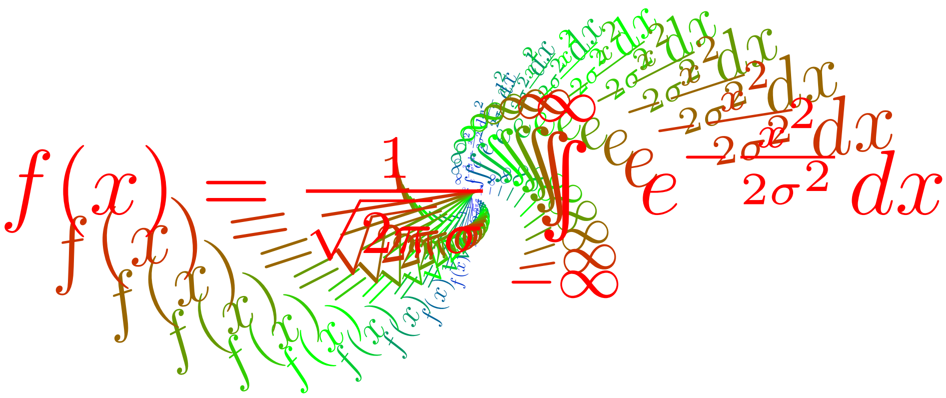
Создание иллюстраций в MetaPost. (Версия 1.01)

🄯 Е.М. Балдин*



Этот текст основан на цикле статей опубликованных с 6го (февральский) по 10ый (июньский) номера русскоязычного журнала Linux Format за 2006 год.

Текст распространяется под лицензией Creative Commons Attribution-Share Alike 3.0 License (CC-BY-SA-3.0). Если будет необходимость перелицензировать всё под другой свободной лицензией, то свяжитесь со мной по электронной почте. Замечания и предложения принимаются с благодарностью.

*e-mail: `mailto:E.M.Baldin@inp.nsk.su`.

Оглавление

Глава 1

Введение в MetaPost

Каждая иллюстрация — это целая история.

Люди делятся на тех, кто умеет рисовать, и тех, кто не умеет. Причём вторых большинство. В этом нет ничего плохого или хорошего — такова жизнь. Хороший иллюстратор — редкость.

Компьютерные технологии дают возможность создавать высококачественные документы особенно не концентрируясь на проблеме оформления. Всю черновую работу делает за вас компьютер. Например, система LaTeX фактически заменяет собой электронную типографию, но, к сожалению, создание иллюстраций выносится за рамки этого процесса. Умение иллюстрировать свой текст является необходимым навыком для тех, кто хочет делать свои книги самостоятельно от начала и до конца. Часто бывает что основной смысл несут в себе именно рисунки.

Создание иллюстраций это длительный и тяжёлый процесс, но если целью стоит совершенство вашей книги, то потраченное время того стоит. Обучение в изостудии в младших классах подняло мой уровень рисования с «никуда не годного» до «терпимого, если отойти на сто метров». Как художник я совершенно безнадёжен, но иллюстрации мне приходится делать довольно часто. В основном это не сложные картинки к задачам по физике. Когда мне говорят, что мои рисунки к задачам вызывают чувство зависти — мне с одной стороны приятно, но с другой стороны очень тяжело объяснить как же я этого достиг. Цель этого текста — популяризовать создание иллюстраций с помощью MetaPost.

Желание контролировать всё в процессе создания книги в случае Дональда Э. Кнута (Donald E. Knuth) привело к созданию программ TeX и METAFONT.

Изначально METAFONT предназначался для создания шрифтов и результатом его работы был растр с изображением шрифта. Позже аспирант Д.Э. Кнута Джон Хобби (John Hobby) модифицировал METAFONT таким образом, что результатом работы программы стала картинка в формате EPS (Encusulated PostScript).

MetaPost — это программа для тех, кто может объяснить компьютеру что он хочет. MetaPost чрезвычайна полезен для случаев, когда картинку проще описать логически, нежели образно. Даже если Вы не умеете рисовать, результат может получиться вполне приличный, потому что «виртуальный карандаш» под управлением компьютера дрожать не будет.

MetaPost проектировался как простая программа, которую можно относительно быстро настроить под свои нужды. Он вполне обозрим и компактен. Многие «правильные» программы, работающие с векторной графикой, такие как `gnuplot` и `xfig`, умеют экспортировать в MetaPost.

1.1. Здравствуй, Мир!

Когда изучается новая программная технология первое что надо сделать, это сказать: «Здравствуй, Мир!». Что и сделаем, правда, пока на английском. Для этого следует создать файл `helloworld.mp` со следующим содержанием:

```
% Для просмотра
prologues := 1;
% Простой Hello, World!
beginfig(1);
  label("Hello,␣World!␣␣␣␣␣",(0,0));
endfig;
% <<конец обработки>> - необходимо вставить в конце файла
end.
```

Далее скомпилируем этот код:

```
> mpost helloworld.mp
```

В результате полученный файл `helloworld.1` можно посмотреть с помощью любой программой понимающей `PostScript`, например, с помощью `gv`:

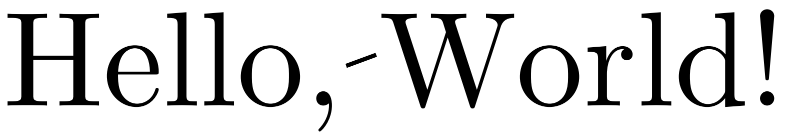


Рис. 1.1. Первый Hello World на MetaPost

Разнообразим пример. Создадим файл `linuxformat.mp` со следующим содержимым:

```
% А теперь мы его завращали и раскрасили
beginfig(1);
  for alpha:=90 step -3 until 0:
    label(btex Linux Format etex
      scaled (5*(1-alpha/100)) rotated alpha,(0,0))
    withcolor (max(1-alpha/45,0)*red+
      min(alpha/45,2-alpha/45)*green+
```

```
    max(alpha/45-1,0)*blue);
  endfor;
endfig;
end.
```

Скомпилируем код и результат преобразуем в pdf:

```
> mpost linuxformat.mp
> pstopdf linuxformat.1
```

Полученный в результате этой операции файл `linuxformat-1.pdf` можно посмотреть, например, с помощью программы `acroread` или `xpdf`.

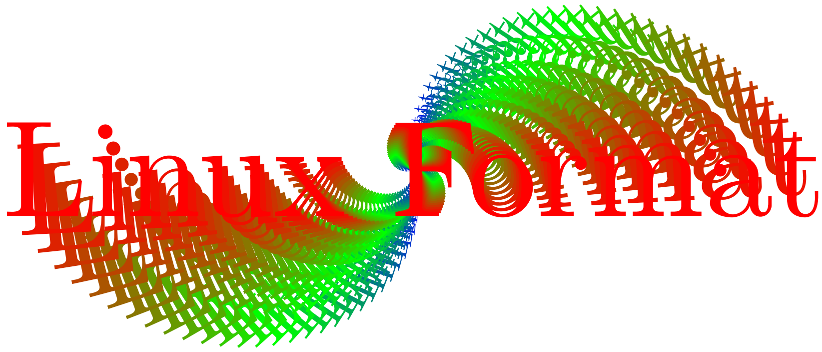


Рис. 1.2. «Продвинутый» Hello World на MetaPost

Всё, что находится между `btex` и `etex` обрабатываются внешней программой. По умолчанию это TEX. Чтобы воспользоваться возможностями LATEX надо в самом начале `linuxformat.mp` строчки вида:

```
verbatimtex \documentclass{article}
\begin{document}
etex;
```

Эти строчки указывают `mpost`, что при обработке любых текстовых вставок им должны предшествовать команды `\documentclass{article}` и `\begin{document}`.

Добавим в `linuxformat.mp` ещё один рисунок. Для этого скопируем уже имеющийся код первого рисунка, вставим его до заключительной команды «`end.`» и немного подправим первые три строчки:

```
% Математика вместо Hello, World!
beginfig(2);
  for alpha:=90 step -9 until 0:
    label(btex \(f(x)=\frac{1}{\sqrt{2\pi}\,\sigma}
             \int\limits_{-\infty}^{\infty}
                e^{-\frac{x^2}{2\sigma^2}}dx\) etex
...
```

Создание новых картинок на базе уже имеющегося кода в MetaPost — обычное дело. Со временем накапливается *своя* библиотечка примитивов, что сильно облегчает создание новых иллюстраций. Кажущаяся сложность выражений компенсируется исключительной гибкостью языка, позволяющая повторно использовать набранный код.

Скомпилируем код ещё раз:

```
> mpost -tex=latex linuxformat.mp
> pstopdf linuxformat.2
```

Таким образом можно отобразить любую сколь угодно сложную математическую конструкцию:

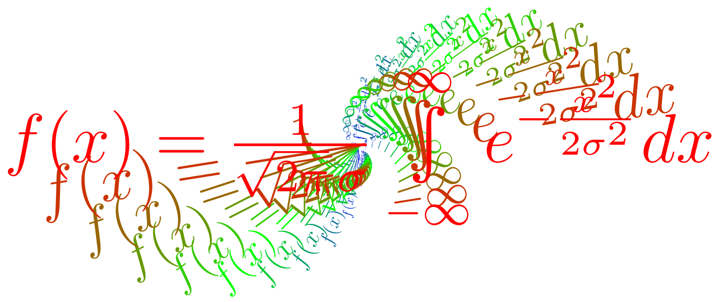


Рис. 1.3. Математическая вставка

Вместо того чтобы запускать `mpost` с опцией `-tex=latex`, можно установить переменную окружения `TEX` равную `latex`. В `bash` это можно сделать так:

```
> export TEX=latex
```

Как видно из предыдущего примера, MetaPost и LaTeX умеют неплохо работать вместе. Усовершенствуем немного `linuxformat.mp`. Заменим первые три строчки в файле на следующий код:

```
verbatimtex \input{preheader}
\begin{document}
etex;
```

и создадим файл `preheader.tex` с примерно следующим содержанием:

```
\documentclass[12pt]{article}
% простейшая кириллизация
\usepackage[koi8-r]{inputenc}
\usepackage[english,russian]{babel}
\usepackage{indentfirst}%first paragraph indent
\usepackage{graphicx}
```

Это стандартный заголовок для файлов LaTeX. Если Вы набираете свои тексты в LaTeX, то можете поместить в `preheader.tex` всю свою преамбулу и включить её с помощью команды `\input`. Это позволит создавать текстовые вставки в картинках MetaPost тем же шрифтом, что и в обычном тексте, а также позволит пользоваться любыми предопределёнными вами командами.

Следует обратить внимание, что чем больше будет разрастаться заголовок, тем дольше будет обрабатываться `META`-код. Для увеличения скорости лучше всего воспользоваться классом `minimal`, вместо `article` и убрать все ненужные для рисования картинок пакеты.

Добавим третью картинку в `linuxformat.mp`, чуть изменив первоначальный код:

```
verbatimtex \input{preheader}
\begin{document}
etex;

...

% изменим направление вращения
beginfig(3) ;
  for alpha:=-90 step 3 until 0:
    label(btex LinuxFormat в России etex
      scaled (5*(1+alpha/100)) rotated alpha,(0,0))
    withcolor (max(1+alpha/45,0)*red+
      min(-alpha/45,2+alpha/45)*green+
      max(-alpha/45-1,0)*blue);
  endfor;
endfig ;
end.
```

Теперь мы можем говорить по русски.

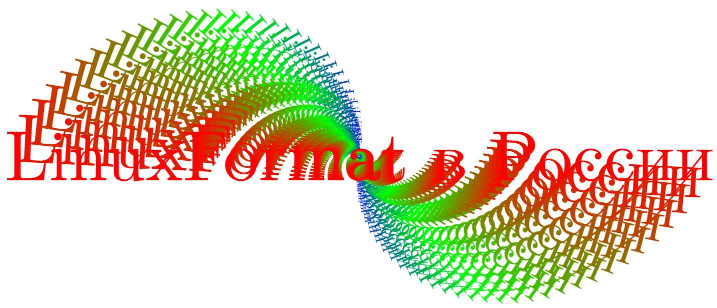


Рис. 1.4. Говорим по русски

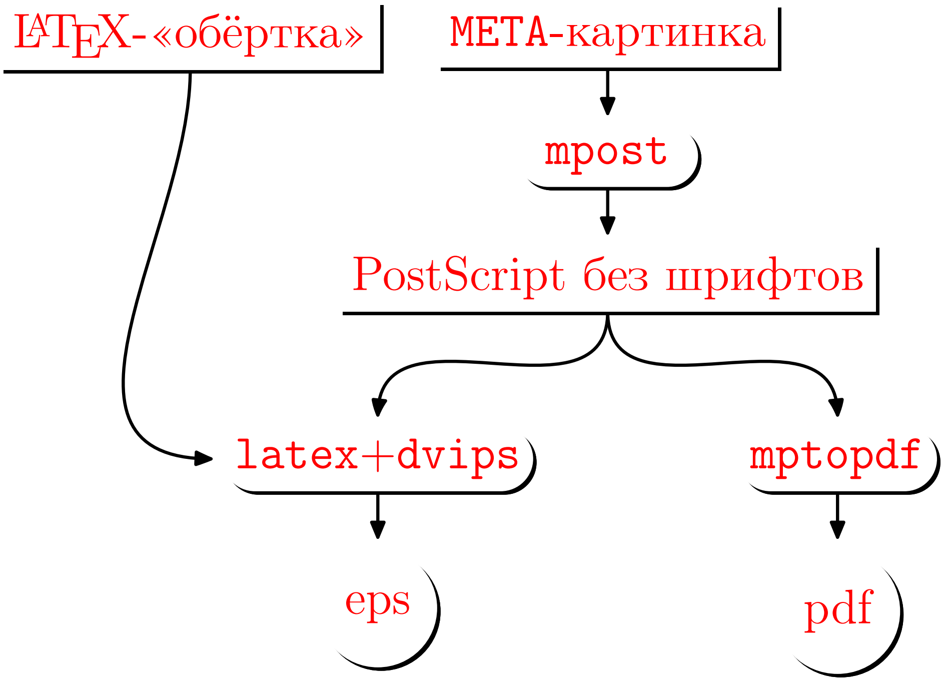


Рис. 1.5. MetaPost-конвейер

1.2. MetaPost-конвейер

Чуть подробнее остановимся на том что же происходит. На вход программы `metapost` подаётся «META-картинка». «META-картинка» — это текстовый mp-файл с инструкциями на языке META. В одном mp-файле можно хранить несколько описаний картинок. При компиляции с помощью `mpost` создаются файлы с тем же именем, что и у исходного файла, но с расширениями в виде чисел, которые указываются в декларации `beginfig`. Результирующие файлы сразу можно вставлять в LaTeX-тексты с помощью обычного `\includegraphics`. Для этого достаточно в заголовок tex-файла добавить команду из LaTeX-пакета `graphicx`:

```
\DeclareGraphicsRule{*}{eps}{*}{}
```

От «правильных» eps-файлов они отличаются только тем, что в них *не* «внедрены» шрифты, поэтому просмотреть их без дополнительной обработки не удастся.

Шрифты можно внедрить посредством программ `latex` и `dvips` с результатом в виде eps-файла или скрипта `mptopdf` с результатом в виде pdf-файла. Эти картинки уже можно использовать независимо любой программой, которая поддерживает эти векторные форматы.

1.3. Среда разработки

Удобная среда разработки понятие весьма относительное. Удобство зависит исключительно от Ваших предпочтений. Здесь будет представлена среда удобная для *меня*. Подробности настройки и установки упоминаемых здесь пакетов выходят за рамки этого теста.

В качестве базовой операционной системы я использую GNU/Linux. Это позволяет мне использовать для своей работы стандартные средства обработки текстовых данных, которые поставляются с этой системой. Например, ниже будет упомянута утилита `make`, которая позволяет автоматизировать ряд стандартных действий.

В качестве базового дистрибутива LaTeX я использую дистрибутив TeX`Live`. На сегодня этот дистрибутив является наиболее полным из дистрибутивов LaTeX. Установочный образ для CD можно взять на любом из `CTAN`-архивов (Comprehensive TeX Archive Network).

В качестве базового редактора я использую редактор `emacs`. Если Вы работаете с текстами в LaTeX-формате, то при настройки этого редактора следует включить пакеты `auctex`[1] и `reftex`[2]. `emacs` имеет простейшую встроенную поддержку для редактирования файлов в формате MetaPost.

Для автоматизации создания eps-картинок я использую `Makefile` примерно следующего вида:

```
# временный файл
tmp_file := tmp_file
# программы
LATEX := latex
MPOST := mpost -tex=latex
DVIPS := dvips

all:
        @echo "run: make mpfile.n.eps"

%.eps:  % preheader.tex
        @echo  "\input{preheader}">$(tmp_file).tex
        @echo  "\DeclareGraphicsRule{*}{eps}{*}{}">>\
                                        $(tmp_file).tex
        @echo  "\nofiles">>$(tmp_file).tex
        @echo  "\begin{document}">> $(tmp_file).tex
        @echo  "\thispagestyle{empty}">> $(tmp_file).tex
        @echo  "\includegraphics{$(basename $@)}">>\
                                        $(tmp_file).tex
        @echo  "\end{document}">> $(tmp_file).tex
        @$(LATEX) $(tmp_file)
        @$(DVIPS) -E -o $@ $(tmp_file)
        @rm $(tmp_file).*

clean:
```

[1] Продвинутая система для работы с проектами LaTeX.

[2] Автоматизация работы со ссылками.

```
        @rm  -f mpx* *~ *.log *.mpx
        @rm -f $(tmp_file).*

# Зависимости для mpost-картинок.
# По одной для каждого числа из beginfig
%.1: %.mp preheader.tex preheader.mp
        $(MPOST) $<
...
%.64: %.mp preheader.tex preheader.mp
        $(MPOST) $<
```

Чтобы на выходе получить готовую eps-картинку с уже «внедрёнными» шрифтами, которую можно вставить уже куда угодно достаточно выполнить следующую команду:

```
> make <имя mp–файлы>.<номер картинки>.eps
```

Обычно, mp-файлам даются короткие имена.

Если в качестве результирующего формата картинок вас интересует pdf-формат, то в `Makefile` можно добавить что-то вроде:

```
MPTOPDF := mptopdf
MV := mv

%.pdf:  % preheader.tex
        @$(MPTOPDF) $<
        @$(MV) `echo $< | sed -e \
                "s/\.\([0-9]\+\)$$/-\1.pdf/"` $<.pdf
```

1.4. Чуть-чуть о META

В качестве базового языка, инструкции которого подаются на вход программы MetaPost используется язык `META`.

Язык `META` поддерживает следующие типы данных:

boolean объект булева типа, то есть либо true, либо false,

numeric обычное число (по умолчанию, если переменная не описана то она считается numeric),

pair точка — пара чисел (x,y) в случае декартовых координат или R$*$dir α в случае полярных координат,

pen перо — то, чем компьютер рисует (в подавляющем большинстве случаев используется круглое перо `pencircle`),

color цвет — тройка чисел (r,g,b),

path путь — совокупность точек с описанием соединения между ними,

picture картинка — совокупность путей и точек,

string строка — ASCII строка,

transform преобразования — линейные преобразования, которые можно применять к объектам типа `pair`, `pen`, `path` и `picture`.

Имена переменных в `META` могут состоять из нескольких лексем. Лексемы могут быть либо буквенными, либо числовыми. Например, переменная `x1l` состоит из трёх лексем. Её можно переписать более понятным способом `x[1].l`, то есть числовая лексема по сути указывает на номер элемента в массиве, а следующая за ней буква уточняет элемент структуры. Возможность упускать «`[].`» в написании имён переменных упрощает в некоторых случаях восприятие кода (x_{1l} — это x-координата границы линии слева по направлению движения для первой точки пути `z[]`) и сокращает объём программы. В замен, если Вам нужны просто переменные без подобных особенностей, то Вам придётся ограничиться только *буквенными* комбинациями.

Все переменные необходимо объявлять перед использованием. Исключением являются переменные типа `numeric`. Массивы объявляются и используются следующим образом:

```
pair w[];
w1:=(10,5);
w[2]=w[1];
```

Взаимодействие переменных, чисел и операторов вполне естественно, но достаточно нетривиально. Описание этого достойно отдельного раздела. В любом случае следует действовать по правилу: если сомневаетесь, то расставляйте скобки в нужных местах.

В `META` можно упускать некоторые из операторов для сокращения записей, например, `2*x` соответствует записи `2x`. Но будьте осторожны, так как `1/2x` это `0.5x`, что более естественно с точки зрения математики, но не программирования. В `META` сначала обрабатываются числовые лексемы.

Набор стандартных вычислительных операций расширен с учётом специализации языка. В частности поддерживаются операции пифагорова сложения $a{+}{+}b = \sqrt{a^2 + b^2}$, пифагорова вычитания $a{+}{-}{+}b = \sqrt{a^2 - b^2}$), целочисленное деление `div` и возведение в степень $x{*}{*}y = x^y$.

В языке присутствуют операторы цикла, условных переходов и тому подобное. Отличительной особенностью `META` является возможность решать линейные уравнения. Например, выражения вида $\mathrm{C} = 1/2[\mathrm{A}, \mathrm{B}]$ означает, что точка C находится ровно посередине между точками A и B.

Программу `metapost` можно использовать в режиме калькулятора для вычислений на языке `META`. Это позволяет проверить правильность ваших предположений относительно языка. Пример сеанса представлен ниже:

```
> mpost
This is MetaPost, Version 0.901 (Web2C 7.5.5)
**\relax

*a:=10;

*b:=8;

*c:=a+-+b;

*show c;
>> 6
*show (3-sqrt 5)/2;
>> 0.38197
*show angle(1,sqrt 3);
>> 60.00008
*show 2**10;
>> 1024.00003
*show infinity;
>> 4095.99998
*show epsilon;
>> 0.00002
*show infinity-infinity;
>> 0
*end
Transcript written on mpout.log.
```

После вывода приглашения `**` следует набрать команду `\relax`. Далее можно вводить команды MetaPost. Делайть это надо аккуратно, так как этот режим не поддерживает «истории команд». В начале не предполагалось, что MetaPost можно использовать и так тоже. С помощью команды `show` можно вывести результат на экран. Закончить сеанс можно с помощью команды `end`.

Обратите внимание, что на просьбу вывести бесконечность (`infinity`) MetaPost выдал 4095.99998 — это максимальное значение, которое может принемать переменная типа `numeric`. Причём в процессе вычисления результат может привышать «бесконечность», но ответ должен быть меньше или равен её, иначе будет выдана ошибка. Минимальный шаг изменения типа `numeric` равен `epsilon`, или точнее 1/256/256. При создании рисунка эти ограничения не существенны, так как диапазон изменения чисел вполне велик, чтобы вместить все элементы. Но в любом случае это тоже необходимо учитывать.

Если Вы хотите вычислить однострочное выражение, то на первоначальное приглашение `**` можно ввести `expr`. В этом случае `mpost` считает файл `expr.mf` и на любое ваше действие будет выдаваться ответ:

```
> mpost
This is MetaPost, Version 0.901 (Web2C 7.5.5)
**expr
(/usr/local/texlive/2005/texmf-dist/metafont/base/expr.mf
gimme an expr: 2(a+3b)-2b
>> 4b+2a
gimme an expr: 1/3[a,b]
>> 0.33333b+0.66667a
```

1.5. Литература

Язык `META`, который используется в MetaPost за некоторыми исключениями полностью соответствует диалекту `META`, который используется в программе создания шрифтов METAFONT.

Основной книгой по языку `META` является «Всё про METAFONT» Дональда Э. Кнута. В 2003 году издательством Вильямс был выпущен русский перевод этой классической книги (ISBN 5-8459-0442-0). Исходники англоязычного оригинала «The METAFONT book» можно найти на любом `CTAN` архиве. Эта книга, как и другие произведения Д.Э. Кнута имеет несколько уровней сложностей. Даже с нулевым начальным уровнем знания предмета Вы можете прочитать книгу полностью — её строение это позволяет, но для дальнейшего продвижения эту книгу придётся перечитывать не один раз. Каждое прочтение приносит новое понимание. Если Вы имеете хоть какое-то отношение к программированию, то книги Д.Э. Кнута *надо* читать. Именно благодаря подобным людям информатика может претендовать на фундаментальность.

На русском языке информацию о MetaPost можно найти в «Путеводителе по пакету LaTeX и его графическим расширениям» М.Гуссенса, С.Ратца и Ф.Миттельбаха от издательства Мир (ISBN 5-03-003388-2).

Все остальные источники в основном англоязычные. Прежде всего это «A User's Manual for MetaPost» Джона Хобби (Jhon D. Hobby) — файл `mpman.pdf`. Этот текст можно найти в документации к дистрибутиву MetaPost. Этому тексту предшествовало несколько «основополагающих» статей, которые при желании можно легко найти в интернете. Очень качественным руководством пользователя отметился Андрэ Хек (André Heck) `http://remote.science.uva.nl/~heck/Courses/mptut.pdf`. Так же представляет интерес книга «`metafun`» от Ганса Хагена (Hans Hagen) — она находится в открытом доступе, имя файла `metafun-p.pdf`.

В интернете можно найти интересный ресурс под названием «Metapost : exemples» от Vincent Zoonekynd `http://zoonek.free.fr/LaTeX/`. Это страничка с огромным количеством простых примеров. Небольшое неудобство состоит в том, что это франкоговорящий ресурс, с другой стороны текста там немного. Зеркало расположено, например, здесь `http://tex.loria.fr/prod-graph/zoonekynd/metapost/metapost.html`.

Ко всей перечисленной электронной документации можно получить доступ через страничку CTAN, посвящённую MetaPost: `http://www.tug.org/metapost.html`

Глава 2

Базовые элементы

Определённо всё из чего-то состоит.
Элементарные кирпичики — вот что интересно.

Объяснять компьютеру что Вы хотите сделать гораздо сложнее, чем нарисовать самому. Компьютеру надо объяснять *абсолютно* всё — телепатические способности у машин на текущий момент отсутствуют. Но, объяснив один раз, все похожие действия выполняются путём небольшой модификации уже готовых инструкций. В результате потраченное на объяснение время себя полностью оправдывает, правда, при этом требуются дополнительные «мозговые усилия».

2.1. Рисуем по точкам

Первое, что надо сделать перед созданием нового рисунка, это сделать его набросок на миллиметровки. Допустим, необходимо нарисовать черепашку:

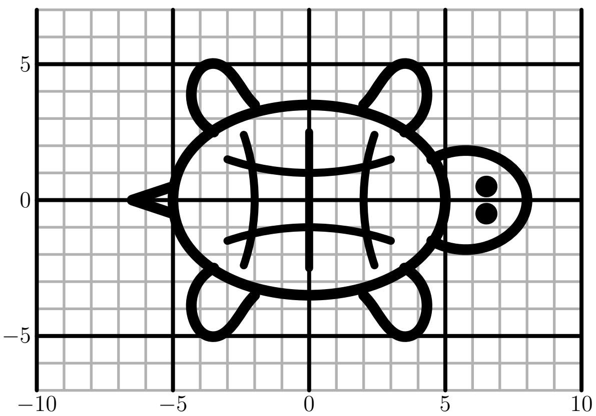


Рис. 2.1. Черепашка

Делаем это, как можем, а затем очень подробно объясняем компьютеру, что мы хотим от него. Инструкции очень простые:

```
% Файл coord.mp
% Черепашка
beginfig(1) ;
 numeric u;u:=5mm;
 draw (-5u,0u){dir 90}..{dir 0}(0,3.5u){dir 0}..
   {dir -90}(5u,0){dir -90}..{dir 180}(0u,-3.5u){dir 180}..
   {dir 90}cycle withpen pencircle scaled 0.4u;
 draw (4.5u,1.5u)..(7u,1.5u)..(8u,0u)..(7u,-1.5u)..
   (4.5u,-1.5u) withpen pencircle scaled 0.4u;
 draw (6.5u,.5u) withpen pencircle scaled 0.8u;
 draw (6.5u,-.5u) withpen pencircle scaled 0.8u;
 draw (-3.5u,-2.5u)..(-4.2u,-4.5u)..(-3.7u,-5u)..
   (-2.4u,-4u)..(-2u,-3.5u) withpen pencircle scaled 0.4u;
 draw (-3.5u,2.5u)..(-4.2u,4.5u)..(-3.7u,5u)..(-2.4u,4u)..
   (-2u,3.5u) withpen pencircle scaled 0.4u;
 draw (3.5u,-2.5u)..(4.2u,-4.5u)..(3.7u,-5u)..(2.4u,-4u)..
   (2u,-3.5u) withpen pencircle scaled 0.4u;
 draw (3.5u,2.5u)..(4.2u,4.5u)..(3.7u,5u)..(2.4u,4u)..
   (2u,3.5u) withpen pencircle scaled 0.4u;
 draw (-5u,0.5u)--(-6.5u,0)--(-5u,-0.5u)
   withpen pencircle scaled 0.4u;
 draw (0u,2.5u)..(0,0u)..(0u,-2.5u)
   withpen pencircle scaled 0.3u;
 draw (-2.4u,2.4u)..(-2u,0u)..(-2.4u,-2.4u)
   withpen pencircle scaled 0.3u;
 draw (2.4u,2.4u)..(2u,0u)..(2.4u,-2.4u)
   withpen pencircle scaled 0.3u;
 draw (-3u,1.5u)..(0,1u)..(3u,1.5u)
   withpen pencircle scaled 0.3u;
 draw (-3u,-1.5u)..(0,-1u)..(3u,-1.5u)
   withpen pencircle scaled 0.3u;
endfig ;
```

Рисуем по точкам, используя команду `draw`. Каждая точка задаётся парой чисел «(x,y)» — x и y координаты соответственно. Точки соединяются либо прямыми линиями «--», либо кривыми «..». Кривые, соединяющие точки описываются полиномом Бернштейна третьей степени (Сергей Николаевич Бернштейн 1912). Часто их называют кубическими кривыми Безье (Piere Bézier 1960).

Компьютер обязательно должен знать каким пером `withpen pencircle` и какой толщины `scaled 0.3u` рисуется текущая линия. Такие объяснения могут показаться избыточными, но всегда можно скопировать предыдущую инструкцию и поправить её под свои нужды, а многословность позволяет легче читать код.

Далее этот рисунок можно использовать многократно, например, для составления какого-либо узора:

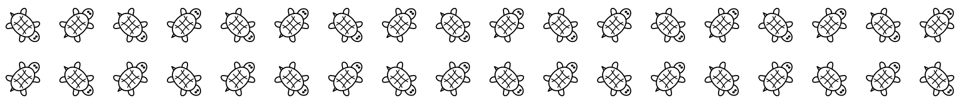


Рис. 2.2. Черепаший узор

Обратите внимание, что почти все числовые значения представляют из себя коэффициент умноженный на параметр, например, «2.4u». Параметру «u» можно присвоить числовое значение в миллиметрах (параметр «mm»). Есть несколько определённых по умолчанию значений длины, например, «cm» соответствует сантиметру.

При программировании на `META` по мере возможности используйте параметрические зависимости. Нет необходимости «прибивать что-то гвоздями». В данном случае наличие параметра позволяет легко изменить масштаб рисунка:

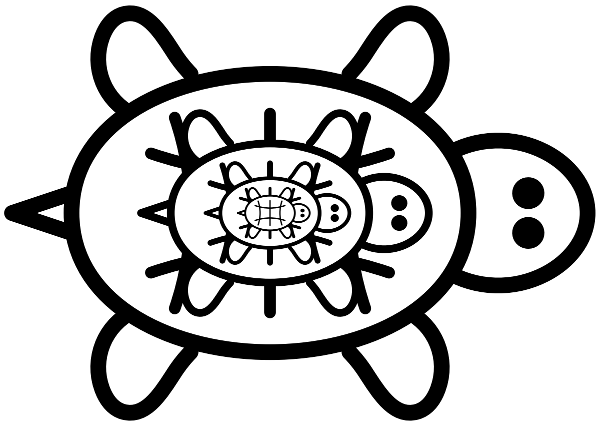


Рис. 2.3. Черепашья семья

Очевидно, что в предложенном решении одного параметра мало, так как «u» контролирует не только геометрические размеры, но и размеры пера.

Точки, тоже можно представить как переменные, имеющие тип `pair`:

```
numeric u;
u:=0.5mm;
pair A,B;
A:=(1u,2u);B=(5u,10u);
draw A--B withpen pencircle scaled 0.3u;
```

Следует обратить внимание, что знак присвоения «:=» в случае переменной A и знак равенства в случае переменной B здесь действует одинаково. Отличия возникают, когда используется уникальная способность `META` воспринимать и решать систему линейных уравнений. В дальнейшем в объяснениях буквы A и B будут использоваться как переменные типа точка.

`META` позволяет создавать свои типы перьев, но для рисования простых рисунков использовать что-то отличное от круглого пера (`pencircle`) нет особой необходимости. Если вам хочется расширить ваши познания в этой области, то обратитесь к любому руководству по METAFONT или MetaPost. Нет необходимости каждый раз указывать каким пером следует рисовать. Достаточно выбрать какое-либо перо по умолчанию с помощью команды `pickup`, например, так:

```
pickup pencircle scaled 0.2u;
```

2.2. Пути

Инструкции `draw` позволяет рисовать сплошную линию. На её основе в MetaPost создана команда для рисования стрелки `drawarrow`. Воспользуемся ей для изображения векторной суммы: Кроме, непосредственно, команды `draw` в этом рисунке

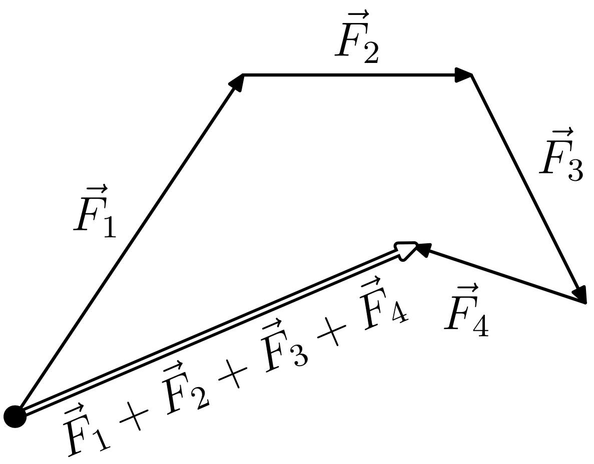


Рис. 2.4. Сложение сил.

необходимо использовать команду для вставки текстовых меток `label`, которая будет подробно разобрана позже:

```
% Файл path.mp
% Закон Ньютона - векторная сумма сил
beginfig(1);
numeric u;
u := 1mm;
drawarrow (0,0)--(20u,30u) withpen pencircle scaled 0.3u;
label.ulft(btex \(\vec{F}_1\) etex,1/2[(0,0),(20u,30u)]);
drawarrow (20u,30u)--(40u,30u) withpen pencircle scaled 0.3u;
label.top(btex \(\vec{F}_2\) etex,1/2[(20u,30u),(40u,30u)]);
drawarrow (40u,30u)--(50u,10u) withpen pencircle scaled 0.3u;
label.urt(btex \(\vec{F}_3\) etex,1/2[(40u,30u),(50u,10u)]);
```

```
drawarrow (50u,10u)--(35u,15u) withpen pencircle scaled 0.3u;
label.llft(btex \(\vec{F}_4\) etex,1/2[(50u,10u),(35u,15u)]);
drawarrow (0u,0u)--(35u,15u) withpen pencircle scaled 0.8u;
drawarrow (0u,0u)--(35u,15u) withpen pencircle scaled 0.3u
  withcolor white;
draw (0u,0u)  withpen pencircle scaled 2u ;
label.bot(btex \(\vec{F}_1+\vec{F}_2+\vec{F}_3+\vec{F}_4\) etex,
  1/2[(35u,15u),(0u,0u)]) rotatedaround
                    (1/2[(35u,15u),(0u,0u)],angle(35,15));
endfig;
```

Конструкция вида «1/2[A,B]» имеет тип `pair` и равна точке, расположенной ровно по середине между точками A и B. Точно так же можно выбрать точку на линии AB, но делящую эту линию 1 к 2: «1/3[A,B]», или 1 к 4: «0.2[A,B]»

Кроме `drawarrow` в MetaPost определена команда `drawdblarrow`, которая рисует кончик стрелки на обоих концах пути.

Команды, типа `draw`, работают с объектами `path` (путь). Путь — это набор из точек (тип `pair`) с описанием как эти точки друг с другом соединяются. Минимальный путь, это одна точка. При рисовании такого пути в рисунке остаётся отпечаток в форме пера. Во всех приведённых здесь примерах перо имеет круглую форму `pencircle`, поэтому в этом случае возникает просто точка. Часто бывает удобно создать переменную типа `path`:

```
% Файл path.mp
% Рис к 1.6.8 б) нить пропущенная через изогнутую трубу
beginfig(2) ;
  numeric u;
  u = 0.8mm;
  path p;
  p:=(5u,0u)--(20u,0u){dir 0}..(20u,20u)..
                        {dir 0}(20u,0u)--(35u,0u);
  cutdraw p  withpen pencircle scaled 1.5u;
  draw p withpen pencircle scaled 1u withcolor white;
  draw p withpen pencircle scaled 0.3u
                        dashed evenly scaled 1/2u;
  drawarrow (-10u,0u)--(0u,0u)
                    withpen pencircle scaled 0.3u;
  draw (-10u,0u) withpen pencircle scaled 1u;
  draw (-10u,0u)--(5u,0u) withpen pencircle scaled 0.3u;
  drawarrow (30u,10u)--(50u,10u)
                    withpen pencircle scaled 0.3u;
  draw (30u,10u) withpen pencircle scaled 1u;
  draw (35u,0u)--(45u,0u) withpen pencircle scaled 0.3u;
  label.top(btex \(\vec{v}\) etex,(-5u,0u));
  label.top(btex \(2\vec{v}\) etex,(40u,10u));
```

```
endfig;
```

В данном примере требовалось изобразить трубу изогнутую буквой «О», сквозь которую продета нить. Для этого определяем путь «p». Затем этот путь используется в трёх командах как переменная: рисуется толстая чёрная труба шириной «1.5u», внутри неё рисуется более тонкая шириной «1u» белого цвета (`withcolor white`) — получается полая труба, а затем внутри трубы отрисовывается нить, причём нить рисуется пунктиром (`dashed evenly`).

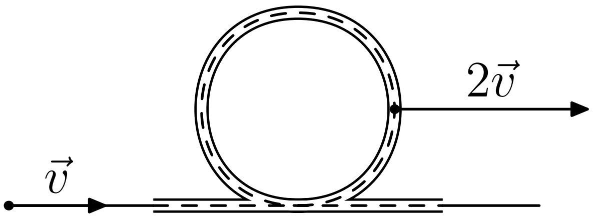


Рис. 2.5. Изогнутая труба.

Первый и последний участок изогнутой трубы прямые линии, поэтому соединение с первой и последней точкой описывается как «--». Чтобы нарисовать кривую, похожую на круг, достаточно двух точек:

```
numeric u,R; u=1mm;R=10u;
pair A,B; A:=(-R,0);B:=(0,R);
path P; P:=A..B..cycle;
draw P withpen pencircle 1u;
```

В этом примере определяются две точки A и B. Путь P строится по этим точкам, при этом выходя из точки A, мы попадаем в точку B, а затем снова в точку A: команда `cycle` позволяет создавать замкнутую кривую. Настройки `META` по умолчанию таковы, что получившаяся кривая достаточно хорошо совпадает с точной окружностью. Чтобы лучше совпадать с окружностью нужно больше «опорных» точек. Просто, обычно, для рисунков точности построения по двум точкам хватает, но всё-таки надо осознавать, что отличие есть.

Для уточнения пути в некоторых точках можно указать под каким углом должна подходить кривая к этой точке. Инструкция вида «$\{\mathrm{dir}\ \alpha\}$» если она находится перед точкой, указывает под каким углом кривая должна подходить, а после точки — под каким углом кривая должна уходить. α — угол в градусах от оси абсцисс. Для указания направления можно так же воспользоваться сокращениями `left`, `right`, `up` и `down`, которые означают, соответственно, влево, вправо, вверх и вниз.

Существует несколько типов соединений между точками. Два из них «..» и «--» уже изучили. Из других типов полезен тип «&» — «сращивание» (объединение двух путей без влияния друг на друга) и «---» — «натянутая» линия (то же, что и «--», но влияет на соседние соединительные участки).

Обратите внимания на команду `cutdraw`. Так как MetaPost рисует с помощью перьев, то в случае круглого пера (`pencircle`), концы линий также округлые. В случае когда необходимо «обрубить» концы, избавиться от округлостей на конца линий, используется эта инструкция. Следует отметить, что эта команда «подчистки» концов использует другую более общую команду cutoff("точка","угол");

2.3. Вставка текста

METAFONT был создан как специализированный инструмент для создания шрифтов. MetaPost создавался как инструмент для рисования любых изображений, поэтому в него был встроен довольно мощный механизм вставки текста. Работая совместно с LaTeX, MetaPost позволяет использовать всю мощь текстового процессора для создания надписей. Кроме всего прочего, иметь в рисунках те же шрифты, что и в тексте просто красиво.

Разберём это на примере треугольник Паскаля[1]:

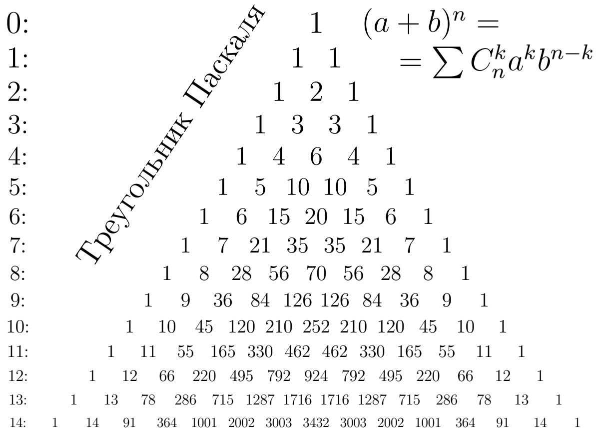


Рис. 2.6. Треуголник Паскаля.

Так как нижние строки из-за увеличения разрядности чисел расползались, то для представления появилась необходимость уменьшить их геометрические размеры. В коде так же использовались циклы, которые будут разобраны позже. Команда `show` позволяет отлаживать код, выдавая значения переменных на экран во время компиляции рисунка с помощью `mpost`.

```
% Файл pic.mp
% Треугольник Паскаля
beginfig(5) ;
  numeric u;
  u = 1mm;
  numeric dy,dx,x,y,n[][] ,i ,j ,sy ,ds ,nlast ;
  dy:=5u;dx:=5u;x=0;y=0;
```

[1] Треугольник Паскаля — треугольная числовая таблица для составления биномиальных коэффициентов. По боковым сторонам треугольника стоят единицы, внутри треугольника числа образуются сложением двух чисел, стоящих над данным.

```
  ds=0.04;sy=0.032;nlast=14;
  picture z;
  for i:=0 upto nlast:
    dy:=dy*(1-sy);
    y:=y-dy;
    for j:=0 upto i:
      if (j=0) or (j=i):
        n[i][j]:=1;
      else:
        n[i][j]:=n[i-1][j-1]+n[i-1][j];
      fi
%       show i,j,n[i][j];
      z:=thelabel(decimal(n[i][j]),(0,0));
      x:=dx*(j-i/2);
      label(z scaled (1-ds*i),(x,y));
    endfor
    z:=thelabel.lft(decimal(i)&":",(0,0));
    label(z scaled (1-ds*i),(dx*(-nlast/2-1),y));
  endfor
  label.rt(btex \((a+b)^n=\) etex,(5u,-5u));
  label.rt(btex \(=\sum C^k_na^kb^{n-k}\) etex,
                                        (10u,-10u));
  label.lft(btex Треугольник Паскаля etex
                            rotated 56,(-5u,-20u));
endfig;
```

Команда `decimal` делает из переменной типа `numeric` строку. Две строки можно слить с помощью операнда «&».

Для того чтобы вставить текстовую метку в рисунок, необходимо получить на входе текстовую строку, которую, возможно, надо будет преобразовать с помощью LaTeX, и точку где эту строку следует расположить:

```
label("text␣string",A);
```

Полезно ещё уточнить с какой стороны от указанной точки расположить текстовую метку. Уточнение производится с помощью суффикса, который добавляется макросу `label` через точку. Всего существует восемь стандартных суффиксов: «.rt» — расположить справа, «.lft» — слева, «.top» — сверху, «.bot» — снизу, «.llft» — расположить снизу и слева по диагонали, «.lrt» — снизу и справа, «.ulft» — сверху и слева, «.urt» — сверху и справа.

Если `label` передаётся просто строка (тип `string`), то текст обрабатывается силами MetaPost и всё, что выходит за пределы ASCII-таблицы с большой вероятностью не отобразится. Для того чтобы строка была обработана LaTeX, необходимо это указать с помощью разделителей `btex` и `etex`. Строка между этими разделителями обрабатывается LaTeX, при этом в качестве заголовка используются инструкции перечисленные в начала mp-файла между `verbatimtex` и `etex`. Таким образом можно использовать кириллицу и любую конструкцию, которую понимает LaTeX.

В некоторых случаях команде `label` удобно передавать не строку, а картинку `picture`. В нашем случае это было необходимо, так как надпись надо было масштабировать. Объект `picture` представляет из себя совокупность примитивов типа путей и точек, поэтому его можно трансформировать. Команда `thelabel` создаёт такую картинку. Объект, заключённый между `btex` и `etex` так же является картинкой.

2.4. Заливка

Кроме рисования кривых часто бывает необходимо закрасить какую-либо замкнутую область.

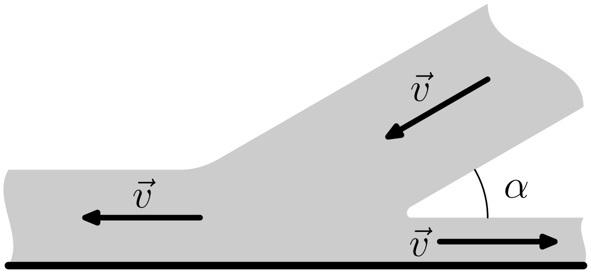


Рис. 2.7. «Струя воды».

Для этого существует команда `fill`. На вход команды `fill` подаётся объект типа `path`, при этом путь должен быть замкнутым, то есть оканчиваться командой `cycle`. Команда `cycle` автоматически замыкает кривую.

```
% Файл pic.mp
% Рис к 1.6.20 а) струя под углом, разбивающаяся о пол
beginfig(10) ;
  numeric u;
  u = 1mm;
  numeric st;st:=u/(sqrt 3);
  path p;
  p:=(0,10u)--(18u,10u){dir 0}..{dir 30}(22u,10u+2st)--
     (50u,10u+30st){dir -90}..{dir -90}(60u,5u+20st)--
     (42u,5u+2st){dir -150}..{dir 0}(42u,5u)--
     (60u,5u){dir -120}..{dir -120}(60u,0u)--
     (0u,0u){dir 60}..{dir 60}cycle;
  fill p withcolor 0.8 white;
  draw ((40u,5u)+10u*dir 30){dir -60}..{dir -90}(50u,5u);
  label(btex \(\alpha\) etex,(53u,8u));
  draw (0u,0)--(60u,0u) withpen pencircle scaled 0.8u;
```

```
  numeric R, alpha ;R:=12u ; alpha=30;
  drawarrow (25u+25u,5u+25st)--
                ((25u+25u,5u+25st)-R*dir alpha)
                         withpen pencircle scaled 0.6u;
  label.ulft(btex \(\vec{v}\) etex,1/2[(25u+25u,5u+25st),
    ((25u+25u,5u+25st)-R*dir alpha)]);
  drawarrow (20u,5u)--(20u-R,5u)
                     withpen pencircle scaled 0.6u;
  label.top(btex \(\vec{v}\) etex,(20u-R/2,5u));
  drawarrow (45u,2.5u)--(45u+R,2.5u)
                     withpen pencircle scaled 0.6u;
  label.lft(btex \(\vec{v}\) etex,(45u,2.5u));
endfig ;
```

Для `fill` существует противоположная по смыслу команда `unfill`, которая, соответственно, убирает заливку в выбранной окрестности. Если вы планируете использовать эти команды, то изучите соответствующий раздел «Всё про METAFONT» Кнута, так как влияние этих команд друг на друга не совсем тривиально. Дело в том, что при наложении друг на друга двух заливок в месте пересечения образуется «двойной слой краски» и для того чтобы убрать его необходимо дважды вызывать команду `unfill`. Команда `unfill` действует схоже, только число «слоёв краски» становится отрицательным. Для сведения всё к ситуации когда краска либо есть (один слой), либо её нет используется команда «выравнивания» `cullit`.

Обратите внимание, что точку можно задать не только парой чисел: «(x,y)» — Декартова система координат, но и радиусом с направлением «R$*$dir α» — полярные координаты.

2.5. Цвета

Ещё одно важное отличие MetaPost от METAFONT это наличие цвета. Работа с цветом, обычно сложнее чем работа с чёрно-белым рисунком. До сих пор процедура переноса электронного цветного рисунка на бумагу не является тривиальной. Но время идёт, и цветные принтеры становятся всё доступней. Кроме того, хорошие чисто электронные тексты, не привязанные к твёрдой копии, так же становятся весьма распространёнными. Поэтому если необходимо, то следует пользоваться цветом, естественно, если вы знаете что делать. В случае простого рисунка цвет, как правило, только отвлекает.

В MetaPost управление цветом реализовано на довольно низком уровне. Цвет определяется объектом типа `color` и представляет из себя тройку чисел принимающих значение от 0 до 1: «(r,g,b)», где r соответствует красной, g — зелёной, а b — голубой компоненте. Существует пять предопределённых цветовых константы: red (1,0,0), green (0,1,0), blue (0,0,1), а так же black (1,1,1) и white (0,0,0).

Нарисовать объект, выбранным цветом, можно с помощью инструкции `withcolor` за которой следует сам цвет. Цвета можно складывать, вычитать, умножать на число. Пользуясь базовыми определениями можно создать более сложные объекта. Например как этот спектр:

Рис. 2.8. Спектр.

Код, создающий эту картинку, далёк от совершенства, но он простой и его можно улучшать в случае необходимости:

```
% Файл colors.mp
def spectrline(expr ic,c,l,w,dw) =
  begingroup
    save i,ifirst,ilast,istep,icc;
    color icc;
    if (ic>0):istep:=1;ifirst:=0;ilast=255;
    else:istep:=-1;ifirst=0;ilast:=-255; fi;
    if (abs(ic)=1):icc:=red;fi;
    if (abs(ic)=2):icc:=green;fi;
    if (abs(ic)=3):icc:=blue;fi;
    for i:=ifirst step istep until ilast:
      draw ((0u,0u)--(0u,l)) shifted (w+(dw/2*abs(i),0))
      withpen pencircle scaled dw withcolor (c+i/255*icc);
    endfor;
  endgroup;
enddef;

% спектр
beginfig(1) ;
  color current;
  u:=1mm;w:=200u;dw:=w/512;l:=5u;
  current:=(1,0,0);
  spectrline(2,current,l,(1/4w,0u),dw)
  current:=(1,1,0);
  spectrline(-1,current,l,(1/2w,0u),dw)
  current:=(0,1,0);
  spectrline(3,current,l,(3/4w,0u),dw)
  current:=(0,1,1);
  spectrline(-2,current,l,(w,0u),dw)
endfig ;
```

В примере используется макрос `spectrline` для уменьшения размера кода.

Глава 3

Начала автоматизации

Компьютер не умеет читать ваши мысли,
зато неукоснительно следует инструкциям.

До сего момента мы концентрировались на том, как объяснить компьютеру, чтобы он сделал то или иное движение. Теперь воспользуемся способностью компьютера помнить предыдущие действия и извлекать их из памяти по мере необходимости. Автоматизация рутинных процедур это то, для чего компьютер и предназначены. Практиковаться в автоматизации следует постоянно. Не смотря на затраченное на обучение время, в результате время же и экономится.

3.1. Объекты picture

В процесс повествования объект `picture` или картинка уже упоминался. Картинка представляет из себя совокупность путей и точек которую можно подвергать трансформации. В уже существующие картинки можно добавлять пути, замкнутые области и другие картинки.

Для начала опять же воспользуемся миллиметровкой для отрисовке какого-либо рисунка, например, ракеты:

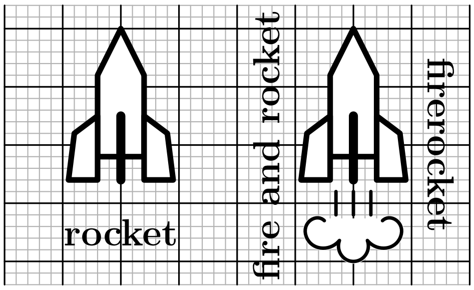


Рис. 3.1. Ракета

Ракета может быть без выхлопа (rocket) и с выхлопом (firerocket). В процессе создания firerocket был использован рисунок самого выхлопа (fire).

```
% Файл picture.1.mp
% Ракета без выхлопа 10x12 Центр у стабилизаторов
picture rocket;
rocket:=nullpicture;
addto rocket contour (-2,-1)--(-2,6)--(0,10)--(2,6)--
  (2,-1)--cycle withpen pencircle scaled 0.4 withcolor white;
addto rocket doublepath (-2,-1)--(-2,6)--(0,10)--(2,6)--
  (2,-1)--cycle withpen pencircle scaled 0.5;% Корпус
addto rocket contour (-2,2.5)--(-4,1)--(-4.5,-3)--
  (-2,-3)--cycle withpen pencircle scaled 0.4 withcolor white;
addto rocket doublepath (-2,2.5)--(-4,1)--(-4.5,-3)--
  (-2,-3)--cycle withpen pencircle scaled 0.5;% левая дюза
addto rocket contour (2,2.5)--(4,1)--(4.5,-3)--(2,-3)--cycle
  withpen pencircle scaled 0.4 withcolor white;
addto rocket doublepath (2,2.5)--(4,1)--(4.5,-3)--
  (2,-3)--cycle withpen pencircle scaled 0.5;% правая дюза
addto rocket doublepath (0,2.5)--(0,-3)
  withpen pencircle scaled 0.8;% центральная дюза

% выхлоп
picture fire;
fire:=nullpicture;
addto fire doublepath (0,-4)--(0,-6)
  withpen pencircle scaled 0.3;%выхлоп 1
addto fire doublepath (-1.5,-4)--(-1.5,-6)
  withpen pencircle scaled 0.3;%выхлоп 2
addto fire doublepath (1.5,-4)--(1.5,-6)
  withpen pencircle scaled 0.3;%выхлоп 3
addto fire contour (-2.5,-6.5){dir 135}..(-4,-8)..
  {dir 50}(-1.2,-8.2){dir -110}..(0,-10)..
  {dir 110}(1.2,-8.2){dir -50}..(4,-8)..
  {dir -135}(2.5,-6.5)--cycle withpen pencircle scaled 0.4
  withcolor white;
addto fire doublepath (-2.5,-6.5){dir 135}..(-4,-8)..
  {dir 50}(-1.2,-8.2){dir -110}..(0,-10)..
  {dir 110}(1.2,-8.2){dir -50}..(4,-8)..{dir -135}(2.5,-6.5)
  withpen pencircle scaled 0.3;% облако

% ракета и выхлоп
picture firerocket;
firerocket:=rocket;
```

```
addto firerocket also fire;
```

Прежде чем к картинке что-то добавить её необходимо инициализировать. В MetaPost есть две определённые по умолчанию картинки: `nullpicture` — пустая картинка и `currentpicture` — текущая картинка. Пользуясь последней переменной, можно в любой момент сохранить результаты промежуточной отрисовки. Добавление элементов к картинке производится с помощью инструкции `addto` далее указывается картинка к которой добавляется тот или иной элемент. Путь добавляется с помощью инструкции `doublepath`, замкнутая область с помощью инструкции `contour`, а другая картинка с помощью инструкции `also`.

Ранее был создан рисунок черепашки. Для его обозначения была выбрана переменная `Turtle`. Теперь с ней можно поработать, как с единым элементом, например, для иллюстрации задачи: «Черепашки расположены в углах правильного треугольника со стороной a и всегда ползут в направлении своей соседки против часовой стрелки со скоростью v. Когда они встретятся?»

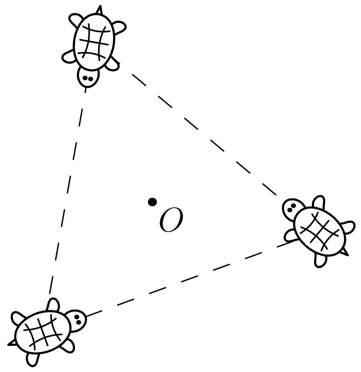


Рис. 3.2. Встреча в центре

Картинку можно отобразить с помощью команды `draw`. Над картинкой можно производить различные преобразования. В данном случае картинка поворачивалась, масштабировалась и сдвигалась.

```
% Файл pic.mp
beginfig(17) ;
  numeric u;u = 0.8mm;
  numeric dphi; dphi=20;
  draw 30u*dir (90+dphi)--30u*dir (210+dphi)--
    30u*dir (330+dphi)--cycle dashed evenly scaled 1u;
  draw Turtle rotated (-120+dphi) scaled 1u
                         shifted (30u*dir (90+dphi));
  draw Turtle rotated dphi scaled 1u
                         shifted (30u*dir (210+dphi));
  draw Turtle rotated (120+dphi) scaled 1u
                         shifted (30u*dir (330+dphi));
```

```
endfig ;
```

Обратите внимание, что линия, соединяющая черепах нарисована пунктиром. Определённая по умолчанию переменная `evenly` тоже является картинкой, поэтому её можно масштабировать с помощью декларации `scaled`. То есть, если вам нужен более широкий шаг пунктира, то вместо масштаба 1u можно указать 2u. Если вас не устраивает где располагаются штрихи у штриховки, то можно воспользоваться декларацией сдвига `shifted`.

Кроме шаблона `evenly` в MetaPost определён шаблон `withdots`, который позволяет рисовать кривую с помощью точек.

Вы можете определить свой шаблон для пунктира примерно следующим образом:

```
picture dash_center;
dash_center:=dashpattern(on 3 off 1.5 on 0.5 off 1.5);
draw 30u*dir (90+dphi)--30u*dir (210+dphi)--
  30u*dir (330+dphi)--cycle dashed dash_center scaled 1u;
```

Функция `dashpattern` принимает список `on/off` с числовой информацией в какой момент рисовать/не рисовать. В этом примере определён шаблон для штрихпунктирной линии, которая обычно используется для обозначения симметрии.

3.2. Трансформация

К задаче №3 варианта ГГФ-51в требовалось изобразить L-образную трубку с водой. По условию трубка сначала стояла вертикально, а потом была положена на стол.

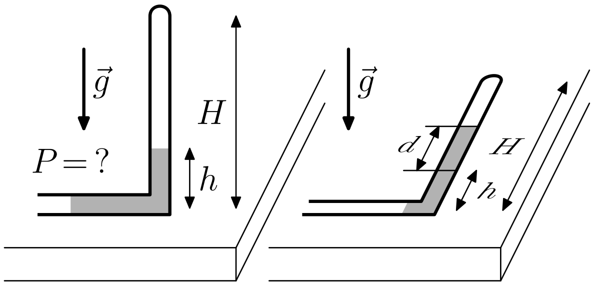


Рис. 3.3. Изогнутая пробирка на столе

Вовсе необязательно работать в трёхмерном редакторе чтобы схематично это изобразить. Ниже идёт код, который рисует вертикально стоящую пробирку с размерами, а затем наклоняет её.

```
% Файл transform.mp
% пример использования slanted
beginfig(1) ;
  numeric u;
```

```
  u = 0.8mm;
  % пробирка
  cutdraw (0u,0u)--(20u,0u)--(20u,30u){dir 90}..
    {dir -90}(17u,30u)--(17u,3u)--(0u,3u)
    withpen pencircle scaled 0.5u;
  drawdblarrow (23u,10u)--(23u,1u);
  label.rt(btex \(h\) etex,1/2[(23u,10u),(23u,1u)]);
  drawdblarrow (30u,30u)--(30u,1u);
  label.lft(btex \(H\) etex,1/2[(30u,30u),(30u,1u)]);
  picture Base;
  Base:=currentpicture; % запоминаем
  clearit; % очищаем текущую картинку
  % рисуем воду когда пробирка будет наклонена
  fill (15u,0u)--(20u,0u)--(20u,20u)--(17u,20u)--
   (17u,3u)--(15u,3u)--cycle withcolor 0.7white;
  draw Base;
  draw (12u,20u)--(20u,20u);draw (12u,10u)--(20u,10u);
  drawdblarrow (14u,20u)--(14u,10u);
  label.lft(btex \(d\) etex,(14u,16u));
  picture Slant;
  Slant=currentpicture; % запоминаем
  clearit; % очищаем текущую картинку
  % рисуем воду когда пробирка стоит
  fill (5u,0u)--(20u,0u)--(20u,10u)--(17u,10u)--
   (17u,3u)--(5u,3u)--cycle withcolor 0.7white;
  % отрисовываем пробирку
  draw Base;
  % отрисовываем пробирку и наклоняем её
  draw Slant yscaled 2/3 slanted 1/2 shifted (40u,0u);
endfig ;
```

В примере применяется возможность сохранить текущее состояние с помощью `currentpicture`, а так же возможность полностью очистить текущую картинку с помощью инструкции `clearit`.

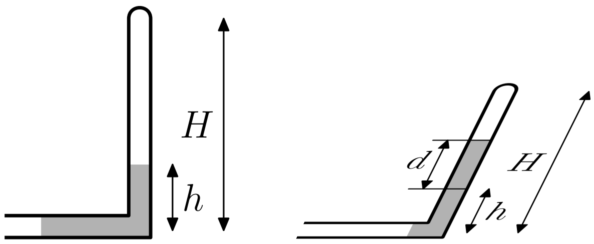


Рис. 3.4. Изогнутая пробирка

Наклон вертикально стоящей пробирки происходит с помощью масштабирования `yscaled` и, собственно, наклона `slanted`.

MetaPost поддерживает следующие базовые линейные преобразования:

$$\begin{array}{ll}
(x,y)\text{ shifted }(a,b) & =(x+a,y+a)\\
(x,y)\text{ scaled }s & =(sx,sy)\\
(x,y)\text{ xscaled }s & =(sx,y)\\
(x,y)\text{ yscaled }s & =(x,sy)\\
(x,y)\text{ slanted }s & =(x+sy,y)\\
(x,y)\text{ rotated }\theta & =(x\cos\theta-y\sin\theta,x\sin\theta+y\cos\theta)\\
(x,y)\text{ zscaled }(a,b) & =(xa-yb,xb+ya)
\end{array}$$

Кроме перечисленных базовых преобразований полезными для использования являются макросы (x,y) rotatedaround $((a,b),\theta)$ — поворот вокруг точки (a,b) на угол θ и (x,y) reflectedabout (z_1,z_2) — отражение относительно линии, проходящей через точки z_1 и z_2.

MetaPost поддерживает объекты типа `transform`, то есть можно определить любое необходимое для вас преобразование, чтобы использовать его в дальнейшем.

```
transform t;
t:= identity yscaled 2/3 slanted 1/2 shifted (40u,0u)
draw Slant transformed t;
```

Используемая при описании преобразования t константа `identity` тоже является преобразованием. `identity` — это «пустое» преобразование, то есть преобразование, которое ничего не делает.

3.3. Циклы и условные операторы

Циклы и условные операторы в `META` отличаются от того, что обычно есть в других языках программирования. Цикл не просто повторяет перечисленные в теле цикла инструкции — он дублирует текст, то есть внутри цикла не обязательно должна находиться синтаксически законченная конструкция. Это же относится и к условным операторам.

«Шарик с постоянной скоростью движется в вдоль спицы, которая с вращается с постоянной угловой скоростью. Требуется изобразить траекторию шарика.»

Для изображения траектории надо построить минимодель явления и задать физические параметры: поступательную скорость вдоль спицы v, угловую частота w и начальные условия r и φ. Сама траектория создаётся с помощью следующего кода:

```
% Файл cycle.mp
v:=27u;w:=360;N:=2.1;phi:=45;r:=5u;n:=100;
path p; pair O;
O:=(r*cosd(phi),r*sind(phi));
p:=O for i=0 upto n:
```

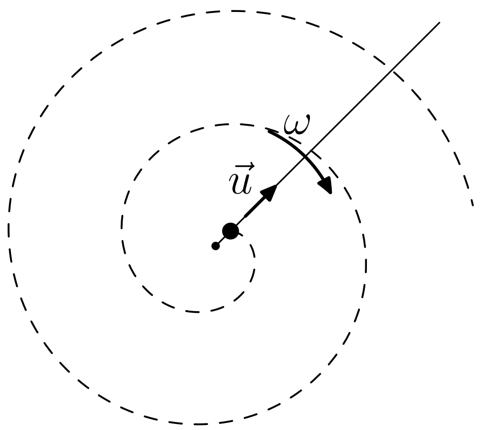


Рис. 3.5. Спираль

```
        ..((r+v*N*i/n)*dir(-w*N*i/n)+phi))
    endfor;
draw p withpen pencircle scaled 0.5u
                             dashed evenly scaled 1u;
```

Декларация `upto` это сокращение для `step 1 until`. Аналогично `downto` является сокращением для `step -1 untull`

Формальный синтаксис цикла представлен ниже:

```
for i=x1 step x2 until x3: text(i) endfor
```

Это одна из форм, которая поддерживается `META`. Ещё одна форма, представляет бесконечный цикл:

```
forever: "текст" endfor
```

Для того чтобы выйти из подобного цикла необходимо воспользоваться конструкцией вида:

```
exitif ("булево_выражение")
```

Булево выражение, это может быть переменная типа `boolean` (true/false) или результат сравнения чисел, точек, путей или преобразований. Операторы сравнения совпадают с операторами сравнения языка «C» за исключением оператора равенства «=» и оператора неравенства «<>». Выражение можно инвертировать с помощью приставки `not` и объединить с другим с помощью приставок `and` или `or`.

Формальный синтаксис условного оператора представлен ниже:

```
if ("булево_выражение1"): "текст1"
elseif ("булево_выражение2"): "текст2"
else: "текст3" fi
```

Воспользуемся циклами для изображения циклоиды — траектория точки на катящемся колесе.

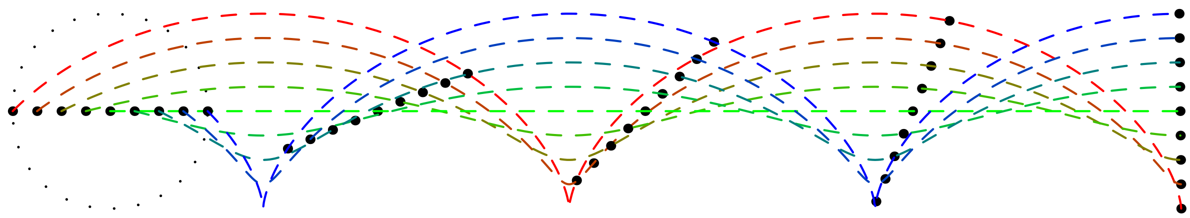


Рис. 3.6. Циклоида

Обратите внимание, что в конце цикла или условного оператора нет необходимости ставить «;» это позволяет использовать их довольно изощрённым образом.

```
% Файл cycle.mp
% Рис к задаче 1.5.8 (20x120) - циклоида
beginfig(1) ;
  numeric u;u = 0.8mm;
  numeric R; R=10u;
  path p,cycl;
  p:=(-R,0u)..(R,0u)..cycle;
  % колесо
  draw p withpen pencircle scaled 0.3u
                                dashed withdots scaled 0.5u;
  numeric j,n,v,w,phi,nsteps;
  j=0;n=100;v=109.8u;w=-(v/R)*180/3.14;phi=180;nsteps=4;
  numeric r,i;
  for i:=0 upto 2*nsteps:
    r:=R-1/nsteps*R*i;
    % метки
    for j:=0 step n/4 until n:
      draw (j*(v/n)+r*cosd(j*(w/n)+phi),r*sind(j*(w/n)+phi))
        withpen pencircle scaled 1u;
    endfor;
    % траектория меток
    cycl:=for j:=0 upto n:
      if j<>0:..fi
          (j*(v/n)+r*cosd(j*(w/n)+phi),r*sind(j*(w/n)+phi))
    endfor;
    draw cycl dashed evenly scaled 1/2u
      withcolor (max(1-i/nsteps,0)*red+
                 min(i/nsteps,2-i/nsteps)*green+
                 max(i/nsteps-1,0)*blue);
  endfor;
endfig ;
```

В этом коде выражение `if j<>0:..fi` использовалось для того, чтобы перед первой точкой пути, описывающем циклоиду, не было декларации соединения. Я не знаю какой из «популярных» на сегодня языков обладает такой способностью.

3.4. Макросы

Пользовательские функции в META фактически заменяются макросами. Как следствие функции могут вернуть любую конструкцию от числа до картинки.

Один из моих ранних рисунков на META был «взрыв» в стакане. Требовалось изобразить траекторию «осколков» которые летят по параболе и найти самую дальнюю точку, которую достигают осколки при таком «взрыве».

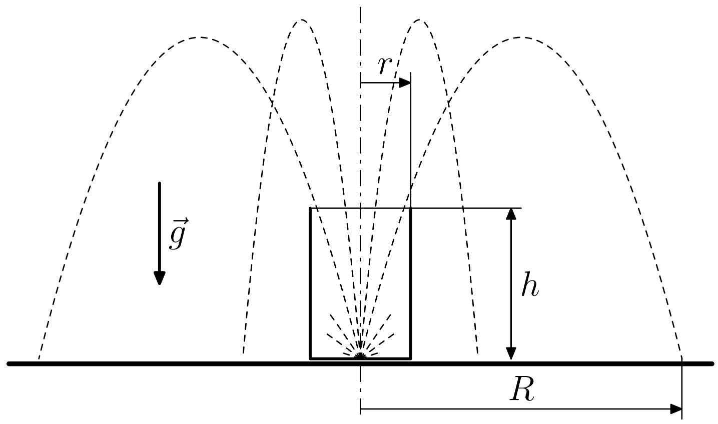


Рис. 3.7. Взрыв в стакане

Была написана процедура, которая рисовала параболу по переданным параметрам. Вызов выглядел примерно следующим образом:

```
Parabola_dashed(0u,0u,-1.25angle(20/sqrt(8),
              10*sqrt(8)),10*sqrt(8)*u,0,100,1);
```

Сам макрос для отрисовки параболы представлен ниже.

```
% Файл macros.mp
% Рисует параболу из точки (x,y) (полёт камня) штриховая
%линия. В качестве параметров передаётся (x,y), ang-угол,
%vel-скорость (100), %from,to - откуда и до куда рисовать
%параболу в процентах [0,100], mag - увеличение (0.8u)
%Для простоты g=10
def Parabola_dashed(expr x,y,ang,vel,from,to,mag) =
  path p;
  numeric t,g,n;
  picture dash_one;
  dash_one:=dashpattern(on 2mag off 2mag);
```

```
  n=100;% число шагов
  g=10.;
  t:=(2*vel*sind(ang)*from)/(g*n);
  p:=(vel*cosd(ang)*t*mag,(vel*sind(ang)*t-g*t*t/2)*mag);
  for i=from+1 upto to:
    t:=(2*vel*sind(ang)*i)/(g*n);
    p:=p..(vel*cosd(ang)*t*mag,
            (vel*sind(ang)*t-g*t*t/2)*mag);
  endfor;
  draw p  shifted  (x*mag,y*mag) dashed dash_one;
enddef;
```

Не самое удачное решение, но оно выполняло то, что от него требовалось. В подобных случаях лучше чтобы в результате деятельности макроса оставался объект, который потом можно нарисовать с помощью команды `draw` и трансформировать по мере необходимости. Тогда функции передавалось бы гораздо меньше параметров, что значительно всё упрощает. Например, вызов для рисования пружинки, которая необходима в следующем примере, использует всего три входных параметра:

```
draw Spring(25u,1.2u,25) rotated -90
  shifted (-12.5u,-20u) withpen pencircle scaled 0.1u;
```

«Два тела, соединённые пружинкой, висят в поле тяжести на нитях, образующих угол в 90°. В какой-то момент нить, крепящую конструкцию к потолку, разрывают.» Надо нарисовать пружинку. Изображение пружинка может пригодится много где

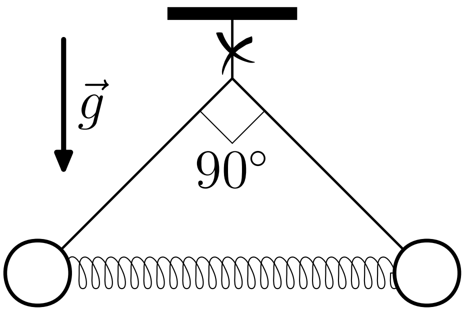


Рис. 3.8. Пружинка

ещё, поэтому оно было оформлено как макрос.

```
% Файл macros.mp
% Создаёт пружину, высоты h, радиуса r, с числом витков n
%(0,0) - в основании пружины
vardef Spring(expr h,r,n) =
  begingroup save i;
    (0,0)--(0,-r/2+0.5h/n){dir 180}
    for i=h/n  step h/n until h:
```

```
      ..tension 1.2..(-r,i-h/n)..tension 1.2 ..
      (0,r/2+i-0.5h/n)..tension 1.2 ..(r,i)..
      tension 1.2 ..(0,-r/2+i+0.3h/n){dir 180}
    endfor--(0,h)
  endgroup
enddef;
```

Обратите внимания на инструкцию `tension` — натяжение. Она говорит с какой «силой» надо «натянуть» соединение между точками. Значение 1.2 означает, что это следует сделать чуть потуже, чем обычно. С помощью этой инструкции описывается соединение между точками в определении пути типа «натянутая прямая»:

```
  def --- = ..tenstion infinity.. enddef;
```

Если отрисовка параболы является аналогом процедуры, то создание пружины аналогом функции. Вызовы `begingroup endgroup` позволяют обособить вычисления, проводящиеся между ними, от «внешнего мира». С помощью команды `save` можно защитить переменные внутри группы — «сохранённые» таким образом переменные восстанавливают свои значения после выхода за пределы `endgroup`.

Параметры, которые передаются внутрь макроса перечисляются после декларации `expr`. Чтобы что-то вернуть в результате исполнения макроса, возвращаемое выражение надо поместить в конце макроса без завершающего символа «;».

Отличие `vardef` от `def` заключается в том, что в случае `def` в качестве названия макроса передаётся «символьная лексема», а в случае `vardef` «объявляемая переменная». Отличие между этими понятиями заключается в том, что объявляемая переменная может состоять из нескольких символьных лексем. Таким образом вы можете создавать переменные с модифицирующимися именами. Если вам этого не надо, то используйте `vardef`.

Средства поддержки макросов в MetaPost исключительно мощные и разнообразные. В частности, с помощью, например, инструкции `primarydef` можно доопределить недостающие бинарные операторы.

3.5. Стандартные функции

Лучший способ облегчить себе жизнь при написании программы, это не писать её, а воспользоваться уже готовыми компонентами. `META` является специализированным языком, поэтому число стандартных функций не очень велико, но их выбор весьма показателен.

«Из точек A и B в море вышли два корабля…» Требуется изобразить поверхность воды: При кодировании этого рисунка использовалась функция генерации случайных чисел:

```
uniformdeviate n
```

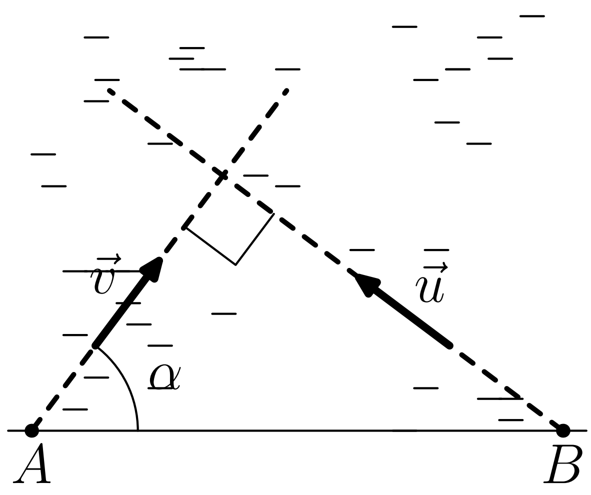


Рис. 3.9. «Водная гладь»

В результате выполнения функции получалось случайное число в интервале $[0, 1]$. Кроме упомянутой функции в `META` есть ещё один генератор случайных чисел `normaldeviate` — генерит числа по распределению Гаусса ($f(x) \sim e^{-x/2}$).

Хотелось бы упомянуть о возможности разлагать сложные объекты на составляющие, например:

```
numeric x[],y[];
pair A,B; A=(x1,y1);A=(x2,y2);
%A=(xpart x1,ypart y1)
color c; c=(r,g,b);
%c=(redpart c,greenpart c,bluepart c)
path p;
p=A--B;
%A = point 0 of p = point 2 of p
%B = point 1 of p = point length p of p
```

Таким образом можно «разобрать» на части любой путь, причём номер точки не обязательно должен быть целым (берётся точка на линии соединения в соответствии с дробной частью). С помощью функции `length` можно узнать число заданных точек в пути, а с помощью `arclength` — его длину.

К уже известным вычислительным функциям `sqrt`, `abs`, `mod`, `round`, `sind` и `cosd` полезно добавить `mlog` ($x = 256 \ln x$) и `mexp` ($x = e^{x/256}$).

Для операций с точками будут полезны функции `angle (x,y)` — вычисления угла наклона к оси абсцисс для вектора (`(0,0)-(x,y)`) в градусах (операция обратная $\mathrm{dir}\,\alpha$) и `unitvector (x,y)` — единичный вектор из начала координат.

Полный список стандартных функций представлен в A User's Manual for MetaPostДжона Хобби. Этот текст идёт со стандартной поставкой LaTeX в виде файл `mpman.pdf`.

Глава 4

Графики и диаграммы

Даже жизнь можно отобразить
в виде графика функции от времени.

Отображение данных на бумаге всегда было и останется не тривиальным процессом. Не смотря на то, что получение данных, как правило, занимает гораздо больше ресурсов, задача оформления графиков достойна автоматизации. Автоматизация, в частности позволяет, поняв как можно представить данные наилучшим образом, в дальнейшем не снижать уровень оформления.

По хорошему графики следует создавать в специализированных приложениях. Автоматизация этого процесса достаточно специфична, чтобы реализовывать его с помощью программ не предназначенных только для этого. Если Вам нужны двумерные графики, то никто не справится с этим лучше чем `gnuplot`. Если вас интересуют гистограммы, то это работа для пакетов анализа `paw/cernlib` или `root`. Но, если Вам хочется разукрасить график, то MetaPost будет для этого очень кстати.

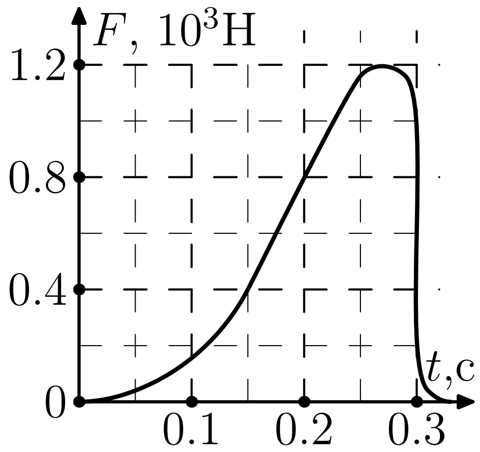


Рис. 4.1. График, созданный в ручную

Ничего не мешает нарисовать график самостоятельно. Всё что для этого надо:

- знать диапазон функции и её аргумента,
- нарисовать оси координат,
- сделать решётку и поставить метки на осях,
- изобразить функцию.

В этом смысле кодирование графика ничем не отличается от кодирования рисунка — всё что надо уметь это рисовать стрелки и линии. Однако, это годится только для простых графиков.

4.1. Графики в MetaPost

Для серьёзной работы с графиками в MetaPost есть довольно продвинутый пакет `graph.mp`. Этот пакет написан «отцом» MetaPost Джоном Хобби и к нему прилагается подробная документация, которую можно найти в стандартной поставке LaTeX в виде файла `mpgraph.pdf`. Если Вы планируете воспользоваться этим пакетом, то прежде изучите этот текст.

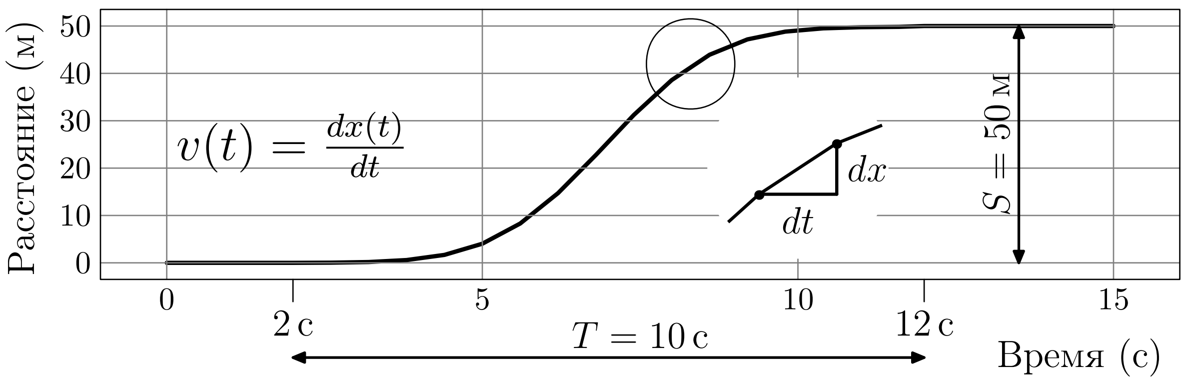


Рис. 4.2. Иллюстрация для объяснения понятия дифференцирования

```
% Файл graphics.mp
input graph;
% График переменной скорости - тема "дифференцирование"
beginfig(2) ;
  numeric u;
  u = 0.6mm;
  % начало графика
  draw begingraph(12cm,3cm);
    path r;
    numeric e,pi,A,sigma,n,scale,mean,y[],x[];
    e:=2.718;pi:=3.14159;A=50;
```

```
    sigma=1.4;n:=100;scale=10;mean=5;
    % создание функции - она использовалась не один раз
    for j:=0 upto n:
      if (j=0):
        x[0]=0;y[0]=0;
      else:
        x[j]:=scale*j/n;
        y[j]:=y[j-1]+A*(e**(-(((scale*j/n-mean)/
        (1.41*sigma))**2)))*
        (x[j]-x[j-1])/(sqrt(2*pi)*sigma);
      fi
    endfor;
    % здесь нужна функция из отрезков, поэтому step 6
    r:=(0,0)for j:=0 step 6 until (n-1):
       --((2,0)+(x[j],y[j])) endfor ..(12,50)--(15,50);
    % отрисовка функции
    gdraw r withpen pencircle scaled 0.8u;
    % сетка
    autogrid(grid.bot,grid.lft) withcolor .5white;
    % дополнительные метки на оси абсцисс
    otick.bot(btex 2\,\text{c} etex,2);
    otick.bot(btex 12\,\text{c} etex,12);
    % текст и стрелка за пределами графика
    gdrawdblarrow (2,-20)--(12,-20)
                        withpen pencircle scaled 0.5u;
    glabel.top(btex \(T=10\,\text{c}\) etex,(7.5,-20));
    gdrawdblarrow (13.5,0)--(13.5,50)
                        withpen pencircle scaled 0.5u;
    % текстовые метки внутри графика
    glabel.lft(btex \(S=50\,\text{м}\) etex
                   rotated 90, 1/2[(13.5,0),(13.5,50)]);
    glabel.rt(btex \(v(t)=\frac{dx(t)}{dt}\) etex
                                  scaled 1.3,(0,25));
    glabel(btex \(dx\) etex,(11.1,20)) ;
    glabel(btex \(dt\) etex,(10,9)) ;
    % подписи к осям
    glabel.rt(btex Время (с) etex,(13,-20)) ;
    glabel.lft(btex Расстояние (м) etex
                    rotated 90,OUT) shifted (0cm,0.5cm);
  endgraph;
endfig ;
```

В graph.mp определено специальное окружение. Между begingraph и endgraph действует своя система координат. Эта система координат привязана ни к геометрическим размерам картинки, а к диапазону осей графика. То есть точка (0,0) соответствует точке пересечения оси абсцисс и оси ординат графика. Размер создаваемого графика указывается сразу после `begingraph`.

Вместо стандартных команд `draw`, `fill`, `label` и `dotlabel` используются `gdraw`, `gfill`, `glabel` и `gdotlabel`, соответственно. Новые команды работают с учётом координат графика. С помощью них можно построить функцию именно по точкам, не заботясь о сдвигах и тому подобное. Для подписи осей существует специальная точка `OUT`, которая в зависимости от суффикса команды `glabel` выносит текст за пределы основной сетки.

Функция отрисовывается точно так же как любая из кривых: создаётся путь и выводится с помощью команды `gdraw`.

Для фиксации диапазона графика используется команда:

```
setrange("нижний левый угол","верхний правый угол");
```

При вызове этой инструкции все последующие функции работают в указанном диапазоне. Таким образом, используя одно и то же пространство можно совместить несколько графиков, имеющих различный диапазон аргументов и функций. Шкалу графика можно сделать логарифмической с помощью инструкции `setcoord`:

```
% x- линейная шкала, y - логарифмическая
setcoord(linear,log);
```

Для создания решётки и меток используется команда `autogrid`

```
% решётка по нижней и левой осям
autogrid(grid.bot,grid.lft) withcolor .5white;
% внешние метки по нижней оси и внутренние метки по правой оси
autogrid(otick.bot,itick.rt);
```

4.2. Работа с файлами данных

С помощью команды `gdraw` можно читать данные из файла. Если в качестве аргумента передаётся текстовая строка, то `gdraw` предполагает, что это имя файла.

```
% Файл graphics.mp
% Число постов на LOR от месяца года
beginfig(4);
  u := 0.4mm;
  draw begingraph(150u,100u);
    pickup pencircle scaled 0.2u;
    gdraw("lor-1998.dat") plot btex \(\circ\) etex;
    gdraw("lor-1999.dat") plot btex \(\circ\) etex;
    gdraw("lor-2000.dat");
    gdraw("lor-2001.dat");
```

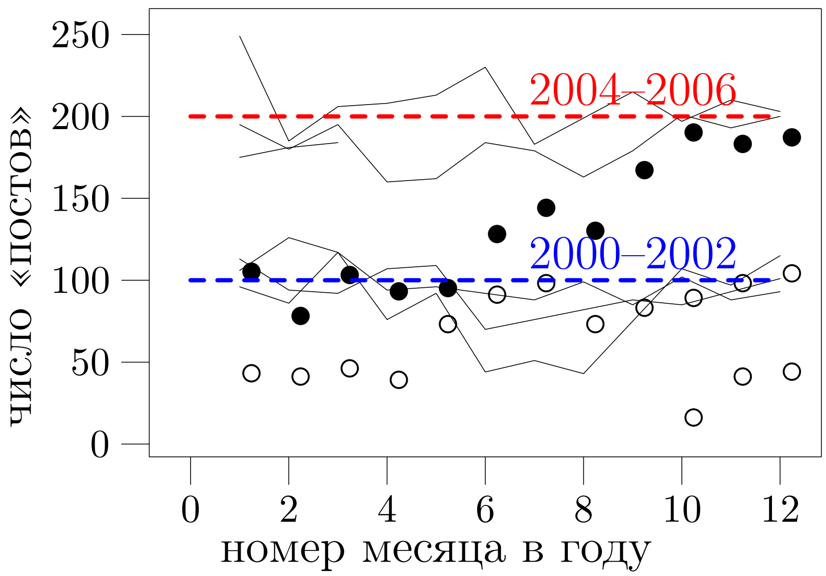


Рис. 4.3. Число постов на LOR

```
    gdraw("lor-2002.dat");
    gdraw("lor-2003.dat") plot btex \(\bullet\) etex;
    gdraw("lor-2004.dat");
    gdraw("lor-2005.dat");
    gdraw("lor-2006.dat");
    gdraw (0,100)--(12,100) withpen pencircle scaled 1u
    dashed evenly scaled 1u withcolor blue;
    glabel.top(btex 2000--2002 etex,(9,100)) withcolor blue;
    gdraw (0,200)--(12,200) withpen pencircle scaled 1u
    dashed evenly scaled 1u withcolor red;
    glabel.top(btex 2004--2006 etex,(9,200)) withcolor red;
    glabel.lft(btex число <<постов>> etex rotated 90,OUT);
    glabel.bot(btex номер месяца в году etex,OUT);
  endgraph;
endfig;
```

`gdraw` так же как и команда `draw` «понимает» инструкции `withpen` (какое перо использовать), `withcolor` (цвет линии) и `dashed` (использовать пунктир). Дополнительно `gdraw` воспринимает команду `plot {picture}`. Указанная в инструкции `plot` картинка выбирается в качестве маркера. На рисунке чёрными маркерами отмечен 2003 год — год, когда произошло удвоение новостных постов в месяц на LOR (`http://www.linux.org.ru`), а белыми маркерами — первый год существования этого ресурса.

По умолчанию предполагается, что входной файл представляет из себя два столбца цифр, разделённые пробелами. Пример файла `lor-2006.dat`:

```
1 175
```

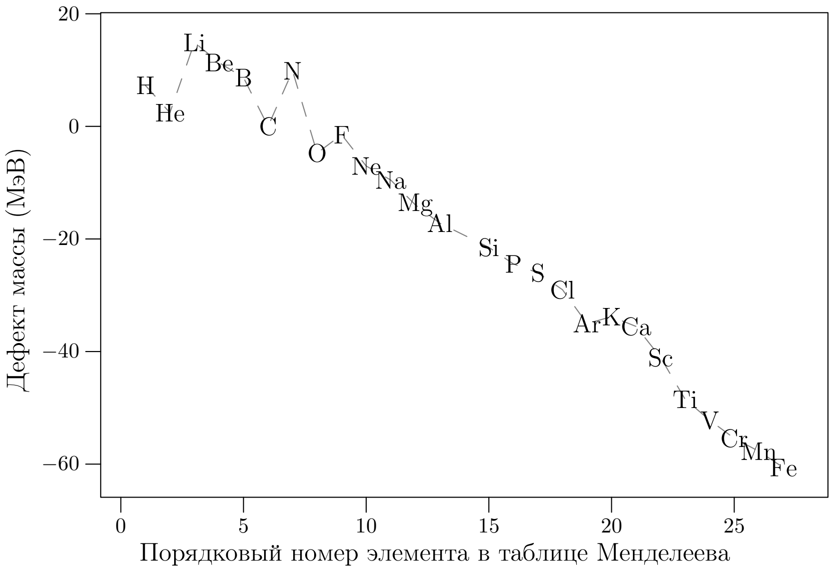


Рис. 4.4. Дефект массы химических элементов от порядкового номера в периодической таблице Меделеева. Приведены элементы от водорода до железа.

```
2 181
3 184
```

Данные взяты с сайта LOR.

Если данные имеют более сложное представление, чем в упомянутом выше файле, то используется команда `gdata`.

```
gdata("имя_файла",переменная куда считываются данные,действие);
```

Если на текстовый файл вида:

```
1 7.289 H
2 2.425 He
—вырезано—
26 −57.710 Mn
27 −60.604 Fe
```

воздействовать с помощью следующего кода:

```
% Файл graphics.mp
% дефект масс
beginfig(5) ;
  u := 0.8mm;
  draw begingraph(150u,100u);
    gdraw "mendeleev.dat" dashed evenly scaled 1u
                                 withcolor 0.5white ;
    gdata("mendeleev.dat", s,
      glabel(s3,(scantokens s1,scantokens s2));
```

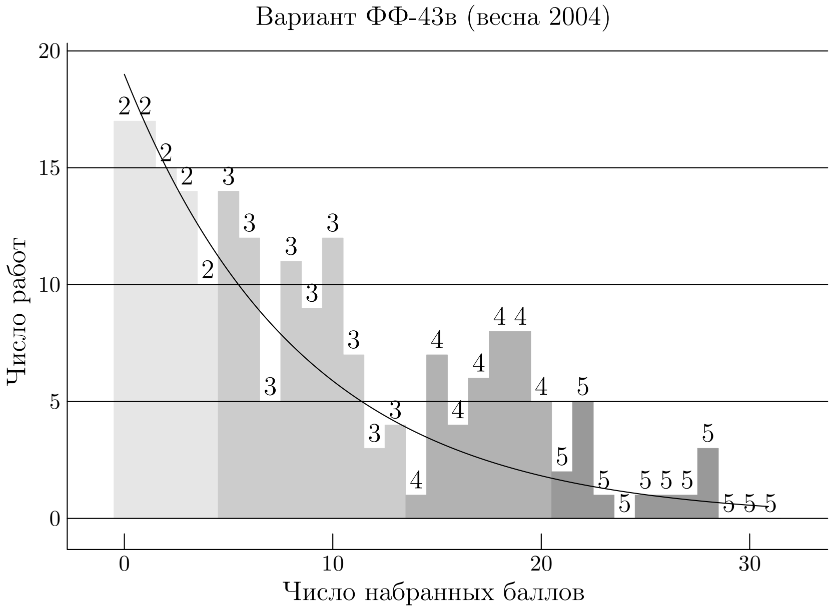


Рис. 4.5. Распределение по баллам, полученных на Открытой олимпиаде по физике в НГУ в 2004 году.

```
    )
    glabel.lft(btex Дефект массы (МэВ) etex rotated 90,OUT);
    glabel.bot(btex Порядковый номер элемента в таблице etex,OUT);
  endgraph;
endfig;
```

то получится простенькая картинка (см. рис. 4.4).

Переменная `s` объявляется массивом в который считывается строка. В качестве разделителя выступает пробел. Запись `s1` соответствует `s[1]` — первому элементу массива. Макрос `scantokens` «встраивает» значение аргумента в код. В данном случае происходит перевод строки в число. В качестве «действия» в команде `gdata` может быть набор из нескольких команд.

4.3. Гистограммы в MetaPost

Если Вы хотите нарисовать гистограмму, то можно воспользоваться следующими макросами:

```
% Файл graphics.mp
% делает профиль гистограммы из пути
def histpath(expr pairs) =
  for i=0 upto length pairs:
    if (i>0):--else:fi((point i of pairs)-
      ((xpart(point 1 of pairs)-
```

```
      xpart(point 0 of pairs))/2,0))--
    ((point i of pairs)+
      ((xpart(point 1 of pairs)-
        xpart(point 0 of pairs))/2,0))
  endfor
enddef;

% делает зацикленный профиль гистограммы для закрашивания
def histpathcycle(expr pairs) =
  ((xpart(point 0 of pairs),0)-
    ((xpart(point 1 of pairs)-
      xpart(point 0 of pairs))/2,0))
  --histpath(pairs)--
  ((xpart(point infinity of pairs),0)+
    ((xpart(point 1 of pairs)-
      xpart(point 0 of pairs))/2,0))--cycle
enddef;
```

Распределение представленное на картинке (рис. 4.5) похожее на убывающую экспоненту. Вот так рисовались «пятёрки»:

```
% Файл graphics.mp
% Оценки
path five;
five:=(21,2)--(22,5)--(23,1)--(24,0)--(25,1)--
      (26,1)--(27,1)--(28,3)--(29,0)--(30,0)--(31,0);
% Заполняем гистограмму
gfill histpathcycle(five) withcolor 0.6white;
```

Для закрашивания контура применяется инструкция `gfill` — аналог команды `fill`.

Данные по оценкам получались в результате обработки текстовых файлов со статистикой с помощью простого скрипта на `perl`. Время, которое потребовалось на автоматизацию составило примерно один человеко-день, что многократно уступает времени написания полного отчёта куда вошли подобные гистограммы и оформлению всех условий и решений Открытой олимпиады и последующих за ней летних вступительных экзаменов.

Создание своих макросов на MetaPost является стандартным действием. `META` не страдает избыточностью, поощряя «затачивать» окружения под свои нужды.

4.4. Круговые диаграммы

Пакет `piechartmp.mp` относительно «молодой» пакет. Замечательная, можно сказать красочная, документация поставляется в виде файла `piechartmp.pdf` и примеров.

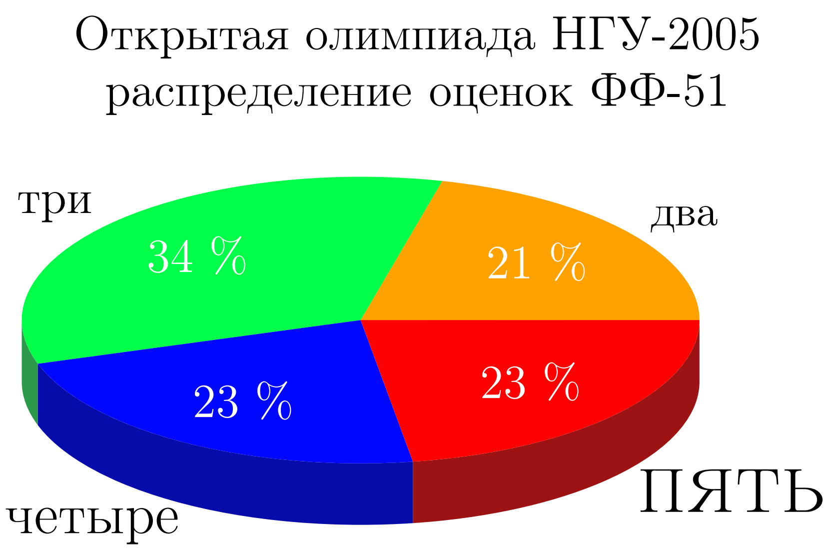


Рис. 4.6. Распределение оценок

Как обычно, за русским языком надо приглядывать. По умолчанию пакет работает только с латиницей.

```
% Файл pie.mp
input piechartmp;
% Круговые гистограммы
% Открытая Олимпиада "ФФ-51"
beginfig(1) ;
  numeric u; u:=1mm;
  % чтобы был русский
  SetupText(1, "\input{preheader-base}", "\begin{document}")
  label(btex Открытая олимпиада НГУ-2005 etex,(5u,25u));
  label(btex распределение оценок ФФ-51 etex,(5u,20u));
  % Чтобы отображался процент
  SetupPercent(this, "␣\%");
  % Опеределение сегментов
  Segment( 15,"\small␣два", auto) ;
  Segment( 24,"три", auto) ;
  Segment( 16,"\large␣четыре", auto) ;
  Segment( 16,"\huge␣пять", red ) ;
  % Создание круговой гистограммы
  PieChart(30u, 0.2, 65, 0, 0);
  % метки
  Label(0)(percent)(inwards,0) withcolor white;
  Label.auto(0)(name)(outwards,0) ;
endfig;
```

Алгоритм создания круговой или секторной диаграммы следующий:

- Если Вы хотите воспользоваться автоматической системой размещения меток, то с помощью команды `SetupText` следует настроить ввод текста. В файле `preheader-base.tex` должна быть минимальная шапка для документа LaTeX. Если Вы хотите использовать русский язык, то там обязательно должны быть строки вида `\usepackage[T2A]{fontenc}` и `\usepackage[koi8-r]{inputenc}`. Вместо `koi8-r` можно поставить свою кодировку.
- С помощью команды `Segment` определить сегменты. В качестве аргументов команде передаётся число (измеряемая величина), текстовая метка и цвет сегмента (можно указать значение по умолчанию `auto`).

  Можно также ввести четвёртый необязательный строковый параметр, который заменяет измеряемую величину при создании меток. Это решает проблему ограничения на диапазон чисел в MetaPost.
- Нарисовать гистограмму. В качестве параметров команде `PieChart` передаётся размер, высота диаграммы, угол под которым мы на неё смотрим (трёхмерия), угол поворота вокруг центральной оси и «смещение» сегментов относительно центра.
- Расставить метки.

Каждому сегменту при создании присваивается порядковый номер начиная с единицы. Все изменения в круговой диаграмме делаются глобально. Пакет написан так, чтобы было удобно работать в одном `mp`-файле ровно с одной диаграммой. Если при описании диаграммы внутри окружения `beginfig` что-то уже определили, то нет необходимости в последующих окружениях это повторять. Этот режим удобен если нужно создать несколько модификаций одной диаграммы, например, для создания «оверлеев» в презентации. Минусом такого подхода является то, что если в этом же `mp`-файле хочется создать ещё одну гистограмму, то уже определённые сегменты необходимо спрятать.

```
% Круговые гистограммы
% Распределение вещества во вселенной
beginfig(2) ;
  numeric u; u:=1mm;
  % Прячем определённые ранее сегменты
  SegmentState(1,hidden,this);
  SegmentState(2,hidden,this);
  SegmentState(3,hidden,this);
  SegmentState(4,hidden,this);
  % Опеределение новых сегментов
  Segment( 65,"\Large \bf тёмная энергия",green,"\LARGE 65\%--70\%");
  Segment( 25,"\Large \bf тёмная материя", (1,0,1),"\LARGE 25\%");
  Segment( 0.5,"звёзды", (1,1,0),"\LARGE 0.5\%");
```

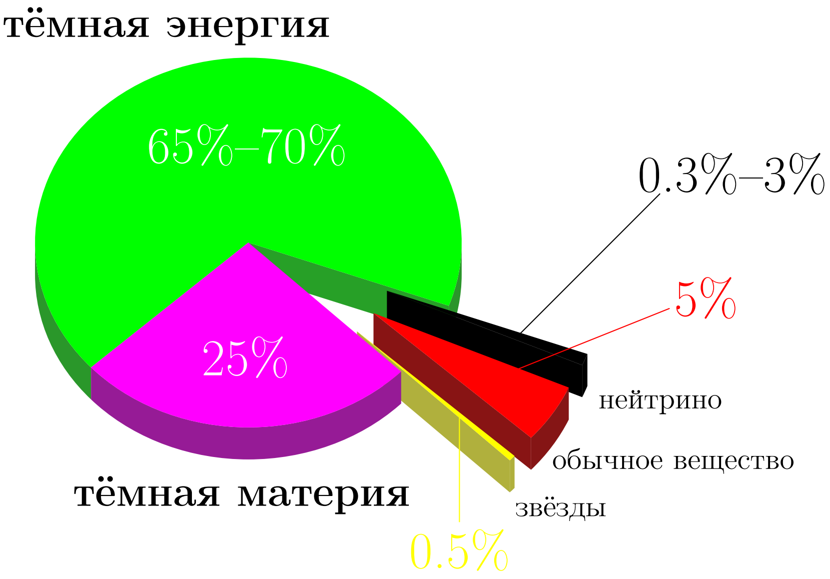


Рис. 4.7. Распределение «вещества» в современной вселенной.

```
  Segment( 5,"обычное␣вещество", red,"\LARGE␣5\%");
  Segment( 1,"нейтрино", black,"\LARGE␣0.3\%--3\%");
  % Выдвигаем сегменты из диаграммы
  SegmentState(7,this,0.7);
  SegmentState(9,this,0.7);
  SegmentState(8,this,0.7);
  % Создаём круговой гистограммы
  PieChart(30u, 0.3, 30, 340, 0);
  % Размещаем метки
  Label.auto(0)(name)(outwards,0) ;
  Label.auto(5)(value)(inwards,0) shifted (7u,-7u) withcolor white;
  Label.auto(6)(value)(inwards,0) shifted (0u,5u) withcolor white;
  Label.auto(7)(value)(inwards,(0u,-15u)) withcolor (1,1,0);
  Label.auto(8)(value)(inwards,(25u,10u)) withcolor red;
  Label.auto(9)(value)(inwards,(20u,20u)) withcolor black;
endfig;
```

Состояние каждого из сегментов можно задавать с помощь команды `SegmentState`.

```
SegmentState(Порядковый номер сегмента,
  устанавливаемое состояние сегмента или this если оно не меняется,
  радиальный сдвиг сегмента или this если он не меняется);
```

Состояние сегмента может быть:

- `normal` — сегмент становится видимым,

- `invisible` — сегмент не рисуется при создании круговой диаграммы (в диаграмме остаётся пустое место),
- `hidden` — при создании диаграммы этот сегмент игнорируется.

Радиальный сдвиг сегмента указывается в процентах, где 1 — это 100%, то есть сегмент полностью «выдвинут».

Для установки меток используется команда `Label`:

```
Label(Порядковый номер сегмента)(метка)
    (базовая точка в системе отсчёта сегмента, сдвиг);
```

`Label` понимает те-же суффиксы, что и обычная команда `label`, плюс она понимает суффикс `auto`. При установки суффикса `auto` `Label` пытается сама угадать где лучше поставить метку.

В качестве порядкового номера сегмента можно передать 0. В этом случае команда `Label` применяется ко всем видимым сегментам. `Label` может принимать в качестве аргумента список порядковых номеров сегментов, разделённых запятыми (например: `5,7,9`).

Метка может быть просто текстовой строкой или одним из стандартных значений:

- `value` — измеряемая величина, которая приписывается сегменту при его создании,
- `percent` — процент занимаемой площади,
- `name` — имя сегмента, которое приписывается ему при его создании.

Базовая точка представляется в виде пары чисел `(x,y)`, где `x` — расстояние от вершины сегмента (0 — это вершина, 1 — это противоположный от вершины край), а y — эквивалент полярному углу (0 — край сегмента по часовой стрелки, 1 — край сегмента против часовой стрелки). В пакете определены константы `inwards=(0.7,0.7)` и `outwards=(1.1,0.5)`. Сдвиг же представлен в терминах всего графика (например, `(1cm,0)` — означает сдвиг на 1 сантиметр вправо).

P.S. При желании тип заливки и порядок цветов по умолчанию можно определять самостоятельно. Подробней об этом рассказано в документации к пакету (`piechartmp.pdf`).

4.5. LaTeX рисует с помощью MetaPost

Для рисования графиков можно воспользоваться и возможностями самого LaTeX. Например, с этой обязанностью прекрасно справляется стандартный пакет *mfpic*. Для «отрисовки» функций с помощью этого пакета достаточно задать саму функцию, а не рисовать по точкам:

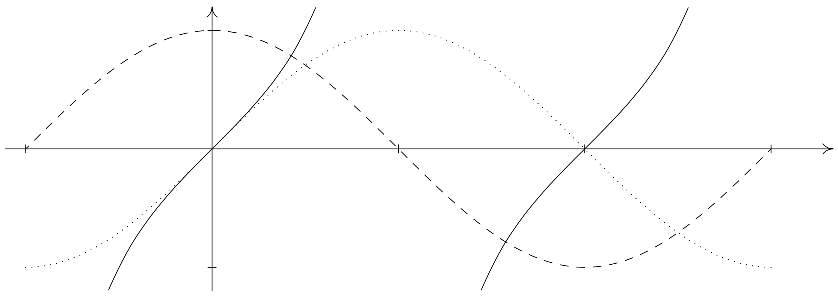


Рис. 4.8. График косинуса, синуса и тангенса.

```
% файл graphics-mfpic.tex
% Пример от Сергея В. Знаменского
\documentclass{article}
\usepackage[koi8-r]{inputenc}
\usepackage[russian]{babel}
\usepackage[metapost]{mfpic}
% Указываем mp-файл куда будет «складываться» код mfpic
\opengraphsfile{graph-mfp}
\begin{document}
% Запрет на печатать номера страницы
\pagestyle{empty}
% Начинаем создание графика
\mfpic[1][57.2]{-100}{300}{-1.2}{1.2}
% Оси координат
\axes
\xmarks{-90,90,180,270}
\ymarks{-1,1}
% Рисуем синус точками
\dotted\function{-90,270,4.5}{sind(x)}
% Рисуем косинус пунктиром
\dashed\function{-90,270,4.5}{cosd(x)}
% Рисуем тангенс сплошной линией
\function{-50,50,10}{tand(x)}
\function{130,230,10}{tand(x)}
% Заканчиваем создание графика
\endmfpic
% Закрываем mp-файл
\closegraphsfile
\end{document}
```

Для получения картинки необходимо выполнить следующие действия:

```
> latex graphics-mfpic.tex
```

```
> mpost graph-mfp.mp
> latex graphics-mfpic.tex
> dvips -E graphics-mfpic.dvi -o graphics-mfpic.eps
```

При компиляции файла `graphics-mfpic.tex` создаётся `graph-mfp.mp`, куда пишутся команды на языке `META`. Если посмотреть на код `graph-mfp.mp`, то можно увидеть что там используется макрос `function`, который позволяет рисовать функции одной командой без всяких циклов. То, как это делается, можно подглядеть в библиотечке `grafbase.mp`, которая входит в состав пакета *mfpic*.

Более подробно про пакет `mfpic` можно узнать из документации которую можно найти в директории `$(TEXMF)/texmf-dist/doc/generic/mfpic/`, где `$(TEXMF)` — директория в которую установлен LaTeX (это заведомо верно для дистрибутива TeX`Live`).

4.6. gnuplot

В случае отображения больших объёмов данных лучше использовать `gnuplot`. Для того чтобы `gnuplot` работал с MetaPost, необходимо определить правильное устройство вывода, то есть `gnuplot` следует передать команду:

```
set terminal mp color latex
```

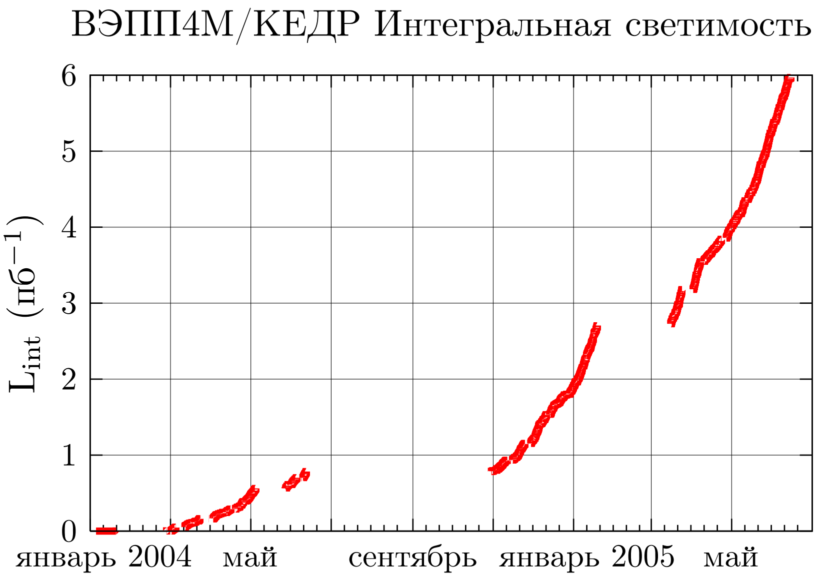


Рис. 4.9. Интегральная светимость от времени. Данные за полтора года (примерно 130 тысяч точек) обработаны с помощью программы `gnuplot`.

В этом случае `gnuplot` будет выводить графики в формате MetaPost. К сожалению, похоже, что вывод в MetaPost давно не поддерживался, поэтому установка кодировки:

```
set encoding koi8r
```

не оказывает должного эффекта. Получившийся mp-файл приходится «дотачивать» в ручную. Благо это не сложно (следует поправить только заголовок), но для автоматизации следует озадачиться исправлением этой неприятной ошибки в `gnuplot`.

В случае если будет желание изменить метки, то следует обратить внимание на переменную `textmag`, которая отвечает за размер текста.

4.7. Подведём итоги

С помощью MetaPost можно создавать двумерные графики и диаграммы любой сложности. Возможности языка `META` и стандартные пакеты позволяют сконцентрироваться на смысловой части и не отвлекаться на отдельные элементы оформления.

Глава 5

Дополнительные главы

Всё описать невозможно,
но ничто не запрещает попробовать.

Возможности `META` в то или иной мере уже изложены. В этой статье будут разобраны полезные приёмы, которые можно применять при кодировании картинки и описаны некоторые из стандартных пакетов, поставляемых с MetaPost.

Исходники стандартных MetaPost пакетов, обычно находится в стандартной же директории `$(TEXMF)/texmf-dist/metapost/`, а документация к ним в директории `$(TEXMF)/texmf-dist/doc/metapost/`, где `$(TEXMF)` — корневая директория для дистрибутива LaTeX. Сказанное верно для TEX`Live`.

5.1. Пакет boxes

`boxes` — один из самых первых пакетов общего назначения, появившихся в MetaPost. Его предназначение — это рисовать простые диаграммы. Он проектировался как более изощрённая замена для пакета Брайана Кернигана `pic` (`man pic`).

С другой стороны функциональности этого пакета достаточно для реализации автоматической генерации достаточно сложных зависимостей. Для примера разберём как была реализована диаграмма объясняющая действие MetaPost-конвейер во введении в цикл статей по MetaPost.

```
% Файл dop.mp
% В отличии от boxes здесь ещё определено окружение rboxit
input rboxes;
% Определяем ещё один вид box'а
% Параметры можно передавать и так
%expr - изолированные выражения
%text - абсолютно всё, что передаётся
vardef drawshadowed(expr dx,dy)(text t) =
  fixsize(t);
```

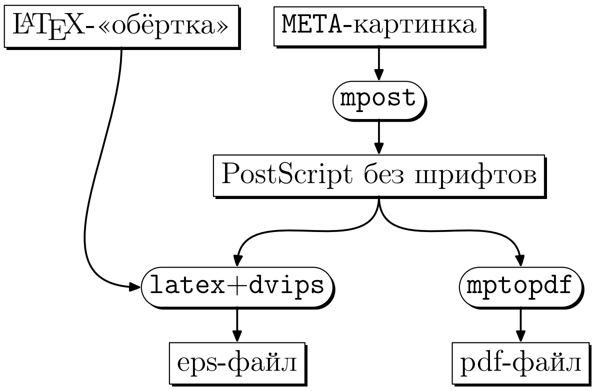


Рис. 5.1. MetaPost-конвейер

```
  forsuffixes s=t:
    fill bpath.s shifted (dx,dy);
    unfill bpath.s;
    drawboxed(s);
% Можно было напечатать только текст
%и не рисовать рамку
%   draw pic(s) withcolor red;
  endfor;
enddef;

% Пример boxes
beginfig(2) ;
  numeric u;u:=1mm;
  % Определяем box'ы
  % Определяем box с прямыми углами
  boxit.a(btex \meta-картинка etex);
  % Определяем box с закруглёнными углами
  rboxit.b(btex \texttt{metapost} etex);
  % Определяем жёсткую связь между a и b
  b.n = a.s - (0,5u);
  boxit.c(btex PostScript без шрифтов etex);
  c.n = b.s - (0,5u);
  rboxit.e(btex \texttt{latex}\(+\)\texttt{dvips} etex);
  e.n=c.s-(20u,10u);
  boxit.d(btex \LaTeX-<<обёртка>> etex);
  d.e=a.w-(5u,0);
  boxit.g(btex eps-файл etex);
  g.n=e.s-(0,5u);
  rboxit.h(btex \texttt{mptopdf} etex);
  h.n=c.s-(-20u,10u);
```

```
  boxit.i(btex  pdf-файл etex);
  i.n=h.s-(0,5u);
  % Разрешаем зависимости и рисуем boxes
  drawshadowed(1/3u,-1/3u,a,b,c,d,e,g,h,i);
  pickup pencircle scaled 0.3u;
  % Рисуем стрелки (сдвижка вида (0u,-1/3u) появилась,
  %так как было определено новое окружение drawshadowed)
  drawarrow a.s -- b.n-(0u,-1/3u);
  drawarrow b.s -- c.n-(0u,-1/3u);
  drawarrow c.s{dir -90} .. {dir -90}e.n-(0u,-1/3u);
  drawarrow c.s{dir -90} .. {dir -90}h.n-(0u,-1/3u);
  drawarrow e.s -- g.n-(0u,-1/3u);
  drawarrow h.s -- i.n-(0u,-1/3u);
  drawarrow d.s{dir -90} .. {dir 0}e.w-(1/3u,0u);
endfig;
```

Обратите внимание на ещё один вид цикла `forsuffixes`. Эта запись позволяет разбирать произвольное число передаваемых аргументов.

Алгоритм создания диаграммы следующий:

- С помощью команд `boxit`, `rboxit` или `circleit` объявляем box. Имя box'a добавляется к команде через точку как суффикс. В данном примере определено 8 box'ов от «a» до «i» включительно.

- Составляем уравнения связей. Через точку к именам box'ов можно добавить один из восьми суффиксов `n` — север (верхняя точка), `s` — юг (нижняя точка), `w` — запад (крайняя точка рамки слева), `e` — восток (крайняя точка рамки справа) и `sw`, `bw`, `se`, `sw` — комбинации уже перечисленных суффиксов, которые соответствуют углам рамки. Уравнение связи `b.n = a.s - (0,5u);` можно описать следующим образом: северная (верхняя) точка box'a «b» находится ниже южной (нижней) точки box'a «a» на `5u`. При составления уравнения связей следует пользоваться только знаком равенства. Это не присваивание, а именно уравнение, которое требуется разрешить.

- С помощью команды `drawboxed` нарисовать «закодированную» диаграмму. При исполнении этой команды разрешается система уравнений. По умолчанию, если не указать специально, box располагается в точке `(0,0)`.

- Нарисовать стрелки между элементами диаграммы. Можно пользоваться именами box'ов со стандартными суффиксами.

Прежде чем использовать этот пакет необходимо прочитать соответствующий раздел в руководстве пользователя по MetaPost Джона Хобби `mpman.pdf`.

5.2. Фейнмановские диаграммы

Простота MetaPost позволяет использовать как базу для построений более высокого порядка. Существует несколько подобных LaTeX-пакетов. О пакете `mfpic` уже упоминалось. В этом разделе описывается ещё один старейший пакет созданный по этой технологии.

В 1995 году Торстен Охл (Torsten Ohl) представил пакет `feynmp` для рисования фейнмановских диаграмм. Фейнмановские диаграммы используются для вычисления сумм большого числа вкладов от элементарных процессов. В своё время эта технология довольно сильно продвинула технику вычислений в физике высоких энергий. Её можно использовать везде, где сложный процесс можно описать с помощью элементарных приближений.

Пакет `feynmp`, это пакет LaTeX, который для рисования диаграмм использует MetaPost.

Интересующий меня с целью извлечения $\Gamma_{J/\psi \to e^+e^-}$ процесс имеет в Борновском приближении следующий вид:

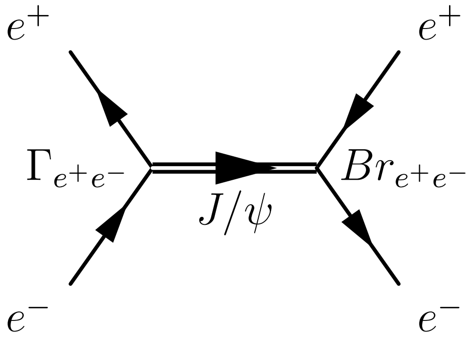


Рис. 5.2. Простейшая фейнмановская диаграмма

Это не сложная диаграмма и если не считать метки, то для её описания требуется всего пять операторов:

```
% Файл eepsiee.tex
% Пакет для рисования фейнмановских диаграмм
\usepackage{feynmp}
...
\begin{fmffile}{ee-psi-ee} % Имя mp-файл
  \begin{fmfgraph*}(110,62) % Размер диаграммы
    \fmfleft{ei,pi}  % Вершины-источники
    \fmfright{eo,po} % Исходящие вершины
    \fmflabel{$e^-$}{ei} % Метка источника e^-
    \fmflabel{$e^+$}{pi} % Метка источника e^+
    \fmflabel{$e^+$}{po} % Метка исходящей вершины
    \fmflabel{$e^-$}{eo} % Метка исходящей вершины
    % Линия соединяющая источники
```

```
    \fmf{fermion}{ei,Ji,pi}
    % Линия соединяющие исходящее вершины
    \fmf{fermion}{po,Jo,eo}
    % Метка для начальной вершины промежуточной частицы
    \fmflabel{$\Gamma_{\ee}$}{Ji}
    % Метка для конечной вершины промежуточной частицы
    \fmflabel{$Br_{\ee}$}{Jo}
    % Соединительная линия
    \fmf{heavy,label=$J/\psi$}{Ji,Jo}
  \end{fmfgraph*}
\end{fmffile}
```

Окружение `\beginfmffile` в качестве параметра требует имя mp-файла в который будут писать команды `META`. В данном примере имя файла определено как `ee-psi-ee.mp`. Для того чтобы получить диаграмму, описанную в файле `eepsiee.tex` были проделаны следующие действия:

```
> latex eepsiee.tex
> mpost ee-psi-ee.mp
> latex eepsiee.tex
```

После выполнения этих команд результат можно посмотреть с помощью программы `xdvi` или преобразовать dvi-файл в PostScript или pdf.

Следующая простейшая диаграмма не имеет особого смысла. Она просто демонстрирует, возможности пакета: «Сотрудничество» с MetaPost даёт возможность

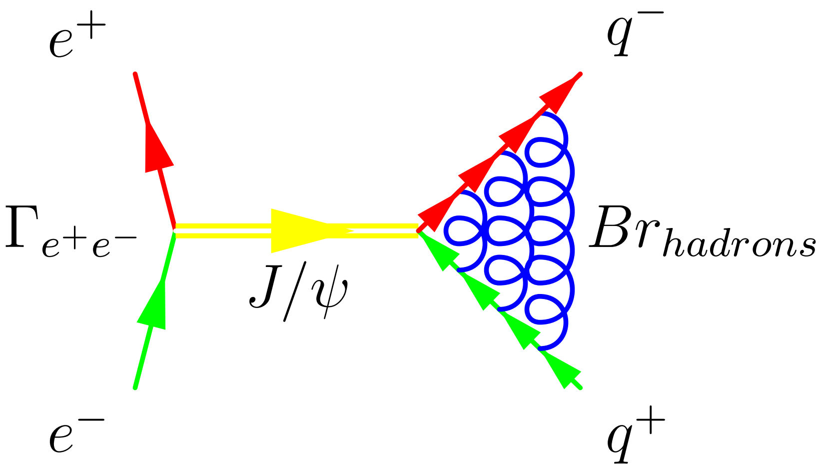


Рис. 5.3. Раскрашенная фейнмановская диаграмма

`feynmp` получить доступ к цвету. Цвет можно определять точно так же, как он определяется в MetaPost.

```
% Файл eepsihadr.tex
\begin{fmffile}{ee-psi-hadr}
```

```
  \begin{fmfgraph*}(110,62)
    \fmfleft{i1,i2}
    \fmfright{o1,o2}
    \fmflabel{$e^-$}{i1}
    \fmflabel{$e^+$}{i2}
    \fmflabel{$q^+$}{o1}
    \fmflabel{$q^-$}{o2}
    \fmf{fermion,foreground=green}{i1,v1}
    \fmf{fermion,foreground=red}{v1,i2}
    \fmf{fermion,foreground=green}{o1,v2,v3,v4,v5}
    \fmf{fermion,foreground=red}{v5,v6,v7,v8,o2}
    % Изменяем натяжение (расталкиваем вершины)
    \fmf{heavy,label=$J/\psi$,tension=1/3,
                              foreground=red+green}{v1,v5}
    \fmffreeze
    \fmf{gluon,foreground=blue}{v2,v8}
    \fmf{gluon,foreground=blue}{v3,v7}
    \fmf{gluon,foreground=blue}{v4,v6}
    \fmfv{label=$\Gamma_{\ee}$}{v1}
    \fmfv{label=$Br_{\hadr}$,label.dist=0.3w}{v5}
  \end{fmfgraph*}
\end{fmffile}
```

При создании диаграммы MetaPost пытается оптимально расположить вершины, минимизируя взвешенную сумму расстояний между ними:

$$L(v_1, \dots, v_n) = \frac{1}{2} \sum_{i,j} t_{i,j} (v_i, v_j)^2,$$

где параметр $t_{i,j}$ — «натяжении» между вершинами v_i и v_j. По умолчанию натяжение равно 1. С помощью опции `tension` натяжение можно изменить. Чем меньше натяжение, тем сильнее расталкиваются вершины.

Обычно, полученная диаграмма со значениями по умолчанию на требует вмешательства в код. В случае если результат не устраивает, как правило, достаточно изменить натяжение для одного, максимум для двух соединений.

Довольно подробную документацию по пакету `feynmp` можно найти в файле `$(TEXMF)/texmf-dist/doc/latex/feynmf/manual.ps.gz`, где `$(TEXMF)` — корень дерева LaTeX (заведомо верно для TeX`Live`).

5.3. Фракталы

MetaPost, естественно, поддерживает рекурсию. Пользуясь этим, а так же генераторами случайных `uniformdeviate` или `normaldeviate` можно организовать фрактальную «лесопосадку».

Рис. 5.4. Фрактальный «лес»

```
% Фрактальная лесопосадка
beginfig(1) ;
  u:=1mm; branchrotation := 50;
  offset := 180-branchrotation;
  thinning := 0.7;
  shortening := 0.8;
  def drawit(expr p, linethickness,col) =
    draw p withpen pencircle scaled linethickness withcolor col;
  enddef;
  % A - основание B - направление роста, n - число бранчей,
  %size - толщина дерева, col - цвет
  vardef tree(expr A,B,n,size,col) =
    save C,D,thickness; pair C,D;
    thickness := size;
    C := shortening[B, A rotatedaround(B,
      offset+uniformdeviate(branchrotation))];
    D := shortening[B, A rotatedaround(B,
     -offset-uniformdeviate(branchrotation))];
    if n>0:
      drawit(A--B, thickness, col);
      thickness := thinning*thickness;
      tree(B, C, n-1, thickness,col);
      tree(B, D, n-1, thickness,col);
    else:
      drawit(A--B,thickness,col);
      thickness := thinning*thickness;
      drawit(B--C, thickness,col);
      drawit(B--D, thickness,col);
    fi;
```

```
  enddef;

  numeric nbr,nx,ny,ell,size;
  color col;
  nx:=10;ny:=5;
  pair A;
  for ix:=1 upto nx:
    for iy:=1 upto ny:
      nbr:=4+uniformdeviate 5;
      ell:=nbr*u;
      x:=ix*(1+1/20*normaldeviate);
      y:=iy*(1+1/20*normaldeviate);
      A:=(20u*(x+y*sqrt(2)/2),20u*y*sqrt(2)/2);
      size:=ell/5;
      col:=(uniformdeviate 1,uniformdeviate 1,uniformdeviate 1);
      show ix,iy,A,ell,nbr,size,col;
      tree(A, A+(0,ell), nbr, size,col);
    endfor;
  endfor;
endfig;
```

Увлечение автоматически создаваемыми картинами влечёт за собой опасность переполнения памяти. MetaPost создавался в то время, когда к используемой памяти относились исключительно бережно.

Если при компиляции картинки будет выдана ошибка `MetaPost capicity exceded`, то необходимо поправить файл конфигурации `texmf.cnf`. Обычно, этот файл можно найти а директории `$(TEXMF)/web2c/texnf.cnf`[1].

За выделяемый объём памяти для MetaPost отвечает переменная `main_memory.mpost`. Значение этой переменной должно быть меньше значения `main_memory`, которая ставит верхнюю границу по использованию памяти для всех TeX-подобных программ. Обратите внимание на то, что в конфигурационном файле может быть несколько переменных с одним и тем же именем — используется последнее значение.

В моём случае ограничение по памяти стояло в 15 Мб. Очевидно, что используемую память можно безболезненно увеличить. С другой стороны, если происходит переполнение, то это возможно значит, что в алгоритм закралась ошибка.

После исправления значения переменных следует перегенерить форматные файлы, например, с помощью команды `texconfig init`.

5.4. Увеличительное стекло

Бывает в процессе создания картинки необходимо увеличить участок для лучшей детализации. MetaPost предоставляет средство для этого в виде команды `clip`,

[1] В TeX`Live` можно создавать локальные настройки

которая позволяет обрезать картинку по любому замкнутому пути. Ниже идёт код, который позволил увеличить кусок циферблата корабельных часов.

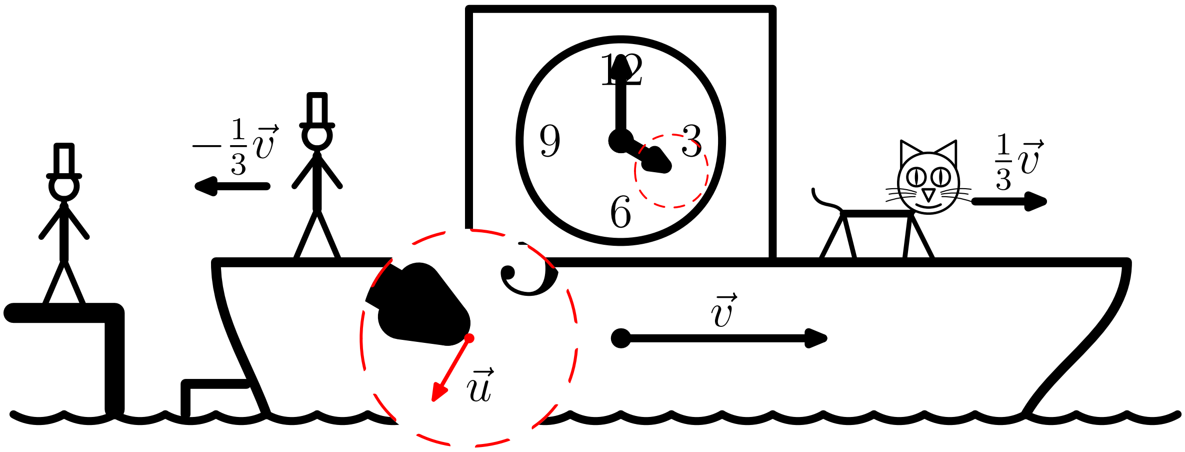


Рис. 5.5. Преобразование Галилея

```
% Файл dop.mp
R:=3.6u;
path q;
% Определяем форму области которую хотим вырезать
q:=(-R,0)..(R,0)..cycle;
% Определяем центр вырезаемой области (orig)
%и местоположение увеличенного участка (copy)
pair orig,copy;
orig=(65u,24u);copy=(45u,7.5u);
% Отмечаем вырезаемую область
draw q  shifted orig dashed evenly scaled 1/2u
         withpen pencircle scaled 0.2u witcholor red;
picture p;
% Сохраняем текущую картинку в переменной p
p:=currentpicture;
% Коэффициент увеличения
numeric scale;scale:=3;
% Обрезаем картинку p по замкнутому пути q
clip p to (q shifted orig);
% Чистим область, где будет нарисована увеличенная копия
fill q scaled scale shifted copy withcolor white;
% Рисуем копию
draw p shifted -orig scaled scale shifted copy;
% Рисуем дополнительную стрелку и метку
%на уже увеличенной копии
draw copy withpen pencircle scaled 1u withcolor red;
```

```
drawarrow copy--(copy+7u*dir -120)
        withpen pencircle scaled 0.4u withcolor red;
label.lrt(btex \(\vec{u}\) etex,
                     1/3[copy,(copy+7u*dir -120)]);
```

Пользуясь этим приёмом можно создать любой фон для какой-угодно фигуры, сделать любую штриховку.

5.5. Штриховка

`clip`-технология, которая упоминалась выше, позволяет создать любой вид штриховки. В стандартной поставке MetaPost идут пакета, которые этим пользуются.

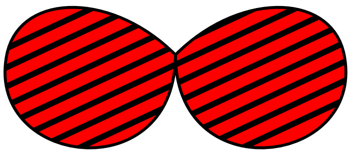


Рис. 5.6. Пакет `hatching`.

Пакет `hatching.mp` представляет из себя обычный «хак»:

```
% Файл dop.mp
% Пример использования пакета hatching
input hatching;
beginfig(3) ;
  numeric u;u=1mm;
  path q;
  q=(10u,0)..{dir -135}(0u,0u){dir 135}..
    (-10u,0)..{dir 90}(0u,0u){dir -90}..cycle;
  hatchfill q withcolor red withcolor (25,1u,-1);
  draw q;
endfig;
```

Функция `hatchfill` берёт информацию о штриховке из данных, которые следуют с декларацией `withcolor`. Сигналом, что эти данные предназначены именно для `hatchfill` является то, что голубая компонента тройки чисел меньше 0. Красная компонента соответствует углу наклона штриховки, зелёная расстоянию между штрихами. Подробности можно узнать в краткой документации к пакету.

Более развитым (возможно излишне) является пакет `mpattern`:

```
% Файл dop.mp
% Пример использования пакета mpattern
input mpattern;
```

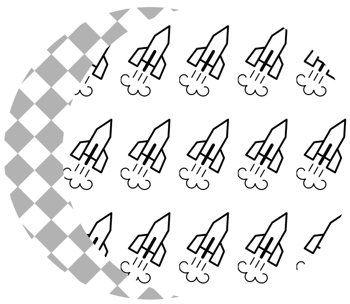


Рис. 5.7. Пакет mpattern

```
% Создаём "обои" ракета
beginpattern(rocket);
 begingroup;save u;
   u:=1mm;
   draw Rocket scaled 2/3u rotated -30;
   patternbbox(-5u,-8u,5u,8u);
 endgroup;
endpattern;

% Создаём "обои" повёрнутая клетка
beginpattern(rotated_checker);
 fill unitsquare scaled 4mm rotated 45 withcolor .7white;
endpattern;

beginfig(4);
 numeric u; u:=1mm;
 path p;
 z1=(10u,0u);
 p=fullcircle scaled 50u;
 % Рисуем фигуру в клетку
 fill p  withpattern rotated_checker;
 unfill p shifted z1;
 % Рисуем фигуру "в ракету"
 fill p shifted z1 withpattern rocket;
endfig;
```

Здесь уже определены свои команды для создания штриховки. Пакет не свободен от недостатков, но он достаточно прост и можно легко «довести» код до необходимой кондиции. Простота — это общее свойство пакетов MetaPost. `META вынуждает` писать кратко. Подробности можно найти в документации к пакету.

5.6. Вставка eps

Пакет `exteps.mp` позволяет включить eps-картинку как единый объект. Краткая документация доступна в файле `exteps.pdf`, поставляемом с этим пакетом.

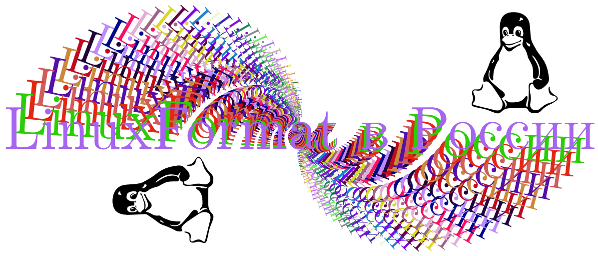


Рис. 5.8. Добавляем чёрно-белых Туксов

```
% Файл dop.mp
input exteps;
% Включение eps-картинки
beginfig(6) ;
  numeric u;u:=1mm;

% Базовая надпись
  for alpha:=-90 step 3 until 0:
    label(btex LinuxFormat в России etex
      scaled (5*(1+alpha/100)) rotated alpha,(0,0))
    withcolor
    (uniformdeviate 1,uniformdeviate 1,uniformdeviate 1);
  endfor;

% Посадим Тукса (penguin.eps) справа
  begineps "penguin.eps" ;
% Ширина картинки
  width:=30u;
% Сдвиг картинки от начала координат
  base:= (60u,5u);
% Можно нарисовать решётку на картинки
%  grid := true;
% Обрезание по bounding box
%  clip := true;
  endeps;

% Посадим Тукса (penguin.eps) слева
  begineps "penguin.eps" ;
% Поворот
```

```
  angle:=90;
  width:=30u;
  base:= (-30u,-40u);
  endeps;
endfig ;
```

5.7. Большие числа

Когда обсуждалась вставка меток в качестве примера был представлен треугольник Паскаля. Из-за ограничений META на максимальный размер числа треугольник Паскаля отрисовывался до 14ой строчки.

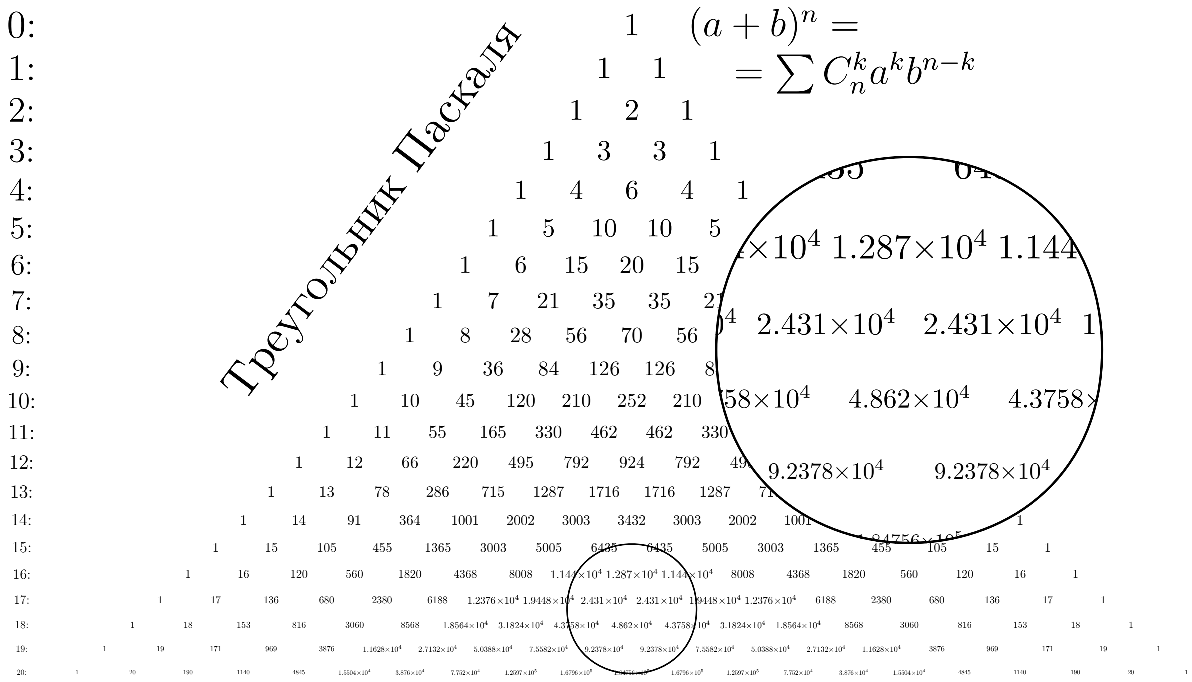


Рис. 5.9. Треугольник Паскаля.

Это ограничение можно обойти воспользовавшись пакетом sarith.mp. Этот пакет был создан специально под пакет graphics.mp. Что вполне естественно, так как если для рисовании небольших рисунков больших чисел быть не может, то при анализе данных это вполне рядовая ситуация.

```
% Файл dop.mp
% Macros for arithmetic on strings that represent big numbers
input sarith;
% Треугольник Паскаля  с большими числами
beginfig(5) ;
  numeric u;
```

```
  u = 1.mm;
  numeric dy,dx,x,y,i,j,sy,ds,nlast;dy:=5u;
  dx:=6u;x=0;y=0;nlast:=20;
  % Для хранения больших чисел нужна строка
  string n[][];
  ds=0.04;sy=0.032;
  picture z;
  for i:=0 upto nlast:
    dy:=dy*(1-sy);
    y:=y-dy;
    for j:=0 upto i:
      if (j=0) or (j=i):
        n[i][j]:=Sabs 1; % Сложение
      else:
        n[i][j]:=n[i-1][j-1] Sadd n[i-1][j]; % Сложение
      fi
      % Формат вывода
      z:=thelabel(format("%6g", n[i][j]),(0,0));
      x:=dx*(j-i/2);
      label(z scaled (1-ds*i),(x,y));
    endfor
    z:=thelabel.lft(decimal(i)&":",(0,0));
    label(z scaled (1-ds*i),(dx*(-nlast/2-1),y));
  endfor
endfig;
```

Для представления больших чисел используется встроенный тип `string` — строка.

Операция	действие
Scvnum <number>	конвертация в numric
Sabs <number>	абсолютное значение
<number> Sadd <number>	сложение
<number> Ssub <number>	вычитание
<number> Smul <number>	умножение
<number> Sdiv <number>	деление
<number> Sleq <number>	сравнение `<=`
<number> Sneq <number>	сравнение `<>`

Таблица 5.1. Операции над большими числами

Документация к пакету `sarith.mp` является частью документации к `graphics.mp` (`mpgraph.pdf` от Джона Хобби).

5.8. Макрос TEX

Вы наверное уже обратили внимание, что метки в MetaPost являются статическими. Всё, что между `btex` и `etex` отдаётся LaTeX для обработки. Это значит, что метку нельзя скомпоновать из различных кусков. Решением является немного модифицированный макрос из стандартного пакета `TEX.mp`:

```
% Файл macros.mp
vardef TEX primary s =
  write "verbatimtex"                to "mptextmp.mp";
  write "\input{preheader-base}"     to "mptextmp.mp";
  write "\begin{document}"           to "mptextmp.mp";
  write "etex"                       to "mptextmp.mp";
  write "btex "&s&" etex"            to "mptextmp.mp";
  write EOF                          to "mptextmp.mp";
  scantokens "input mptextmp"
enddef;
```

В процессе вызова макроса `TEX`, всё что находится внутри него записывается в `mptextmp.mp`, а затем этот файл включается в основной файл прямо во время компиляции. То есть, происходит модификация программы прямо во время компиляции. Пример использования макроса идёт ниже:

```
% Файл coord.mp
numeric u;u:=2mm;
for i:=-15 step 5 until 15:
  label.lft(TEX("\("&decimal(i)&"\)"),(-10u,i*u));
endfor;
```

При формировании ASCII строчки использовался оператор & для объединения. Хотя, в данном случае можно было обойтись только статическими записями, но иногда этот приём может пригодиться для целей автоматизации, где скорость компиляции не главное.

Основной минус этого способа в том, что он очень «мееедленный». Для ускорения следует оставить в `preheader-base.tex` только самые необходимые инструкции. Что-то вроде:

```
\documentclass[12pt]{article}
\usepackage[warn]{mathtext}
\usepackage[T2A]{fontenc}
\usepackage[koi8-r]{inputenc}
\usepackage[english,russian]{babel}
```

Глава 6

Заключение

Естественно, в таком кратком рассказе невозможно изложить *всё*. Основные понятия и приёмы уже рассмотрены, но множество вещей выпало из обзора. В частности, совсем не рассмотрены 3D объекты, создание геометрических чертежей, параметризация пути и векторные поля в MetaPost. Это не смертельно, так как есть весьма качественная свободная литература, посвящённая этим предметам.

MetaPost достаточно легко использовать после обучения, но обучение перед использованием просто необходимо. Это свойство всех стоящих технологий, которые позволяют делать то, для чего компьютеры и существуют: *автоматизировать* рутинные действия, для выполнения которых не требуется задумываться. Думать же, в любом случае, прерогатива человека.