутые кривые. Представим себе, что пространство рисунка существует в некоторой координатной системе. Тогда можно описать это изображение, как совокупность простых объектов, вышеперечисленных типов, координаты узлов которых заданы вектором относительно точки начала координат (рис. 1.5).

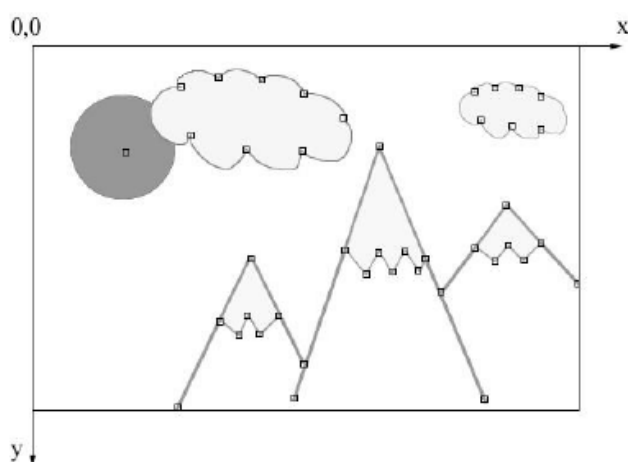


Рис. 1.5. Векторное изображение и узлы его примитивов

Проще говоря, чтобы компьютер нарисовал прямую, нужны координаты двух точек, которые связываются по кратчайшей прямой. Для дуги задается радиус и т. д. Таким образом, векторная иллюстрация – это набор геометрических примитивов.

Важной деталью является то, что объекты задаются независимо друг от друга и, следовательно, могут перекрываться между собой.

При использовании векторного представления изображение хранится в памяти как база данных описаний примитивов. Основные графические примитивы, используемые в векторных графических редакторах: точка, прямая, кривая Безье, эллипс (окружность), полигон (прямоугольник). Примитив строится вокруг его узлов (*nodes*). Координаты узлов задаются относительно координатной системы макета.

А изображение будет представлять из себя массив описаний – нечто типа:

- отрезок (20,20-100,80);
- окружность (50,40-30);
- кривая_Безье (20,20-50,30-100,50).

Каждому узлу приписывается группа параметров, в зависимости от типа примитива, которые задают его геометрию относительно узла. Например, окруж-

ность задается одним узлом и одним параметром – радиусом. Такой набор параметров, которые играют роль коэффициентов и других величин в уравнениях и аналитических соотношениях объекта данного типа, называют аналитической моделью примитива. Отрисовать примитив – значит построить его геометрическую форму по его параметрам согласно его аналитической модели.

Векторное изображение может быть легко масштабировано без потери деталей, так как это требует пересчета сравнительно небольшого числа координат узлов. Другой термин – *«object-oriented graphics»*.

Самой простой аналогией векторного изображения может служить аппликация. Все изображение состоит из отдельных кусочков различной формы и цвета (даже части растра), «склеенных» между собой. Понятно, что таким образом трудно получить фотореалистичное изображение, так как на нем сложно выделить конечное число примитивов, однако существенными достоинствами векторного способа представления изображения, по сравнению с растровым, являются:

- векторное изображение может быть легко масштабировано без потери качества, так как это требует пересчета сравнительно небольшого числа координат узлов;
- графические файлы, в которых хранятся векторные изображения, имеют существенно меньший, по сравнению с растровыми, объем (порядка нескольких килобайт).

На самом деле размер векторного изображения зависит от количества объектов на изображении. И чем ближе качество векторного рисунка будет приближаться к фотореалистичному изображению, тем большей размер будет у файла.

Сферы применения векторной графики очень широки. В полиграфии – от создания красочных иллюстраций до работы со шрифтами. Все, что мы называем машинной графикой, 3D-графикой, графическими средствами компьютерного моделирования и САПР – все это сферы приоритета векторной графики, ибо эти ветви дерева компьютерных наук рассматривают изображение исключительно с позиции его математического представления.

Как видно, векторным можно назвать только способ описания изображения, а само изображение для нашего глаза всегда растровое. Таким образом, задачами векторного графического редактора являются растровая прорисовка графических примитивов и предоставление пользователю сервиса по изменению параметров этих примитивов. Все изображение представляет собой базу данных примитивов и параметров макета (размеры холста, единицы измерения и т. д.). Отрисовать изображение – значит выполнить последовательно процедуры прорисовки всех его деталей.

С другой стороны, если изображение состоит из простых объектов, то для его хранения в векторном виде необходимо не более нескольких килобайт.

1.3. Представление изображений с помощью фракталов

В последнее время фракталы стали очень популярны. Большую роль в этом сыграла книга франко-американского математика Бенуа Мандельброта "Фрактальная геометрия природы", изданная в 1975 году. Что же такое фрактал?

Фрактал (лат. fractus — дробленный, сломанный, разбитый) — сложная геометрическая фигура, обладающая свойством самоподобия, то есть составленная из нескольких частей, каждая из которых подобна всей фигуре целиком. Свойство самоподобия характерно для многих природных объектов. Таким свойством обладают, например, ветки деревьев, снежинки, границы облаков и морских побережий, трещины в камнях, структуры некоторых веществ, полученных с помощью электронного микроскопа и т. д.

Фрактальная геометрия позволяет описать и получить изображения таких природных объектов с помощью математических средств. В компьютерной графике фракталы, могут использоваться не только для генерации изображений сложных объектов, но и для сжатия изображений.

Для классификации фракталов часто используют деление на следующие классы:

- геометрические фракталы;
- алгебраические фракталы;
- стохастические фракталы.

Существуют и другие классификации фракталов, например, деление фракталов на детерминированные (алгебраические и геометрические) и недетерминированные (стохастические). Подробно рассмотрим геометрические и алгебраические фракталы.

1.3.1. Геометрические фракталы

Фракталы этого класса самые наглядные. В двухмерном случае их получают с помощью некоторой ломаной (или поверхности в трехмерном случае), называемой генератором. За один шаг алгоритма каждый из отрезков, составляющих ломаную, заменяется на ломаную генератора, в соответствующем масштабе. В результате бесконечного повторения этой процедуры, получается геометрический фрактал.

Рассмотрим один из таких фрактальных объектов - *триадную кривую Коха*. Построение кривой начинается с отрезка единичной длины (рис. 1.6) - это 0-е поколение кривой Коха. Далее каждое звено (в нулевом поколении один отрезок) заменяется на образующий элемент, обозначенный на рис. 1 через $n=1$. В результате такой замены получается следующее поколение кривой Коха. В 1-ом поколении - это кривая из четырех прямолинейных звеньев, каждое длиной по $1/3$.

Для получения 3-го поколения проделываются те же действия - каждое звено заменяется на уменьшенный образующий элемент. Итак, для получения каждого последующего поколения, все звенья предыдущего поколения необходимо заменить уменьшенным образующим элементом. Кривая n -го поколения при любом конечном n называется **предфракталом**. На рис. 1.6 представлены пять поколений кривой. При n стремящемся к бесконечности кривая Коха становится фрактальным объектом.

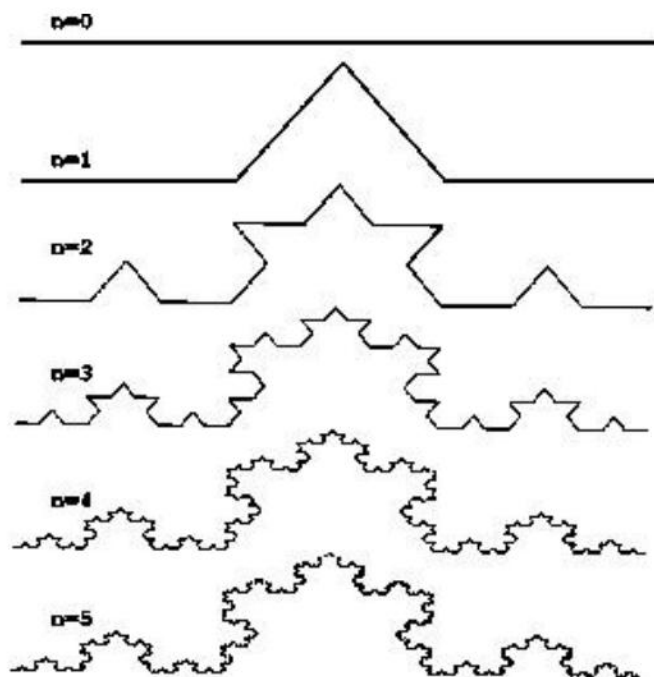


Рис. 1.6. Построение триадной кривой Коха

Три копии кривой Коха, построенные (остриями наружу) на сторонах правильного треугольника, образуют замкнутую кривую, называемую *снежинкой Коха*.

Алгоритм построения фрактала можно описать словесно, как показано на примере построения кривой Коха, либо математически, как будет показано в следующем разделе для алгебраических фракталов. Но существует еще один способ описания алгоритма построения геометрических фракталов с помощью *L-систем (системы Линдемайера)*. *L-система* это формальная грамматика, используемая для моделирования процессов роста и развития растений.

Изначально L -системы были введены при изучении формальных языков, а также использовались в биологических моделях селекции. С их помощью можно строить многие известные самоподобные фракталы, включая снежинку Коха и ковер Серпинского.

L -система определяется как кортеж $G=(V, w, P)$, где V – алфавит, представляющий набор символов, содержащих элементы которые могут быть представлены графически, w – аксиома, которая представляет собой строку символов из V , определяющая начальное состояние системы, P – представляет собой набор правил производства новой строки, путем замены символов текущего состояния системы на ряд новых. Правила грамматики L -системы применяются итеративно, начиная с начального состояния.

Рассмотрим L -систему, где алфавит состоит всего из двух символов $V=\{a,b\}$. Аксиома $w=a$. Правила производства описываются как $p1: a \rightarrow ab$, $p2: b \rightarrow ab$. Следуя итеративному принципу, каждое поколение кривой удваивается на каждом этапе: $a, ab, abab, abababab$. Графически a и b отображаются как два равных отрезка, соединенных под прямым углом (рис. 1.7).

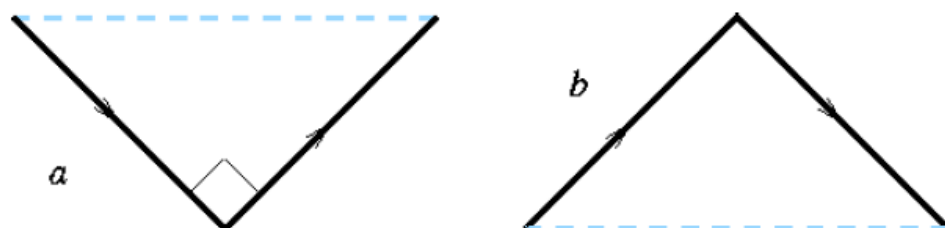


Рис. 1.7. Графическое представление алфавита L -системы

Графическое представление правил $p1$ и $p2$, приведено на рис. 1.8. Таким образом, правила описывают замену каждого из отрезков двумя другими, соединенных под прямым углом. При этом, каждый раз, меняется угол поворота новых отрезков относительно исходного.

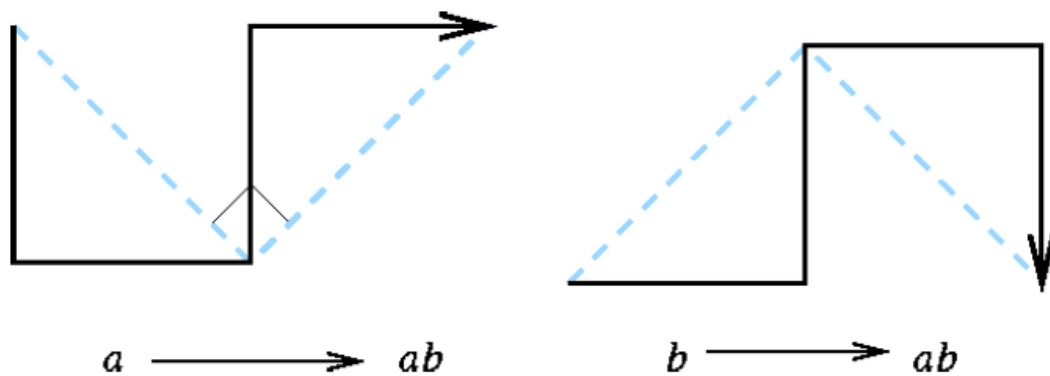


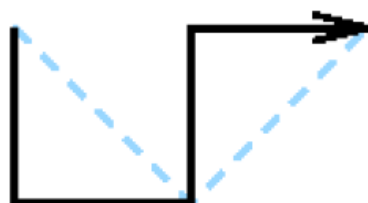
Рис. 1.8. Графическое представление правил L -системы

На рис. 1.9 изображены пять первых итераций процесса построения фрактала.

Gen. 1



Gen. 2



Gen. 3



Gen. 4

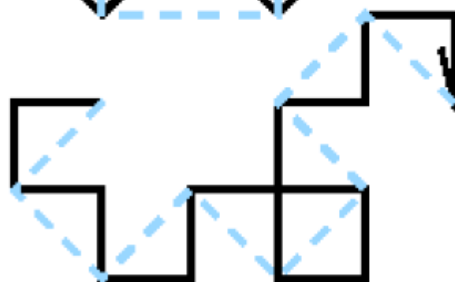


Рис. 1.9. Первые итерации построения фрактальной кривой

Такая предельная фрактальная кривая (при числе итераций стремящимся к бесконечности) называется *драконом Хартера-Хейтуэя* и представлена на рис. 1.10.

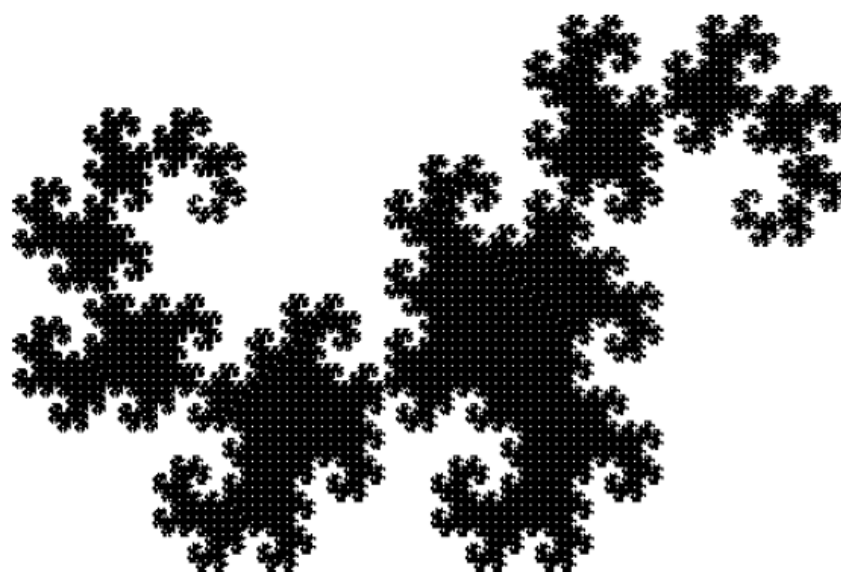


Рис. 1.10. Дракон Хартера-Хейтуэя

Некоторые из геометрических фракталов можно построить исходя из первоначально закрашенной плоской фигуры. Примером этого может служить метод построения треугольника Серпинского. Процесс построения можно описать следующим образом.

Равносторонний треугольник M_0 делится прямыми, параллельными его сторонам, на 4 равных равносторонних треугольника. Из треугольника удаляется центральный треугольник. Получается множество M_1 , состоящее из 3 оставшихся треугольников «первого ранга». Поступая точно так же с каждым из треугольников первого ранга, получим множество M_2 , состоящее из 9 равносторонних треугольников второго ранга. Продолжая этот процесс бесконечно, получим бесконечную последовательность $M_0, M_1, \dots, M_n, \dots$ пересечение членов которой есть треугольник Серпинского (рис. 1.11).



Рис. 1.11. Построение салфетки Серпинского

2. Представление цвета в компьютере

2.1. Свет и цвет

Понятия света и цвета в компьютерной графике тесно связаны и являются основополагающими.

Свет может рассматриваться либо как электромагнитная волна, либо как поток фотонов. Одной из характеристик электромагнитной волны является ее длина λ . Видимый свет имеет длину волн в диапазоне 400-700 нм. Свет принимается либо непосредственно от источника, например, от экрана монитора, либо косвенно при отражении от поверхности объекта или преломлении в нем.

На практике мы редко сталкиваемся со светом какой-то определенной длины волны. Напротив, видимый свет практически всегда состоит из сочетания фотонов разных длин волн.

Источник или объект является *ахроматическим*, если наблюдаемый свет содержит все видимые длины волн в приблизительно равных количествах. Ахроматический источник кажется белым, а отраженный или преломленный ахроматический свет — белым, черным или серым. Белыми выглядят объекты, ахроматически отражающие более 80% света белого источника, а черными — менее 3%.

Промежуточные значения дают различные оттенки серого.

Если воспринимаемый свет содержит длины волн в произвольных неравных количествах, то он называется *хроматическим*.

Монохроматический - это такой свет, который имеет одну длину волны или частоту.

Понятие цвета тесно связано с тем, как человек воспринимает свет. Можно сказать, что ощущение цвета формируется человеческим мозгом в результате анализа электромагнитного излучения (света), попадающего на сетчатку глаз.

Считается, что в глазе человека существует три группы цветовых рецепторов (колбочек), каждая из которых чувствительна к определенному диапазону длин волны. Каждая группа формирует один из трех основных цветов: красный, зеленый, синий.

Если длины волн светового потока сконцентрированы у верхнего края видимого спектра (около 700 Нм), то свет воспринимается как красный. Если длины волн сконцентрированы у нижнего края видимого спектра (около 400 Нм), то свет воспринимается как синий. Если длины волн сконцентрированы в середине видимого спектра (около 550 Нм), то свет воспринимается как зеленый.

С помощью экспериментов, построенных на этой гипотезе, были получены кривые реакции глаза, показанные на рис. 2.1.

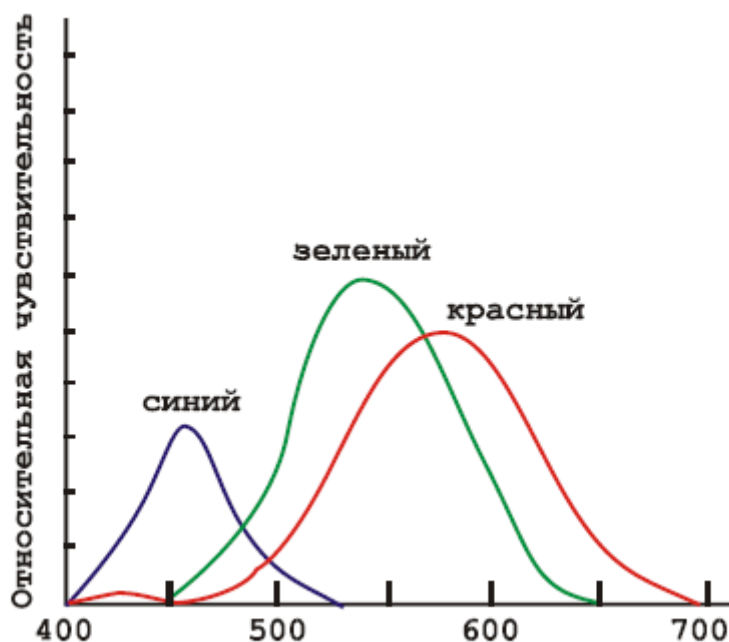


Рис. 2.1. Кривые реакции глаза

При описании цвета используют три его субъективных атрибута: **цветовой тон**, **насыщенность** и **светлоту**. Разделение признака цвета на эти взаимосвязанные компоненты есть результат мысленного процесса, существенно зависящего от навыка и обучения.

Наиболее важный атрибут цвета – **цветовой тон (hue)** – ассоциируется в человеческом сознании с обусловленностью окраски предмета определенным типом пигмента, краски, красителя. Тон определяется характером распределения излучения в спектре видимого света. Именно тон определяет название цвета, например, «красный», «синий», «зелёный».

Насыщенность (saturation) характеризует степень, уровень, силу выражения цветового тона. Этот атрибут в человеческом сознании связан с количеством (концентрацией) пигмента, краски, красителя. Можно сказать, что насыщенность цвета показывает, насколько данный цвет отличается от монохроматического («чистого») излучения того же светового тона. Насыщенность характеризует степень ослабления (разбавления) данного цвета белым и позволяет отличать розовый от красного, голубой от синего. Считается, что серые тона (ахроматические) не имеют насыщенности и различаются лишь по светлоте.

Светлота (lightness, value) – это различимость участков, сильнее или слабее отражающих свет. Уровень светлоты окрашенных объектов определяется при сравнении их с ахроматическими объектами и при выявлении степени их приближения к белому цвету, отражающему максимум света.

Со светлотой тесно связано физическое понятие **яркости света (luminance)**. Чем больше яркость, тем больше светлота. Поэтому можно сказать, что светлота есть мера ощущения яркости. С другой стороны, во многих источниках, вместо понятия светлота часто используют субъективное понятие **яркости (brightness)**.

2.2. Цветовые модели и пространства

Как видим из вышеизложенного, описание цвета может опираться на составление любого цвета на основе основных цветов или на такие понятия, как светлота, насыщенность, цветовой тон. Применительно к компьютерной графике описание цвета также должно учитывать специфику аппаратуры для ввода/вывода изображений. В связи с необходимостью описания различных физических процессов воспроизведения цвета были разработаны различные цветовые модели. Цветовые модели позволяют с помощью математического аппарата описать определенные цветовые области спектра. Цветовые модели описывают цветовые оттенки с помощью смешивания нескольких основных цветов.

Основные цвета разбиваются на оттенки по яркости (от темного к светлому), и каждой градации яркости присваивается цифровое значение (например, самой темной – 0, самой светлой – 255). Считается, что в среднем человек способен воспринимать около 256 оттенков одного цвета. Таким образом, любой цвет можно разложить на оттенки основных цветов и обозначить его набором цифр – цветовых координат.

Таким образом, при выборе цветовой модели можно определять обычно трехмерное цветовое координатное пространство, внутри которого каждый цвет представляется точкой. Такое пространство называется пространством цветовой модели.

Профессиональные графические программы обычно позволяют оперировать с несколькими цветовыми моделями, большинство из которых создано для специальных целей или особых типов красок: CMY, CMYK, CMYK256, RGB, HSB, HLS, L*a*b, YIQ, Grayscale

(Оттенки серого) и Registration color. Некоторые из них используются редко, диапазоны других перекрываются.

1. PIL – библиотека Python Imaging Library

Библиотека *Python Imaging Library (PIL)* содержит общие средства для обработки изображений и разнообразных полезных операций, в том числе: изменение размера, кадрирование, поворот, преобразование цветов и т. д. Библиотека распространяется бесплатно, ее можно скачать с сайта <http://www.pythonware.com/products/pil/>.

PIL позволяет читать изображения, записанные в большинстве существующих форматов, и сохранять в наиболее популярных. Наиболее важен модуль `Image`. Для чтения изображения служит такой код:

```
from PIL import Image

pil_im = Image.open('empire.jpg')
```

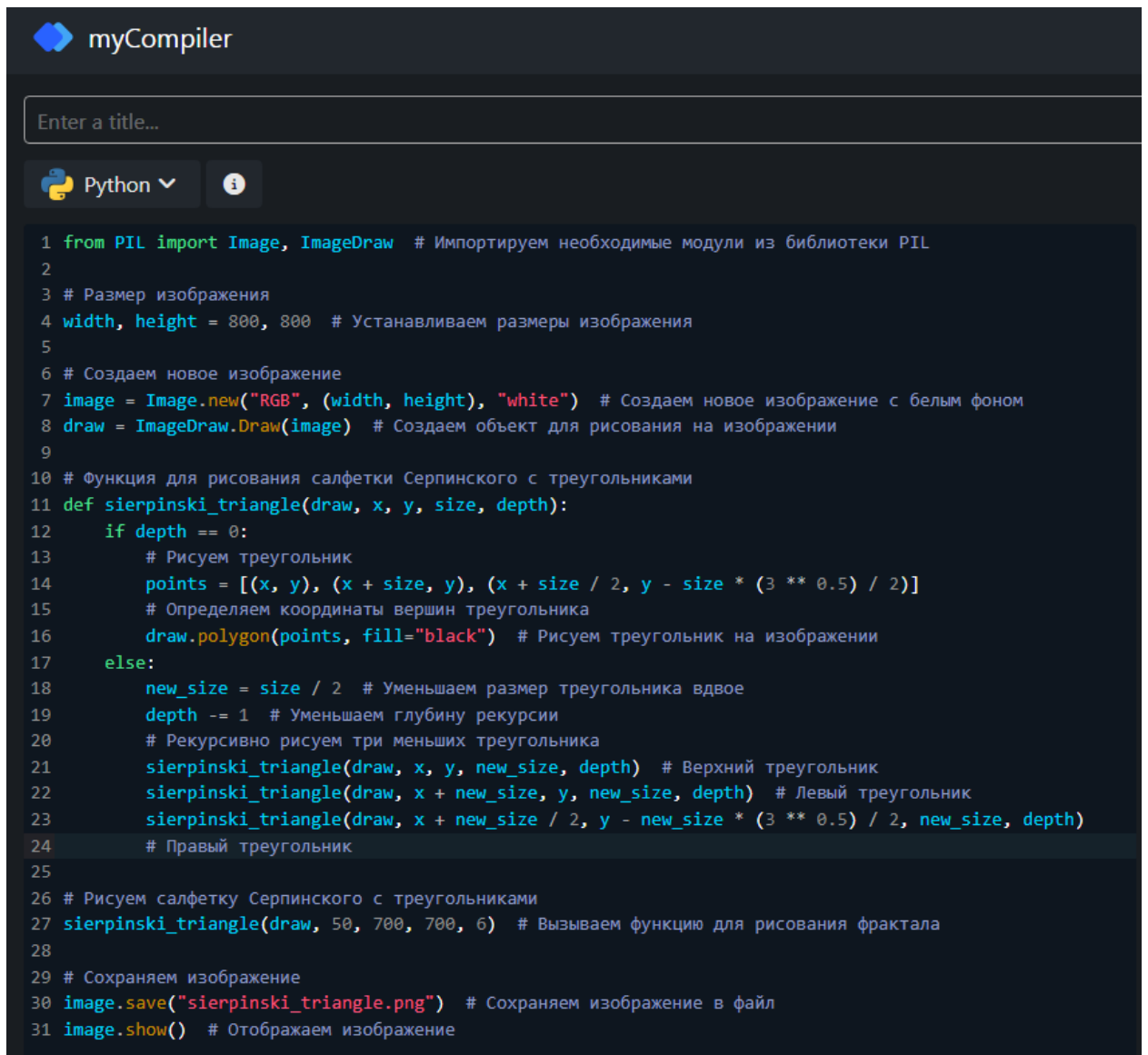
В качестве значения метод возвращает объект изображения *`pil_im`*.

Пакет NumPy

Пакет `NumPy` (<http://www.scipy.org/NumPy/>) часто используется для научных расчетов на Python. NumPy включает ряд полезных концепций, в частности объекты-массивы (для представления векторов, матриц, изображений и т. п.), и функции из области линейной алгебры. Массивы NumPy будут использоваться почти во всех примерах ниже. Объект массива позволяет выполнять такие важные операции, как умножение матриц, транспонирование, решение систем линейных уравнений, скалярное умножение и нормировку векторов. Все это необходимо для таких вещей, как совмещение изображений, деформирование, различные виды моделирования, классификация и группировка изображений и т. д.

Пакет NumPy можно бесплатно скачать по адресу <http://www.scipy.org/Download>, а в документации (<http://docs.scipy.org/doc/numpy/>) имеются ответы на большинство вопросов.

Задание 1. Используя онлайн-компилятор myCompiler, получить изображение салфетки Серпинского. Сохранить изображение в формате PNG (рис. 1).



```
1 from PIL import Image, ImageDraw # Импортируем необходимые модули из библиотеки PIL
2
3 # Размер изображения
4 width, height = 800, 800 # Устанавливаем размеры изображения
5
6 # Создаем новое изображение
7 image = Image.new("RGB", (width, height), "white") # Создаем новое изображение с белым фоном
8 draw = ImageDraw.Draw(image) # Создаем объект для рисования на изображении
9
10 # Функция для рисования салфетки Серпинского с треугольниками
11 def sierpinski_triangle(draw, x, y, size, depth):
12     if depth == 0:
13         # Рисуем треугольник
14         points = [(x, y), (x + size, y), (x + size / 2, y - size * (3 ** 0.5) / 2)]
15         # Определяем координаты вершин треугольника
16         draw.polygon(points, fill="black") # Рисуем треугольник на изображении
17     else:
18         new_size = size / 2 # Уменьшаем размер треугольника вдвое
19         depth -= 1 # Уменьшаем глубину рекурсии
20         # Рекурсивно рисуем три меньших треугольника
21         sierpinski_triangle(draw, x, y, new_size, depth) # Верхний треугольник
22         sierpinski_triangle(draw, x + new_size, y, new_size, depth) # Левый треугольник
23         sierpinski_triangle(draw, x + new_size / 2, y - new_size * (3 ** 0.5) / 2, new_size, depth)
24         # Правый треугольник
25
26 # Рисуем салфетку Серпинского с треугольниками
27 sierpinski_triangle(draw, 50, 700, 700, 6) # Вызываем функцию для рисования фрактала
28
29 # Сохраняем изображение
30 image.save("sierpinski_triangle.png") # Сохраняем изображение в файл
31 image.show() # Отображаем изображение
```

Рис.1. Программный код для построения салфетки Серпинского

Задание 2. Используя онлайн-компилятор myCompiler, получить фрактальное изображение листа папоротника (Barnsley fern). Сохранить изображение в формате PNG (рис. 1). Прокомментируйте каждую строку кода. Чем замечательно изображение папоротника Барнсли?

myCompiler

Enter a title...

Python

```
1 from PIL import Image, ImageDraw
2 import random
3
4 # Размер изображения
5 width, height = 800, 800
6
7 # Создаем новое изображение
8 image = Image.new("RGB", (width, height), "white")
9 draw = ImageDraw.Draw(image)
10
11 # Начальные координаты
12 x, y = 0, 0
13
14 # Функции преобразования для фрактала
15 def transform(x, y):
16     r = random.random()
17     if r < 0.01:
18         return (0, 0.16 * y)
19     elif r < 0.86:
20         return (0.85 * x + 0.04 * y, -0.04 * x + 0.85 * y + 1.6)
21     elif r < 0.93:
22         return (0.2 * x - 0.26 * y, 0.23 * x + 0.22 * y + 1.6)
23     else:
24         return (-0.15 * x + 0.28 * y, 0.26 * x + 0.24 * y + 0.44)
25
26 # Рисуем фрактал
27 for _ in range(100000):
28     x, y = transform(x, y)
29     px = int(width / 2 + x * width / 11)
30     py = int(height - y * height / 11)
31     draw.point((px, py), fill="green")
32
33 # Сохраняем изображение
34 image.save("barnsley_fern.png")
35 image.show()
```

Задание 3. Получить фрактальное изображение триадной кривой Коха. Сохранить изображение в формате PNG.

Задание 4. Получить фрактальное изображение дракона Хартера-Хейтуэя. Сохранить изображение в формате PNG.

Задание 5*. Получить цветное фрактальное изображение множества Мандельброта. Сохранить изображение в формате PNG.

Задание 6*. Получить цветное фрактальное изображение множества Жулия. Сохранить изображение в формате PNG.