КОМПЬЮТЕРНАЯ ПОЛИГРАФИЯ*

А. С. Зациорский
amartel@yandex.ru

6 апреля 2023 г.

1°. Дональд Кнут начал создавать систему TEX в конце 1970-ых. В это время графические дисплеи уже появились, но текстовый режим ещё был доминирующим. Однако т. к. основной задачей было получение высококачественного математического текста, TEX разрабатывался как полностью графическая система (в математике используется множество символов, имеющих разные размеры и формы, что создаёт проблемы и с помещением символа в знакоместо, и со вместимостью алфавита). Кнут даже создал специальный язык программирования META, использовавшийся в системе METAFONT для создания параметрических векторных шрифтов (металшрифтов).

Учитывая вышесказанное, можно ожидать, что TEX обладает богатыми возможностями по работе с графикой. Однако на самом деле есть всего два базовых способа включения в документ графических объектов. Первый — создание специального шрифта и использование его символов как элементов изображения. Второй способ заключается в использовании команды `\special`.

2°. Первый подход был применён Лесли Лэмпортом при создании LATEX в 1984 году (окружение `picture`). Использование специального шрифта позволило рисовать некоторые графические примитивы: отрезки, стрелки, окружности, «овалы» (на самом деле прямоугольники с закруглёнными углами) и квадратичные кривые Безье. Однако это накладывало существенные ограничения. TEX использует внутреннюю кодировку символов и изначально она была 7-битной, что позволяло иметь только 128 символов. В результате удавалось обеспечить поддержку ограниченного количества углов наклона для отрезков и стрелок: параметры принимали значения от -6 до 6 в первом случае и всего лишь значения от -4 до 4 во втором (рис. 1). Диаметры окружностей допускались равными 1–40 пунктам, а кругов — 1–15 пунктам. Некая комбинация окружностей, кругов и «овалов» представлена на рис. 2.

*Семинар по оптимизации, машинному обучению и искусственному интеллекту «O&ML» `http://oml.cmlaboratory.com/`

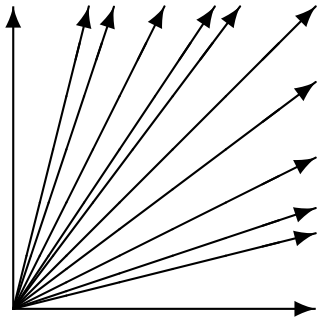

Рис. 1. Все допустимые углы наклона стрелок в первом квадранте

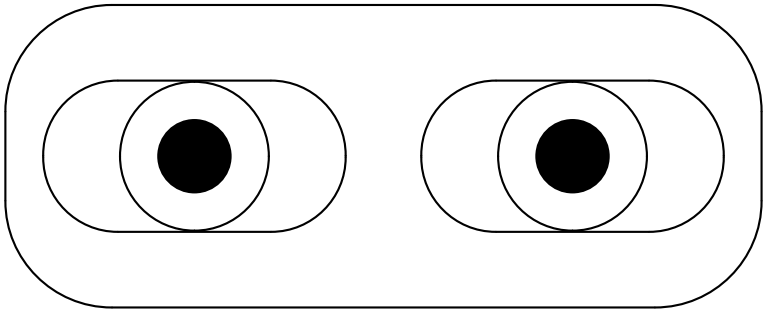

Рис. 2. Окружности, круги и «овалы»

Но у окружения `picture` были и существенные плюсы: возможность помещать текст на рисунок, используя при этом шрифты из документа, точно этот текст позиционировать, а главное — картинка получалась переносимой. К тому же, этот подход не требовал разработки дополнительного ПО, а лишь применения имеющихся средств.

3°. Несмотря на описанные достоинства, использование шрифтов накладывало слишком много ограничений. Рассмотрим второй подход, основанный на применении команды `\special`. Для этого необходимо в общих чертах разобраться с тем, какие основные этапы, а их обычно несколько, есть в процессе генерации документа в TEX. Изначально пользователь создаёт так называемый исходный файл. Это обычный текстовый файл, содержащий непосредственно текст и команды TEX, которые будут определять, как именно данный текст в итоге должен выглядеть. Исходный файл передаётся на вход TEX, и после обработки мы получаем файл с расширением .dvi, что означает device independent. Можно было бы ожидать, что этот результат работы системы TEX и есть готовый файл с нашим красивым текстом, но на самом деле никакого видимого текста и никаких изображений в dvi-файле нет. Это бинарный файл, содержащий коды символов в специальной внутренней кодировке, а также информацию о расположении и размерах объектов. Непосредственное построение по данному описанию изображения и вывод его на устройство обес-

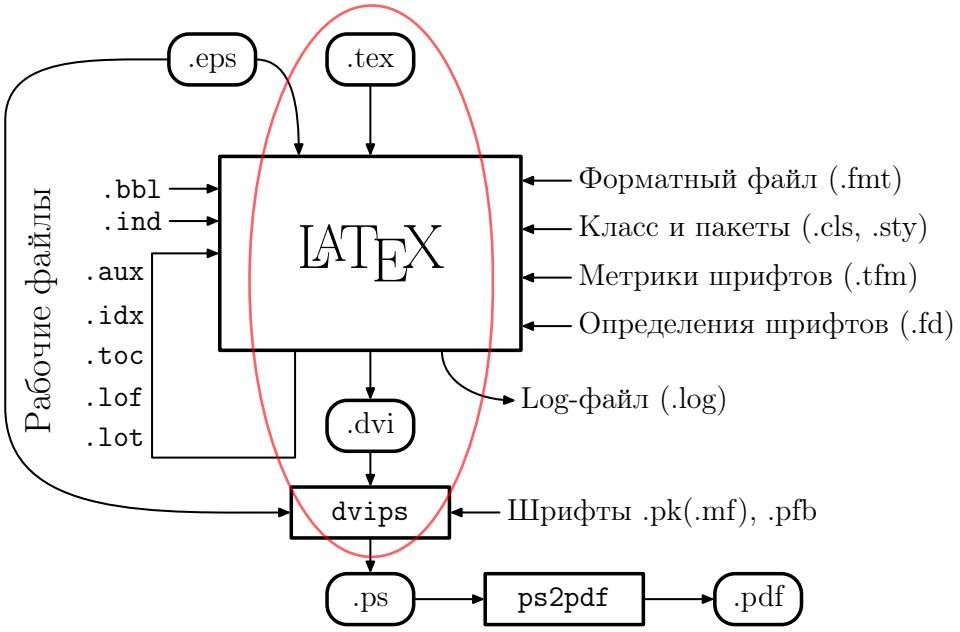

Рис. 3. Пример возможного TEX-конвейера [1]

печивает драйвер — специальная программа. В примере на рис. 3 используется популярный драйвер dvips, преобразующий dvi-файл в программу на языке PostScript. Википедия использует драйвер, выдающий результат в растровом формате png. Но драйвер может и непосредственно формировать изображение на экране (просмотрщик Yap в дистрибутиве MiKTeX). Такой подход применён для достижения переносимости — TEX не нужно учитывать аппаратные особенности различных систем, достаточно чтобы там существовал драйвер, способный осуществлять вывод на имеющиеся в данной системе устройства.

Вернёмся к команде `\special`. Что же TEX делает с её аргументом? А ничего не делает, просто оставляет как есть. И именно драйверу предстоит решать, что с этим делать. Если драйвер «понимает», как это обработать — он генерирует изображение, а вот если нет — придётся ему как-то выкручиваться. Игнорировать всё или то, что он не понял, сообщать пользователю об ошибке и т. п. Ситуацию существенно осложняет то, что аргументом `\special` может быть что угодно, а различных драйверов существует очень много. Поэтому возникают проблемы с совместимостью. За прошедшие десятилетия ситуация несколько улучшилась, однако и сегодня некоторые пакеты могут полностью или частично не работать с некоторыми драйверами. Но если всё же удаётся обеспечить «понимание» драйвером всего того, что Вы включаете в документ с помощью команды `\special`, то появляется возможность достигать очень высокого качества графики.

4°. В 1984 году был создан язык PostScript, предназначенный для вывода высококачественной векторной графики. Он был платформонезависимым и мог работать с устройствами, обладающими абсолютно отличающимися возможностями. Учитывая открытость спецификаций, он быстро завоевал популярность. Применялся для описания страниц, передаваемых принтерам на печать и одно время практически в каждом принтере был интерпретатор PostScript. Да и сейчас во многих принтерах он имеется. Естественно, что и сообществом TEXников он не остался незамеченным. Именно программы на этом языке и стали тем, что включает в документ команда `\special`, став де-факто стандартом. Драйвер, реализовавший поддержку PostScript, мог корректно обрабатывать большинство изображений.

Данный язык является стековым и поэтому использует постфиксную нотацию. Это позволяет обходиться без скобок, но воспринимаются такие программы непривычно. Для того, чтобы поразмять мозги и, в частности, для использования в учебном процессе это и хорошо, а вот для того, чтобы быстро и без усилий нарисовать какую-нибудь картинку — не очень хорошо. PostScript — достаточно низкоуровневый язык и больше ориентирован на машинную обработку, чем на чтение человеком. Поэтому сейчас чаще всего другие средства, более высокоуровневые, генерируют программы на PostScript.

В качестве примера рассмотрим программу рисования правильной звезды с n вершинами, в данном случае $n = 21$ (рис. 4). Программа довольно лаконична, но не слишком понятна на первый взгляд. Зато нет никаких проблем с углами наклона, можем рисовать что угодно, вопрос только в трудозатратах.

```
/ngramm {
  /n exch def
  /l exch def
  /a {
    180 n div
    180 add
  } def

  n 1 sub {
    l 0 rlineto
    a rotate
  } repeat

  closepath
  stroke
} def
```

Листинг 1. Построение правильной звезды с помощью PostScript

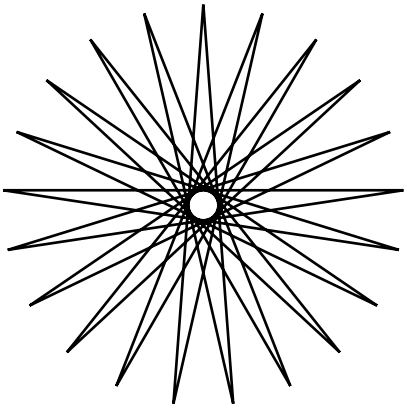

Рис. 4. Правильная звезда $\{21/10\}$, нарисованная с помощью PostScript

5°. На PostScript стали писать различные подпрограммы, подпрограммы накапливались и объединялись в модули, называемые графическими пакетами. Было создано много графических пакетов, одни забрасывались, другие создавались. Некоторые из них:

1) TikZ
2) PSTricks
3) curves
4) epic and eepic
5) PiCTEX
6) XY-pic
7) и многие другие...

В настоящее время по большому счёту осталось два основных графических пакета — TikZ и PSTricks. Пакет PSTricks, как можно догадаться исходя из его названия, основан на PostScript. Пакет TikZ ведёт себя чуть более хитро. Это связано с появлением хорошо всем известного формата PDF — Portable Document Format. Данный формат основан на PostScript-подобном языке, дополненном различными расширенными возможностями (не только описание страницы, но и мета-информация: шрифты, ссылки и т.п.). Популярность PDF росла и это повлекло появление драйвера pdfTeX, создающего файлы в данном формате напрямую, минуя все остальные стадии конвейера. Это удобно, но т. к. пакет PSTricks основан на включении в документ PostScript-кода с помощью команды `\special`, а pdfTeX не создаёт dvi-файла и не передаёт

его какому-то драйверу, который бы этот PostScript-код понимал, то в итоге PSTricks с pdfTeX не работает. А хитрость TikZ заключается в том, что он определяет, какой драйвер используется, и в зависимости от этого генерирует «правильные» команды `\special`.

Вообще, если присмотреться, возможности большинства графических пакетов схожи. Начинается всё с графических примитивов — это отрезки, дуги, окружности, эллипсы, кривые Безье. Появляются возможности управлять свойствами линий: цветом, толщиной, стилем (различные пунктиры); заливкой областей. А далее на основе всех этих базовых вещей разрабатываются различные дополнительные модули, позволяющие изображать более сложные объекты: графики функций; узлы и их соединения, деревья, т. е. различные графы; трёхмерные изображения; геометрические построения. И множество ещё более специализированных пакетов для рисования всего подряд вплоть до жёлтых уточек.

По причине имеющегося сходства, как правило, достаточно владеть одним графическим пакетом, и на роль этого пакета можно рекомендовать TikZ. Помимо поддержки драйвера pdfTeX у него есть и ещё одно важное преимущество — пакет beamer, предназначенный для подготовки презентаций, создан на базе TikZ и поэтому они прекрасно работают совместно, а вот с PSTricks beamer взаимодействует не так хорошо.

Приведём несколько базовых конструкций пакета TikZ и используем их для создания простого примера.

1) Координаты:
   - декартовы `(x, y)` и полярные `(alpha: r)`
   - абсолютные, относительные `+(x, y)` и... ещё одни относительные `++(x, y)`
2) Траектории:
   - определение `\path (0, 0) -- (1, 1);`
   - изображение `[draw]`, заливка `[fill]` и кое-что ещё
3) Параметры:
   - цвет `[color = red]`
   - толщина линии `[line width = 2pt]`
   - стиль линии `[dashed]` и многое другое

```
\begin{tikzpicture}
  \draw[color = purple, fill = yellow, dashed]
    (0, 0) -- (60: 3) -- +(-60:3) -- cycle;
\end{tikzpicture}
```

Листинг 2. Построение равностороннего треугольника с помощью TikZ

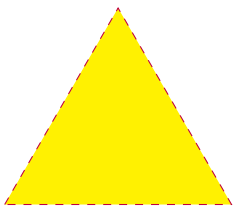

Рис. 5. Равносторонний треугольник, нарисованный с помощью TikZ

Команды пакета TikZ помещаются в окружение `tikzpicture`. Они определяют траекторию и её свойства. В данном случае траектория является ломанной, соединяющей три точки, задаваемые в декартовых, полярных и относительных полярных координатах соответственно. Траектория замыкается с помощью специальной точки `cycle`, что позволяет избежать дублирования координат начальной точки. Свойствами траектории являются её цвет (ключ `color`) и стиль (ключ `dashed`). Ключ `fill` определяет цвет заливки и одновременно необходимость её выполнения, а команда `draw`, являющая короткой формой записи основной команды задания траектории `path` с ключом `draw`, изображает саму траекторию.

В качестве второго примера попробуем ещё раз нарисовать правильную звезду, но уже с помощью TikZ, а не PostScript.

```
% #1 - количество вершин
% #2 - радиус
\newcommand{\ngramm}[2]{
  \begin{tikzpicture}[magenta, dotted, line width = 0.5pt]
    \pgfmathsetmacro\a{180 - 180/#1}
    \foreach \i in {0,...,#1}{
      \draw +(\i*\a:#2) -- +(\a + \i*\a:#2);
    }
  \end{tikzpicture}
}
\ngramm{21}{20mm}
```

Листинг 3. Построение правильной звезды с помощью TikZ

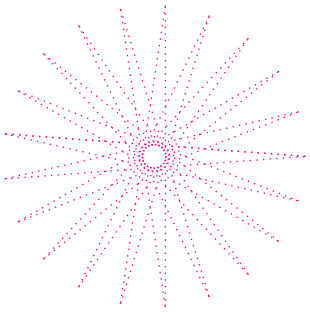

Рис. 6. Правильная звезда $\{21/10\}$, нарисованная с помощью TikZ

Эта программа, по существу, состоит из трёх содержательных строк (строки 5, 6, 7). Первая из них определяет угол поворота, вторая реализует цикл по множеству вершин, а последняя — соединяет пару вершин отрезком.

6°. В предшествующих примерах все координаты задавались явно. Однако при подготовке иллюстраций часто возникает необходимость внесения изменений для достижения требуемого визуального качества. Причём прежде чем всё наконец-то станет выглядеть симпатично, может понадобиться несколько итераций. А каждая из них может приводить к изменению и необходимости пересчёта части или даже всех координат. Чем сложнее рисунок и чем больше имеется различных объектов — тем более трудоёмким становится внесение изменений. Конечно, очень хочется переложить всю или хотя бы большую часть этой работы на компьютер.

Если наше изображение имеет структуру (наиболее выгодный пример — это различные геометрические построения на плоскости), которую мы могли бы как-то описать, то можно использовать следующий подход. Определим небольшое количество базовых объектов, явным образом указав необходимые параметры (координаты, длины, углы и т. п.). А все остальные объекты будем определять относительно базовых в геометрических терминах — как параллельные или перпендикулярные чему-то, являющееся пересечением чего-то с чем-то или симметричные чему-то. В этом случае для внесения корректировок в наше изображение будет достаточно изменить один или несколько базовых объектов, а координаты всех зависимых объектов автоматически пересчитает компьютер по имеющемуся описанию.

Реализовать такой подход можно по-разному. Можно расширить имеющийся графический пакет набором макросов, реализующих необходимые геометрические операции. Преимущество этого подхода в том, что пользователю проще освоить это расширение, т. к. основные подходы и синтаксис наследуются от графического пакета и остаётся только выяснить, как называется макропрограмма, выполняющая требуемую операцию. Однако получающийся код содержит и описание структуры изображения, и команды рисования. Всё это может быть намешано совершенно невообразимым образом.

В программировании такие вещи не приветствуются, там как правило выполняется декомпозиция, всё разбивается на небольшие блоки, отвечающие за что-то одно. Так называемый принцип единой ответственности. Поэтому альтернативный подход — это разработка специального геометрического языка. Он будет отвечать за описание структуры изображения. И на основе этого описания уже будет выполняться построение изображения, например, посредством генерации необходимого набора команд какого-то графического пакета.

Работа с геометрическим языком обычно выглядит следующим образом: пользователь создаёт описание желаемого рисунка, определяет значения параметров и даёт команду на обработку в результате чего получает изображение. Оценивает его и при необходимости меняет значения параметров. И так до тех пор, пока результат его не устроит. Т. е. всё происходит в духе TEX — сначала создаём описание желаемого, потом рассматриваем получившийся результат.

Альтернатива — визуальный подход WYSIWYG (What you see is what you get) в духе, например, MS Word. В этой ситуации пользователь имеет возможность непосредственно наблюдать и изменять изображение — перетаскивать с помощью мыши точки, растягивать окружности или многоугольники и всё в этом роде. Такие системы называются системами динамической или интерактивной геометрии. При этом было бы не правильно противопоставлять их геометрическим языкам, т. к. на самом деле эти системы сами используют какой-то геометрический язык для описания чертежа и обычно предлагают возможность изменять значения параметров или дополнять описание в текстовом режиме, как и остальные геометрические языки. Просто появляется дополнительная возможность взаимодействовать с рисунком в визуальном ключе. Это весьма наглядно и удобно, хотя если важна точность задания параметров, то точные значения проще ввести непосредственно, чем пытаться мышью растянуть окружность до необходимого радиуса в 3.1415. Основная же идея геометрических языков — параметризация изображений — сохраняется и здесь. Одни объекты определяются относительно других, просто вместо текстового описания используются визуальные средства, какие-то кнопочки на панели инструментов и т. п.

Рассмотрим несколько конкретных примеров пакетов и программ, использующих описанные подходы.

7°. Необходимость в использовании геометрических операций весьма насущна, поэтому неудивительно, что оба основных графических пакета имеют геометрические расширения. Их названия не блещут оригинальностью и образованы от названий графических пакетов и имени Евклида.

- TikZ: пакет `tkz-euclide`
- PSTricks: пакет `pst-eucl`

Не будем перечислять все возможности этих пакетов — в каждом из них реализована поддержка основных геометрических понятий. Поэтому можно предположить, что пакет умеет стоить перпендикуляры или симметричные объекты, и предположение это окажется верным. Рассмотрим пример создания изображения с помощью пакета tkz-euclide и на примере оценим и его преимущества, и его недостатки.

ЛЕММА 1. *Пусть две окружности касаются в точке A. Их параллельными диаметрами являются отрезки CD, EF. Тогда точки A, C и E лежат на одной прямой.*

```
\tkzDefPoints{0/0/O_1,0/1/O_2,0/3/A}
\tkzDefPoint(15:3){F}
\tkzInterLC(F,O_1)(O_1,A) \tkzGetSecondPoint{E}
\tkzDefLine[parallel=through O_2](E,F) \tkzGetPoint{x}
\tkzInterLC(x,O_2)(O_2,A) \tkzGetPoints{D}{C}
\tkzDrawCircles(O_1,A O_2,A)
\tkzDrawSegments[new](O_1,A E,F C,D)
\tkzDrawSegments[purple](A,E A,F)
\tkzDrawPoints(A,O_1,O_2,E,F,C,D)
\tkzLabelPoints(A,O_1,O_2,E,F,C,D)
```

Листинг 4. Пример из документации пакета `tkz-euclide`

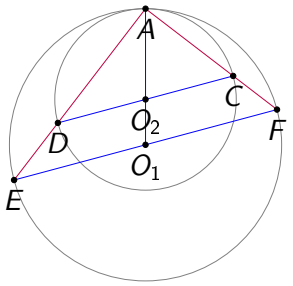

Рис. 7. Иллюстрация к лемме

TEX основан на макрокомандах и при этом не имеет механизмов для реализации пространств имён. Поэтому для предотвращения конфликтов между объявленными в разных пакетах идентификаторами разработчики пакетов обычно используют производные от имени пакета префиксы (tkz в случае пакета tkz-euclide). Кроме того, в TEX отсутствует механизм возвращаемых макрокомандами значений. Поэтому требуется использовать специальную макрокоманду, которая связывает возвращаемый результат предыдущей макрокоманды с идентификатором для последующего использования. Всё это делает код громоздким. Однако использование «геометрических» макрокоманд, автоматически определяющих точки пересечения окружности с прямой или

строящих отрезок по ранее определённым или вычисленным точкам, позволяет существенно упростить описание рисунка и сделать его независимым от конкретных координат базовых точек. Данный пример мог бы быть более универсальным. Например, точки A и F лежат на окружности радиуса 3 с центром в начале координат и задаются своими декартовыми и полярными координатами соответственно с явным указанием числа 3. Однако несогласованное изменение этого значения приведёт к тому, что точка F перестанет на окружности находиться (т. к. окружности в программе определяются проходящими через точку A). Введение переменной r со значением 3 и её использования для определения точек A и F сделало бы программу более устойчивой к изменениям.

Приведём ещё один пример изображения, описанного с помощью пакета tkz-euclide. Иллюстрация к теореме Морли о трисектрисах сложнее иллюстрации к лемме, поэтому и описание её более объёмно, но качественно более сложным оно не является.

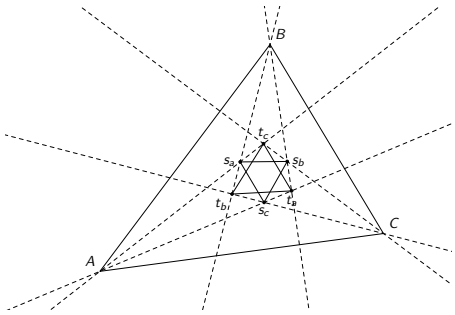

Рис. 8. Теорема Морли о трисектрисах (из документации пакета `tkz-euclide`)

8°. Самостоятельный геометрический язык, в отличие от расширений пакетов, не обязан иметь согласованный с чем-то синтаксис, и, таким образом, его можно спроектировать более лаконичным и читаемым. Сама концепция геометрического языка выглядит весьма привлекательно, поэтому их тоже было разработано немало. Вот некоторые из них:

- Diag (Benjamin R. Seyfarth) — препроцессор и одноимённый геометрический язык, предназначенный для описания и рисования диаграмм;
- DROL (Бухвалова В. В.) — геометрический язык спецификаций, предназначенный для описания геометрических построений на плоскости и записи алгоритмов вычислительной геометрии;
- Eukleides (Christian Obrecht) — кроссплатформенный геометрический язык с открытым исходным кодом;
- PIC (Brian Kernighan) — ещё один геометрический язык для описания диаграмм.

Продемонстрируем использование геометрического языка DROL на следующем примере: построим описанную окружность равнобедренного треугольника с заданным основанием и углом при основании (рис. 9).

```
1  real l = 20;
2  real alpha = 60;
3
4  point A = origin;
5  point B = rt A on l;
6  ray r1 = A dir cr(alpha), r2 = B dir cr(180 - alpha);
7  point C = r1 intersect r2;
8  lineseg AB = cr(A, B), AC = cr(A, C), BC = cr(B, C);
9  point M = middle(A, B), N = middle(B, C);
10 line k1 = M perp AB, k2 = N perp BC;
11 point O = k1 intersect k2;
12 % point O = (middle(A, B) perp AB) intersect (middle(B, C) perp BC);
13
14 circle circum = cr(O, dist(A, O));
15 % circle circum = compl(cr(A, B, C));
16
17 draw(A, B, C, AB, BC, AC, circum);
```

Листинг 5. Построение описанной окружности с помощью DROL

Поясним данное описание. Сначала определяются параметры — длина основания l и угол при основании α. Точка A совпадает с началом координат, а точка B лежит правее на l единиц (в данном случае — миллиметров). Лучи r_1 и r_2 выходят из точек A и B под углами α и $\pi - \alpha$ соответственно. Далее строится центр описанной окружности как точка пересечения срединных перпендикуляров и сама окружность. Заметим, что язык DROL поддерживает выражения, и, если вспомогательные построения нам не нужны, строки 9–11 можно заменить строкой 12 (раскомментировав последнюю). Более того, возможно и альтернативное построение описанной окружности: вместо использования срединных перпендикуляров можно построить дугу, соединяющую точки A и C через точку B, после чего с помощью функции `compl` дополнить эту дугу до окружности (в этом случае строки 9–14 можно заменить строкой 15).

Иногда возникает необходимость в подготовке серии связанных рисунков. Например, это могут быть иллюстрации шагов некоторого алгоритма. Зачастую существенная часть элементов остаётся неизменной от итерации к итерации. В этом случае возможно сократить трудозатраты следующим образом: единожды опишем все геометрические объекты, появляющиеся на рисунках, и установим связь между объектами и рисунками (в языке DROL это можно сделать с помощью условного оператора — в зависимости от задаваемого

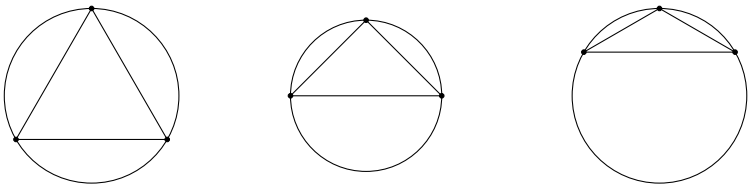

Рис. 9. Описанные окружности, нарисованные с помощью DROL: $\alpha = \frac{\pi}{3}, \frac{\pi}{4}, \frac{\pi}{6}$

номера рисунка можно изображать соответствующие ему объекты). Описание того, какие объекты присутствуют на каждом из рисунков, требует некоторых дополнительных усилий. Но простое копирование повторяющихся объектов — плохая идея, т. к. если вдруг понадобится что-то изменить — придётся менять во многих местах сразу, что и трудоёмко, и влечёт ошибки.

Продемонстрируем результат применения данного подхода для иллюстрирования шагов алгоритма Грэхема построения выпуклой оболочки множества точек. Алгоритм включает две фазы: сортировку точек по полярному углу и обход отсортированных точек с анализом угла поворота, образованного тремя текущими точками. В случае поворота влево происходит переход на одну точку вперед, в случае поворота вправо — средняя точка является внутренней точкой и удаляется из списка. Результат применения алгоритма Грэхема к некоторому множеству точек представлен на рис. 10.

9°. Различных систем динамической геометрии было создано тоже довольно много. Неполный список представлен ниже:

1) Cabri Geometry
2) C.a.R.
3) CaRMetal
4) Cinderella
5) DrGeo
6) **GeoGebra**
7) Geom
8) The Geometer's Sketchpad
9) Geometry Expert (GEX)
10) GEUP
11) Kig Free
12) KSEG

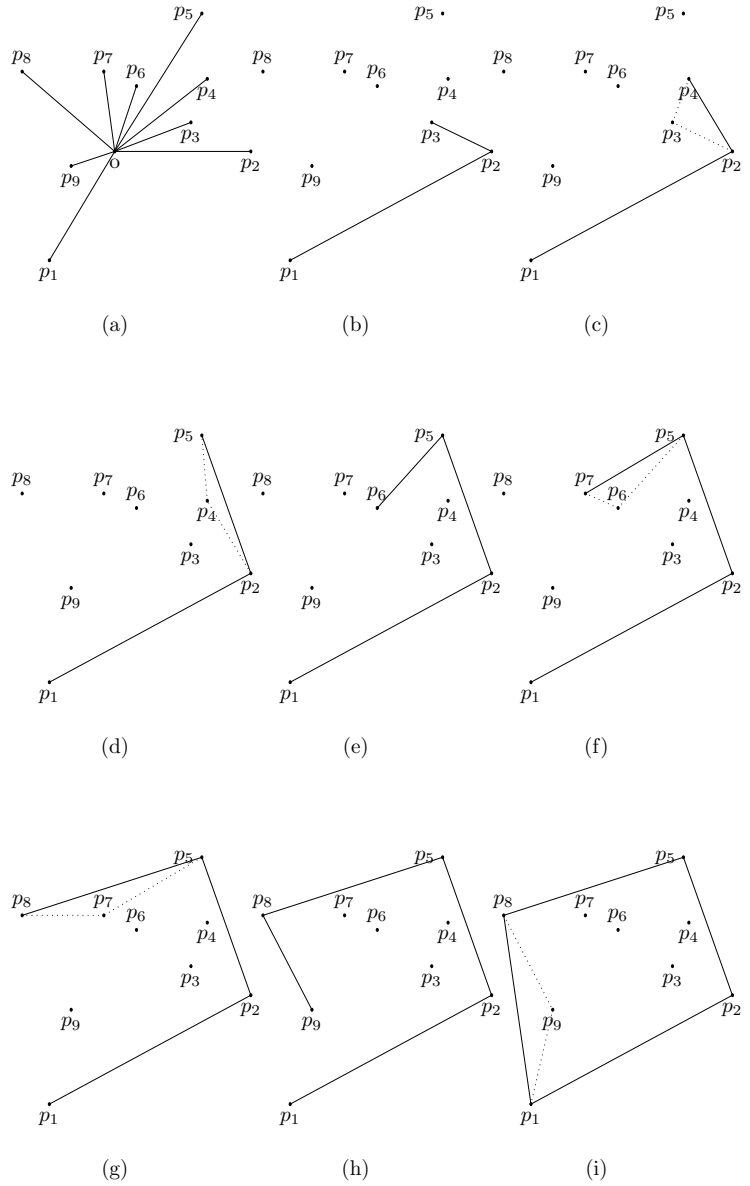

Рис. 10. Шаги алгоритма Грэхема. (a) — сортировка точек по углу; (b) — угол $p_1p_2p_3$ — левый; (c) — угол $p_2p_3p_4$ — правый. Исключаем p_3. Угол $p_1p_2p_4$ — левый; (d) — угол $p_2p_4p_5$ — правый. Исключаем p_4. Угол $p_1p_2p_5$ — левый; (e) — угол $p_2p_5p_6$ — левый; (f) — угол $p_5p_6p_7$ — правый. Исключаем p_6. Угол $p_2p_5p_7$ — левый; (g) — угол $p_5p_7p_8$ — правый. Исключаем p_7. Угол $p_2p_5p_8$ — левый; (h) — угол $p_5p_8p_9$ — левый; (i) — угол $p_8p_9p_1$ — правый. Исключаем p_9. Угол $p_5p_8p_1$ — левый, выпуклая оболочка построена.

Многие из этих систем давным-давно не обновлялись. Впрочем, это ещё не значит, что они совершенно бесполезны и неработоспособны — возможно, авторы реализовали всё, что считали нужным, и теперь система просто работает. Однако, учитывая тенденции развития IT-отрасли (старое ПО может не поддерживаться современными ОС, не работать с новыми версиями библиотек; исходные коды могут отсутствовать и внесение исправлений может быть проблематичным), то, что работает сейчас, запросто может перестать работать завтра. И в этом плане активно развивающаяся GeoGebra выглядит наиболее перспективно.

ЛИТЕРАТУРА

1. Балдин Е. М. *Компьютерная типография LaTeX.* — СПб.: БХВ-Петербург, 2008.

2. Гуссенс М., Ратц С., Миттельбах Ф. *Путеводитель по пакету LaTeX и его графическим расширениям.* — М.: Мир: Бином ЛЗ, 2002.

3. Кнут Д. Э. *Всё про METAFONT.* — М.: Издательский дом «Вильямс», 2003.

4. Кнут Д. Э. *Всё про TeX.* — М.: Издательский дом «Вильямс», 2003.