Asymptote для начинающих

Создание рисунков
на языке векторной графики `Asymptote`

Ю.Г. Крячков

17 июля 2015 г.

Оглавление

Введение

Изначально средства для создания рисунков в LaTeX-документах были довольно ограничены, но усилиями многих людей арсенал таких средств существенно расширился. Благодаря идее интегрировать LaTeX и `PostScript` были созданы такие графические пакеты, как, например, `xypic`, `PSTricks`, `PGF/TikZ` и, наконец, программа `MetaPost`. На русском языке немного книг о `PostScript`-графике и её использовании в LaTeX-документах. Отметим перевод книги [1] и книги [2], ещё [5] и [8] из списка в конце пособия. Главы, посвящённые графике, есть в [3] и [4].

Язык векторной графики `Asymptote` создан в 2004 году А. Хаммерлиндлом, Д. Боуменом и Т. Принсом под идейным влиянием программы `MetaPost`. На русском языке `MetaPost` довольно подробно описан в [7] и [8]. Для желающих прикоснуться к идейным истокам мы рекомендуем познакомиться с переводом книги Д. Кнута [6]. А вот по `Asymptote` литературы на русском языке практически нет.

Это пособие рассчитано на начинающих и предназначено в какой-то мере восполнить этот пробел. Для более глубокого изучения языка есть руководство [9] (на английском), которое можно найти на головном сайте `Asymptote`: `http://asymptote.sourceforge.net`. Там же можно найти галерею примеров и ссылки на другие источники.

Язык `Asymptote` по сути является своеобразным "интерфейсом" к языку `PostScript` и, по задумке авторов, должен обеспечивать некий "стандарт" для вёрстки математических фигур так же, как TeX(LaTeX) фактически является стандартом при вёрстке сложных формул и уравнений. `Asymptote` ориентирован на пользователей, владеющих математикой, в нем используются, например, аффинные преобразования, комплексные переменные и т.д. Его будет легче осваивать людям, имеющим хотя бы минимальный опыт программирования и использования LaTeX, поскольку текстовые метки, формулы и уравнения могут набираться с его помощью — это обеспечивает соответствующее качество рисунков. По умолчанию `Asymptote` производит выходные файлы формата `eps`, но существуют и другие возможности, например, `pdf`.

`Asymptote` построен по модульному принципу. Примеры, имеющиеся в пособии, раскрывают прежде всего возможности модуля `plain`. Использовались также модули `geometry`, `unicode` и `graph`.

Содержание пособия отражает личный опыт автора по освоению технологии создания рисунков на языке `Asymptote` и его понимание того, что нужно для этого освоения. Изложение материала строится на самых простых примерах, доступных начинающим пользователям. Более сложные примеры адресованы преподавателям математики. За рамками пособия остались такие темы, как создание `pdf`-анимации, 3D-графика и многие другие. Они несомненно заслуживают отдельного рассмотрения.

Текст состоит из двух глав. В первой главе описаны установка и запуск `Asymptote`, основы рисования и работа с `Asymptote` внутри latex-файла. Освоение материала этой главы позволит пользователю, не вникая в тонкости языка, создавать простые изображения.

Вторая глава адресована тем, кто уже имеет некоторый опыт рисования с помощью `Asymptote` и хочет глубже освоить возможности этого замечательного языка. Завершается пособие кратким обзором модулей.

Глава 1

Начальный курс

В начальном курсе коротко описывается установка `Asymptote` и LaTeXa, их запуск, на примерах излагается работа с основными чертёжными командами и включение кода `Asymptote` в LaTeX-файл. Отметим, что создавать картинки можно и без использования LaTeXa, но часто он требуется для создания текстовых меток, формул и символов.

1.1 Установка и запуск

В этом пособии автором отражен опыт освоения `Asymptote`. Это освоение началось с установки дистрибутива Linux **Ubuntu 10.04**, набора пакетов издательской системы TeX Live и **Kile**, среды разработки LaTeX.

Это не означает, что пользователи **Windows** не могут установить и использовать `Asymptote`. В операционной системе **Windows** также можно установить интерпретатор `Asymptote` и TeX Live, а вместо среды **Kile** установить **Texmaker**, поскольку это многоплатформенное программное обеспечение. Но мы не даём здесь подробного описания установки и работы с `Asymptote` в **Windows**.

1.1.1 Установка TeX Live в Ubuntu

В дистрибутиве **Ubuntu 10.04** установить TeX Live очень просто. В окне Ubuntu на верхней панели расположено главное меню GNOME. Сделайте, например, следующее:

1. Откройте пункт *Приложения/Центр приложений Ubuntu/Предоставляемые Ubuntu*
2. Найдите в списке строку с приложением TeX Live и выделите строку щелчком левой кнопки мыши. Появятся две кнопки: Подробнее и Установить.
3. Установите пакет, нажав на кнопку Установить.

В дистрибутивах **Ubuntu 12.04** и **Ubuntu 14.04** с интерфейсом Unity на экране монитора после загрузки отображается рабочий стол с с двумя панелями: верхней и левой боковой. На верхней панели слева написано *Рабочий стол Ubuntu*, а на левой боковой панели расположены кнопки запуска, назначение которых вы можете узнать из всплывающих подсказок при наведении на кнопку указателя мыши[1].
Для установки TeX Live можно использовать *Центр приложений Ubuntu*:

1. Найдите кнопку *Центр приложений Ubuntu*
2. После нажатия кнопки в открывшемся окне в правом верхнем углу есть строка поиска. Наберите в строке tex live. Появится список приложений (пакетов TeXLive). Он довольно обширный.
3. Найдите приложение TeX Live: *популярный набор пакетов издательской системы* TeX Live и выделите строку щелчком левой кнопки мыши. Из появившихся двух кнопок: Подробнее и Установить щелкните на Установить.
4. Установите пакет, нажав кнопку Установить.

[1] Более подробно о Unity вы узнаете, если последовательно нажмете кнопки *Главное меню* (верхняя кнопка на панели запуска), а затем откроете справку (голубой кружок со знаком вопроса в строке *Недавние приложения*)

1.1.2 Запуск Asymptote в редакторах Kile и Texmaker

Запуск `Asymptote` в редакторе Kile.
Установите **Kile**, среду разработки LaTeX. Для этого в **Ubuntu 10.04**:

1. Откройте в главном меню *Приложения/Центр приложений Ubuntu/Офис*
2. Найдите в списке **Kile** и установите его, выделив и нажав кнопку Установить.

Открывается редактор так: *Приложения/Офис/Kile* (При желании можно вынести кнопку запуска на панель)
Чтобы установить **Kile** в **Ubuntu 12.04** и **Ubuntu 14.04**:

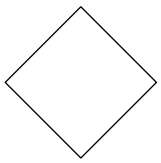
1. Нажмите кнопку *Центр приложений Ubuntu*.
2. Наберите в правом верхнем углу в строке поиска слово kile.
3. Найдите и выделите строку с приложением Kile и установите его, нажав кнопку Установить.

Теперь Вы установили все необходимое, чтобы использовать `Asymptote`.

Итак, приступаем. Откройте меню *File* редактора **Kile**, выберите *New/Empty Document*, откроется чистое поле документа и на верхней панели редактора появится значок не сохраненного файла с именем `Untitled`.
Теперь можно создать некоторый рисунок, для чего введите в поле документа, например, следующий код:

```
unitsize(1cm);
draw((1,0)--(0,1)--(-1,0)--(0,-1)--cycle);
```

Затем откройте меню *File* редактора **Kile**, выберите *Save As*. Откроется диалоговое окно. Внизу окна в строке Filter выберите All Filles и сохраните файл под именем, например, `test.asy`. По умолчанию файл сохранится где-то в папке Home, но можно при сохранении выбрать любую нужную вам папку. После сохранения вы увидете, что имя `Untitled` заменилось на имя `test.asy`.
Теперь прокомпилируем этот файл. Найдите на панели редактора пункт Build, щелкните по нему и найдите в выпадающем меню пункт Compile и щелкните по кнопке Asymptote. Вы получите в той же папке, где расположен файл `test.asy` новый файл `test.eps`. Откройте эту папку с помощью обозревателя файлов и дважды щелкните по имени файла `test.eps`. Откроется просмотрщик Okular и диалоговое окно. Выберите модуль ghostscript и нажмите Ok. Асимптота "вписывает" созданный рисунок в прямоугольный бокс и располагает его вверху страницы по центру. Вы увидете такую картинку:



Итак, при создании рисунка указанным способом используют следующую последовательность шагов:

- в редакторе набираем текст с кодом рисунка
- сохраняем файл с расширением .asy
- обрабатываем файл с кодом рисунка интерпретатором `Asymptote`
- просматриваем полученный eps-файл.

Запуск `Asymptote` в редакторе Texmaker.
Этот многоплатформеный редактор для LaTeXa в дистрибутиве Ubuntu Linux устанавливается аналогично среде **Kile**, через Центр приложений Ubuntu.
В этом редакторе все делается очень просто:

- Создаете новый или открываете готовый файл с расширением .asy (например, `test.asy`), состоящий только из кода `Asymptote`.

- На панели редактора нажимаете кнопку *Инструменты* и в выпадающем меню нажимаете кнопку *Asymptote*
- С помощью обозревателя файлов откройте папку с файлом `test.asy`.
- В ней должен появиться файл `test.eps`, найдите его.
- Дважды щелкните левой кнопки мыши по имени файла. Откроется просмотрщик по умолчанию. Если это `Okular`, в диалоговом окне выберите модуль `Ghostscript` и нажмите Ok. Вы увидете картинку.

Можно настроить редактор, чтобы одной кнопкой запускать интерпретатор `Asymptote`, а другой просматривать полученный `eps`-файл либо `pdf`-файл.

Как включить изображения непосредственно в ваш LaTeX-документ — будет описано далее.

Переходим к изучению языка `Asymptote` на примерах создания очень простых фигур.

1.2 Основные чертежные команды

Несколько предварительных замечаний общего характера.

Представьте прямоугольную декартову систему координат. Начальная точка O, ось x горизонтальна, а ось y направлена вверх. Каждая точка рисунка определяется, как это принято в плоской аналитической геометрии, парой действительных чисел (x, y).

Нужный вам рисунок содержит точки, отрезки, окружности или дугие геометрические образы. Для их воспроизведения служат команды языка `Asymptote`. Например, команда `dot((0,0));` начертит точку O. *Наличие точки с запятой после команды обязательно*, иначе интерпретатор выдаст сообщение об ошибке. Допустим, что вы хотите начертить два отрезка, один из которых соединяет точку `(0,0)` с точкой `(20,20)`, а другой точку `(10,0)` с точкой `(0,10)`. Тогда нужен код:

```
draw((0,0)--(20,20)); draw((10,0)--(0,10));
```

В одной строке не должно быть двух команд, не разделенных точкой с запятой. В то же время расположение команд в различных строках не влияет на результат, так что бывает полезно для лучшего восприятия вашего кода поместить команды в разные строки. Пробелы до и после команд в `Asymptote` не читаются, так что любой код может быть как угодно разбит на строки. Так же для удобства чтения кода используют различные переменные.

Например, точки в команде `draw((0,0)--(20,20));` можно заменить переменными O и A: `draw(O--A);` Но в этом случае переменные O и A должны быть *предварительно* заданы, то есть указан тип данных и данным должны быть приданы значения:

```
pair O=(0,0); pair A=(20,20); draw(O--A);
```

Более подробно о типах данных читайте во второй части.

Комментарии в коде выделяются или двойным слэшем `//` *или символами* `/*` и `*/`.

Вот пример, в котором не компилируется всё, что находится после двойного слэша до конца строки.

```
draw((0,0)--(50,50)); // Чертим отрезок от точки (0,0) до точки (50,50)
```

Ещё пример:

```
draw((0,0)--(50,50)); /* Чертим прямолинейный отрезок с начальной  точкой (0,0) и
конечной точкой  (50,50)*/
```

Здесь не компилируется всё то, что расположено между символами `/*` и `*/`.

Размеры чертежа

Единицей длины в x и y направлениях по умолчанию служит `PostScript`-овский большой пункт, `bigpoint`, сокращенно `1 bp = 1/72 inch` (дюйма). Таким образом, если вы не указали единицу длины, то между точками `(0,0)` и `(72,0)` ровно один дюйм. Однако эта единица длины далеко не единственная. В `Asymptote` используются встроенные константы `pt`(1/72.27 дюйма), `inch, cm` и `mm`.

Для указания единицы длины, отличной от `bp`, используется функция `unitsize`. В круглых скобках нужно указать аргумент команды, единицу длины. Например

```
unitsize(1cm);    unitsize(5mm);
```

Рисунок может масштабироваться по осям путем указания единицы длины в направлении оси x и единицы длины в направлении оси y. Например

```
unitsize(x=1cm,y=0.5cm);
```

Если приводится только один аргумент, то единицы измерения равны в обоих направлениях. Таким образом, команда `unitsize(72);` означает, что единица длины — 1 дюйм, что равносильно `unitsize(1inch);`.

Другой полезной функцией является `size`, которая показывает точную ширину и высоту бокса, в котоый будет вписана ваша картинка (по умолчанию это снова будет `currentpicture`). Если задается только одно число, то ширина и высота будут установлены в этом размере.

Например, команда `size(6cm,6cm);` или просто `size(6cm);` будет соответствовать боксу 6×6 см в окне независимо от указанного `unitsize`.

Создайте файл с программой из двух строк:

```
unitsize(2inch);
draw(unitsquare);
```

и посмотрите, что происходит при изменении размера 2 дюйма на другие значения.

1.2.1 Команды draw и dot

- Команда ***draw***.

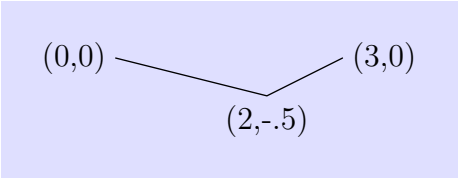
  Рассмотрим одну из основных команд, команду `draw`. У нее может быть много аргументов, но обязательным аргументом команды `draw` является путь. Путь нужно либо явно прописать в круглых скобках (как в прмере 1), либо определить заранее (как в прмере 2).

  Проиллюстрируем сказанное на ряде примеров.

  Пример 1

  Путь g в данном примере есть ломаная с началом $(0,0)$, концом $(3,0)$, проходящая через точку $(2,-.5)$. Для этого можно указать координаты точек и двумя черточками указать способ соединения точек. Приведем соответствующий код:

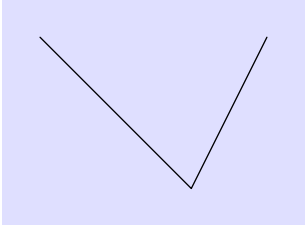
```
unitsize(1cm); // Задаем пользовательские единицы(одновременно по x и по y).
draw((0,0)--(2,-.5)--(3,0)); // Чертим простую двузвенную ломаную.
shipout(bbox(5mm,Fill(paleblue+white))); // Делаем вокруг поля в 5mm и создаём фон.
```



  Далее, как правило, мы не будем указывать команду создания фона рисунка и координаты точек.

  Пример 2

```
unitsize(x=1cm,y=4cm);
/* Задаем пользовательские
единицы(отдельно по x и по y).*/
path g=(0,0)--(2,-.5)--(3,0);
// Определяем путь g как ломаную.
draw(g); // Чертим путь g.
```

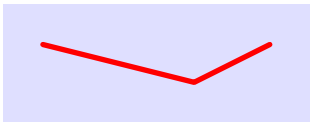


  Сравните результаты этих двух примеров. В примере 2 рисунок примера 1 растянут в четыре раза по вертикали.

Пример 3

Мы можем указать необходимую толщину линии и ее цвет.

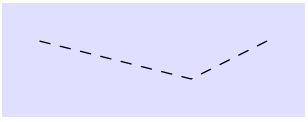
```
unitsize(1 cm);
path g=(0,0)--(2,-.5)--(3,0);
// Определяем путь.
draw(g,2bp+red);
/* Чертим путь, указывая через запятую
толщину линии и ее цвет.*/
```



Пример 4

Заменим сплошную линию на пунктирную:

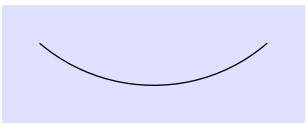
```
unitsize(1 cm);
path g=(0,0)--(2,-.5)--(3,0);
draw(g,dashed);
// Задаем характер линии(пунктир).
```



Пример 5

Заменим ломаную на кривую линию, проходящую через те же точки. Для этого вместо черточек укажем точки:

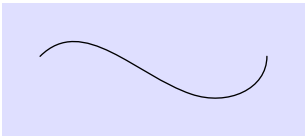
```
unitsize(1 cm);
path g=(0,0)..(2,-.5)..(3,0);
draw(g); // Задаем и чертим кривую.
```



Пример 6

Покажем, как легко можно менять форму кривой, указывая направления выходящих и входящих касательных векторов.

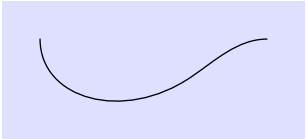
```
unitsize(1 cm);
path g=(0,0){dir(45)}..(2,-.5)..{up}(3,0);
draw(g);
```



Пример 7

Направления касательных векторов, выходящих и входящих, можно задавать словами `up`, `down`, `left`, `right`.

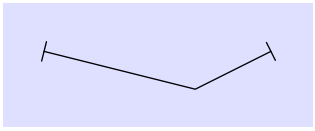
```
unitsize(1 cm);
path g=(0,0){down}..(2,-.5)..{right}(3,0);
draw(g);
```



В трёх следующих примерах показано, как на концах линии можно рисовать полоски, стрелки и т.п. Например, зададим на концах ломаной полоски:

Пример 8

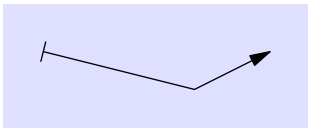
```
unitsize(1 cm);
path g=(0,0)--(2,-.5)--(3,0);
draw(g,Bars);
// Задаем "полоски" на концах.
```



Пример 9

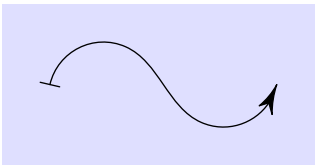
Покажем, как можно сочетать полоски и стрелки на концах пути:

```
unitsize(1 cm);
path g=(0,0)--(2,-.5)--(3,0);
draw(g,BeginBar,EndArrow);
/* Задаем на одном конце стрелку,
а на другом - "полоску".*/
```



Пример 10

```
unitsize(1 cm);
path g=(0,0)..(1,.5)..(2,-.5)..(3,0);
draw(g,BeginBar,EndArrow(arrowhead=HookHead));
/* Дополнительно указываем вид конца
стрелки.*/
```



Структура команды `draw` описывается функцией `void draw`:

```
void draw(picture pic=currentpicture, Label L="", path g,
        align align=NoAlign, pen p=currentpen,
        arrowbar arrow=None, arrowbar bar=None, margin margin=NoMargin,
        Label legend="", marker marker=nomarker);
```

В круглых скобках через запятую перечислены аргументы функции, причем слева от знака равенства стоит аргумент, затем его имя, а справа от знака равенства — значение по умолчанию:

- `picture pic=currentpicture` : рисуется картинка pic, по умолчанию это `currentpicture`.
- `Label L=` : позволяет наложить на картинку текст или другую метку. Для этого текст нужно расположить в двойных кавычках.
- `path g` : изображаемый путь.
- `pen p=currentpen` : используемое перо, по умолчанию currentpen.
- `arrowbar arrow=None` : рисуется стрелка (по умолчанию рисуется просто путь).
- `arrowbar bar=None` : рисуются полоски на концах пути (по умолчанию полосок нет).

В частности, аргументы функции `void draw`, приведённые в рамке, можно расшифровать так: начертить путь `g` на рисунке `pic=currentpicture` пером `p=currentpen` (без меток Label L, а значит и выравнивания `align`, стрелок `arrow`, полосок `bar`, полей `margin`, легенды `legend`, и маркеров `marker`). Используется лишь один параметр, путь.

Аргументы `arrow` и `bar` могут быть указаны в любом порядке. Аргумент `legend` является меткой `Label`.

Варианты стрелок : `None, Blank, BeginArrow, MidArrow, Arrow` или `Arrows` (стрелки на обоих концах). Для стрелок можно указать тип наконечника (`DefaultHead,SimpleHead,HookHead,TeXHead`), его размер и угол, тип заполнения (`FillDraw, Fill, NoFill, UnFill, Draw`) и позицию расположения. Существуют стрелки для дуг : `BeginArcArrow, MidArcArrow, ArcArrow` или `ArcArrows`.

Полоски полезны для указания размеров. возможные значения `bar` — `None, BeginBar, EndBar` или `Bars` (или, что эквивалентно, `Bar`), и `Bars` (когда чертятся полоски на обоих концах пути). Каждый из этих спецификаторов (за исключением `None`) будет принимать дополнительный действительный аргумент, который обозначает длину полоски в `PostScript`-координатах. Длина полоски по умолчанию `barsize (pen)`.

Поля `margin` могут быть использованы для сокращения видимой части пути (например, для избежания наложений на другие объекты). Типичными значениями `margin` являются следующие: `NoMargin, BeginMargin, EndMargin`(что эквивалентно `Margin`) и `Margins` (которые оставляют поля на обоих концах пути).

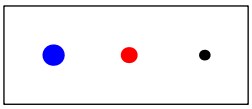
Более подробную информацию об аргументах команды `draw` можно найти:

1. В пособии *Christophe Grospellier* **Asymptote. Démarrage «rapide»**
   (Смотри http://www.cgmaths.fr/Atelier/Asymptote/Asymptote.html)
2. В руководстве по Asymptote, раздел Documentation.
   Смотри http://asymptote.sourceforge.net/index.html

Команду `draw` можно использовать для изображения точек. Чтобы нарисовать точку, можно просто нарисовать путь, состоящий из единственной точки.

Пример 11

```
unitsize(1cm);// Определяем единицу измерения.
pair A=(2,0);// Задаем точку A.
draw((0,0),8bp+blue);// Рисуем точку (0,0) синего цвета размером 8bp.
draw((1,0),6bp+red);//  Рисуем точку (1,0) красного цвета размером 6bp.
draw(A,linewidth(4bp));//  Рисуем точку A размером 4bp.
shipout(bbox(5mm));
```



- Команда ***dot***.
  Хотя изображать точки можно командой `draw`, в языке `Asymptote` предусмотрена специальная команда, рисующая точки, команда `dot`, которая имеет несколько форм. Вот одна из них:

```
void dot(picture pic=currentpicture, pair z, pen p=currentpen,
         filltype filltype=Fill);
```

В этом случае команда `dot`, определенная в модуле `plain`, рисует точку с диаметром, связанным с явной толщиной пера, используя указанный `filltype`(при этом толщина пера умножается на `dotfactor`, который по умолчанию равен 6):

Пример 12

```
dot((0,0));//Рисуем точку.
```


Пример 13

В этом примере показывется, как получить тот же результат другим способом.

```
real w=linewidth();/*Задаем действительное число w,
равное толщине пера.*/
dot((0,0),linewidth(6w));//Рисуем точку с диаметром 6w.
```


Пример 14

Изменим тип заполнения пера. Точка в виде закрашенного круга заменяется точкой в виде окружности. Разумеется, что можно менять диаметр точки и её цвет.

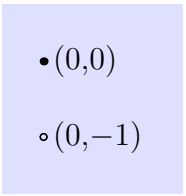

```
dot((0,0),filltype=Draw());
```

Следующая структура применяется, когда нужно нарисовать точку с текстовой меткой:

```
void dot(picture pic=currentpicture, Label L, pair z, align align=NoAlign,
         string format=defaultformat, pen p=currentpen, filltype filltype=Fill);
```

Пример 15

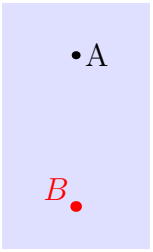
Здесь метка `Label L`, а это всё, что находится внутри двойных кавычек, пуста. По умолчанию показывается пара координат точки привязки.




```
dot("",(0,0));
dot("",(0,-1),filltype=Draw());
```

Во второй строке кода кроме пустой метки и точки привязки указан тип заполнения пера.

Пример 16



Первая строка кода.
Простейшая метка: буква A. Указана точка привязки, остальные аргументы по умолчанию.

```
dot("A",(0,0));
dot(Label("$B$",(0,-2),align=NW),4bp+red);
```

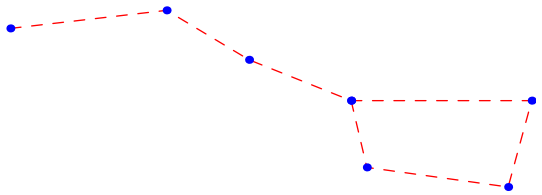
Вторая стока кода.
Метка: буква B в TEX-формате и математической моде. Точка привязки (0,-2), компасное указание позиции, явно указан размер пера и его цвет.

Для изображения узловых точек пути можно использовать структуру:

```
void dot(picture pic=currentpicture, Label[] L=new Label[],path g, align
         align=RightSide, string format=defaultformat,pen p=currentpen,
         filltype filltype=Fill);
```

Пример 17

```
size(7cm,0); // размер рисунка
pair A=(-4.44,1.02),B=(-2.45,1.25),C=(-1.4,0.62),D=(-.1,.1),E=(.1,-.75),F=(1.9,-1.),
G=(2.2,.1);
path bm=A--B--C--D--E--F--G--D; /* определяем путь */
draw(bm,dashed+red);/* чертим путь пунктиром пером красного цвета*/
dot(bm,blue);/* рисуем узлы пером синего цвета */
```



Есть и более общие методы для маркировки узлов пути.

1.2.2 Команда fill

Структура команды `fill` намного проще, чем у команды `draw`

```
void fill(picture pic=currentpicture, path g, pen p=currentpen);
```

Команда `fill` заполняет циклический путь `g` в картинке `pic` указанным пером `p` (чаще всего с аргументом цвет).

Пример 18

```
unitsize(0.75 inch);
fill((0,0)--(0,1)--(1,1)--(1,0)--cycle,red);
```

Существует также удобная команда `filldraw`, которая заполняет путь, а затем очерчивает границу. Можно указать отдельные перья `pen fillpen` и `pen drawpen` для каждой операции:

```
void filldraw(picture pic=currentpicture, path g, pen fillpen=currentpen,
                                pen drawpen=currentpen);
```

Пример 19

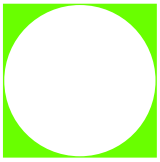
```
unitsize(1.5cm);
path g=(0,0)--(2,0)--(2,1)--(0,1)--cycle;
filldraw(g,fillpen=gray,drawpen=linewidth(1mm)+.95red);
```

Команда `filloutside` позволяет заполнить внешнюю область пути `g`, внутри бокса с рисунком `pic`.

```
void filloutside(picture pic=currentpicture, path g, pen p=currentpen);
```

Пример 20

```
size(3cm); // Размер рисунка
path c=unitcircle; // единичная окружность
filloutside(c,green+.7yellow);
/* картинка заполняется вне окружности
цветным пером */
```



Когда мы имеем несколько замкнутых путей, то возможны различные варианты заполнения областей, которые при этом получаются. Для этого используют перо `p` с правилом заполнения(закрашивания).

По умолчанию используется правило `pen zerowinding=fillrule(0)`.

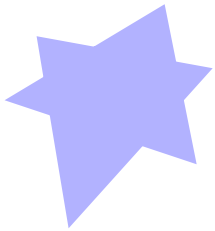
Ещё одно правило `pen evenodd=fillrule(1)`, закрашивание с чередованием. Рассмотрим несколько примеров.

Пример 21

Даны два замкнутых пути, каждый из которых является треугольником. Треугольники накладываются друг на друга. Используем правило заполнения `zerowinding`.

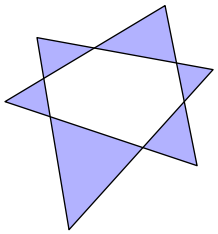
Заметим, что пути `l` и `l1` соединены знаком `^^`. Этот знак позволяет рассматривать два пути как один.

```
unitsize(12);
pair [] z;
z[0]=(-1,-3); z[1]=(2,4); z[2]=(-3,1);
z[3]=(3,-1); z[4]=(3.5,2); z[5]=(-2,3);
path l=z[0]--z[4]--z[5]--cycle;
path l1=z[1]--z[2]--z[3]--cycle;
fill(l^^l1,blue+.7yellow);
//filldraw(l^^l1,blue+.7yellow);
```



Пример 22

Для тех же двух треугольников используем правило `evenodd`.



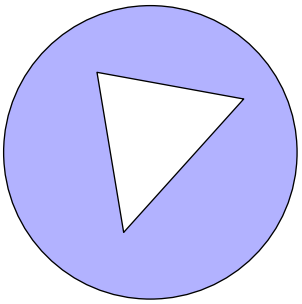
```
unitsize(12);
pair [] z;
z[0]=(-1,-3); z[1]=(2,4); z[2]=(-3,1);
z[3]=(3,-1); z[4]=(3.5,2); z[5]=(-2,3);
path l=z[0]--z[4]--z[5]--cycle;
path l1=z[1]--z[2]--z[3]--cycle;
filldraw(l^^l1,blue+.7yellow+evenodd);
```

Пример 23

Теперь даны два замкнутых пути, один из которых треугольник, а второй является окружностью. Они расположены так, что вершины треугольника лежат внутри окружности. Нужно закрасить область внутри окружности, но вне треугольника.

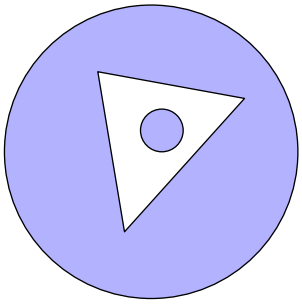
Эту задачу можно решить двумя способами. Первый из них использует функцию `reverse` для треугольника. Она меняет направление обхода треугольника. По умолчанию применяется правило заполнения `zerowinding`. Второй способ заключается в применении правила заполнения `evenodd`.

```
unitsize(10);
pair [] z;
z[0]=(-1,-3); z[1]=(2,4); z[2]=(-3,1);
z[3]=(3,-1); z[4]=(3.5,2); z[5]=(-2,3);
path l=z[0]--z[4]--z[5]--cycle;
path l1=z[1]--z[2]--z[3]--cycle;
path [] C;
C[0]=shift(.3*z[1])*scale(.8)*unitcircle;
C[1]=scale(5.5)*unitcircle;
filldraw(l^^C[1],blue+.7yellow+evenodd);
```



Пример 24

В этом примере уже три пути, две окружности и треугольник.

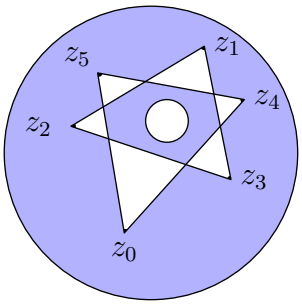


```
unitsize(10);
pair [] z;
z[0]=(-1,-3); z[1]=(2,4); z[2]=(-3,1);
z[3]=(3,-1); z[4]=(3.5,2); z[5]=(-2,3);
path l=z[0]--z[4]--z[5]--cycle;
path l1=z[1]--z[2]--z[3]--cycle;
path [] C;
C[0]=shift(.2*z[1])*scale(.8)*unitcircle;
C[1]=scale(5.5)*unitcircle;
filldraw(l^^C[0]^^C[1],blue+.7yellow+evenodd);
```

Наконец, пример с четырьмя путями:

Пример 25

```
unitsize(10);
pair [] z;
z[0]=(-1,-3); z[1]=(2,4); z[2]=(-3,1);
z[3]=(3,-1); z[4]=(3.5,2); z[5]=(-2,3);
path l=z[0]--z[4]--z[5]--cycle;
path l1=z[1]--z[2]--z[3]--cycle;
dot("$z_{0}$",z[0],S); dot("$z_{1}$",z[1]);
dot("$z_{2}$",z[2],2W);
dot("$z_{3}$",z[3]); dot("$z_{4}$",z[4]);
dot("$z_{5}$",z[5],NW);
path [] C;
C[0]=shift(.3*z[1])*scale(.8)*unitcircle;
C[1]=scale(5.5)*unitcircle;
filldraw(l^^l1^^C[0]^^C[1],blue+.7yellow+evenodd);
```



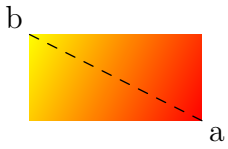
Кроме закрашивания, при котором краска распределяется по области равномерно, есть так называемое затенение (`shading`). Вот примеры некоторых способов затенения.

Осевое плавное градиентное затенение, которое варьируется от `pena` до `penb` в направлении линии сегмента `a -- b` может быть достигнуто так

```
void axialshade(picture pic=currentpicture, path g, bool stroke=false,
pen pena, pair a, pen penb, pair b);
```

Пример 26

```
size(3cm,0);
transform t=xscale(2); // Задаем преобразование(сжатие к оси x)
pen p1=red, p2=yellow; // Определяем перья
pair pa=t*dir(0), pb=t*dir(90); // Задаем концы диагонали
axialshade(t*unitsquare,p1,pa,p2,pb); // Путем является сжатый единичный квадрат
draw(pa---pb, dashed); // Рисуем диагональ (пунктиром)
```

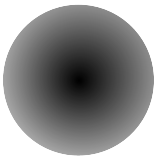


Ещё существует радиальное плавное градиентное затенение, которое варьируется от `pena` на окружности с центром `a` и радиусом `ra` до `penb` на окружности с центром `b` и радиусом `rb`:

```
void radialshade(picture pic=currentpicture, path g, bool stroke=false,
pen pena, pair a, real ra, pen penb, pair b, real rb);
```

Пример 27

```
size(3cm,0);
pair A=(0,0), B=(0.1,0.1);
radialshade(unitcircle,black,A,0,lightgrey,B,1.5);
shipout(bbox(5mm));
```



1.2.3 Команда clip

Структура команды `clip` еще проще:

```
void clip(picture pic=currentpicture, path g, pen p=currentpen);
```

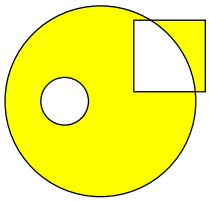
Эта команда обрезает изображение по границе пути `g` пером `p` с правилом заполнения(заливки). Как и в команде `fill` по умолчанию используется правило `pen zerowinding=fillrule(0)`.

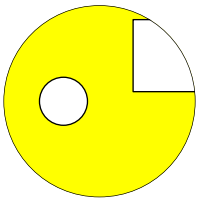
Пример 28

С помощью кода рисуем желтый круг с небольшой белой дыркой и квадрат, который наложен на него.

```
size(0,72);
path unitcircle=E..N..W..S..cycle;// Рисуется единичная окружность.
path g=scale(2)*unitcircle;// Она увеличивается в два раза. Получаем путь g.
path a=shift(-.75,0)*scale(.5)*unitcircle;// Рисуется дырка. Получаем путь a.
path b=shift(.7,.2)*scale(1.5)*unitsquare;/* Рисуется единичный квадрат,
     он увеличивается в 1.5 раза и сдвигается на вектор (.7,.2). Получаем путь b.*/
filldraw(a^^b^^g,evenodd+yellow,black);/* Рисуются пути: окружность, дырка,
     квадрат и заполняются желтым цветом с чередованием и черным контуром.*/
```

Для того, чтобы обрезать все, что находится вне желтого круга, к указанному коду нужно приписать команду `clip(g)`. В итоге получим картинку справа:



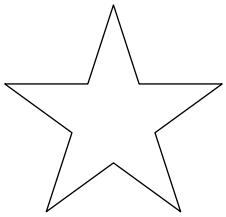


Пример 29

Он взят со wiki-странички:

http://www.artofproblemsolving.com/Wiki/index.php/Asymptote

Рассмотрим смайлик и звезду:



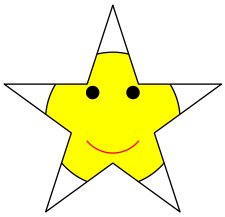
Смайлик нарисован кодом:

```
import graph;// Подключаем модуль graph, чтобы воспользоваться функцией
//черчения окружности по заданному центру и радиусу и функцией arc.
unitsize(.35inch);
filldraw(Circle((0,0),1),yellow,black);
fill(Circle((-.3,.4),.1),black);//Рисуем левый глазик.
fill(Circle((.3,.4),.1),black);//Рисуем правый глазик.
draw(Arc((0,0),.5,-140,-40),red);// Рисуем ротик
```

Звезда чертится кодом

```
import graph;
unitsize(.35inch);
path star;
star=expi(0)--(scale((3-sqrt(5))/2)*expi(pi/5))--expi(2*pi/5)--
     (scale((3-sqrt(5))/2)*expi(3*pi/5))--expi(4pi/5)--
     (scale((3-sqrt(5))/2)*expi(5*pi/5))--expi(6*pi/5)--
     (scale((3-sqrt(5))/2)*expi(7*pi/5))--expi(8*pi/5)--
     (scale((3-sqrt(5))/2)*expi(9*pi/5))--cycle;
draw(scale(1.7)*rotate(18)*star);
```

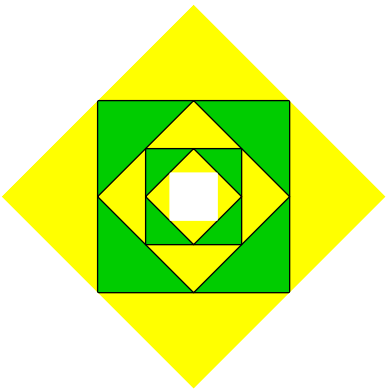
Мы хотим обрезать этот смайлик по границе звезды. К коду смайлика мы добавляем код звезды и команду:
`clip(currentpicture,scale(1.7)*rotate(18)*star)`.
В результате получаем:



Пример 30

Если мы хотим обрезать рисунок с чередованием, то мы можем использовать правило заполнения `evenodd`:

```
path s0=(-18,-18)--(-18,18)--(18,18)--(18,-18)--cycle;
path s1=(0,-18)--(-18,0)--(0,18)--(18,0)--cycle;
path s2=scale(2)*s0;
path s3=scale(2)*s1;
path C0=scale(.5)*s0;
path C2=scale(2)*s3;
path [] zones=C0^^C2;
fill(zones,yellow+evenodd);
filldraw(s0^^s1^^s2^^s3,.8green+evenodd);
clip(currentpicture,zones,evenodd);
```



1.2.4 Команда label

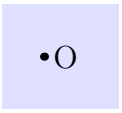
После того, как создан рисунок или прямо в процессе создания, на него можно наложить текст, какие-либо обозначения, формулы и тому подобное. Всё это мы будем называть **метками**.

Добавлять метки к рисунку можно различным образом. Для этого существует команда `label`. Сначала на ряде простых примеров покажем, как снабдить метками простейшие пути — точки.

Пример 31

Создадим код, который добавляет к точке $(0,0)$ ее обозначение, например букву O, которая расположена справа от точки.

```
unitsize(1cm);
dot((0,0));// Рисуем точку.
label("O",(0,0),E);/* Указываем строку с буквой "O",
точку привязки (0,0) и, как в компасе, букву E.*/
```

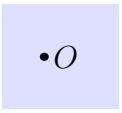


Мы видим, что рисунок немного отличается от задуманного, поскольку буква O — обычная заглавная, а мы хотели наклонную. Чтобы это сделать, используем TEX, изменим строку на O.

Пример 32

Посмотрите код и полученное изображение:

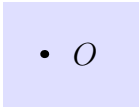
```
unitsize(1cm);
dot((0,0));// Рисуем точку.
label("$O$",(0,0),E);/* Указываем тех-код буквы "O",
точку привязки (0,0) и, как в компасе, букву E.*/
```



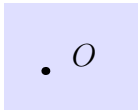
Вместо компасного обозначения можно более точно указать направление и изменить расстояние от метки до точки привязки.

Пример 33

```
unitsize(1cm);
dot((0,0));
label("$O$",(0,0),3*E);/* Увеличиваем расстояние
от метки до точки привязки в три раза.*/
```



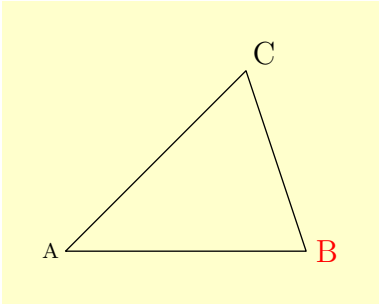

```
unitsize(1cm);
dot((0,0));
label("$O$",(0,0),3*dir(23));
// Указываем направление в градусах.
```



Пример 34

Можно менять размер шрифта текстовой метки и цвет.

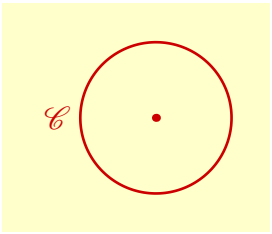
```
size(6cm);
pair A, B, C;
A=(-2,0); B=(2,0); C=(1,1);
draw(A--B--C--cycle); /*Рисуем треугольник
с вершинами в указанных точках.*/
label("A",(-2,0),W,fontsize(8pt));
// Указываем размер шрифта.
label("B",(2,0),E,10pt+red);
//Указываем размер шрифта и его цвет.
label("C",C,NE);
//Используется размер шрифта по умолчанию.
```



Пример 35

Можно менять шрифт текстовой метки.

```
import geometry;//Подгружаем модуль geometry.
unitsize(1cm);
point O=origin;
usepackage("mathrsfs");//Подгружаем шрифтовый пакет.
real r=1;
circle Ci= circle(O,r);/*Определяем окружность
Ci с центром (0,0) и радиусом 1.*/
dot(Ci.C,.8*red);//Рисуем центр темнокрасным пером.
draw("$\mathscr{C}$",Ci,bp+.8*red);/*Рисуем
окружность Ci тем же пером и ее метку.*/
```



Более подробно о метках и команде `label` можно прочитать в *главе 2, раздел 2.5*, а пока покажем, как метку можно присоединить не к точке, а к пути.

Для присоединения метки к пути используют команду `label` со следующей структурой:

```
void label(picture pic=currentpicture, Label L, path g, align align=NoAlign,
           pen p=nullpen, filltype filltype=NoFill);
```

По умолчанию точка привязки находится на середине пути. Альтернативное положение точки привязки можно указать заданием действительного значения в конструкции метки.

Аргумент `position=Relative(real)` определяет место точки привязки по отношению к общей длине (`arclength`) пути. Есть удобные сокращения:

```
position BeginPoint=Relative(0);
position MidPoint=Relative(0.5);
position EndPoint=Relative(1);
```

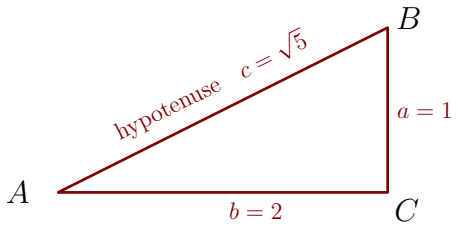
Метки путей позиционируются в направлении `align` от точки привязки, которое может быть задано как абсолютное направление по компасу или относительное направление `Relative(pair)`, где `pair` является радиус-вектором в локальном репере $(O, \vec{i}, \vec{j})$ (правом) с началом O в точке привязки, а вектор $\vec{j}$ направлен по касательной к пути по направлению возрастания параметра пути.

Для удобства `LeftSide`, `Center`, и `RightSide` определяется как `Relative(W)`, `Relative((0,0))`, и `Relative(E)`, соответственно. Умножение `LeftSide`, `Center`, и `RightSide` слева на действительный масштабирующий коэффициент будет двигать метку дальше или ближе к пути.

Пример 36

В этом примере показывается, как можно формульные и буквенные метки присоединять к сторонам треугольника. Обратите внимание, что метки можно различным образом выравнивать, масштабировать, поворачивать, словом подвергать преобразованиям.

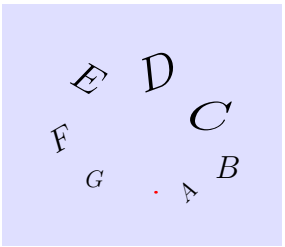
```
size(6cm);
pair A=(0,0), B=(2,1), C=(2,0); //Определяем вершины треугольника.
path tri= A--B--C--cycle, f=A--C, g=C--B, h=A--B; //Определяем треугольник
//tri и его стороны.
label("$A$",A,3*dir(A-C)); //Обозначаем вершину A.
label("$B$",B,dir(B-A)); //Обозначаем вершину B.
label("$C$",C,dir(-45)); //Обозначаем вершину C.
draw(tri,bp+brown); //Чертим треугольник tri коричневым пером.
draw(scale(.75)*Label("$b=2$", position=Relative(.6)),f,brown); //Наносим метку длин-
//ного катета.
draw(scale(.75)*Label("$a=1$",position=Relative(.5)),g,brown); //Наносим метку второго
//катета.
draw(scale(.75)*Label("$\mathrm{hypotenuse}\quad c=\sqrt{5}$",Rotate(dir(h))),align=
1.5*LeftSide,h,brown); //Наносим надпись вдоль
//гипотенузы с левой стороны и увеличиваем расстояние от надписи до гипотенузы.
```



Текст метки `Label` может быть подвергнут преобразованиям. Следующий пример показывет некоторые варианты преобразований.

Пример 37

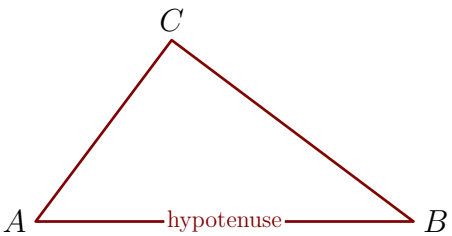
```
unitsize(1cm);
pair O=(0,0);
dot(O,bp+red);
label(rotate(45)*scale(.75)*"$A$",O,2*dir(0));
label(shift(10,0)*"$B$",O,4*dir(30));
label(xscale(1.5)*scale(1.25)*"$C$",O,8*dir(55));
label(shift(0,10)*rotate(20)*scale(1.5)*"$D$",O,8*dir(90));
label(rotate(-30)*scale(1.25)*"$E$",O,12*dir(120));
label(rotate(30)*"$F$",O,10*dir(150));
label(shift(-5,5)*scale(.75)*"$G$",O,4*dir(180));
```



В следующих примерах показано применение опций `filltype` и `position`. В примере 32 можно увидеть результат действия комбинации опций `filltype=UnFill` и `align=Center`. Первая из них создает бокс для метки. Вторая располагает бокс по линии гипотенузы так, что она является его осью симметрии. Значение `UnFill` очищает его перед вложением метки. Поскольку опция `position` отсутствует, то по умолчанию метка располагается по центру гипотенузы.

Пример 38

```
size(6cm);
pair A=(0,0),B=(5,0), C=(1.8,2.4);
path tri= A---B---C---cycle,
f=A---C, g=C---B, h=A---B;
label("$A$",A,W);label("$B$",B,E);
label("$C$",C,N);draw(tri,bp+brown);
draw(scale(.75)*Label("$\mathrm{hypotenuse}$",
align=Center,filltype=UnFill), h,brown);
/*опция filltype=UnFill разрывает гипотенузу
и вставляет надпись.*/
```



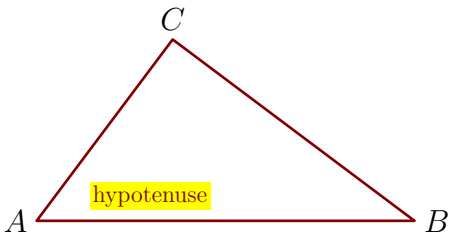
Пример 39

В примере другое сочетание значений опций `filltype`, `align` и `position`.

Значение `align=1.5*LeftSide` означает, что бокс с меткой расположен по левую сторону от гипотенузы, если двигаться от точки A к точке B, причем расстояние по умолчанию увеличено в полтора раза.

Опция `position=Relative(.3)` означает, что центр бокса расположен от точки A на расстоянии 0.3 длины гипотенузы. Наконец опция `filltype=Fill(yellow)` означает, что бокс заполняется пером желтого цвета.

```
size(6cm);
pair A=(0,0),B=(5,0),C=(1.8,2.4);
path tri= A---B---C---cycle,
f=A---C,g=C---B,h=A---B;
label("$A$",A,W);label("$B$",B,E);
label("$C$",C,N);draw(tri,bp+brown);
draw(scale(.75)*Label(
"$\mathrm{hypotenuse}$",align=
1.5*LeftSide, position=Relative(.3),
filltype=Fill(yellow)), h,brown);
```

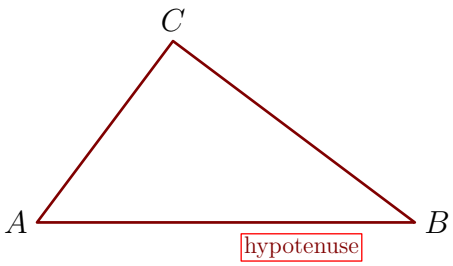


Пример 40

```
size(6cm);
pair A=(0,0),B=(5,0), C=(1.8,2.4);
path tri= A---B---C---cycle, f=A---C, g=C---B, h=A---B;
label("$A$",A,W);label("$B$",B,E);label("$C$",C,N);
draw(tri,bp+brown);
draw(scale(.75)*Label("$\mathrm{hypotenuse}$",align=1.5*RightSide,
position=Relative(.7),filltype=Draw(red)), h,brown);
```

Аналогично предыдущему, но другое сочетание значений опций `filltype`, `align` и `position`.

Значение `align=1.5*RightSide` означает, что бокс с меткой расположен с правой стороны от гипотенузы, если двигаться от точки A к точке B, причем расстояние по умолчанию умножается на 1.5. Опция `position=Relative(.7)` означает, что центр бокса расположен от точки A на расстоянии 0.7 длины гипотенузы. Наконец, опция `filltype=Draw(red)` означает, что чертится только граница бокса пером красного цвета.



1.2.5 Метки с кириллицей

Помимо обычных TEX-овских меток с латиницей, приходится использовать и метки, содержащие текст на русском языке. Обычными средствами LATEX'а сделать это трудно. В `Asymptote` для этого используют модуль *unicode* (Смотри *Глава 3. Краткий обзор модулей*). Команда `import unicode` в начале файла предписывает `LaTeX`'у принять стандартизированные международные символы `unicode` (UTF-8). Для использования кириллических шрифтов, вам необходимо будет изменить кодировку шрифтов:

```
import unicode;
texpreamble("\usepackage{mathtext}\usepackage[russian]{babel}");
defaultpen(font("T2A","cmr","m","n"));
```

При использовании **Ubuntu 14.04** можно использовать следующий код без подгружения модуля *unicode*:

```
texpreamble("\usepackage[T2A]{fontenc}\usepackage[utf8]{inputenc}
\usepackage{mathtext}\usepackage[russian]{babel}");
```

Замечания:

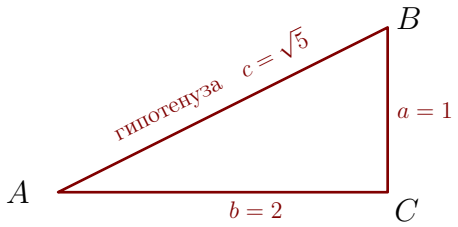
1) Подгружение пакета \usepackage{mathtext} не обязательно. Так же не обязательна третья строка defaultpen(font("T2A","cmr","m","n");

2) В математической моде (например, между знаками $$) русский текст всё равно не воспроизводится.

Пример 41

Попробуем в предыдущем примере заменить английское hypotenuse на русское. Мы видим, что русский текст не воспроизводится. Исправим код следующим образом:

```
texpreamble("\usepackage[T2A]{fontenc}\usepackage[utf8]{inputenc}
\usepackage{mathtext}\usepackage[russian]{babel}");
size(6cm);
pair A=(0,0), B=(2,1), C=(2,0);//Определяем вершины треугольника.
path tri= A--B--C--cycle, f=A--C, g=C--B, h=A--B;//Определяем треугольник
//tri и его стороны.
label("$A$",A,3*dir(A-C));//Обозначаем вершину A.
label("$B$",B,dir(B-A));//Обозначаем вершину B.
label("$C$",C,dir(-45));//Обозначаем вершину C.
draw(tri,bp+brown);//Чертим треугольник tri коричневым пером.
draw(scale(.75)*Label("$b=2$", position=Relative(.6)),f,brown);
//Наносим метку длинного катета.
draw(scale(.75)*Label("$a=1$"),g,brown);//Наносим метку второго катета.
draw(scale(.75)*Label("гипотенуза$\quad c=\sqrt{5}$",Rotate(dir(h))),align=
1.5*LeftSide, h,brown);//Наносим надпись вдоль
//гипотенузы с левой стороны и увеличиваем расстояние от надписи до гипотенузы.
```



1.3 Интеграция Asymptote и LaTeXa

С помощью LaTeXa можно верстать документы, содержащие рисунки. Рисунки могут быть заранее созданы внешними программыми средствами и затем включены в документ. Рисунки могут иметь различные форматы. Общие правила внедрения графики можно прочитать, например, в книге [4], разделах 10.3 – 10.6.

Далее в подразделах рассмотрены два способа интеграции рисунков и текста. На выбор способа могут влиять: количество рисунков, производительность компьютера, стадия работы над документом (предварительная или окончательная), объём самого документа и т.д.

В первом подразделе расскажем, как включить заранее заготовленные с помощью `asymptote` рисунки в документ. Во втором подразделе расскажем, как включить код `asymptote` в исходный tex-файл документа. Мы уже говорили, что по умолчанию выходным форматом `asymptote` является EPS, но может быть и PDF. От этого и от того, в каком формате нам нужен выходной файл документа, будет зависеть последовательность действий при компиляции исходного tex-файла документа.

1.3.1 Включение графики в документ

1) Предположим, что нам нужно получить выходной `dvi`-файл документа с несколькими рисунками. Мы заранее заготовили рисунки в формате `eps`, используя `asymptote`, и теперь можем просто включить их в текст, используя команду `\includegraphics{}`. Пусть `picture.eps` — файл с картинкой.

Для вставки рисунка в `latex`-документ последовательность действий такова:

- Располагаем файл `picture.eps` в одной папке с исходным файлом документа.
- В преамбулу исходного файла документа вносим команду подгружения графического пакета:

  ```
  \usepackage[dvips]{graphicx}
  ```

- В нужном месте текста, там, где должна быть картинка, вставляем команду

  ```
  \includegraphics{picture.eps}
  ```

- Обрабатываем исходный `tex`-файл документа компилятором `latex` и затем просматриваем полученный `dvi`-файл.

Замечания :

1. Расширение `.eps` можно не указывать.
2. Полученный рисунок будет прижат к левому краю страницы. Чтобы расположить его по центру, используйте стандартные средства LaTeXa следующим образом:

   ```
   \begin{center}
    \includegraphics{picture.eps}
   \end{center}
   ```

3. Если нужно изменить размер рисунка, есть два способа:

   сменить `unitsize` в коде `asymptote`(изменить внутренний масштаб) или использовать необязательный аргумент `scale` в команде `\includegraphics`. Например, мы можем увеличить рисунок в два раза:

   ```
   \includegraphics[scale=2]{picture.eps}
   ```

Если же вас не устраивает `dvi`-файл в качестве выходного, а хотелось бы иметь `ps`-файл или `pdf`-файл, можно использовать конверторы `DVItoPS`, `PStoPDF` или `DVItoPDF` на панели инструментов среды **Kile** или редактора **Texmaker**. Для облегчения работы можно использовать функцию быстрой сборки, но для этого нужно предварительно настроить редактор.

Однако использование конверторов — не самый лучший вариант. Есть способ прямого использования компилятора `pdflatex`.

2) Пусть мы хотим получить на выходе документ с рисунками в формате PDF. Последовательность действий такова:

- Изготовим с помощью `asymptote` рисунки в формате PDF. Для этого в начале кода рисунка нужно вставить команду

  ```
  settings.outformat="pdf";
  ```

  сохранить файл с кодом как `picture.asy` и обработать интерпретатором `asymptote`. Тогда после компиляции файл рисунка получится в формате PDF.

- Располагаем файл `picture.pdf` в одной папке с исходным файлом документа.
- В нужном месте текста, там, где должна быть картинка, вставляем команду

  ```
  \includegraphics{picture.pdf}
  ```

- Обрабатываем исходный файл документа компилятором `pdflatex`. Документ готов.
- Просматриваем полученный `pdf`-файл.

К сказанному в пункте 2 применимы замечания, указанные в пункте 1.

1.3.2 Включение кода asymptote в tex-файл документа

Можно сразу включить код asymptote в исходный tex-файл документа. Пусть имя tex-файла будет example.tex.

1. **Способ первый**. Создаём dvi-файл документа. Компилируем, используя latex

   Для включения кода asymptote необходимо сделать следующее:

   - Сначала вставим в преамбулу файла example.tex команду \usepackage{asymptote}.
   - Добавляем в преамбулу файла example.tex команду, подгружающую пакет graphicx \usepackage[dvips]{graphicx}.
   - Вставляем в нужное место файла example.tex следующий блок

     ```
     \begin{asy}
        asymptote-код рисунка
     \end{asy}
     ```

   - Компилируем tex-файл документа, последовательно используя latex → asymptote → latex. Файл рисунка будет в формате EPS.
   - Просматриваем dvi-файл документа.

2. **Способ второй**. Создание pdf-документа.

   - Вставим в преамбулу файла example.tex команду \usepackage{asymptote}.
   - В нужное место файла example.tex вставляем asymptote-код рисунка. В этом случае код рисунка должен начинаться указанием выходного формата:

     ```
     \begin{asy}
        settings.outformat="pdf";
        asymptote-код рисунка
     \end{asy}
     ```

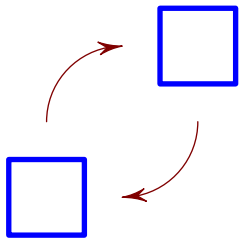
   - Компилируем последовательно, используя pdflatex → asymptote → pdflatex.
   - Просматриваем pdf-файл документа.

Для нужного позиционирования рисунков окружение \begin{asy}...\end{asy} помещаем внутри окружения \begin{center}...\end{center} или ему подобных. При желании можно всё это поместить внутри окружения \begin{figure}...\end{figure} — это в том случае, когда вы хотите вставить картинку как плавающий объект.

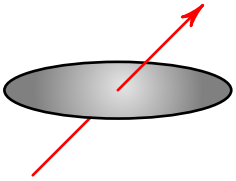
Приведем простой пример: нарисуем картинку из двух квадратов и поместим слева от картинки код, который её рисует.

Пример 42

```
\begin{asy}
unitsize(1cm);
draw(unitsquare,2bp+blue);
draw(shift(2,2)*unitsquare,2bp+blue);
path p=(.5,1.5){(0,1)}..{(1,0)}(1.5,2.5);
path q=(2.5,1.5){(0,-1)}..{(-1,0)}(1.5,.5);
draw(p,brown,Arrow(HookHead,size=2mm));
draw(q,brown,Arrow(HookHead,size=2mm));
\end{asy}
```



Вот второй пример, в котором картинка вставляется как плавающий объект.



Пример 43

Приведем код к этой картинке:

```
\begin{figure}[hb]
\begin{center}
\begin{asy}
settings.outformat="pdf"; unitsize(1cm);
pair A=(0,0), B=(0.1,0.1);
path e=scale(2)*unitcircle; path e1=yscale(.25)*e;
path ar1=(0,0)--(1.5,1.5); path ar2=(-2,-2)--(0,0);
radialshade(e1,grey,A,1.5,lightgrey,B,0);
pair P=intersectionpoint(ar2,e1);
draw((-1.5,-1.5)--P,bp+red); draw(e1,linewidth(bp));
draw(ar1,bp+red,Arrow(HookHead,size=2mm));
shipout(bbox(3mm));
\end{asy}
\end{center}
\end{figure}
```

Глава 2

Элементы программирования

Предлагаем некоторые краткие примеры использования языка программирования Asymptote, управляющие структуры которого имеют большое сходство со структурами языков C, C++ и Java:

```
real x; // Объявляется x как действительная переменная;

x=1.0; //  Действительной переменной x присваивается значение 1.

if(x == 1.0) {
  write("x equals 1.0");
} else {
  write("x is not equal to 1.0");
}
//  Проверка условия x = 1.0 или нет.

for(int i=0; i < 10; ++i) {
  write(i);
}
//  Цикл в 10 шагов
```

В Asymptote , как в C/C++, есть конструкции while, do, break и continue. Кроме того, в нем поддерживается Java-стиль сокращений для прохода по всем элементам массива:

```
 // Проход по массиву (Iterate over an array)
int[] array={1,1,2,3,5};
for(int k : array) {
  write(k);
}
```

В Asymptote используется множество функций, которые есть в языках C, C++ и Java. Мы перечислим следующие типы данных: void, bool, int, real, pair, triple, string, path, guide, pen, transform, frame, picture. Это неполный список. Отметим, что пользователи могут дополнительно определять свои собственные типы.

void	Этот тип используется только для функций, которые не имеют аргументов или не возвращают никаких значений.
bool	Данный тип принимает только два значения, true либо false. Например: bool b=true; Определяется логическая (булева) переменная b и ей присваивается значение true. Если присвоения не присходит: bool b;, то предполагается еще и значение false.
int	Целое число; если число не инициализировано, то предполагается неявное значение 0. Минимально допустимое значение целого числа является intMin, а максимальное значение — IntMax.

real	Действительное число. Представляется в виде десятичной дроби с плавающей точкой. Неявно инициализируется значением `0.0`. Действительные числа имеют точность `realEpsilon`; число значащих цифр `realDigits`. Наименьшее положительное действительное число — `realMin`, а наибольшее положительное действительное число — `realMax`.
pair	Комплексное число, то есть упорядоченная пара `(x,y)` действительных чисел (компонентов). Действительную и мнимую части пары `z` обозначают `z.x` и `z.y`. Неявно инициализируется значением `(0.0, 0.0)`.
triple	Упорядоченная тройка `(x,y,z)` действительных чисел (компонент), используется для трехмерного черчения. Компоненты тройки v обозначают `v.x, v.y` и `v.z`. Неявно тройка инициализируется как `(0.0,0.0,0.0)`.
string	Строка символов, реализованная с использованием STL класса `string` .
path	Кубический сплайн, представляющий путь. Неявным представителем пути является `nullpath`
guide	Нерешенный кубический сплайн (список узлов и контрольных точек кубического сплайна). Неявной инициализатор для `guide` есть `nullpath`. `guide` похож на путь за исключением того, что вычисление кубического сплайна откладывается до времени рисования (когда он будет преобразован в путь);
pen	Перо. В `Asymptote` перья обеспечивают стиль рисования для основных команд. Для изменения стиля указываются атрибуты пера. Перо, используемое в чертежных процедурах, называется `currentpen`. Если оно не указано явно, оно принимает значение `defaultpen`.
transform	Преобразование. К этому типу данных относятся частные виды аффинных преобразований.
picture	Рисунок, картинка. Рисунки являются структурами высокого уровня, определенными в модуле `plain` и являются основой рисования в пользовательских координатах. Рисунком, используемым по умолчанию, является `currentpicture`.
frame	Полотно (холст). Холсты являются основой для рисования в `PostScript` координатах.

2.1 Некоторые типы данных

2.1.1 Точки и комплексные числа

С помощью прямоугольной декартовой системы координат можно установить взаимно-однозначное соответствие между точками плоскости и комплексными числами. Тем, кто знаком с геометрическим истолкованием комплексных чисел, это известно.

Данные типа `pair` представляют собой упорядоченные пары `(x,y)` действительных чисел, поэтому на `pair z=(x, y)` можно смотреть как на точку с декартовыми координатами `(x, y)` или как на радиус-вектор или, что тоже самое, как на комплексное число.

Пары можно покомпонентно складывать, а также покомпонентно умножать на действительное число, как показано ниже.

1. `(a,b)+(c,d)=(a+c, b+d)`

   Легко показать, что для сложения выполняются переместительный и сочетательный законы. Можно говорить и об операции вычитания пар.

2. `r*(a,b)=(r*a, r*b)`

   Можно так же легко проверить, что выполняются два распределительных закона.

Если переменная `z` имеет тип `pair`, то первый элемент пары обозначается `z.x`, а второй `z.y`.

Рассмотрим два способа явного задания точек.

Способ первый.

Чтобы задать точку `z`, нужно сначала указать тип, затем имя переменной, знак равенства и саму пару, например `pair z=(2, -5);`

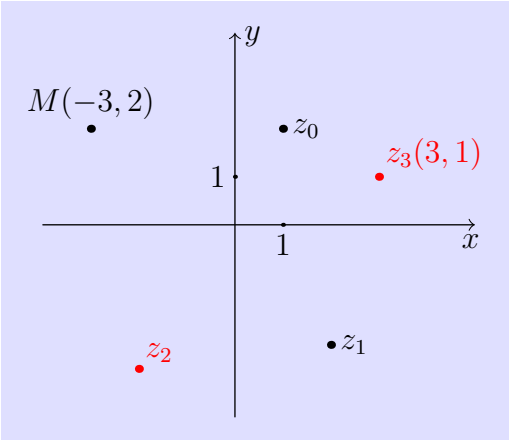
Пример 44

Зададим конкретную точку M и изобразим её:

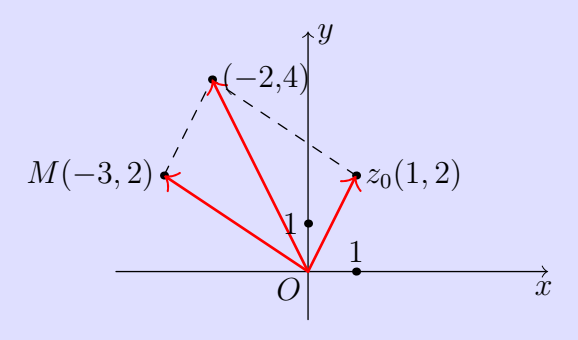
```
pair M=(-3, 2);
dot("$M(-3,2)$",M,N);
```

Для задания несколько точек используем одномерный массив. В фигурных скобках через запятую перечисляем элементы массива — пары, определяющие точки. Изобразить эти точки можно командами `dot` и `draw`.

```
pair [] z={(1,2),(2,-2.5),(-2,-3),(3,1)};
dot("$z_{0}$",z[0]); dot("$z_{1}$",z[1]);
draw("$z_{2}$", z[2],NE,3bp+red);
draw("$z_{3}(3,1)$", z[3],NE,3bp+red);
```



Пример 45



На картинке слева показано, что комплексные числа `z[0]` и `M` складываются по правилу параллелограмма.

```
draw((0,0)--z[0],bp+red,Arrow(TeXHead));
draw((0,0)--M,bp+red,Arrow(TeXHead));
pair K=z[0]+M;
dot("",K);
draw(M--K--z[0],dashed);
draw((0,0)--K,bp+red,Arrow(TeXHead));
```

Способ второй.

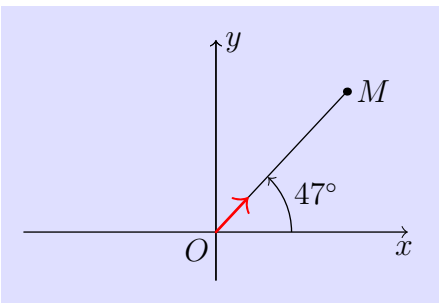
Используется принцип, лежащий в основе полярных координат: для задания точки M нужно указать направление на M и расстояние до неё. Направление указывается, например, с помощью функции `dir`. Действительный агумент `degrees` этой функции — величина угла в градусах между положительным лучом оси `Ox` и лучом OM.

Функция возвращает пару координат единичного вектора луча OM.

Пример 46

Зададим точку M, которая:
1) Лежит на луче h, который выходит из точки O под углом 47° к оси x.
2) Расстояние `|OM|=4cm`.

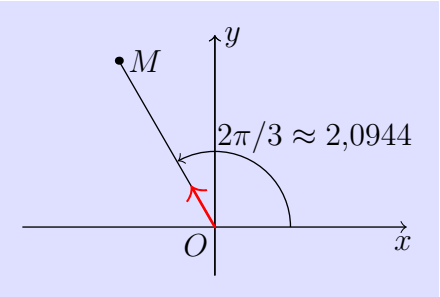
```
unitsize(1cm);
real r=4;
pair M=r*dir(47);
dot("$M$",M);
draw((0,0)--dir(47),bp+red,Arrow(TeXHead));
```



Для указания направления, кроме функции `dir`, можно воспользоваться функцией `expi`. Действительный агумент `angle` этой функции — величина угла в радианах между положительным направлением оси `Ox` и лучом OM.

Функция `expi` возвращает пару координат единичного вектора луча OM.

Пример 47



Например, величина угла задана в радианной мере: $\frac{2\pi}{3} = 120°$, расстояние `|OM|=4cm`.

```
unitsize(1cm);
real r=4;
real a=2*pi/3;
pair M=r*expi(a);
dot("$M$",M);
draw((0,0)--expi(a)),bp+red,Arrow(TeXHead));
```

Перечислим ещё встроенные функции для пар, включая уже отмеченные `dir` и `expi`:

- `pair conj(pair z)`
  возвращает сопряженное (conjugate) к `z`;
- `real length(pair z)`
  возвращает модуль $|z|$ комплексного числа z.
  Пример: `pair z=(3,4);` Функция `length(z);` возвращает результат 5.
  Синоним `length(pair)` — `abs(pair)`;
- `real angle(pair z, bool warn=true)`
  возвращает аргумент `z` в радианах из интервала $[-\pi, \pi]$ или 0, если `warn` есть `false` и `z=(0,0)` (вычисление приводит к ошибке);
- `real degrees(pair z, bool warn=true)`
  возвращает аргумент `z` в градусах из интервала `[0,360)` или `0`, если `warn` есть `false` и `z=(0,0)` (вычисление приводит к ошибке);
- `pair unit(pair z)`
  возвращает единичный вектор направления `z`;
- `pair expi(real angle)`
  возвращает единичный вектор направления `angle`, заданного в радианах;
- `pair dir(real degrees)`
  возвращает единичный вектор направления `degrees`, заданного в градусах;
- `real xpart(pair z)`
  возвращает `z.x`;
- `real ypart(pair z)`
  возвращает `z.y`;
- `pair realmult(pair z, pair w)`
  возвращает `(z.x*w.x,z.y*w.y)` — поэлементное произведение пар;
- `real dot(explicit pair z, explicit pair w)`
  возвращает скалярное произведение `z.x*w.x+z.y*w.y`;

- `pair minbound(pair z, pair w)`

  возвращает (`min(z.x,w.x),min(z.y,w.y)`);

- `pair maxbound(pair z, pair w)`

  возвращает (`max(z.x,w.x),max(z.y,w.y)`).

2.1.2 Пути. Основы работы

Путем в `Asymptote` называют некоторую линию, созданную определенным образом посредством указанием точек, через которые она проходит, и способов их соединения. Существуют также предопределенные пути, такие как окружность, квадрат, прямая, отрезок и т.п.

Путь (`path`) в `Asymptote` — это кусочно-кубическая функция параметра t. Точки могут рассматриваться как пути нулевой длины.

Операторы соединения путей

Самый простой способ создать новый путь, это соединить точки или пути следующими операторами:

```
p--q
p..q
p^^q,
```

где p и q — пути.

В конце подраздела рассматриваются еще два способа соединения путей:

```
p---q
p & q.
```

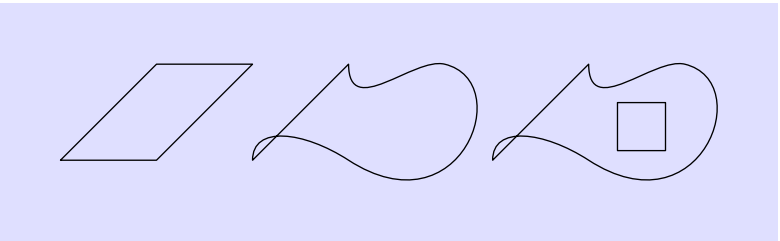
Первый из перечисленных операторов (--) соединяет конец пути p с началом пути q прямой линией. Второй из указанных операторов (..) плавно соединяет их кубическим сплайном.

Третий оператор (^^) перепараметризует эти два пути так, что они рассматриваются как один. Символ цикла(`cycle`) приказывает Асимптоте завершить путь переходом от конечной точки к начальной.

Рассмотрим примеры, разъясняющие сказанное. Начнем с примера, который включает в себя особенности всех трех первых типов соединения путей:

Пример 48

```
unitsize(.5inch);
path T,ct,tt;
T=(0,0)--(1,0)--(2,1)--(1,1)--cycle;
ct=(0,0){up}..(1,0)..(2,1)..{up}(1,1)--cycle;
tt=scale(.5)*unitsquare;
draw(T);
draw(shift(2*right)*ct);
draw((shift(4.5*right)*ct)^^(shift(5.8,.1)*tt));
```



Настройка параметров `direction`, `tension`, `curl`

Пользователь может изменить параметры `path` (или `guide`), указав значения `direction` (направление), `tension` (натяжение) и `curl` (изгиб) .

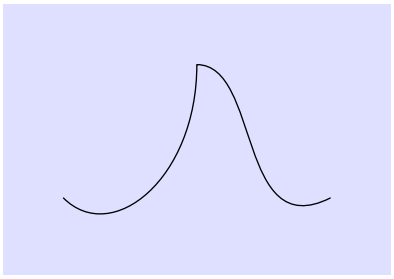
Первый параметр, `direction`, задает направление касательной в заданной точке гладкой кривой. Значение `direction` указывают в фигурных скобках.

В зависимиости от того, стоит ли указатель направления перед точкой или после точки, указывается направление левой или правой касательной.

Пример 49

В этом примере направление указывается с помощью задания вектора касательной. Для этого задается пара (a,b) координат этого вектора и указывается в фигурных скобках перед или после узла пути.

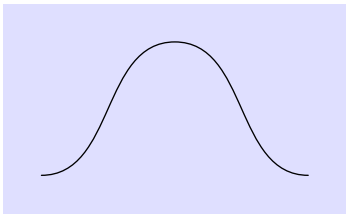
```
draw((0,0){(1,-1)}..{(0,1)}(50,50){1,0}..{(1,.5)}(100,0));
shipout(bbox(5mm,Fill(paleblue+white)));
```



Пример 50

Вместо указания вектора `(1,0)` можно в фигурных скобках написать просто `right`.

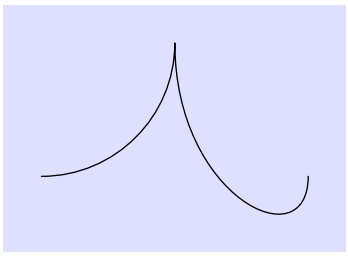
```
draw((0,0){right}..{right}(50,50)..{right}(100,0));
```



Пример 51

Аналогично употребляются слова `left, up, down`.

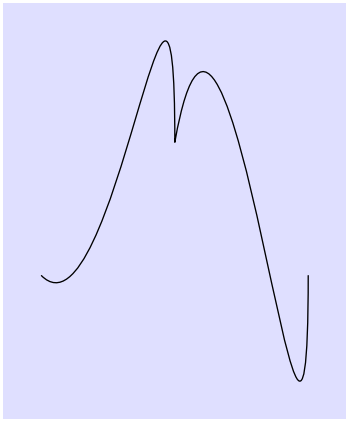
```
draw((0,0){right}..{up}(50,50){down}..{up}(100,0));
```



Пример 52

Направление касательной можно задать в градусной мере, указав в фигурных скобках `dir()`, а внутри круглых — нужное число градусов.

```
draw((0,0){dir(-45)}..{dir(-90)}(50,50){dir(80)}..{dir(90)}(100,0));
```

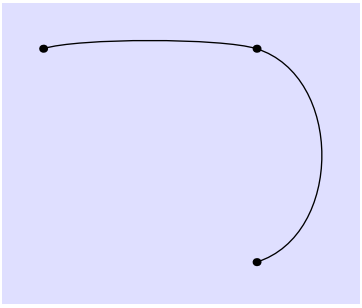


Параметр `tension` определенным образом меняет кривизну линии между двумя точками. Чем больше натяжение (`tension`) линии, тем больше она по виду приближается к прямой линии. Можно заменить натяжение сплайна (его значение по умолчанию 1) на любое действительное значение, большее или равное 0.75.

Пример 53

В этом примере между точками указано ровно по одному значению параметра `tension`:

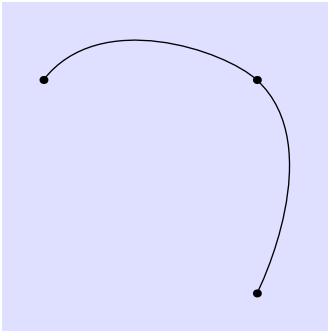
```
draw((100,0)..tension 1.2 ..(100,100)..tension 2 ..(0,100));
dot((100,100));dot((100,0));dot((0,100));
```



Пример 54

В этом примере между точками указано два значения параметра `tension`, поэтому кривизна линии между точками неодинакова.

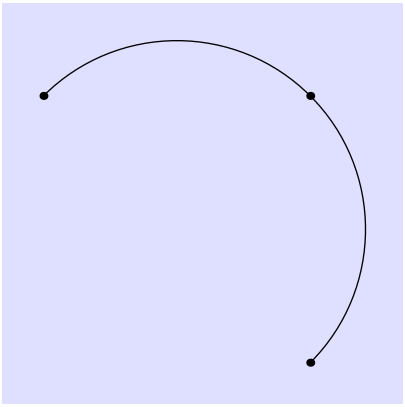
```
dot((100,100));dot((100,0));dot((0,100));
draw((100,0)..tension 25 and .8 ..(100,100)..tension 2 and .9 ..(0,100));
```



Пример 55

Значение `tension atleast 1` означает приблизительно 1. Поэтому это значение дает такой же эффект, как значение по умолчанию, равное 1.

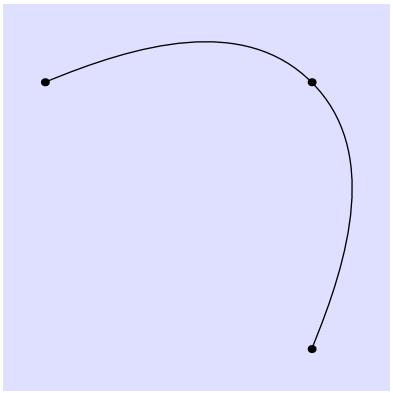
```
draw((100,0)..tension atleast 1 ..(100,100)..(0,100));
```



Параметр `curl` определяет кривизну в конечных точках пути (0 означает прямую; по умолчанию значение 1 означает примерно дугу окружности):

Пример 56

```
draw((100,0){curl 0}..(100,100)..{curl 0}(0,100));
```



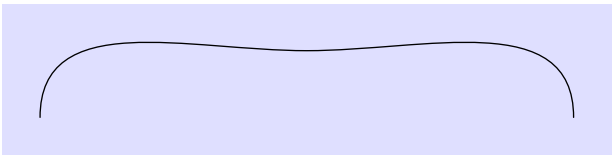
В `MetaPost` есть оператор ... соединения путей, который позволяет, если возможно, без точек перегиба, заключить кривую в треугольник, определенный конечными точками и направлениями.

В Асимптоте этот оператор реализуется удобной аббревиатурой `::` взамен более длинного обозначения — `..tension atleast 1..` (многоточие ... употребляется в Асимптоте, чтобы указать число переменных аргументов).

Пример 57

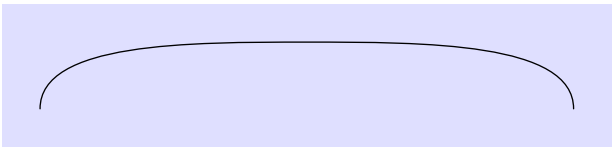
Например, сравните результат выполнения кода

```
draw((0,0){up}..(100,25){right}..(200,0){down});
```



с результатом выполнения кода

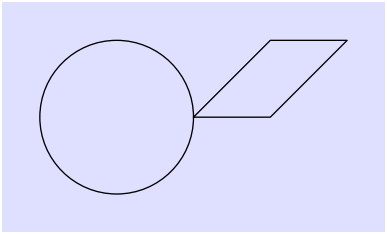
```
draw((0,0){up}::(100,25){right}::(200,0){down});
```



Символ соединения `---` является аббревиатурой для `.. tension atleast infinity ..` и соединение & объединяет два пути, предварительно сняв последний узел первого пути (который, как правило, должен совпадать с первым узлом второго пути)

Пример 58

```
unitsize(.4inch);
path T,tt;
T=(0,0)--(1,0)--(2,1)--(1,1)--cycle;
tt=shift(-1,0)*unitcircle;
draw(tt&T);
shipout(bbox(5mm));
```



2.1.3 Перья

В `Asymptote` перья обеспечивают процесс выполнения четырех основных команд рисования. В процессе рисования они реализуют следующие свойства: цвет (`color`), тип линии (`line type`), толщину линии (`line width`), форму концов (`line cap`), тип соединения (`line join`), правило заполнения (заливки) (`fill rule`), способ выравнивания текста (`text alignment`), вид шрифта (`font`), размер шрифта (`font size`), штриховка (`pattern`), режим замены(`overwrite mode`) и каллиграфические преобразования острия пера (`calligraphic transforms on the pen nib`).

Перо (`pen`), используемое по умолчанию в чертежных процедурах, называется `currentpen`. Это обеспечивает функциональность, аналогичную команде `pickup` в `MetaPost`.

Цвет

Цвет по умолчанию — `black`; его можно изменить с помощью процедуры `defaultpen (pen)`. Функция `colorspace(pen p)` возвращает цвет пера `p` в виде строки

```
("gray", "rgb", "cmyk" или "").
```

Функция `real[] colors(pen)` возвращает компоненты цвета пера. Такие функции, как `pen gray(pen)`, `pen rgb(pen)` и `pen cmyk(pen)` возвращают новые перья, полученные путем преобразования их аргументов в соответствующие цветовые пространства.

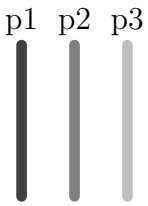
Цвета задаются с использованием одного из следующих цветовых пространств:

- `pen gray(real g);`

  Дает различные оттенки серого цвета; интенсивность `g` лежит в интервале [0,1], при этом действительное число 0.0 обозначает черный и 1.0 — белый цвет.

Пример 59

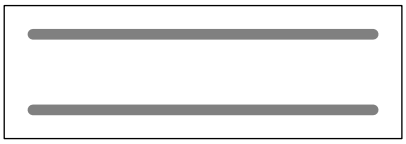
```
unitsize(1cm);
pen p1=gray(.25);
pen p2=gray(.5);
pen p3=gray(.75);
draw((0,0)--(4.5,0),p1+4bp);
draw((0,1)--(4.5,1),p2+4bp);
draw((0,2)--(4.5,2),p3+4bp);
```



Пример 60

Заметим, что предопределенный цвет `gray` соответствует перу `pen p=gray(.5)`

```
unitsize(1cm);
pen p=gray(.5); // Задаем текущее перо
draw((0,0)--(4.5,0),4bp+p); // Указываем его, добавляя с помощью оператора +
draw((0,1)--(4.5,1),4bp+gray); // Указываем предопределенное перо gray
shipout(bbox(3mm));
```

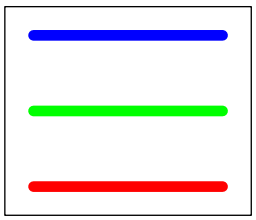


- `pen rgb(real r, real g, real b);`

  Перо производит цвета пространства `RGB`, где каждый параметр `real r`, `real g`, `real b` является действительным числом в из интервала [0,1], определяющим интенсивность красного (red), зеленого (green) и синего (blue) цвета.

Пример 61

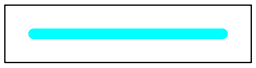
```
unitsize(1cm);
pen p1=rgb(1,0,0);pen p2=rgb(0,1,0);pen p3=rgb(0,0,1);
draw((0,0)--(2.5,0),p1+4bp);// Соответствует предопределенному значению  red
draw((0,1)--(2.5,1),p2+4bp);// Соответствует предопределенному значению   green
draw((0,2)--(2.5,2),p3+4bp);// Соответствует предопределенному значению  blue
shipout(bbox(3mm));
```



Эти три основных цвета можно сочетать, получая другие цвета, в том числе и предопределенные. Приведем несколько примеров.

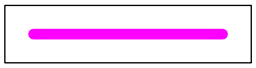
Пример 62

```
unitsize(1cm);
pen p=rgb(0,1,1); // иначе green+blue
draw((0,0)--(2.5,0),p+4bp);// Соответствует предопределенному значению cyan
shipout(bbox(3mm));
```



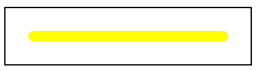
Пример 63

```
unitsize(1cm);
pen p=rgb(1,0,1); // иначе red+blue
draw((0,0)--(2.5,0),p+4bp);// Соответствует предопределенному значению
                           // magenta
shipout(bbox(3mm));
```



Пример 64

```
unitsize(1cm);
pen p=rgb(1,1,0); // иначе red+green
draw((0,0)--(2.5,0),p+4bp);// Соответствует предопределенному значению
                           // yellow
shipout(bbox(3mm));
```



- `pen cmyk(real c, real m, real y, real k);`

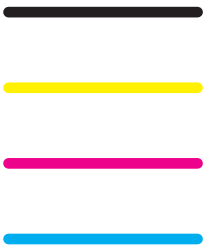
  Перо производит цвета CMYK, где каждый параметр c(cyan), m(magenta), y(yellow), k(black) является числом в интервале [0,1], определяющим интенсивность голубого, пурпурного, желтого, черного цвета.

Приведем несколько примеров.

Пример 65

Пример иллюстрирует четыре базовых цвета.

```
unitsize(1cm);
pen p1=cmyk(1,0,0,0); pen p2=cmyk(0,1,0,0);
pen p3=cmyk(0,0,1,0); pen p4=cmyk(0,0,0,1);
draw((0,0)--(2.5,0),p1+4bp);// Соответствует предопределенному
                                          //значению cyan (нижняя)
draw((0,1)--(2.5,1),p2+4bp);// Соответствует предопределенному
                                          //значению magenta (вторая снизу)
draw((0,2)--(2.5,2),p3+4bp);// Соответствует предопределенному
                                          //значению yellow (третья снизу)
draw((0,3)--(2.5,3),p4+4bp);// Соответствует предопределенному
                                          //значению black (верхняя)
```



- `pen invisible;`

  Это специальное перо пишет невидимыми чернилами, но настраивает ограничивающий бокс, как если что-то было нарисовано (подобно команде \phantom в TEX). Функция `bool invisible(pen)` может быть использована для проверки, является ли перо невидимым.

Пример 66

В примере показано, как убрать рамку с помощью `pen invisible;`

```
unitsize(1cm);
pen p1=gray(.25);
draw((0,0)--(4,0),p1+4bp);
shipout(bbox(3mm+invisible));
```


Тип линии

Типы линий можно указать с помощью функции

`pen linetype(real[] a, real offset=0, bool scale=true, bool adjust=true)`,

в которой переменная `a` обозначает массив действительных чисел. Первое число массива указывает размер штриха (если scale есть true, в единицах ширины линии пера, в противном случае в PostScript-единицах), второе число массива определяет длину пробела, и так далее.

Если adjust есть true, эти расстояния автоматически регулируются `Asymptote`, чтобы соответствовать длине дуги пути. Необязательный параметр `offset` определяет смещение в шаблоне штриховки. Таким образом можно определить "рисунок" пунктира

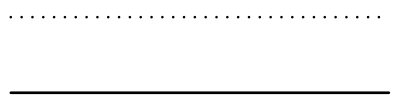
Существуют предопределенные типы линий:

```
pen solid=linetype(new real[]); // Сплошная линия

pen dotted=linetype(new real[] {0,4}); // Линия, состоящая из точек

pen dashed=linetype(new real[] {8,8}); // Пунктирная линия

pen longdashed=linetype(new real[] {24,8}); // Длинный пунктир

pen dashdotted=linetype(new real[] {8,8,0,8});

pen longdashdotted=linetype(new real[] {24,8,0,8});
```

Приведем ряд примеров для иллюстрации сказанного.

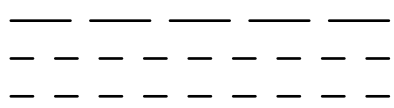
Пример 67

```
unitsize(1cm);
pen p1=linetype(new real[]); // Определяем тип solid
pen p2=linetype(new real[] {0,4}); // Определяем тип dotted
pen p3=linewidth(1); // Определяем толщину линии
draw((0,0)--(5,0),p1+p3); // Чертим сплошную линию (нижняя)
draw((0,1)--(5,1),p2+p3); // Чертим линию из точек (верхняя)
```



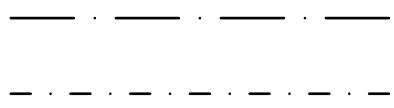
Пример 68

```
unitsize(1cm);
pen p1=linetype(new real[]{8,8}); // Пунктирная линия
pen p2=linetype(new real[] {24,8}); // Длинный пунктир
pen p3=linewidth(1); // Определяем толщину линии
draw((0,0)--(5,0),p1+p3); // Чертим пунктирную линию (нижняя), иначе
draw((0,.5)--(5,.5),dashed+p3);
draw((0,1)--(5,1),p2+p3); // Чертим длинный пунктир (верхняя)
```



Пример 69

```
unitsize(1cm);
pen p1=linetype(new real[]{8,8,0,8}); // Чередование пунктира и точек
pen p2=linetype(new real[] {24,8,0,8}); // Чередование длинного пунктира и точек
pen p3=linewidth(1);
draw((0,0)--(5,0),p1+p3);  // Нижняя линия
draw((0,1)--(5,1),p2+p3); // Верхняя линия
```



По умолчанию употребляется тип `solid` (сплошная линия), но его можно изменить с помощью команды `defaultpen (pen)`. Тип линии пера может быть определен функциями

`real[] linetype(pen p=currentpen), real offset(pen p), bool scale(pen p),`
и `bool adjust(pen p)`.

Толщина линии

Толщина линии в `Asymptote` определяется толщиной пера и указывается в PostScript-единицах `pen linewidth(real)`. По умолчанию толщина линии составляет `0.5 bp`; это значение может быть изменено с помощью команды `defaultpen(pen)`. Толщина линии возвращается функцией

`real linewidth(pen p=currentpen)`.

Для удобства в модуле `plain` определяется:

- `static void defaultpen(real w) defaultpen(linewidth(w));`
- `static pen operator +(pen p, real w) return p+linewidth(w);`
- `static pen operator +(real w, pen p) return linewidth(w)+p;`

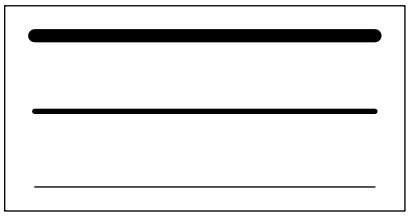
так что можно определить толщину линии примерно так:

`defaultpen(2); pen p=red+0.5;`

Пример 70

В этом примере показано использование пера по умолчанию (первая линия снизу) и пера с явно указанной толщиной:

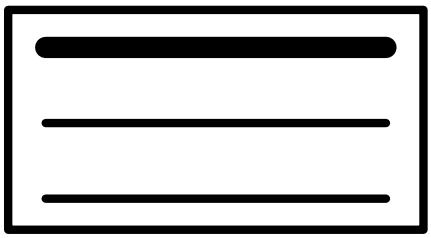
```
unitsize(1cm);
draw((0,0)--(4.5,0));//используется перо по умолчанию толщиной 0.5bp (нижняя линия)
draw((0,2)--(4.5,2),linewidth(1.75mm));// явно указываем толщину в \texttt{mm}
draw((0,1)--(4.5,1),linewidth(2bp));//явно указываем толщину в \texttt{bp}
shipout(bbox(3mm));
```



Пример 71

Пример, в котором мы переопределяем перо по умолчанию

```
unitsize(1cm);
real w=3.14;// Задаем действительное число
defaultpen(w);// Переопределяем толщину пера по умолчанию
draw((0,0)--(4.5,0));// Используем вновь определенное перо по умолчанию
draw((0,1)--(4.5,1),linewidth(w));// Явно указываем толщину пера
draw((0,2)--(4.5,2),linewidth(2.54*w));// Увеличиваем указанную толщину в 2.54 раза
shipout(bbox(3mm));
```



Смотрим на полученный результат и делаем вывод, что

1) Поскольку перо по умолчанию и явно указанное используют одно и то же число w, результаты не отличаются.

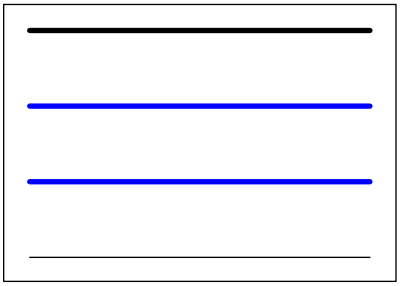
2) Рамка начерчена пером по умолчанию, поэтому получилась толще, чем нужно.

3) Толщину линии можно увеличивать, просто умножая w на числовой действительный множитель.

Пример 72

Толщину пера можно задать, добавив необходимое значение `w` к определенному перу `p` с помощью оператора +:

```
unitsize(1cm);
real w=2;
pen p=blue;// Определяем перо синего цвета
draw((0,0)--(4.5,0));// Используем перо по умолчанию
draw((0,1)--(4.5,1),p+w);// Применяем оператор +, задавая и цвет и толщину w
draw((0,2)--(4.5,2),w+p);// Оператор + коммутативен
draw((0,3)--(4.5,3),linewidth(2bp));// Явно задаем толщину
shipout(bbox(3mm));
```



Другие атрибуты пера

В Асимптоте концы линий могут иметь различный вид. Этот вид задается пером с атрибутом `linecap`. Перо с конкретным видом конца в `PostScript` возвращается на вызове `linecap` с целым аргументом:

```
pen squarecap=linecap(0);
pen roundcap=linecap(1);
pen extendcap=linecap(2);
```

По умолчанию конец линии есть `roundcap`. Вид конца линии может быть изменен с помощью команды `defaultpen(pen)`.

Различия в очертаниях концов можно увидеть из следующего примера:

Пример 73

```
unitsize(1cm);
pen p=6bp+blue;// Определяем перо синего цвета
draw((0,0)--(2.5,0),p+linecap(0));// перо с концом squarecap (первое снизу)
draw((0,.5)--(2.5,.5),p+linecap(2));// перо с концом extendcap (второе снизу)
draw((0,1)--(2.5,1),p+linecap(1));// перо с концом roundcap (второе сверху)
draw((0,1.5)--(2.5,1.5),p);// перо с концом по умолчанию (первое сверху)
```

Если нам нужно, чтобы линии имели только один тип конца, можно просто переопределить перо. Это показано в следующем примере:

Пример 74

```
unitsize(1cm);
defaultpen(6bp+blue+linecap(2));
pen p=6bp+blue;// Определяем перо синего цвета
draw((0,0)--(4.5,0),p+linecap(2));// Используем перо с концом extendcap
draw((0,.5)--(4.5,.5));// Используем перо с концом по умолчанию
```

Об оставшихся атрибутах пера, таких как: тип соединения (`line join`), правило заполнения (заливки) (`fill rule`), выравнивание текста (`text alignment`), вид шрифта (`font`), размер шрифта (`font size`), штриховка (`pattern`), режим замены (`overwrite mode`) и каллиграфические преобразования острия пера (`calligraphic transforms on the pen nib`) можно прочитать в авторском руководстве [9].

2.1.4 Преобразования

Термин преобразование (`transform`) в `Asymptote` понимается как аффинное преобразование. Для определения преобразования t нужно задать шесть чисел: $t = (t.x, t.y, t.xx, t.xy, t.yx, t.yy)$.

На языке математики это означает, что пара (x, y) преобразуется преобразованием t в пару (x', y'), где

$$\begin{cases} x' &= t.x + t.xx * x + t.xy * y \\ y' &= t.y + t.yx * x + t.yy * y \end{cases}$$

Преобразования можно применять к таким типам данных, как `pair`, `guides`, `paths`, `pens`, `strings`, `transforms, frames` и `pictures`. Для этого служит бинарный оператор (*).

Можно создавать композиции преобразований и находить обратное преобразование с помощью функции `transform inverse(transform t)`; их также можно возводить в степень с целым показателем с помощью оператора `^`.

Заметим, что мы рассматриваем здесь только самые простые примеры применения преобразований. Для углубленного изучения рекомендуем познакомиться с ресурсами

http:www.piprime.fr/asymptote/ и http://asy.marris.fr/asymptote/

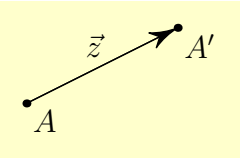
Перечислим следующие частные виды преобразований:

1. `transform identity();`
   тождественное преобразование;
2. `transform shift(pair z);`
   параллельный перенос на вектор `z`;
3. `transform shift(real x, real y);`
   параллельный перенос на вектор `(x,y)`
4. `transform xscale(real x);`
   растяжение в число `x` в направлении оси `x`;
5. `transform yscale(real y);`
   растяжение в число `y` в направлении оси `y`;
6. `transform scale(real s);`
   растяжение в число `s` в обоих направлениях;
7. `transform scale(real x, real y);`
   растяжение в число `x` в направлении оси `x` и одновременно растяжение в число `y` в направлении оси `y`;
8. `transform slant(real s);`
   отображение $(x, y) \to (x + s * y, y)$;
9. `transform rotate(real angle, pair z=(0,0));`
   поворот вокруг точки `z` на угол `angle` в градусах
10. `transform reflect(pair a, pair b);`
    симметрия относительно прямой `a--b`.

Начнем с простых примеров. Проще всего рассмотреть действие преобразований на точки.

Пример 75

Рассмотрим картинку, которая иллюстрирует параллельный перенос точки A на вектор $\vec{z}(2, 1)$. Образ точки A обозначим A'.



Эту картинку можно нарисовать различным кодом. Первый код использует `shift(z)`

```
unitsize(1cm);
pair A=(3,0), z=(2,1), B=shift(z)*A;//Задаем точку A, вектор переноса z и
//точку B как образ точки A при переносе z.
path g=A--B;//Определим путь из A в B.
dot(Label("$A$",A,SE));//Рисуем точку A.
dot(Label("$A^{\prime}$",B,SE));//Рисуем образ точки A.
draw(g,Arrow(HookHead));//Рисуем вектор переноса.
draw(Label("$\vec{z}$",align=LeftSide),g);//Обозначаем его z
```

Второй код использует `shift((x,y))`

```
unitsize(1cm);
pair A=(3,0);//Задаем точку A.
dot(Label("$A$",A,align=SE));//Рисуем точку A.
dot(shift((2,1))*A);//Рисуем образ точки A.
label("$A^{\prime}$",shift((2,1))*A,SE);//Обозначаем образ как A'.
draw(A--shift((2,1))*A,Arrow(HookHead));//Рисуем вектор переноса.
draw(Label("$\vec{z}$",align=LeftSide),A--shift((2,1))*A);//Обозначаем его z
```

Третий вариант кода такой:

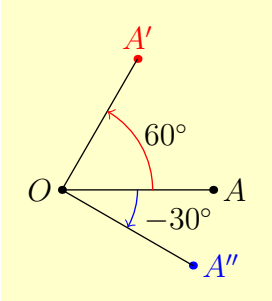
```
unitsize(1cm);
pair A=(3,0),z=(2,1);//Задаем точку A и вектор z.
transform t=shift((2,1));//Задаем преобразование t.
dot(Label("$A$",A,align=SE));//Рисуем точку A.
dot(Label("$A^{\prime}$",t*A,SE));//Рисуем образ точки A.
draw(A--t*A,Arrow(HookHead));//Рисуем вектор переноса.
draw(Label("$\vec{z}$",align=LeftSide),A--t*A);//Обозначаем его z
```

Рассмотрим преобразование, называемое поворотом или вращением, причем рассмотрим сначала поворот точки вокруг начала координат.

Пример 76

В этом примере первый поворот совершается вокруг точки $O(0,0)$ на угол $60°$, причем точка A преобразуется в точку A'. При втором повороте, который совершается вокруг точки $O(0,0)$ на угол $-30°$, точка A преобразуется в точку A''.

Приведем рисунок и затем код к рисунку.

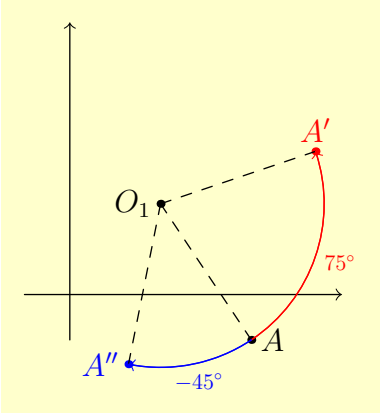



```
unitsize(1cm);// задаем единицу измерения
real a=60, b=-30;//задаем углы поворота в градусах
pair O=(0,0), A=(2,0);// задаем центр поворота z и точку A.
```

```
transform r1=rotate(a,O);//задаем поворот на 60 градусов(против часовой стрелки)
transform r2=rotate(b,O);//задаем поворот на 30 градусов(по часовой стрелке)
dot("$O$",(0,0),W); dot("$A$",A,E);//рисуем точки O и A
draw("$A^{\prime}$",r1*A,N,3bp+red);//рисуем образ A при повороте на 60 градусов
draw("$A^{\prime\prime}$",r2*A,E,3bp+blue);/*рисуем образ A при повороте на -30
градусов. Далее в последующих строках изображаем обозначения углов поворота */
draw(O--A); draw(O--r1*A); draw(O--r2*A);
path p= scale(1.2)*(dir(A){up}..dir(r1*A));
draw(p,red,Arrow(TeXHead)); draw(dir(A){down}..dir(r2*A),blue,Arrow(TeXHead));
label("$60^{\circ}$",(1,.5),NE); label("$-30^{\circ}$",(1,-.15),SE);
shipout(bbox(3mm,Fill(yellow+.8*blue)));
```

Пример 77

В примере поворот делается вокруг точки O_1 на углы $75°$ и $-45°$. Вот картинка:



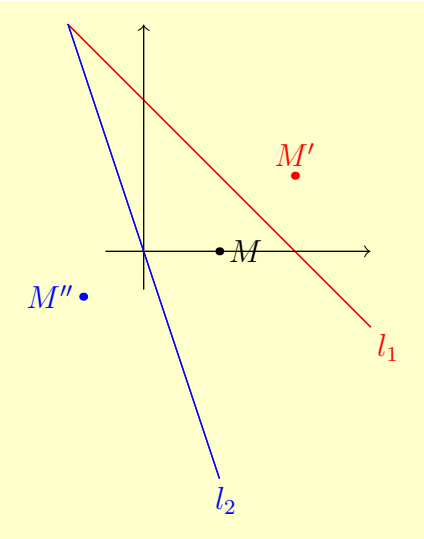
Приведем код к ней.

```
unitsize(1cm);// задаем единицу измерения
real a=75, b=-45;//задаем углы поворота в градусах
pair O1=(1,1), A=(2,-.5);// задаем центр поворота и точку A
transform r1=rotate(a,O1);//задаем поворот на 75 градусов(против часовой стрелки)
transform r2=rotate(b,O1);//задаем поворот на 45 градусов(по часовой стрелке)
dot("$O_{1}$",(1,1),W); dot("$A$",A,E);//рисуем центр поворота и точку A
draw("$A^{\prime}$",r1*A,N,3bp+red);//рисуем образ A при повороте на 75 градусов
draw("$A^{\prime\prime}$",r2*A,W,3bp+blue);//рисуем образ A при повороте на -45
//градусов
draw((-.5,0)--(3,0),Arrow(TeXHead));//рисуем первую координатную ось
draw((0,-.5)--(0,3),Arrow(TeXHead));//рисуем вторую координатную ось
//Далее в последующих строках изображаем
//обозначения углов поворота
draw(O1--A,dashed); draw(O1--r1*A,dashed); draw(O1--r2*A,dashed);
path p=A{dir(rotate(90)*(A-O1))}..r1*A;
path q=A{dir(rotate(-90)*(A-O1))}..r2*A;
draw(p,red,Arrow(TeXHead));
draw(scale(.7)*Label("$75^{\circ}$"),align=RightSide,p,red);
draw(q,blue,Arrow(TeXHead));
draw(scale(.7)*Label("$-45^{\circ}$"),align=LeftSide,q,blue);
```

Далее рассмотрим осевую симметрию.

Пример 78

В примере заданы две оси симметрии, красная и синяя. Произвольная точка M отображается относительно красной оси в точку M', а относительно синей оси в точку M''.



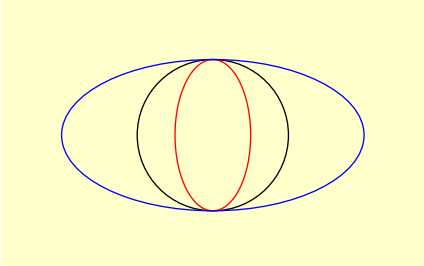
Приведем код к рисунку:

```
unitsize(1cm);// задаем единицу измерения
pair O=(0,0), A=(-1,3), B=(3,-1), C=(1,-3), M=(1,0);// задаем точки O, A, B, C и M.
transform r1=reflect(A,B);//задаем симметрию относительно прямой AB
transform r2=reflect(O,A);//задаем симметрию относительно прямой OA
dot("$M$",M,E);//рисуем точку M
path a=A--B,b=A--C;//определяем оси симметрии
draw(A--B,red); draw(A--C,blue);// рисуем оси симметрии
draw(Label("$l_{1}$",EndPoint),a,red);// обозначаем первую ось
draw(Label("$l_{2}$",EndPoint),b,blue);//обозначаем вторую ось
draw("$M^{\prime}$",r1*M,N,3bp+red);//рисуем образ M при симметрии
//относительно прямой AB
draw("$M^{\prime\prime}$",r2*M,W,3bp+blue);//рисуем образ M при симметрии
//относительно прямой OA
draw((-.5,0)--(3,0),Arrow(TeXHead));//рисуем первую координатную ось
draw((0,-.5)--(0,3),Arrow(TeXHead));//рисуем вторую координатную ось
```

Рассмотрим преобразования растяжения и сжатия вдоль осей X и Y. Обратите внимание на значения коэффициентов a и b

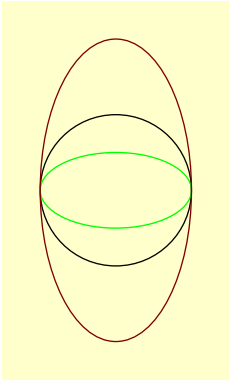
Пример 79

```
unitsize(1cm);// задаем единицу измерения
real a=0.5; real b=2;//задаем числа (коэффициенты )
path p=unitcircle;//задаем единичную окружность как путь
transform t1=xscale(a);//задаем сжатие вдоль оси x (a<1)
transform t2=xscale(b);//задаем растяжение вдоль оси x (b>1)
draw(p);//рисуем единичную окружность
draw(t1*p,red);//рисуем образ окружности при сжатии вдоль оси x
draw(t2*p,blue);//рисуем образ окружности при растяжении вдоль оси x
```



Пример 80

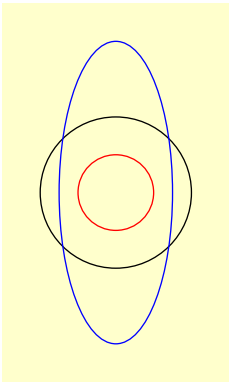
```
unitsize(1cm);// задаем единицу измерения
real a=0.5; real b=2;//задаем число
path p=unitcircle;//задаем единичную окружность как путь
transform s1=yscale(a);//задаем сжатие вдоль оси y (a<1)
transform s2=yscale(b);//задаем растяжение вдоль оси y (b>1)
draw(p);//рисуем единичную окружность
draw(s1*p,green);//рисуем образ окружности при сжатии вдоль оси y
draw(s2*p,brown);//рисуем образ окружности при растяжении вдоль оси y
```



В следующем примере показано действие преобразования сжатия в обоих направлениях с одним и тем же коэффициентом (красный образ) и различными коэффициентами вдоль x и вдоль y (синий образ).

Пример 81

```
unitsize(1cm);// задаем единицу измерения
real a=.5, b=2, c=.75;//задаем действительные числа, коэффициенты
path p=unitcircle;//задаем единичную окружность как путь
transform r1=scale(a);//задаем сжатие
transform r2=scale(c,b);//задаем сжатие вдоль оси x и растяжение вдоль оси y
draw(p);//рисуем единичную окружность
draw(r1*p,red);//рисуем образ единичной окружности при сжатии
draw(r2*p,blue);/*рисуем образ единичной окружности при сжатии вдоль оси x и
растяжении вдоль оси y */
```



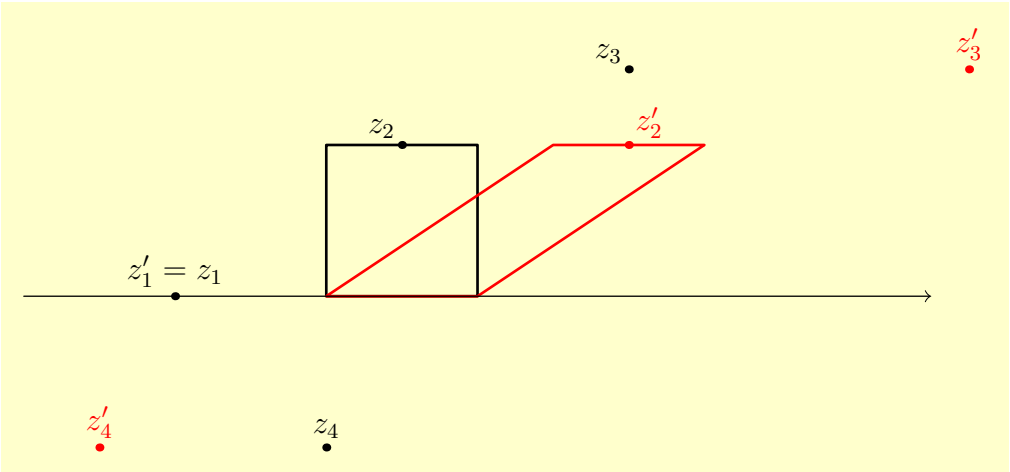
Последнее преобразование называется сдвигом вдоль оси x. Это частный случай сдвига вдоль прямой и математически определяется формулой

$$(x, y) \to (x + s * y, y)$$

в которой есть параметр s — действительное число. Приведем пример.

Пример 82

```
unitsize(2cm);// задаем единицу измерения
real s=1.5;//задаем действительное число
pair z1=(-1,0), z2=(.5,1), z3=(2,1.5), z4=(0,-1);// задаем точки z1, z2, z3, z4
path p=unitsquare;//задаем единичный квадрат как путь
transform sl=slant(s);//задаем сдвиг
//затем рисуем точки
dot("$z_{2}$",z2,NW); dot("$z_{3}$",(2,1.5),NW); dot("$z_{4}$",(0,-1),N);
draw(p,linewidth(1bp));//рисуем единичный квадрат
draw(sl*p,bp+red);//рисуем образ квадрата при сдвиге sl
draw(sl*z1);//рисуем образ точки z1 при сдвиге sl
draw(sl*z2,red);//рисуем образ точки z2 при сдвиге sl
draw(sl*z3,red);//рисуем образ точки z3 при сдвиге sl
draw(sl*z4,red);//рисуем образ точки z4 при сдвиге sl
//далее рисуем обозначения образов
dot("$z_{1}^{\prime}=z_{1}$",sl*z1,N);
dot("$z_{2}^{\prime}$",sl*z2,NE,red);
dot("$z_{3}^{\prime}$",sl*z3,N,red);
dot("$z_{4}^{\prime}$",sl*z4,N,red);
draw((-2,0)--(4,0),Arrow(TeXHead));//рисуем координатную ось x
```



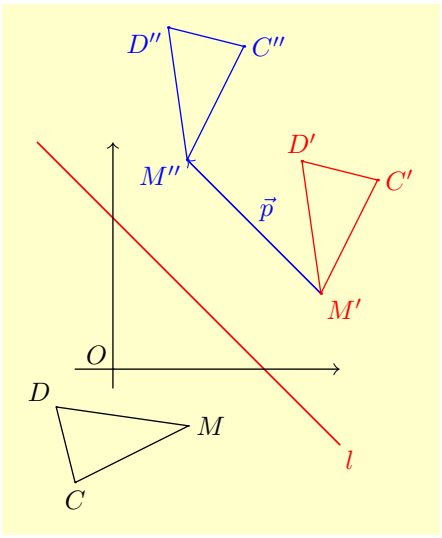
Вот несколько примеров действия преобразований на замкнутый путь `path`, в качестве которого выступает треугольник.

Пример 83

Зададим и изобразим треугольник DMC. Поставим цель изобразить образ треугольника DMC при скользящей симметрии, которая определена осью симметрии l и вектором $\vec{p}$, параллельным этой оси. Рассмотрим следующий код:

```
unitsize(1cm);// задаем единицу измерения
defaultpen(fontsize(10pt));//определяем размер шрифта по умолчанию
pair O=(0,0), A=(-1,3), B=(3,-1), C=(-.5,-1.5), D=(-.75,-.5), M=(1,-.75);
// задаем точки O, A, B, C, D, M
transform rf=reflect(A,B);//задаем симметрию относительно прямой AB,
//обозначая ее l
transform sh=shift(2.5dir(A-B));//задаем параллельный перенос на
//вектор, параллельный оси симметрии l
draw("$M$",M,E); draw("$O$",O,NW); draw("$C$",C,S); draw("$D$",D,NW);
```

```
//рисуем точки M, O, C, D
path h=A--B,g=D--M--C--cycle;//определяем ось l и треугольник DMC
draw(h,red); draw(g);//рисуем ось и треугольник
draw(Label("$l$",EndPoint),h,red);//рисуем обозначение оси
draw(rf*g,red);//рисуем образ треугольника при симметрии
draw(sh*rf*g,blue);//рисуем образ треугольника при переносе
draw(shift(rf*M-O)*(O--sh*O),blue,Arrow(TeXHead));
//рисуем изображение вектора переноса
draw(Label("$\vec{p}$",MidPoint),shift(rf*M-O)*(O--sh*O),blue);
//обозначаем вектор переноса
draw("$M^{\prime}$",rf*M,SE,bp+red);//рисуем образ M при симметрии
draw("$D^{\prime}$",rf*D,N,bp+red);//рисуем образ D при симметрии
draw("$C^{\prime}$",rf*C,E,bp+red);//рисуем образ C при симметрии
draw("$M^{\prime\prime}$",sh*rf*M,SW,bp+blue);//рисуем образ M при переносе
draw("$D^{\prime\prime}$",sh*rf*D,SW,bp+blue);//рисуем образ D при переносе
draw("$C^{\prime\prime}$",sh*rf*C,E,bp+blue);//рисуем образ C при переносе
draw((-.5,0)--(3,0),Arrow(TeXHead));//рисуем первую координатную ось
draw((0,-.25)--(0,3),Arrow(TeXHead));//рисуем вторую координатную ось
```

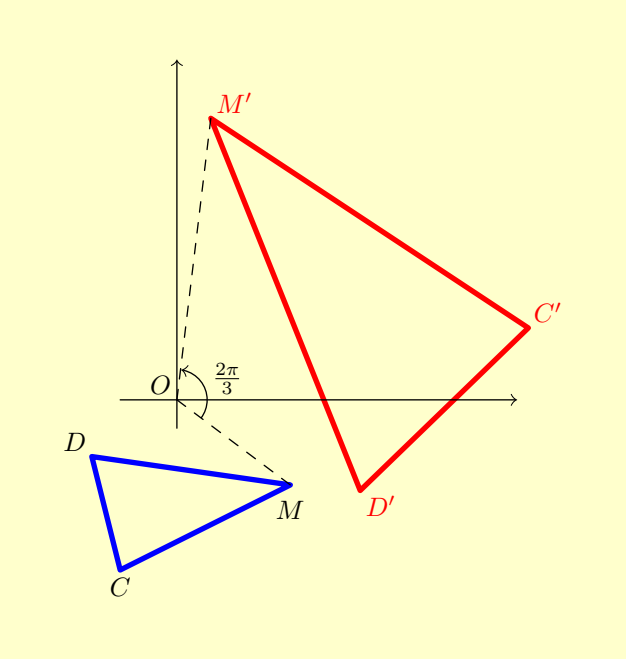


Пример 84

Зададимся целью изобразить образ треугольника DMC при преобразовании, которое является композицией поворота и гомотетии относительно начала координат.

```
unitsize(1cm);// задаем единицу измерения
defaultpen(fontsize(10pt));//определяем размер шрифта по умолчанию
import markers;
real a=120, b=2;
pair O=(0,0), A=(-1,3), B=(3,-1), C=(-.5,-1.5), D=(-.75,-.5), M=(1,-.75);
// задаем точки O, A, B, C, D, M
transform scrot=scale(b)*rotate(a,O);//определяем преобразование
draw("$M$",M,2*S); draw("$O$",O,NW); draw("$C$",C,S); draw("$D$",D,NW);
//рисуем точки M, O, C, D
path g=D--M--C--cycle;//определяем треугольник DMC
draw(g,2bp+blue);//рисуем  треугольник
draw(scrot*g,2bp+red);//рисуем образ треугольника при заданном преобразовании
draw("$M^{\prime}$",scrot*M,NE,bp+red);//рисуем образ M
```

```
draw("$D^{\prime}$",scrot*D,SE,bp+red);//рисуем образ D
draw("$C^{\prime}$",scrot*C,NE,bp+red);//рисуем образ C
draw(O--M,dashed);draw(O--scrot*M,dashed);//draw(D--scrot*D,dashed);
draw((-.5,0)--(3,0),Arrow(TeXHead));//рисуем первую координатную ось
draw((0,-.25)--(0,3),Arrow(TeXHead));//рисуем вторую координатную ось
markangle("$\frac{2\pi}{3}$",M,O,scrot*M,radius=4mm,Arrow(TeXHead));
```



2.1.5 Рисунки

Картинка, полученная командами рисования, называется `currentpicture`. Однако картинке можно давать собственное имя, например, `pic`. Вы можете нарисовать новую картинку, которую можно преобразовывать и добавить к `currentpicture`, позиционируя её определённым образом относительно `currentpicture`.

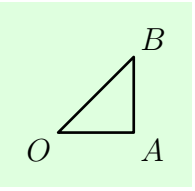
Новая картинка (`pic1`) может быть добавлена к предыдущей (`pic`) командой `add(pic,pic1)` и по умолчанию команда `add(pic)` просто добавляет картинку `pic` к `currentpicture`.

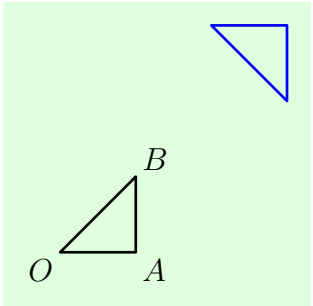
Более подробно о работе с картинками (тип данных `picture`) будет рассказано в *Главе 2*, а сейчас два примера объединения нескольких картинок в одно изображение.

Пример 85

Здесь в качестве основы выступает треугольник OAB черного цвета. Его изображение выступает в роли `currentpicture`.

```
unitsize(1cm);
pair O=(0,0), A=(1,0), B=(1,1);
draw(O--A--B--cycle,linewidth(bp));
label("$O$",O,SW );label("$A$",A,SE );label("$B$",B,NE );
```



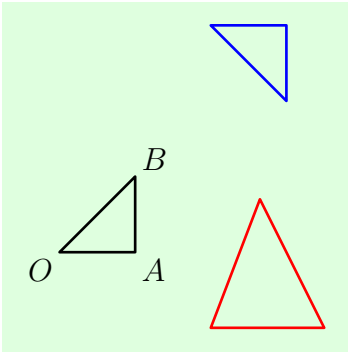


Дополним предыдущий код следующими строками:

```
real a=90;
picture pic;
draw(pic,rotate(a,O)*(O--A--B--cycle),1bp+blue);
add(shift(3,2)*pic);
```

Прокомментируем эти четыре строки. Задаём угол поворота и новую картинку `pic`. Помещаем в неё исходный треугольник, повернутый на 90° в позиции `(3,2)`. Рисуем его пером синего цвета.

Дополним имеющийся код ещё несколькими строками:




```
picture pic1;
path tri=0--A--B--cycle;
transform t=slant(s)*scale(1.5,1.7);
draw(pic1,t*(tri),1bp+red);
add(shift(2,-1)*pic1);
```

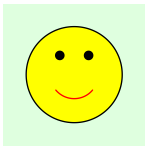
Задаём вторую картинку, `pic1`. Для компактности кода определим путь `tri` — контур треугольника и преобразование. Подвергнем исходный треугольник преобразованию, начертим его на картинке `pic1` и добавим `pic1` к текущей картинке в позиции `(2,-1)`.

Пример 86

В этом примере исходной картинкой является пустая картинка. Она играет роль `currentpicture`. Мы создаем новую картинку `pic` — смайлик. После выполнения команды `add(pic)` смайлик становится текущей картинкой (`currentpicture`). Картинку `pic` можно подвергать различным преобразованиям и добавлять к текущей картинке. Прокомментируем код:

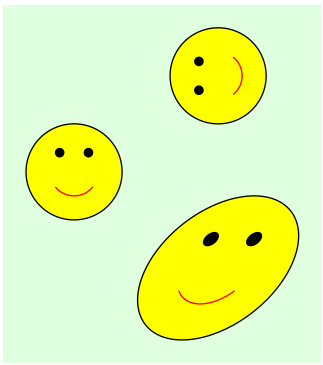
```
import graph;
unitsize(.25inch);
real a=90,s=.5;
pair 0=(0,0);
```

Подгружаем модуль `graph`, задаем единицу измерения, угол поворота и параметр сдвига. Задаем центр поворота. Далее определяем смайлик как картинку `pic`.



```
picture pic;
filldraw(pic,Circle((0,0),1),yellow,black);
fill(pic,Circle((-.3,.4),.1),black);
fill(pic,Circle((.3,.4),.1),black);
draw(pic,Arc((0,0),.5,-140,-40),red);
add(pic); //рисуем смайлик
```

Добавляем к текущей картинке смайлик, повернутый на 90 градусов и сдвинутый на вектор `(3,2)`.



```
add(shift(3,2)*rotate(a,0)*pic);
```

Добавляем к текущей картинке смайлик, подвергнутый преобразованию `shift(3,-2)*slant(s)*scale(1.5)`.

```
add(shift(3,-2)*slant(s)*scale(1.5)*pic);
shipout(bbox(3mm,Fill(palegreen+white)));

//рисуем box с фоном
```

2.2 Массивы

Добавление скобок `[]` ко встроенному или пользовательскому типу приводит к созданию массива. Элементами массивов могут быть числа, точки, пути, перья, преобразования и т.д. Элемент `i` массива `A` обозначается `A[i]`. По умолчанию, попытки доступа или присвоения элементу массива отрицательного индекса генерируют сообщение об ошибке. Чтение элемента массива с индексом за пределами длины массива также генерирует сообщение об ошибке; однако, определение элемента за пределами длины массива служит причиной изменения размеров массива, в котором будет размещен новый элемент. Можно также проиндексировать массив `A` целочисленным массивом `B`: массив `A[B]` формируется путем индексации массива `A` последовательными элементами массива `B`.

Декларация `real[] A;` инициализирует `A` как пустой (нулевой длины) массив. Пустые массивы следует отличать от null-массивов, которые не допускают ссылок, не имеют длины, не читаются и не присваиваются.

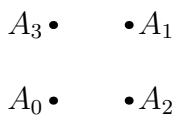
Массивы могут быть инициализированы явно или иным способом. Рассмотрим примеры задания массивов. Начнём с очень простых примеров.

Пример 87

`real[] A={0,1,2};` Пример явного задания числового массива. В данном примере числа действительные (real). Элементы массива перечислены в фигурных скобках после знака равенства.

Пример 88

`pair[] A={(0,0),(1,1),(1,0),(0,1)};` Явно задан массив из четырех точек. Изобразим их:



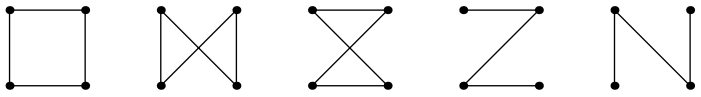
Пример 89

Из точек предыдущего примера создадим массив `path[] q` из пяти путей.

```
unitsize(1cm);
pair[] A={(0,0),(1,1),(1,0),(0,1)};
path[] q; q[0]=A[0]--A[2]--A[1]--A[3]--cycle;
          q[1]=A[0]--A[1]--A[2]--A[3]--cycle;
          q[2]=A[0]--A[2]--A[3]--A[1]--cycle;
          q[3]=A[3]--A[1]--A[0]--A[2];
          q[4]=A[0]--A[3]--A[2]--A[1];
```

Начертим пути, добавив в код следующие строчки:

```
draw(q[0]); draw(shift(2,0)*q[1]); draw(shift(4,0)*q[2]);
draw(shift(6,0)*q[3]);draw(shift(8,0)*q[4]);
dot(q[0]); dot(shift(2,0)*q[1]); dot(shift(4,0)*q[2]);
dot(shift(6,0)*q[3]); dot(shift(8,0)*q[4]);
shipout(bbox(4mm,invisible));
```

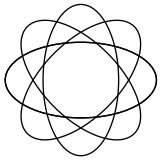


Пример 90

Задаем числовой массив `real[] beta={0,45,90,135,180}` из углов поворота и массив преобразований `transform[] T; T[i]=rotate(beta[i])`.

Затем определяем массив путей `path[] q; q[i]=T[i]*yscale(.5)*unitcircle;` и чертим пути:

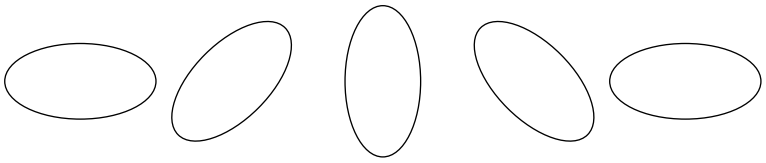
```
unitsize(1cm); real[] beta={0,45,90,135,180};
for(int i=0; i<=4; ++i){
transform[] T; T[i]=rotate(beta[i]);
path[] q; q[i]=T[i]*yscale(.5)*unitcircle;
draw(q[i]);}
```



Пример 91

Аналогично предыдущему примеру определяются два массива действительных чисел, массив преобразований и массив путей:

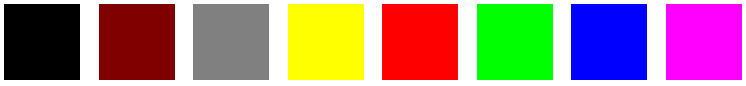
```
unitsize(1cm);
real[] s={0,2,4,6,8}; real[] beta={0,45,90,135,180};
for(int i=0; i<=4; ++i){
transform[] T; T[i]=rotate(beta[i]);
path[] q; q[i]=shift(s[i],0)*T[i]*yscale(.5)*unitcircle;
draw(q[i]);}
```



Пример 92

В этом примере явно задается массив из семи перьев различного цвета. Это `pen [] p;` Далее задается массив путей из семи квадратов. Это `path [] s;` Каждый квадрат закрашивается пером из указанного массива. Однако, если первый массив задан явно, то второй — описательно.

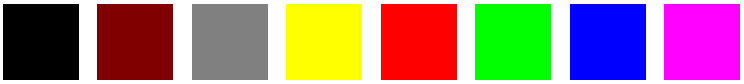
```
unitsize(1cm);
pen[] p={black,brown,gray,yellow,red,green,blue,magenta};
for(int i=0; i<=7;++i){
path [] s; s[i]=shift(1.25*i,0)*unitsquare;
fill(s[i],p[i]);}
shipout(bbox(4mm,invisible));
```



Пример 93

Чтобы получить такой результат, не обязательно задавать два массива. В коде этого примера показано, как обойтись всего одним.

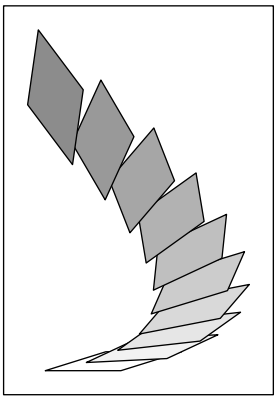
```
unitsize(1cm);
path s=unitsquare;
pen[] p={black,brown,gray,yellow,red,green,blue,magenta};
for(int i=0; i<=7;++i)
      {fill(shift(1.25*i,0)*s,p[i]);}
shipout(bbox(4mm,invisible));
```



Пример 94

В примере определены четыре массива: два числовых `real [] a;` и `real [] g;`, затем массив перьев `pen[] p;` и массив преобразований `transform [] t;`

```
unitsize(1cm);
pair O=(0,0);
real [] a={0,10,20,30,40,50,60,70,80,90};
real [] g={1,.95,.9,.85,.8,.75,.7,.65,.6,.55};
path  rec=shift(3,-.5)*slant(.6)*yscale(.5)*unitsquare;
for(int i=0; i<=9;++i)
{
pen[] p; p[i]=gray(g[i]);
transform [] t;
t[i]=yscale(.5+.1*i)*slant(1-.2*i)*rotate(.5*a[i],O);
filldraw(shift(i/5,0)*t[i]*rec,p[i]);
}
shipout(bbox(3mm));
```



Теперь приведем два примера двумерных массивов. Первый иллюстрирует возможности функции `pair[] z=quarticroots(a,b,c,d,e)` из пакета `math`.

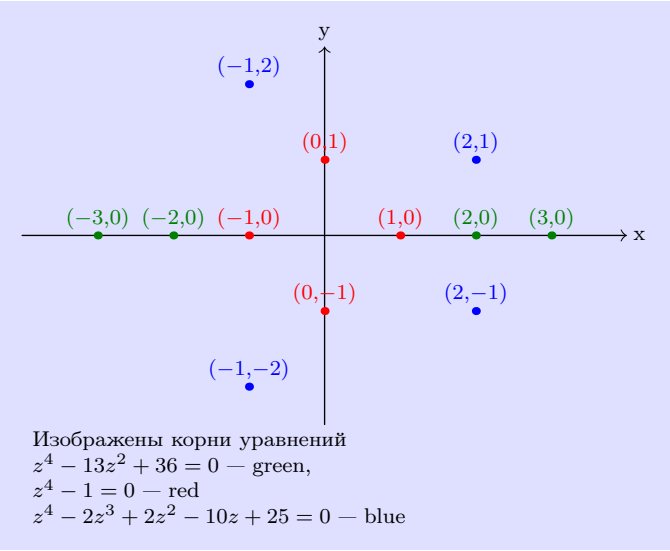
Пример 95

Функция `pair[] z=quarticroots(a,b,c,d,e)` возвращает комплексные корни уравнения

$$ax^4 + bx^3 + cx^2 + dx + e = 0, \text{ где } a, b, c, d, e \in \mathbb{R}.$$

Мы задаём с помощью массива
`real [][] A={{1, 0, 0, 0, -1},{1, -2, 2, -10, 25},{1, 0, -13, 0, 36}}`
действительные коэффициенты трёх уравнений одновременно.

```
texpreamble("\usepackage[T2A]{fontenc}\usepackage[utf8]{inputenc}
\usepackage[russian]{babel}\usepackage{amsfonts}");
unitsize(1cm);
path x=(-4,0)--(4,0);
path y=(0,-2.5)--(0,2.5);
draw(Label("x",position=EndPoint),x,Arrow(TeXHead));
draw(Label("y",position=EndPoint),y,Arrow(TeXHead));
import math;
for(int i=0; i<=2;++i){
pen [] p={red,blue,.5green};
real [][] A={{1, 0, 0, 0, -1},{1, -2, 2, -10, 25},{1, 0, -13, 0, 36}};
pair[] z=quarticroots( A[i][0],A[i][1],A[i][2],A[i][3],A[i][4]);
for(int k=0; k<=3;++k){dot("",z[k],N,p[i]);}}
label(minipage("Изображены корни уравнений\\
$z^4 -13z^2 +36 = 0$ --- green,\\
$z^4 -1 = 0$ --- red\\
$z^4 -2z^3 +2z^2 -10z +25 = 0$ --- blue ",width=8cm),truepoint(S), S);
shipout(bbox(5mm, invisible));
```



Следующий пример — пример двумерного массива, элементами которого являются перья.

Он употребляется тогда, когда плавное градиентное затенение делается перьями `p` разных цветов с помощью правила заливки `fillrule`

```
void latticeshade(picture pic=currentpicture, path g, bool stroke=false,
pen fillrule=currentpen, pen[][] p);
```

Пример 96

```
size(3cm);
path g=(0,0)--(0,3)--(1,3)--(1,0)--cycle;//Задаем циклический путь.
pen [][] p={{rgb(.8gray)},{rgb(red)}};//Задаем массив перьев.
latticeshade(g,p);//Задаем градиентное затенение указанными перьями.
shipout(bbox(5mm,invisible));
```

Перья в массиве `p` должны принадлежать одному и тому же цветовому пространству. Можно использовать функции `rgb(pen)` или `cmyk(pen)`.

Дальнейшие примеры использования массивов приведены в *подразделе 2.4.3 Массивы точек пересечения*.

2.3 Управляющие структуры

Программа есть последовательность инструкций. Простая программа может быть линейной последовательностью инструкций. Ещё программа может разветвляться, то есть в зависимости от условий происходит выполнение тех или иных инструкций, а также инструкции программы могут повторяться.

Для этих целей в `Asymptote` существуют структуры управления, указывающие, что должно быть сделано в данной программе, когда и при каких обстоятельствах.

В структурах управления используют понятие *оператор соединения* или *блок*. Блок — это группа инструкций, которые разделяются точкой с запятой (;). По аналогии с выражениями в C++, инструкции, объединенные в блок, заключаются в фигурные скобки: { }:

{ оператор 1; оператор 2; оператор 3; }

Большинство структур управления этого раздела содержат общее высказывание, как часть своего синтаксиса. Высказывание может быть либо простым высказыванием (простой инструкцией, заканчивающийся точкой с запятой), либо составным оператором (из нескольких команд, сгруппированных в блок), как только что описано.

В случае, если мы имеем простое высказывание, нам не обязательно заключать его в фигурные скобки ({}). Но составное непременно должно быть заключено в фигурные скобки, образуя блок.

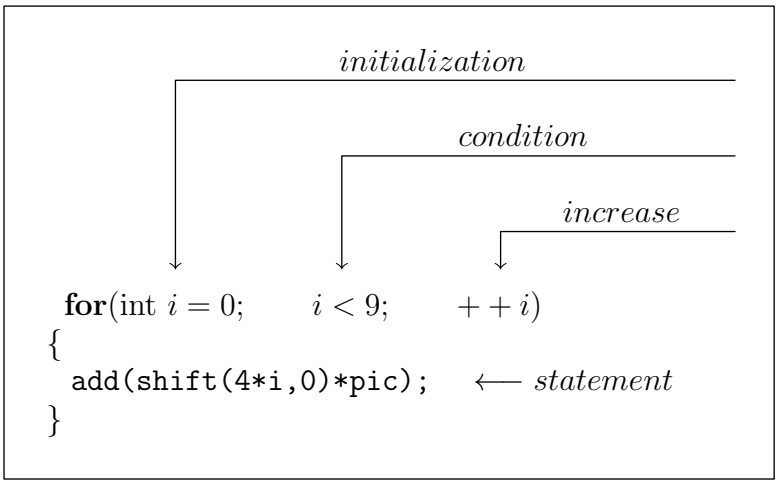
2.3.1 Циклы

Циклы имеют цель повторить инструкцию либо некоторое определенное число раз, либо пока выполняется указанное условие. Соответственно есть три варианта циклов: **for**-цикл, **while**-цикл и **do-while**-цикл.

- **for цикл**

  for цикл имеет следующий формат:

for `(initialization; condition; increase) statement;`



Его главная функция — в повторении *инструкции* (statement), если *условие* (condition) остается истиным. Кроме того, эта форма цикла имеет поля *инициализации* (initialization) и *шаг* (increase). Этот цикл разработан специально для выполнения повторяющихся действий со счетчиком, который инициализируется и увеличивается (либо уменьшается) при каждом повторении (итерации).

Последовательность действий такова:

1. Выполняется *инициализация* (присвоение начального значения переменной-счетчику). Выполняется только один раз, после чего следует переход к пункту 2.
2. Проверяется *условие*. Если условие истинно, то следует переход к пункту 3, в противном случае цикл заканчивается и инструкция не выполняется.
3. Выполняется *инструкция* (оператор). Как обычно, это может быть либо один оператор, либо блок, заключенный в фигурные скобки { }.
4. Наконец, выполняется то, что указано в поле *шаг* и цикл возвращается к пункту 2.

Поля "Инициализация" и "шаг" являются необязательными. Они могут оставаться пустыми, но во всех случаях они должны быть отделены точкой с запятой.

Например можно написать: **for** `(;n<10;)`, если мы хотим сказать об отсутствии инициализации и шага, или **for** `(;n<10;n++)`, если мы хотим сказать, что есть шаг, но нет инициализации (может быть, потому, что переменная ранее уже инициализирована).

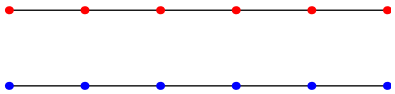
При желании можно использовать запятую (,) для указания более одного выражения в любом из полей, включенных в **for**-цикл, например, в поле инициализации. Оператор запятая (,) является выражением сепаратором, служит для отделения одного выражения от другого, например, для инициализации более чем одной переменной в цикле.

Рассмотрим примеры использования **for**-циклов. Сначала несколько примеров, в которых используются целые значения переменных.

Пример 97

Обратите внимание, что в примере инструкции разные, но приводят к одной и той же картинке, различен лишь цвет изображенных точек. Эта равносильность возможна только для прямолинейных путей.

```
unitsize(1cm);
pair pA=(0,0), pB=(5,0);
draw(pA--pB);
for (int k=0; k<=5; ++k) {
    dot(relpoint(pA--pB,k/5),red);
}
draw(pC--pD);
for (int j=0; j<=5; ++j) {
    dot(interp(pC,pD,j/5),blue);
}
```



Пример 98

Однако путь не обязан быть прямолинейным. В этом случае можно применить только первую форму инструкции из предыдущего примера.

```
unitsize(1cm);
pair pA=(0,0), pB=(5,0);
path p=pA{dir(30)}..{dir(60)}pB;
draw(p);
for (int k=0; k<=5; ++k)
 { dot(relpoint(p,k/5),red); }
```

Пример 99

В этом примере переменная k принимает действительные значения, которое изменяются в промежутке от 0 до $length(p)$ с шагом 0.25.

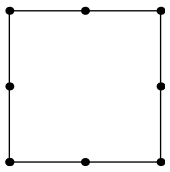
```
unitsize(1cm);
pair pA=(0,0), pB=(5,0);
path p=pA{dir(30)}..{dir(60)}pB;
draw(p);
for (real k=0; k<=length(p); k+=.25)
   { dot(point(p,k),red); }
```

Обратите внимание, что $length(p)$ (длина пути p) в примере 99 равна 1, поэтому число отрезков, на которые разбивается путь, можно определить, разделив длину на шаг.

Пример 100

В данном примере длина пути sq равна 4, поскольку предопределённый путь `unitsquare` имеет четыре узла (вершины квадрата). Следовательно, число точек должно быть равно 8.

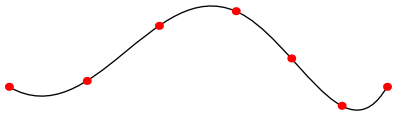
```
unitsize(1cm);
path sq=scale(2)*unitsquare;
draw(sq);
for(real i=0; i<=length(sq); i+=.5)
  { dot(point(sq,i));}
```



Пример 101

В этом примере путь незамкнутый, проходит через три точки, следовательно длина пути $length(p)$ равна 2.

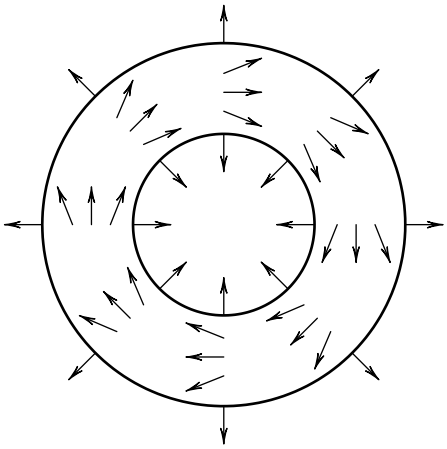
```
unitsize(1cm);
pair pA=(0,0), pC=(3,1), pB=(5,0);
path p=pA{dir(-30)}..pC..{dir(60)}pB;
draw(p);
for (real k=0; k<=length(p); k+=1/3) {
    dot(point(p,k),red);
}
```



Пример 102

Более сложный пример, в котором используются два цикла, один из которых вложен в другой. Один из циклов служит для рисования группы из пяти стрелок, а с помощью второго эта группа поворачивается вокруг центра концентрических кругов.

```
import geometry;
unitsize(1cm);
circle C=circle((0,0),1.2);
draw(C,linewidth(bp));
circle D=circle((0,0),2.4);
draw(D,linewidth(bp));
pair [] A; pair [] B;
A[1]=(0,1.2); B[1]=(0,.7);
A[2]=(0,1.5); B[2]=(.5,1.3);
A[3]=(0,1.75);B[3]=(.5,1.75);
A[4]=(0,2);   B[4]=(.5,2.2);
A[5]=(0,2.4); B[5]=(0,2.9);
for(int i=1; i<=5; ++i)
{for(int j=0; j<=7; ++j)
  {draw(rotate(j*45)*(A[i]--B[i]),
     Arrow(HookHead,size=1mm));}}
```

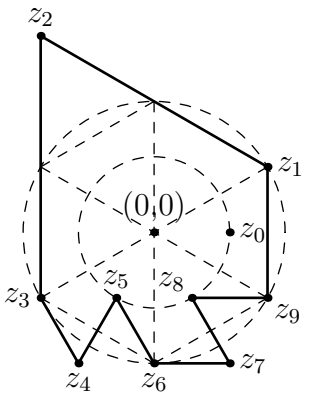


Ещё два примера вложенных циклов. Картинки, нарисованные с их помощью, представляют собой фрагменты паркетов. Плоскость заполняется целиком повторяющимися многоугольными фигурами. Одной из таких фигур является многоугольник, изображённый справа в следующем примере:

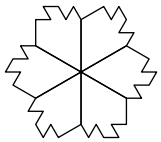
Пример 103

```
unitsize(1cm);
real r=1; pair [] z;/* Задание массива вершин многоугольника */
z[0]=r*dir(0);z[1]=r*sqrt(3)*dir(30);
z[2]=3*r*dir(120);z[3]=rotate(180)*z[1];
z[4]=2*r*dir(-120);z[5]=rotate(-120)*z[0];
```

```
z[6]=rotate(60)*z[3];z[7]=2*r*dir(-60);
z[8]=rotate(60)*z[5];z[9]=rotate(60)*z[6];
dot("",(0,0),N); dot("$z_{0}$",z[0]);
// Обозначения вершин многоугольника
dot("$z_{1}$",z[1],E);
dot("$z_{2}$",z[2],N);
dot("$z_{3}$",z[3],W);
dot("$z_{4}$",z[4],S);
dot("$z_{5}$",z[5],N);
dot("$z_{6}$",z[6],S);
dot("$z_{7}$",z[7],E);
dot("$z_{8}$",z[8],NW);
dot("$z_{9}$",z[9],SE);
/* Пунктиром изображаются вспомогательные линии */
draw(circle((0,0),r),dashed);
draw(circle((0,0),r*sqrt(3)),dashed);
draw(.5*(z[1]+z[2])--.5*(z[2]+z[3]),dashed);
draw(z[1]--z[3],dashed);
draw(z[3]--z[6],dashed);
draw(z[6]--z[9],dashed);
draw(.5*(z[1]+z[2])--z[6],dashed);
draw(.5*(z[2]+z[3])--z[9],dashed);
// Путь, являющийся контуром многоугольника
path t=z[1]--z[2]--z[3]--z[4]--z[5]--z[6]--z[7]--z[8]--z[9]--cycle;
draw(t);
```

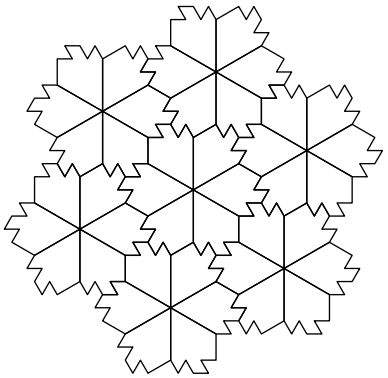


Многоугольниками такой формы можно замостить целиком всю плоскость, причём сделать это можно различными способами. Способ первый состоит в том, что сначала шесть многоугольников объединяют в группу, напоминающую снежинку. Для этого нужен первый цикл.



Такими "снежинками" затем можно замостить всю плоскость. Для этого служит второй цикл, вложенный в первый. На рисунке изображен фрагмент замощения.

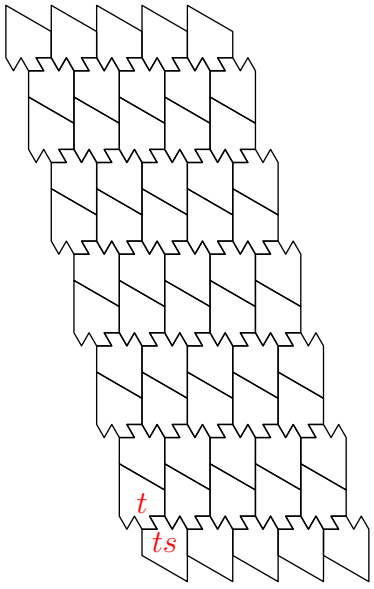
```
unitsize(1cm);
real r=.2; pair [] z;
z[0]=r*dir(0);
z[1]=r*sqrt(3)*dir(30);
z[2]=3*r*dir(120);
z[3]=rotate(180)*z[1];
z[4]=2*r*dir(-120);
z[5]=r*dir(-120);
z[6]=rotate(-120)*z[1];
z[7]=2*r*dir(-60);z[8]=r*dir(-60);
z[9]=rotate(-60)*z[1];
path t=z[1]--z[2]--z[3]--z[4]--z[5]
--z[6]--z[7]--z[8]--z[9]--cycle;
for(int i=0; i<=5; ++i){ path [] f;
f[i]=rotate(60*i,z[2])*t; draw(f[i]);
for(int j=0; j<=5; ++j){pair [] A;
A[j]=rotate(60*j,z[2])*z[9];
pair [] B; B[j]=rotate(180+60*j,
z[2])*z[1];
draw(shift(B[j]-A[j])*f[i]);
}}
shipout(bbox(2mm,invisible));
```



Пример 104

Второй способ замощения плоскости такими многоугольниками состоит в следующем: образуется фигура из двух многоугольников, основного t и центрально-симметричного ему ts, которая затем двумя циклами распространяется на всю плоскость.

```
unitsize(1cm);
real r=.2; pair [] z;
/* Задаём вершины многоугольника и
вспомогательные точки*/
z[0]=r*dir(0); z[1]=r*sqrt(3)*dir(30);
z[2]=3*r*dir(120); z[3]=rotate(180)*z[1];
z[4]=2*r*dir(-120); z[5]=r*dir(-120);
z[6]=rotate(-120)*z[1];
z[7]=2*r*dir(-60); z[8]=r*dir(-60);
pair M=.5*(z[7]+z[8]);
z[9]=rotate(-60)*z[1];
z[10]=rotate(180,M)*z[1];
/* задаём два основных контура */
path t=z[1]--z[2]--z[3]--z[4]--z[5]--z[6]
               --z[7]--z[8]--z[9]--cycle;
path ts=rotate(180,M)*t;
/* Задаём вектор переноса */
pair v=z[2]-z[10];
// первый цикл
for(int i=0; i<=4; ++i){
path [] f; f[i]=shift((3*i)*r,0)*t;
path [] g; g[i]=shift((3*i)*r,0)*ts;
// второй цикл
for(int j=0; j<=5; ++j){
draw(shift(j*v)*(f[i]^^g[i]));}};
```



- **while цикл**

  Рассмотрим **while**-цикл. Его формат:

  while (expression) statement

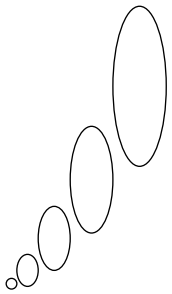
  Смысл в том, чтобы повторять инструкцию, пока условие, заданное в выражении (expression), истинно. Еще его называют циклом с предусловием.

  При создани `while`-цикла мы должны понимать, что в когда-нибудь он закончится, поэтому нам необходимо предусмотреть в рамках блока некоторый метод, заставляющий условие в какой-то момент становится ложным, в противном случае цикл будет продолжаться бесконечно.

Пример 105

Пример, в котором условие наложено на коэффициент k, который входит в параметры преобразований. Преобразования повторяются, пока k не превосходит значения "5".

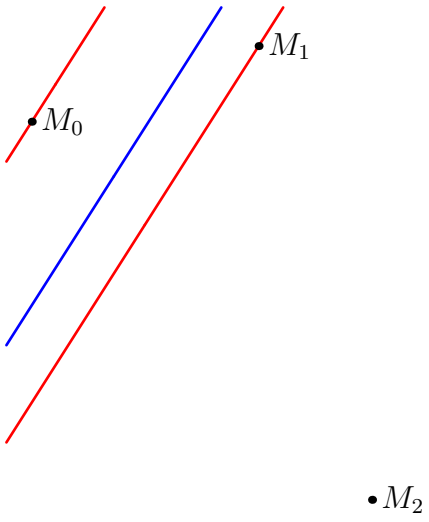
```
unitsize(12);
real k;  path p=unitcircle;
while (k<=5){
draw(yscale(.5+k/2)*scale(k/6)*shift(k,k/2)*p);
++k;}
shipout(bbox(5mm));
```



Пример 106

В этом примере рисуются прямые, проходящие через данные точки и параллельные данной прямой, но только если расстояние d от данной точки до данной прямой не превосходит определенного положительного значения (в примере $d = 3$).

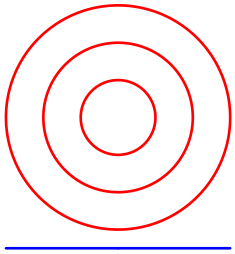
```
import geometry;
unitsize(1cm);
point A=(-4,-3); point B=(-.5,2.5);
point [] M;
M[0]=(-3,1); M[1]=(0,2); M[2]=(1.5,-4);
dot(A); dot(B);
line l=line(A,B);
draw(l,bp+blue); int i;
//--------------------------------
while (distance(M[i],l)<=3.0)
{
line p=parallel(M[i],l);
dot(M[i]);
draw(p,bp+red);
++i;
}
//--------------------------------
shipout(bbox(5mm));
```



Пример 107

Изображаются концентрические окружности, если они не имеют с прямой общих точек.

```
import geometry;
size(4cm,0);
point A=(-2.5,0); point B=(2.5,0);
point Co=(0,3.5); int i;
real [] r; r[0]=1.0; r[1]=2.0; r[2]=3.0;
r[3]=4.0; r[4]=5.0;
line l=line(A,B); draw(l,bp+blue);
while (distance(Co,l)-r[i]>0)
{draw(circle(Co,r[i]),bp+red);++i;}
shipout(bbox(5mm));
```



- **do-while цикл**

Формат этой формы цикла:

do statement while (condition);

Его функция точно такая же, как у `while`-цикла, за исключением того, что условие `do-while`-цикла проверяется после исполнения инструкции, а не до. Инструкции выполняются, по крайней мере, один раз, даже если условие никогда не выполняется.

`Do-while`-цикл обычно используется тогда, когда условие, определяющее конец цикла, задается в пределах инструкции самого цикла. Вот пример использования этого вида цикла:

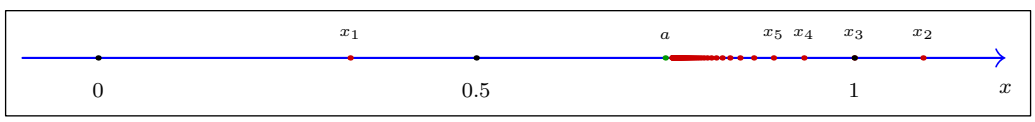
Пример 108

На рисунке изображена числовая ось и на ней красные точки последовательности действительных чисел:

$$x_n = \frac{6n^2+5n+1}{8n^2+3}$$

Зеленой точкой отмечен предел $a = \lim\limits_{n\to\infty}\{x_n\} = .75$. Изображены члены последовательности, для которых $x_n - a > 0.009$.

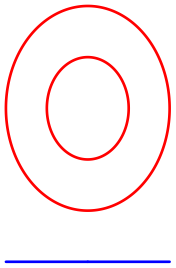
```
unitsize(10cm,2cm); path p=(-.1,0)--(1.2,0);
draw(p,.8bp+blue,EndArrow(arrowhead=TeXHead));
int n; real s(int n){return (6*n*n+5*n+1)/(8*n*n+3);}
dot((.75,0),2bp+.6*green);
do {dot((s(n),0),2bp+.8*red);++n;}
while ((s(n)-.75)>.009);
draw((0,0),linewidth(2bp)); dot((.5,0),linewidth(2bp));
dot((1,0),linewidth(2bp));
label("0",(0,0),4*S,fontsize(8pt));
label("$0.5$",(.5,0),4*S,fontsize(8pt));
label("1",(1,0),4*S,fontsize(8pt));
label("$x$",(1.2,0),4*S,fontsize(8pt));
label("$a$",(.75,0),4*N,fontsize(6pt));
shipout(bbox(2mm));
```



Пример 109

Изображаются концентрические эллипсы до тех пор, пока они не имеют с прямой общих точек.

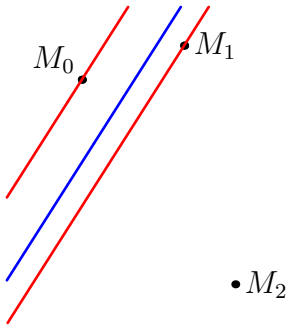
```
import geometry;
size(3cm,0);
point A=(-2.5,.5); point B=(2.5,.5);
point Co=(0,3.5);
int i;real [] r; r[0]=1.0; r[1]=2.0;
r[2]=3.0; r[3]=4.0; r[4]=5.0;
line l=line(A,B);
draw(l,bp+blue);
do
{draw(xscale(.8)*circle(Co,r[i]),bp+red);++i;}
while (distance(Co,l)-r[i]>0);
shipout(bbox(5mm));
```



Пример 110

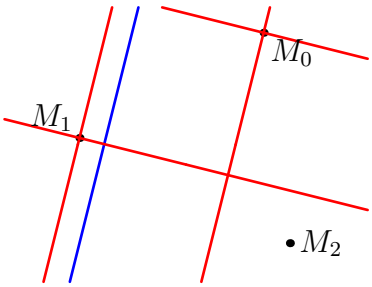
Через данные точки проводятся красные прямые, параллельные синей прямой, отстоящие от нее на расстоянии $\leqslant 3.0$

```
import geometry;
size(0,5cm);
point A=(-4,-3);point B=(-.5,2.5);
point [] M; M[0]=(-3,1); M[1]=(0,2);
M[2]=(1.5,-5);dot(A); dot(B);
line l=line(A,B);linemargin=-.3cm;
draw(l,bp+blue); int i;
do{
line p=parallel(M[i],l);
dot(M[i]);draw(p,bp+red); ++i;
}
while (distance(M[i],l)<=3.0);
shipout(bbox(5mm));
```



Пример 111

```
import geometry;
size(5cm,0);int i;
point A=(-4,-3);point B=(-2.5,3);
point [] M; M[0]=(-.5,1); M[1]=(-4,-1);
M[2]=(0,-3);dot(A); dot(B);
dot("M[0]",M[0]); dot("M[1]",M[1]);
dot("M[2]",M[2]);line l=line(A,B);
linemargin=-.5cm;draw(l,bp+blue);
do{line p=perpendicular(M[i],l);
dot(M[i]); draw(p,bp+red);++i;}
while (distance(M[i],l)<=1.5);
shipout(bbox(5mm));
```



2.3.2 Условный переход: if and else

Ключевое слово **if** используется для выполнения оператора или блока, только если условие выполнено. Условный переход используется в форме:

if (condition) statement

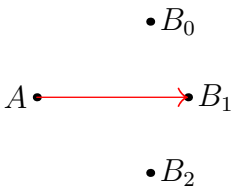
Здесь condition (условие) является выражением, которое оценивается. Если оценка этого условия `true` (истина), выполняется statement (инструкция, оператор). Если оценка `false` (ложь), инструкция игнорируется (не выполняется) и программа продолжает работу сразу после этой условной конструкции.

Пример 112

Например, представленный далее фрагмент кода рисует красную стрелку, если значение переменной `i` действительно равно 1:

```
1 if (i == 1)
2  draw(A--B[i],red,Arrow(TeXHead,size=.6mm));
```

Изобразим точку A и точки: B_0, B_1, B_2. Изобразим вектор $\overrightarrow{AB_1}$.



Приведём полный код этого изображения. В нём можно увидеть конструкцию условного перехода трижды:

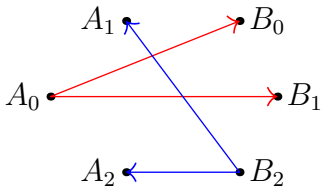
```
unitsize(1cm);
pair A=(1,0); pair [] B; B[0]=(2.5,1);
B[1]=(3,0); B[2]=(2.5,-1);
dot("$A$",A,W);
for(int j=0; j<=2; ++j)
{
    if(j==0) dot("$B_{0}$",B[j]);
    if(j==1) dot("$B_{1}$",B[j]);
    if(j==2) dot("$B_{2}$",B[j]);
}
for(int i=0; i<=2; ++i )
{
  if (i == 1)
draw(A--B[i],red,Arrow(TeXHead,size=.6mm));
}
```

Вот пример, в котором условие выглядит более сложно. При отрисовки стрелок в условии используется конъюнкция.

В примере задаются два массива точек. Нужно нарисовать стрелки, которые связывают точки разных массивов. Условие легко изменять, чтобы рисовались различные стрелки.

Пример 113

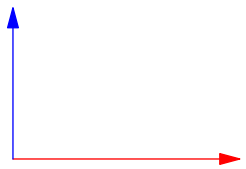
```
settings.outformat="pdf";
unitsize(1cm);
pair [] A; A[0]=(0,0); A[1]=(1,1); A[2]=(1,-1);
pair [] B; B[0]=(2.5,1); B[1]=(3,0); B[2]=(2.5,-1);

for(int k=0; k<=2; ++k){
   if(k==0) dot("$A_{0}$",A[k], W);
   if(k==1) dot("$A_{1}$",A[k], W);
   if(k==2) dot("$A_{2}$",A[k], W);
}
for(int j=0; j<=2; ++j){
   if(j==0) dot("$B_{0}$",B[j]);
   if(j==1) dot("$B_{1}$",B[j]);
   if(j==2) dot("$B_{2}$",B[j]);
}
for(int i=0; i<=2; ++i )
{
for(int k=0; k<=2; ++k){
  if (i <= 1 & k == 0 )
draw(A[k]--B[i],red,Arrow(TeXHead,size=.6mm));
  if (i == 2 & k >= 1 )
draw(B[i]--A[k],blue,Arrow(TeXHead,size=.6mm));
}}
shipout(bbox(5mm,invisible));
```



Пример 114

Если мы имеем несколько инструкций, которые будут выполняться в том случае, если условие истинно, мы можем указать блок, используя фигурные скобки { }:

```
1  if (i == 1)
2  {
3     draw(A--B[i],red,Arrow);
4     draw(A--C[i],blue,Arrow);
5  }
```



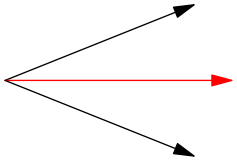
Используя ключевое слово **else**, мы можем указать, что нужно делать, если условие не выполняется. Такая форма может использоваться в сочетании со словом **is**:

if (condition) statement1 else statement2

Пример 115

Следующий фрагмент кода будет изображать три стрелки, одна красным цветом, остальные черным.

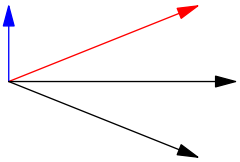
```
unitsize(1cm);
pair A=(0,0); pair [] B; B[0]=(2.5,1);
B[1]=(3,0); B[2]=(2.5,-1);
for (int i=0; i<=2; ++i){path [] q;
q[i]=A--B[i];
if (i==1) draw(q[i],red,Arrow);
else draw(q[i],Arrow);}
```



Помните, что в случае двух и более инструкций мы должны объединить их в блок, заключив в фигурные скобки { }. В качестве примера приведем код, который нарисует уже четыре стрелки: вместе с красной будет дополнительно изображена синяя стрелка.

Пример 116

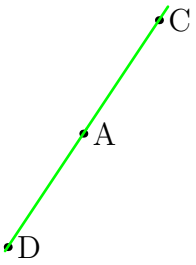
```
unitsize(1cm);
pair [] B;
B[1]=(2.5,1); B[2]=(3,0);B[3]=(2.5,-1);
pair A=(0,0), C=(0,1);
for (int i=1; i<4; ++i){
path [] q; q[i]=A--B[i];
if (i==1) {draw(q[i],red,Arrow);
draw(A--C,blue,Arrow);}
else draw(q[i],Arrow);}
```



Пример 117

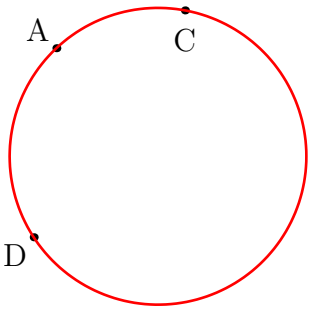
В этом примере проверяемое условие — это условие коллинеарности трех точек, которое технически сводится к проверке коллинеарности двух векторов. Для этого использована стандартная функция `collinear(vector u,vector v)` пакета `geometry`. Коллинеарность точек A, C и D приводит к картинке:

```
import geometry;
unitsize(1cm);
point A=(-1,.5); point C=(0,2);
point D=(-2,-1);
vector u=A; vector v=C;
vector w=D;
dot("A",A); dot("C",C);
dot("D",D);
if (collinear(v-u,w-u))
draw(line(A,C),bp+green);
else draw(circle(C,A,D),bp+red);
shipout(bbox(5mm));
```



Если изменить, например, координаты точки A, не меняя остального кода, это приведёт уже к совершенно другому изображению:

```
import geometry;
unitsize(1cm);
point A=(-1.7,1.5); point C=(0,2);
point D=(-2,-1);
vector u=A; vector v=C;
vector w=D;
dot("A",A);  dot("C",C);
dot("D",D);
if (collinear(v-u,w-u))
draw(line(A,C),bp+green);
else draw(circle(C,A,D),bp+red);
shipout(bbox(5mm));
```



2.4 Пути. Расширенные возможности

2.4.1 Некоторые функции путей

Вот несколько полезных функций для путей:

`int length(path p);`

Число (линейных или кубических) сегментов в пути `p`. Если же путь `p` циклический, это то же самое, что и число узлов в `p`.

`pair point(path p, real t);`

Возвращает координаты точки, расположенной между узлами `floor(t)` и `floor(t)+1`, соответствующей на кубическом сплайне параметру `t-floor(t)`

`pair dir(path p, real t, bool normalize=true);`

Возвращает направление касательной к пути `p` (как пару) в точке между узлами `floor(t)` и `floor(t)+1`, соответствующее значению параметра `t-floor(t)`.

`real arclength(path p);`

возвращает длину (в пользовательских координатах) кусочно-линейной кубической кривой, представляющей путь `p`.

`pair relpoint(path p, real l);`

возвращает точку на пути `p` как относительнуя долю `l` от своей `arclength`.

`path reverse(path p);`

Возвращает путь, который проходится в обратном направлении

`path subpath(path p, int a, int b);`

Возвращает подпуть пути `p` от узла `a` до узла `b`. Если `a > b`, то направление подпути является обратным.

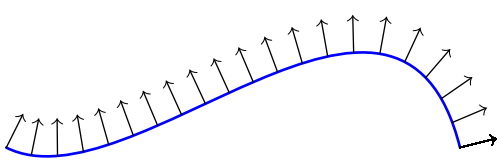
`path subpath(path p, real a, real b);`

Возвращает подпуть пути `p` от времени `a` до времени `b`, in the sense of `point(path, real)`. Если `a > b`, то направление подпути является обратным.

Перечислены далеко не все функции, некоторые другие будут рассмотрены в следующих подразделах. Рассмотрим несколько примеров.

Пример 118

На рисунке изображено векторное поле нормалей к линии. Как видно из кода на следующей странице, используются функции `real arclength(path p)`, `pair relpoint(path p,real i)` и `pair dir(path p, real t, bool normalize=true);`.



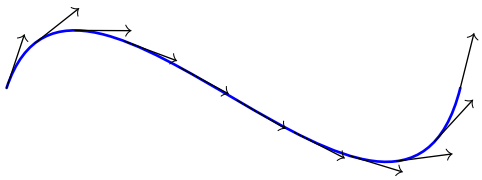
```
unitsize(1cm);
pair [] A; A[1]=(-3,0); A[2]=(3,0); //начало и конец линии
path p= A[1]{(1,-.5)}..{(.25,-1)}A[2]; // Определяем линию
draw(p,bp+blue); // Рисуем линию
// Ниже рисуется векторное поле
for(real i=0; i<=arclength(p); i+=.1)
{
pair [] B; pair [] C; pair [] D;
int j;
B[j]=relpoint(p,i); // Точки на линии
C[j]=.5*dir(p,i); // Направления касательных
D[j]= shift(rotate(90)*C[j])*B[j]; // Концы нормалей
draw(B[j]--D[j],Arrow(TeXHead)); // Рисуем векторы нормалей
}
shipout(bbox(5mm,invisible));
```

Пример 119

Следующий код позволяет изобразить векторное поле, касательное к линии.

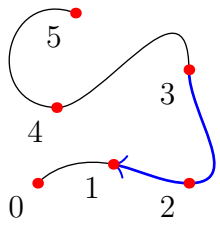
Помимо функций `int length(path p)` и `pair dir(path p, real t, bool normalize=true)`, используется функция `pair point(path p,real i)`.

```
unitsize(1cm);
pair [] A; A[1]=(-3,0); A[2]=(3,0); //начало и конец линии
path p= A[1]{(.5,1.5)}..{(.25,1)}A[2]; // Определяем линию
draw(p,bp+blue); // Рисуем линию
// Ниже рисуется векторное поле
for(real i=0; i<=length(p); i+=.1)
{
pair [] B; pair [] C; pair [] F;
int j;
B[j]=point(p,i); // Точки на линии
C[j]=.75*dir(p,i); // Направления касательных
F[j]= shift(C[j])*B[j]; // Концы касательных векторов
draw(B[j]--F[j],Arrow(TeXHead)); // Рисуем векторы касательного поля
}
shipout(bbox(5mm,invisible));
```



Пример 120

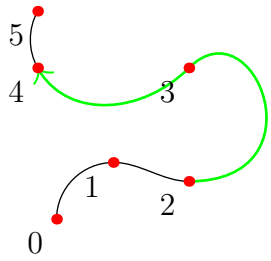
В этом примере иллюстрируется функция `subpath`. Здесь она выделяет подпуть исходного пути `c` между двумя узлами, `3` и `1`. Поэтому подпуть имеет направление, противоположное направлению исходного пути. Подпуть выделен синим цветом и стрелкой.




```
unitsize(.5cm);
path c=(-2,-2){(1,1)}..(0,-1.5)..{(1,0)}(2,-2)..
(2,1){(0,1)}..(-1.5,0)..(-1,2.5);
draw(c);
for (int k; k<=length(c); ++k)
label(string(k),point(c,k),2*SW);
draw(subpath(c,3,1),bp+blue,Arrow(TeXHead));
dot(c,4bp+red);
shipout(bbox(1mm,white));
```

В следующем примере подпуть между двумя узлами обозначен зелёным цветом и стрелкой.

Пример 121



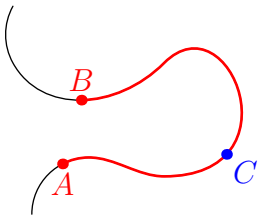

```
unitsize(.5cm);
path c=(-1.5,-3){(0,1)}..(0,-1.5)..{(1,0)}(2,-2)..
(2,1){(-1,-1)}..(-2,1)..(-2,2.5);
draw(c);
for (int k; k<=length(c); ++k)
label(string(k),point(c,k),2*SW);
draw(subpath(c,2,4),bp+green,Arrow(TeXHead));
dot(c,4bp+red);
shipout(bbox(1mm,white));
```

Пример 122

Здесь иллюстрируется функция `subpath`, но в отличие от предыдущих двух примеров подпуть `f` выделяется двумя значениями действительного параметра, `0.7` и `3.5`, которые означают "время" пути.

Далее используется функция `relpoint`, примененная к подпути `f`. В ней параметр имеет другой смысл (относительная доля от `arclength`), он меняется от `0` до `1`.

```
unitsize(.5cm);
path c=(-1.5,-3){(0,1)}..(0,-1.5)..{(1,0)}(2,-2)..
(2,1){(-1,-1)}..(-2,1)..(-2,2.5);
draw(c); path f=subpath(c,.7,3.5);
draw(f,bp+red);
pair A=relpoint(f,0); pair B=relpoint(f,1);
pair C=relpoint(f,.4);
dot("$A$",A,S,4bp+red); dot("$B$",B,N,4bp+red);
dot("$C$",C,SE,4bp+blue);
shipout(bbox(1mm,white));
```



2.4.2 Точки пересечения

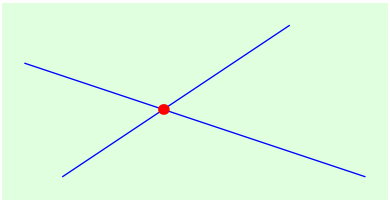
В этом подразделе мы рассмотрим точки пересечения различных графических объектов, которые, как правило, предствляют собой пути. В простейших случаях это будут отрезки. Определение точек пересечения основано на использовании функций путей. Начнем с рассмотрения самых простых примеров.

Например, нужно отметить пересечение двух отрезков, имеющих единственную общую точку. Для этого можно использовать различные средства. Сначала используем следующую функцию:

```
pair intersectionpoint(path p, path q, real fuzz=-1);
```

Подробности можно посмотреть здесь: http://asymptote.sourceforge.net
Andy Hammerlindl, John Bowman, Tom Prince **Asymptote. Vector Graphics Language**

Пример 123



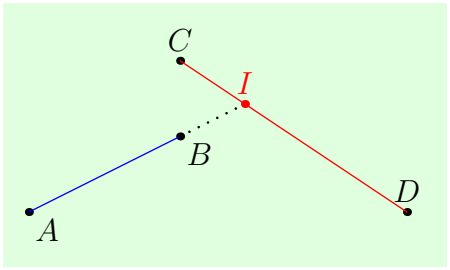
```
unitsize(1cm);
//Определяем отрезки как пути seg1,seg2
path seg1=(-1,-1)--(2,1), seg2=(-1.5,.5)--(3,-1);
draw(seg1^^seg2,blue);// Чертим отрезки
dot(intersectionpoint(seg1,seg2),4bp+red);
// Отмечаем точку пересечения красным цветом
```

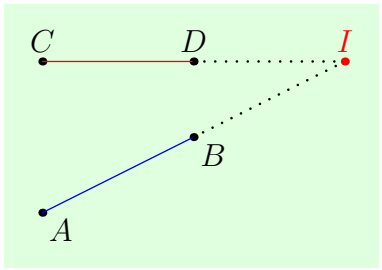
В следующем примере отрезки $[AB]$ и $[CD]$ не имеют общих точек.

Используем функцию `pair extension(pair P, pair Q, pair p, pair q);` [1] Она возвращает точку пересечения прямых, содержащих отрезки `PQ` и `pq`, или `(infinity,infinity)`, если прямые параллельны.

Пример 124

```
unitsize(1cm);
pair A=(0,1),B=(2,2),C=(2,3),D=(5,1);
dot("$A$",A,SE); dot("$B$",B,SE);
dot("$C$",C,N); dot("$D$",D,N);
// Чертим [AB] и [CD]
draw(A--B,blue); draw(C--D,red);
// Создаем точку пересечения
// отрезков [AB] и [CD]
pair pI=extension(A,B,C,D);
dot("$I$",pI,N,red);
draw(B--pI,1pt+dotted);
```

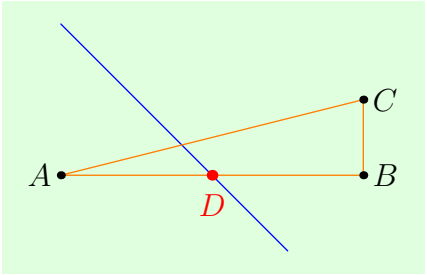





```
unitsize(1cm);
pair A=(0,1),B=(2,2),C=(0,3),D=(2,3);
dot("$A$",A,SE); dot("$B$",B,SE);
dot("$C$",C,N); dot("$D$",D,N);
// Чертим [AB] и [CD]
draw(A--B,blue); draw(C--D,red);
// Создаем точку пересечения
// отрезков [AB] и [CD]
pair pI=extension(A,B,C,D);
dot("$I$",pI,N,red);
draw(B--pI,1pt+dotted);
draw(D--pI,1pt+dotted);
```

Пример 125

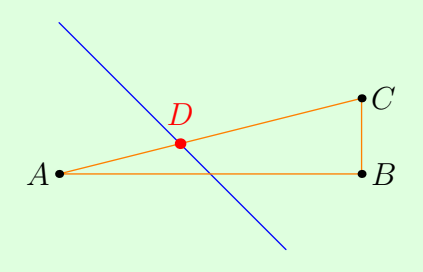
```
unitsize(1cm);
path droite=(-2,2)--(1,-1);
path tri=(-2,0)--(2,0)--(2,1)--cycle;
pair pD=intersectionpoint(droite,tri);
draw(droite,blue);
draw(tri,orange);
dot("$A$",(-2,0),W);
dot("$B$",(2,0),E);
dot("$C$",(2,1),E);
dot("$D$",pD,2S,4bp+red);
```



[1] Смотри [9], Глава 6, раздел 6.2, стр 34.

В этом примере функция `intersectionpoint` возвращает первую точку пересечения путей. Заметьте, что треугольник обходится против часовой стрелки, начиная с точки A.

Изменим направление обхода треугольника. Для этого служит команда `reverse`.



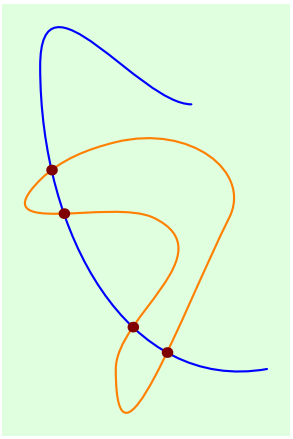

```
unitsize(1cm);
path droite=(-2,2)--(1,-1);
path tri=(-2,0)--(2,0)--(2,1)--cycle;
pair pD=intersectionpoint(droite,reverse(tri));
draw(droite,blue);
draw(tri,orange);
dot("$A$",(-2,0),W);
dot("$B$",(2,0),E);
dot("$C$",(2,1),E);
dot("$D$",pD,2N,4bp+red);
```

Мы видим, что точка пересечения отрезка и треугольника изменилась.

В следующем примере используется другая функция, связанная с пересечением путей. Эта функция `pair[] intersectionpoints(path p, path q, real fuzz=-1)`, которую легко спутать с функцией предыдущего примера. Обратите внимание на наличие буквы "s" в конце названия команды. Она возвращает все точки пересечения двух путей.

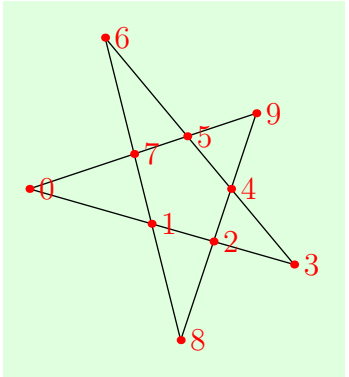
Пример 126

```
unitsize(1cm);
defaultpen(.8bp);
pair [] A;
             A[1]=(-1,.5); A[2]=(0,1); A[3]=(1.5,0);
             A[4]=(0,-2); A[5]=(.5,0);
path l1=
(1,1.5){dir(180)}..(-1,2){dir(-90)}..{dir(10)}(2,-2);

path l2=A[1]{(1,1)}..A[2]..{(-1,-2)}A[3]..
                           {(0,1)}A[4]..A[5]..cycle;

draw(l1,blue); draw(l2,orange);
dot(intersectionpoints(l1,l2),4bp+brown);
```

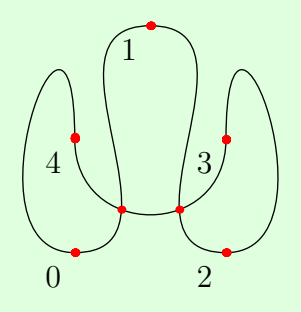


Пример 127

```
// Ломаная с самопересечениями.
unitsize(1cm);
// ~~~~~~~~~ Определение ~~~~~~~~~
pair A=(-1,0), B=(2.5,-1), C=(0, 2),
     D=(1,-2), E=(2,1);
path c = A--B--C--D--E--cycle;
pair[] p = intersectionpoints(c,c);
// ~~~~~~~~~ Построение ~~~~~~~~~
draw(c);
for (int k=0; k<p.length; ++k)
dot(format("%i",k),p[k],red);
shipout(bbox(2mm,white));
```



Пример 128

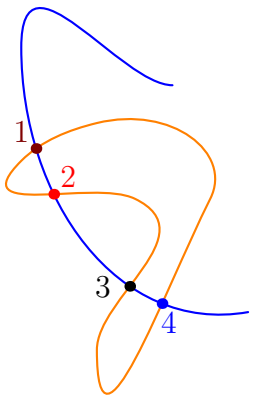



```
// Кривая с самопересечением.
unitsize(.5cm);
// ~~~~~~~~~ Определения ~~~~~~~~~
path c=(-2,-2){(1,0)}..(0,4)..{(1,0)}(2,-2)..
(2,1){(0,-1)}..{(0,1)}(-2,1)..{(1,0)}(-2,-2);
pair[] p = intersectionpoints(c,c);
// ~~~~~~~~~ Построения ~~~~~~~~~
draw(c);
dot(intersectionpoints(c,c),3bp+red);
for (int k; k<length(c); ++k)
label(string(k),point(c,k),2*SW);
//отмечаем узлы пути цифрами; красные точки
//без цифр - точки самопересечения
shipout(bbox(1mm,white));
```

2.4.3 Массивы точек пересечения

В следующем примере показано, что из массива точек пересечения двух путей можно извлечь только нужные точки. Для этого обратите внимание на четыре последних команды в указанном коде.

Пример 129




```
unitsize(1cm);
defaultpen(.8bp);
pair [] A; A[1]=(-1,.5); A[2]=(0,1); A[3]=(1.5,0);
      A[4]=(0,-2); A[5]=(.5,0);
path l1=(1,1.5){dir(180)}..(-1,2){dir(-90)}..
                                {dir(10)}(2,-1.5);
path l2=A[1]{(1,1)}..A[2]..{(-1,-2)}A[3]..
                        {(0,1)}A[4]..A[5]..cycle;
draw(l1,blue); draw(l2,orange);
// Функция intersectionpoints возвращает массив
//точек пересечения
pair [] tp=intersectionpoints(l1,l2);
/* Первая точка массива имеет индекс 0, вторая -
индекс 1, третья - индекс 2 и так далее.*/
dot("$1$",tp[0],NW,4bp+brown);
dot("$2$",tp[1],NE,4bp+red);
dot("$3$",tp[2],2W,4bp+black);
dot("$4$",tp[3],SSE,4bp+blue);
```

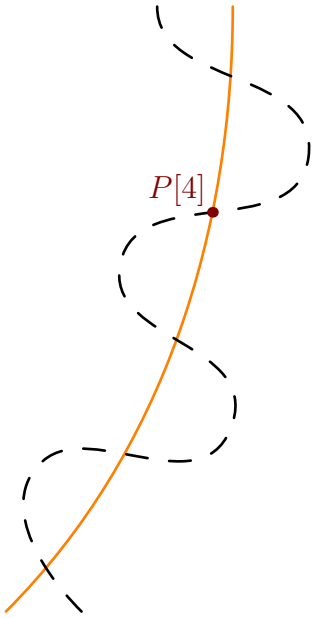
В следующем примере показана функция `real[][] intersections(path p, path q, real fuzz=0)`, которая выдает все времена пересечения путей `p` и `q` как некоторый упорядоченный массив (смотри `[sort]`, [9], Глава 6, раздел 6.12, стр 71).

Пример 130

Порядок в двумерном массиве времен `[][]` определен так: поскольку у нас пять точек пересечения, то первый столбец содержит пять элементов, а пути всего два, значит в строке два элемента.

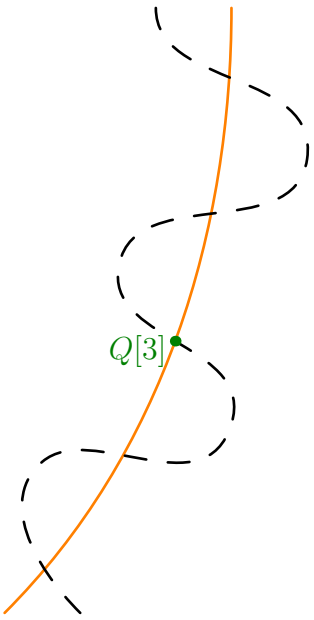
Если мы хотим выделить четвертую точку на первом пути, нужно поступить следующим образом: `P[4]=point(l1,parametres[3][0]);`, так как нумерация в массиве начинается с нуля.

```
unitsize(1cm);
pair A=(0,-4), B=(-.5,-2), C=(2,-1.5),
D=(.5,.5), E=(3,2), F=(1,4);
path l1=(-1,-4){(1,1)}..{(0,1)}(2,4);
path l2=A{(-1,1)}..B..C..D..E..{(0,1)}F;
real[][] parametres=intersections(l1,l2);
pair P[]; pair Q[];
//P[1]=point(l1,parametres[0][0]);
//P[2]=point(l1,parametres[1][0]);
//P[3]=point(l1,parametres[2][0]);
P[4]=point(l1,parametres[3][0]);
//P[5]=point(l1,parametres[4][0]);
draw(l1,bp+orange);
draw(l2,bp+dashed);
//dot("$P[1]$",P[1],W,4bp+blue);
//dot("$P[2]$",P[2],NW,4bp+red);
//dot("$P[3]$",P[3],E,4bp+.5green);
dot("$P[4]$",P[4],NW,4bp+brown);
//dot("$P[5]$",P[5],NE,4bp+purple);
shipout(bbox(3mm));
```



Пример 131

По аналогии с предыдущим примером третья точка на втором пути определяется так:
`Q[3]=point(l2,parametres[2][1]);`




```
unitsize(1cm);
pair A=(0,-4), B=(-.5,-2), C=(2,-1.5),
D=(.5,.5), E=(3,2), F=(1,4);
path l1=(-1,-4){(1,1)}..{(0,1)}(2,4);
path l2=A{(-1,1)}..B..C..D..E..{(0,1)}F;
real[][] parametres=intersections(l1,l2);
pair P[]; pair Q[];
//Q[1]=point(l2,parametres[0][1]);
//Q[2]=point(l2,parametres[1][1]);
Q[3]=point(l2,parametres[2][1]);
//Q[4]=point(l2,parametres[3][1]);
//Q[5]=point(l2,parametres[4][1]);
draw(l1,bp+orange);
draw(l2,bp+dashed);
//dot("$Q[1]$",Q[1],W,4bp+blue);
//dot("$Q[2]$",Q[2],S,4bp+red);
dot("$Q[3]$",Q[3],NE,4bp+.5green);
//dot("$Q[4]$",Q[4],NW,4bp+brown);
//dot("$Q[5]$",Q[5],NE,4bp+purple);
shipout(bbox(3mm));
```

Мы закомментировали в коде все точки, сняв комментарий только с четвертой точки пересечения на первом пути и с третьей точки пересечения на втором пути.

2.4.4 Фрагменты пути

В этом подразделе рассматриваются примеры, иллюстрирующие структуру `slice` (фрагмент), связанную с точками пересечения двух путей. Тип `struct slice {path before,after;}` определен в модуле `plain` и используется со следующими функциями путей: `slice cut(path p, path knife, int n);`, `slice firstcut(path p, path knife);`, `slice lastcut(path p, path knife);`.

`slice cut(path p, path knife, int n);`

возвращает часть пути `p` до и после `n`-го пересечения `p` с путем `knife`(нож), как структуру `slice` (при отсутствии пересечения, весь путь считается "до" пересечения). Аргумент `n` считается по модулю числа пересечений.

`slice firstcut(path p, path knife);`

эквивалентна `cut(p, knife, 0)`; Заметьте, что функция `firstcut.after` играет роль команды `cutbefore` в `MetaPost`.

`slice lastcut(path p, path knife);`

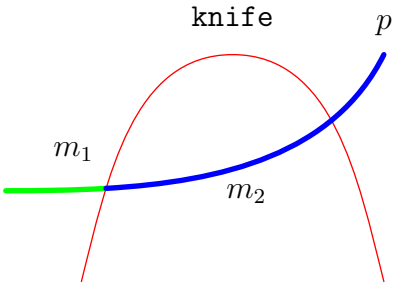
эквивалентна `cut(p, knife, -1)`; Заметьте, что функция `lastcut.before` играет роль команды `cutafter` в `MetaPost`.

Пример 132

Пример иллюстрирует использование функции `slice firstcut(p,knife).before` и функции `slice firstcut(p,knife).after`. Они используются для выделения (вырезания) фрагментов пути p до и после первого пересечения с другим путем `knife` (ножом).

Роль ножа, вырезающего зеленый фрагмент синего пути p, играет путь `knife` красного цвета.

```
unitsize(1cm);
path p=(-3.5,.2){(1,0)}..{(1,2)}(3,3);
/*задаем единицу длины и определяем
путь, из которого будет вырезаться фрагмент*/
path kn=(-2,-1){(1,4)}..(0,2)..{(1,-4)}(2,-1);
// определяем путь, играющий роль ножа
draw(p,bp+blue); draw(kn,red);
// рисуем путь и нож
path m1=firstcut(p,kn).before,
              m2=firstcut(p,kn).after;
/*определяем путь m1 (до первого пересечения)
и m2 (после первого пересечения)*/
draw(m1,2bp+green); draw(m2,2bp+blue);
// чертим фрагменты и наносим метки
label("$p$",point(p,1),3*N);
label("$knife$",point(kn,1),3*N);
label("$m1$",point(p,.2),3*S);
label("$m2$",point(p,.4),2*S);
shipout(bbox(3mm,invisible));
// чертим белые поля в 3mm
```

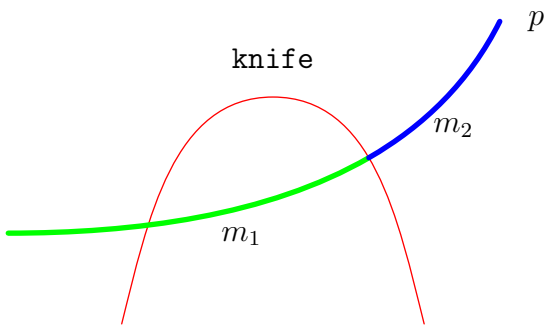


Пример 133

Аналогичен предыдущему. В нем показано применение функций `slice lastcut(p,knife).before` и `slice lastcut(p,knife).after`. Они используются для выделения (вырезания) фрагментов пути p до и после последнего пересечения с другим путем `knife` (ножом).

```
unitsize(1cm);// задаем единицу длины
path p=(-3.5,.2){(1,0)}..{(1,2)}(3,3);// определяем путь, из
// которого будет вырезаться фрагмент
path kn=(-2,-1){(1,4)}..(0,2)..{(1,-4)}(2,-1);// определяем нож
draw(p,bp+blue);draw(kn,red);// рисуем путь и нож
path m1=lastcut(p,kn).before, m2=lastcut(p,kn).after;
// определяем пути m1(до последнего пересечения)
// и m2(после последнего пересечения)
draw(m1,2bp+green); draw(m2,2bp+blue);// чертим фрагменты
```

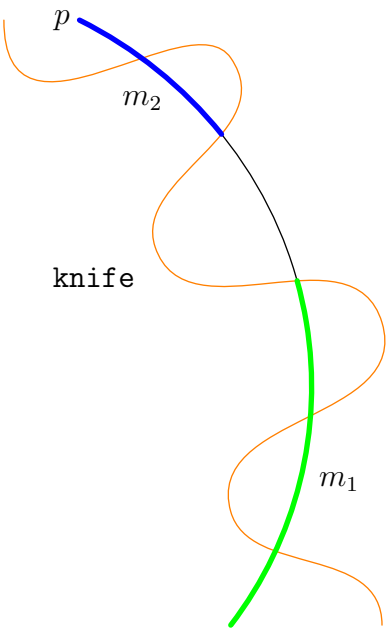
```
label("$p$",point(p,1),3*N);// наносим метку пути
label("\texttt{knife}",point(kn,1),3*N);// наносим метку нож
label("$m1$",point(p,.4),2*S);// наносим метку фрагмента
label("$m2$",point(p,.85),3*S);// наносим метку фрагмента
shipout(bbox(3mm,invisible));// чертим рамку в 3mm
```



Пример 134

В данном примере путь-нож пересекает путь p более, чем два раза. В примере выделен зеленый фрагмент до третьего пересечения и синий фрагмент после четвертого пересечения.

```
unitsize(1cm);
path p=(2,0)..(3,4)..{(-2,1)}(0,8);
/* задаем единицу измерения и путь */
pair A=(4,0), B=(2,1.5), C=(4,4), D=(1,5),
                      E=(2,7.5), F=(-1,8);
path kn=A{dir(90)}..B..C..D..E..{dir(90)}F;
/*задаем точки и путь, играющий роль ножа*/
//---- определения фрагментов
path m1=cut(p,kn,2).before;/*определяем
зеленый фрагмент*/
path m2=cut(p,kn,3).after;/*определяем
синий фрагмент*/
/* далее чертим пути и фрагменты */
draw(p);
draw(kn,orange);
draw(m1,2bp+green);
draw(m2,2bp+blue);
/* ставим метки */
label("$p$",point(p,2),W);
label("\texttt{knife}",point(kn,3),3*SW);
label("$m_1$",point(p,.5),2*dir(0));
label("$m_2$",point(p,1.65),3*W);
shipout(bbox(3mm,invisible));
```



Обратите внимание, что нумерация точек пересечения начинается с нуля. Поэтому, фрагмент от начала до третьей точки пересечения пути p и ножа `knife` задается именно так:

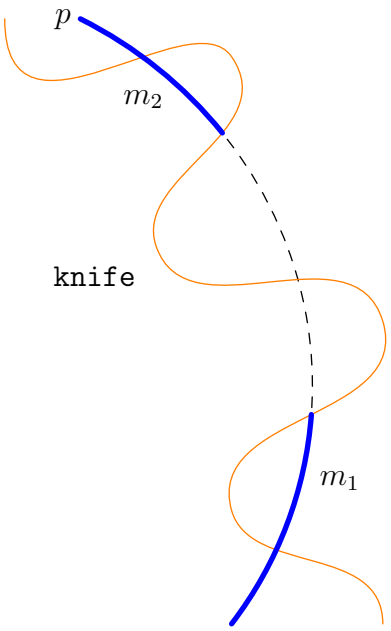
```
path m1=cut(p,kn,2).before;
```

От четвёртой точки до конечной так:

```
path m2=cut(p,kn,3).after;
```

Пример 135

Функцию `slice cut(path p, path knife, int n);` можно использовать для выделения (удаления) части пути. В данном примере пунктиром выделена часть пути p от второй до четвертой точки пересечения с путём `knife`.



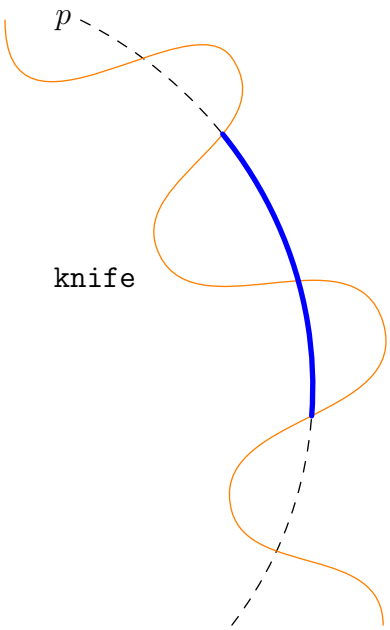

```
unitsize(1cm);
pair A=(4,0), B=(2,1.5), C=(4,4), D=(1,5),
E=(2,7.5), F=(-1,8);
/*---- Определяем пути и фрагменты ---*/
path p=(2,0)..(3,4)..{(-2,1)}(0,8);
path kn=A{dir(90)}..B..C..D..E..{dir(90)}F,
     m1=cut(p,kn,1).before,
     m2=cut(p,kn,3).after;
/*---- чертим пути и фрагменты ---*/
draw(p,dashed);
draw(kn,orange);
draw(m1,2bp+идгу);
draw(m2,2bp+blue);
label("$p$",point(p,2),W);
label("\texttt{knife}",point(kn,3),3*SW);
label("$m_1$",point(p,.5),2*dir(0));
label("$m_2$",point(p,1.65),3*W);
shipout(bbox(3mm,invisible));
```

Покажем, как можно выделить внутренний фрагмент одного из путей между точками пересечения этих двух путей.

Пример 136

В этом примере путь p выделен пунктиром, а средний фрагмент между второй и четвертой точкой пересечения p с путем `knife` выделен пером синего цвета

```
unitsize(1cm);
pair A=(4,0), B=(2,1.5), C=(4,4), D=(1,5),
E=(2,7.5), F=(-1,8);
/* определяем пути p и knife*/
path p=(2,0)..(3,4)..{(-2,1)}(0,8);
path kn=A{dir(90)}..B..C..D..E..{dir(90)}F;
draw(p,dashed); /* чертим путь p */
draw(kn,orange); /* чертим путь knife */
real [] [] A = intersections(p,kn);
/* определяем массив времен точек
пересечения */
path q=subpath(p,A[1][0],A[3][0]);
/* определяем фрагмент */
draw(q,2bp+blue); /*чертим фрагмент */
label("$p$",point(p,2),W);
label("\texttt{knife}",point(kn,3),3*SW);
shipout(bbox(3mm,invisible));
```



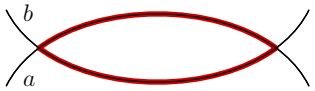
2.4.5 Создание замкнутых путей

Довольно часто возникает необходимость создать замкнутый путь из кусочков уже готовых, имеющихся путей. Рассмотренные функции пересечения путей и создания подпути позволяют это сделать. Однако в Асимптоте предусмотрена очень удобная функция `buildcycle`, подходящая как раз для решения этой задачи.

Рассмотрим применение ее на очень простом примере:

Пример 137

```
unitsize(1cm);
path a,b;
a=(-2,-.5){(.5,1)}..tension 1.3 .. (2,-.5){.5,-1};
b=reflect((-1,0), (1,0))*a;
draw(a^^b);
draw(buildcycle(a,b),2pt+.8red);
draw(scale(.75)*Label("$a$",
                      position=Relative(.05)),a);
draw(scale(.75)*Label("$b$",
        position=Relative(.05),align=LeftSide),b);
```

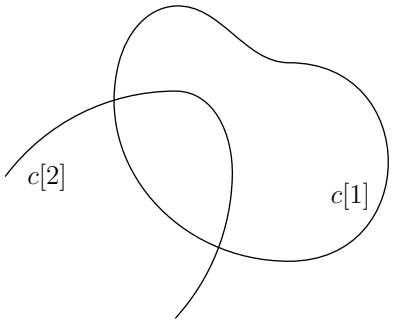


Если поменять местами аргументы a и b функции `buildcycle`, можно заметить, что рисунок остался прежним.

Но так бывает далеко не всегда.

Пример 138

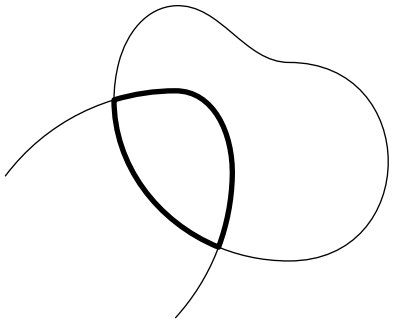
Разберём более сложный пример, когда незамкнутый путь $c[1]$ разбивает область, ограниченную замкнутым путем $c[2]$, на две области. Вот рисунок, иллюстрирующий такую ситуацию:



Чтобы получить замкнутый путь, ограничивающий одну из двух областей, нужно применить функцию `buildcycle`.

```
unitsize(1cm);// Задаем пользовательскую единицу.
// Следующие две строки кода определяют два массива точек.
pair [] A; A[4]=(0,2); A[1]=(1,3); A[2]=(3,2); A[3]=(3,-1.5);
pair [] B; B[1]=(-2,0); B[2]=(1,1.5); B[3]=(2,0); B[4]=(1,-2.5);

path [] c;// определяем массив путей.
// Следующие строки задают два пути из этого массива, причем путь c[1] - замкнутый.
c[1] = A[1]..{(1,0)}A[2]..{(-1,0)}A[3]..A[4]..cycle;
c[2] = B[1]..{(1,0)}B[2]..B[3]..B[4];

c[3] = buildcycle(c[1],c[2]);// Определяем замкнутый путь c[3], используя buildcycle.

draw(c[1]^^c[2]);// Рисуем исходные пути.
draw(c[3], 2bp+black);// Рисуем замкнутый путь черным жирным пером.
```

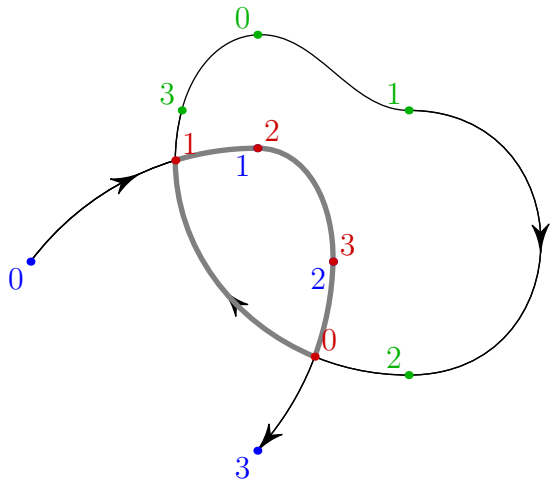
Прокомпилируем код и получим рисунок:



На рисунке мы видим два пути, один из которых замкнутый. Жирным черным пером выделен замкнутый путь, полученный применением функции `buildcycle` к этим двум путям.

Внимательно проанализируем код, который производит этот рисунок.

Чтобы понять механизм действия `buildcycle`, нанесем на рисунок узлы и направления путей. Узлы первого пути обозначены зелеными цифрами, а направление обхода пути указано стрелкой. Узлы второго пути обозначены синими цифрами. Направление также указано стрелками. (На самом деле указание стрелок избыточно, направление определяется по возрастанию номеров узлов)



В примере у функции `buildcycle` два аргумента `c[1]` и `c[2]`, причем их порядок имеет значение. Движение пера начинается с нулевого узла в указанном направлении пути. Этот нулевой узел определяется следующим образом.

Поскольку первый аргумент `buildcycle` — это `c[1]`, движемся вдоль первого пути от нулевой (зеленой) точки первого пути до первой точки пересечения его с путем `c[2]`.

Эта точка пересечения (начальная точка формируемого замкнутого пути) на рисунке обозначена нулем красного цвета. От неё перо движется вдоль первого пути до второй точки пересечения путей. Она получает номер 1 красного цвета. Далее перо перемещается вдоль второго пути по соответствующему направлению, проходя через узлы 1 и 2 второго пути (синего цвета). Эти узлы становятся узлами нового замкнутого пути, поэтому получают номера 2 и 3 красного цвета. Движение пера заканчивается в начальной точке с номером 0.

Мы получили замкнутый путь, выделенный жирным пером серого цвета. Полный код, создающий картинку с нумерацией узлов и стрелками, выглядит так:

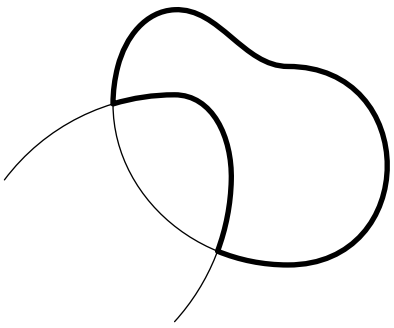
```
unitsize(1cm);
pair [] A; A[4]=(0,2); A[1]=(1,3); A[2]=(3,2); A[3]=(3,-1.5);
pair [] B; B[1]=(-2,0); B[2]=(1,1.5); B[3]=(2,0); B[4]=(1,-2.5);
path [] c;
```

```
c[1] = A[1]..{(1,0)}A[2]..{(-1,0)}A[3]..A[4]..cycle;
c[2] = B[1]..{(1,0)}B[2]..B[3]..B[4];
c[3] = buildcycle(c[1],c[2]);

draw(subpath(c[1],1,2),MidArrow(HookHead));
draw(subpath(c[1],2,3),MidArrow(HookHead));
draw(subpath(c[2],0,1),MidArrow(HookHead));
draw(c[2],EndArrow(HookHead));
draw(c[1]^^c[2]);
draw(c[3], 2bp+gray);
for(int i=0; i<length(c[1]); ++i)
    dot(string(i),point(c[1],i),NW, .7green);
for(int i=0; i<=length(c[2]); ++i)
    dot(string(i),point(c[2],i),SW, blue);
for(int i=0; i<length(c[3]); ++i)
    dot(string(i),point(c[3],i),NE, .8red);
shipout(bbox(3mm,invisible));
```

Пример 139

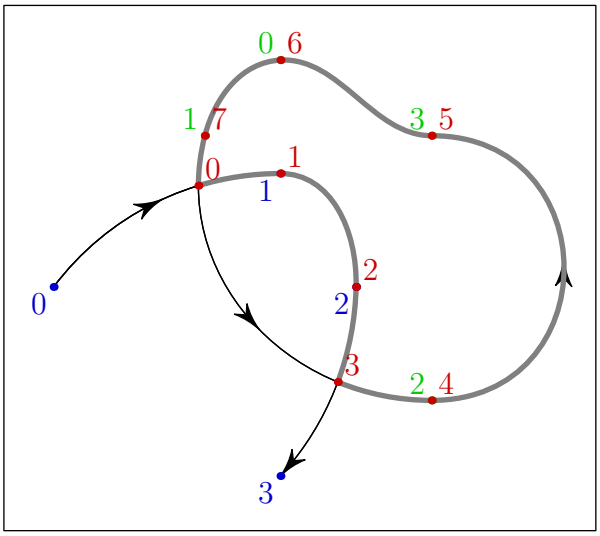
Начертим второй замкнутый путь, который определяется пересечением путей `c[1]` и `c[2]`. Вот код картинки:

```
unitsize(1cm);
pair [] A; A[4]=(0,2); A[1]=(1,3); A[2]=(3,2); A[3]=(3,-1.5);
pair [] B; B[1]=(-2,0); B[2]=(1,1.5); B[3]=(2,0); B[4]=(1,-2.5);
path [] c;
c[1] = A[1]..{(1,0)}A[2]..{(-1,0)}A[3]..A[4]..cycle;
c[2] = B[1]..{(1,0)}B[2]..B[3]..B[4];
c[3] =reverse(c[1]);
c[4] = buildcycle(c[2],c[3]);
draw(c[2]^^c[3]); draw(c[4], 2bp+black);
shipout(bbox(3mm));
```



Как можно заметить, вместо пути `c[1]` используется путь `c[3]`, который отличается от первого только направлением обхода. А еще важно, что порядок аргументов другой: сначала идет путь `c[2]`, а затем путь `c[3]`.

Как и в предыдущем примере, нанесем на чертеж узлы и направления обхода путей. Поскольку первым аргументом является путь `c[2]`, то новый циклический путь начинается с нового узла, обозначенного красным нулем. Затем перо движется вдоль пути `c[2]` до точки с красным номером 3, а затем поворачивает вдоль пути `c[3]`, двигаясь вдоль него через точки с красными цифрами 4, 5, 6 и 7 до исходной точки 0.



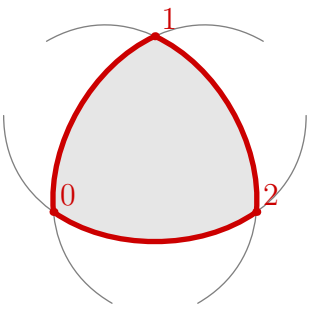
Приведём подробный код, рисующий эту картинку:

```
unitsize(1cm);
pair [] A; A[4]=(0,2); A[1]=(1,3); A[2]=(3,2); A[3]=(3,-1.5);
pair [] B; B[1]=(-2,0); B[2]=(1,1.5); B[3]=(2,0); B[4]=(1,-2.5);
path [] c;
c[1] = A[1]..{(1,0)}A[2]..{(-1,0)}A[3]..A[4]..cycle;
c[2] = B[1]..{(1,0)}B[2]..B[3]..B[4];
c[3] =reverse(c[1]);
c[4] = buildcycle(c[2],c[3]);
draw(subpath(c[3],1,2),MidArrow(HookHead));
draw(subpath(c[3],2,3),MidArrow(HookHead));
draw(subpath(c[2],0,1),MidArrow(HookHead));
draw(c[2],EndArrow(HookHead));
draw(c[3]^^c[2]);
draw(c[4], 2bp+gray);
for(int i=0; i<length(c[3]); ++i) {dot(string(i),point(c[3],i),NW, .8green);}
for(int i=0; i<=length(c[2]); ++i){dot(string(i),point(c[2],i),SW, .8blue);}
for(int i=0; i<length(c[4]); ++i) {dot(string(i),point(c[4],i),NE, .8red);}
shipout(bbox(3mm));
```

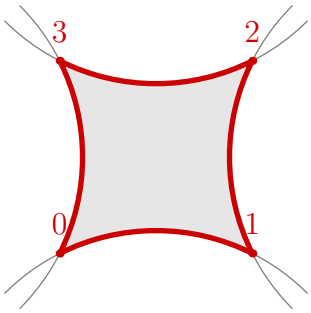
Функция `buildcycle` может иметь более двух аргументов.

Пример 140

```
unitsize(1cm);
path [] c;
c[1] = (-2,-.5){up}..tension 1.2..(2,-.5){down};
c[2] = rotate(60)*c[1];
c[3] = rotate(-60)*c[1];
c[4] = rotate(180)*c[1];
draw(c[2]^^c[3]^^c[4],grey);
c[5] = buildcycle(c[2],c[3],c[4]);
fill(c[5], lightgrey);
draw(c[5], .8red+2bp);
for(int i=0; i<length(c[5]); ++i)
    {dot(string(i),point(c[5],i),NE, .8red);}
shipout(bbox(3mm));
```



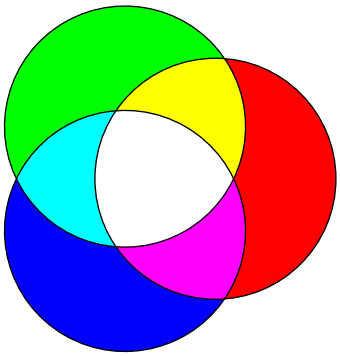
Пример 141



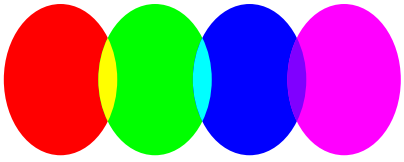

```
unitsize(1cm);
path [] c; path [] h;
c[1] = (-2,0){(1,1)}..{(1,-1)}(2,0);
h[1] = shift(0,-1.8)*c[1];
h[2] = shift(1.8,0)*rotate(90)*c[1];
h[3] = shift(0,1.8)*rotate(180)*c[1];
h[4] = shift(-1.8,0)*rotate(-90)*c[1];
draw(h[1]^^h[2]^^h[3]^^h[4],grey);
c[5] = buildcycle(h[1],h[2],h[3],h[4]);
fill(c[5], lightgrey);
draw(c[5], .8red+2bp);
for(int i=0; i<length(c[5]); ++i)
    {dot(string(i),point(c[5],i),2*N, .8red);}
shipout(bbox(3mm));
```

Пример 142

```
size(5cm,0);
path a,b,c;
a = shift(1,0)*scale(2)*unitcircle;
b = rotate(120)*a;
c = rotate(120)*b;
fill(a, red);
fill(b, green);
fill(c, blue);
fill(buildcycle(a,b), red + green);
fill(buildcycle(b,c), green + blue);
fill(buildcycle(c,a), blue + red);
fill(buildcycle(a,b,c), white);
draw(a^^b^^c);
```



Пример 143



```
unitsize(1cm);
pen[] p={red,green,blue,magenta};
path [] s;
for(int i=0; i<=3;++i){
s[i]=shift(1.25*i,0)*xscale(.75)*unitcircle;
fill(s[i],p[i]);}
for(int i=0; i<=2;++i){
fill(buildcycle(s[i+1],s[i]),p[i]+p[i+1]);}
shipout(bbox(4mm));
```

2.5 Метки. Расширенные возможности

После того, как создан рисунок или прямо в процессе создания, на него можно наложить текст, какие-либо обозначения, формулы и тому подобное. Всё это мы будем называть **метками**.

Добавлять метки к рисунку можно командой `label`. Вот две первые формы команды:

```
void label(picture pic=currentpicture, Label L, pair position,
           align align=NoAlign, pen p=nullpen, filltype filltype=NoFill)
```

или

```
void label(picture pic=currentpicture, Label L, pair position,
           align align=NoAlign, pen p=currentpen, filltype filltype=NoFill)
```

Разъясним аргументы команд.

- `picture pic=currentpicture` — метка наносится на текущую картинку.
- `Label L` — сама метка. Метка `L` может быть либо строкой, либо структурой, полученной вызовом одной из функций

  ```
  Label Label(string s="", pair position, align align=NoAlign,
              pen p=nullpen, embed embed=Rotate, filltype filltype=NoFill);

  Label Label(string s="", align align=NoAlign,
              pen p=nullpen, embed embed=Rotate, filltype filltype=NoFill);

  Label Label(Label L, pair position, align align=NoAlign,
              pen p=nullpen, embed embed=L.embed, filltype filltype=NoFill);

  Label Label(Label L, align align=NoAlign,
              pen p=nullpen, embed embed=L.embed, filltype filltype=NoFill);
  ```

- `pair position` — точка привязки метки в пользовательских координатах.
- `align align=NoAlign` — аргумент `align` позиционирует метку относительно точки привязки, то есть сдвигает центр прямоугольного бокса, содержащего метку, в некотором направлении. Направление можно задать различным образом.

  Метка будет смещена на `PostScript` расстояние `align*labelmargin(p)`. Постоянную `Align` можно использовать для выравнивания метки по нижнему левому углу в точке `position`.

  Если `align` есть `NoAlign`, центр прямоугольного бокса с меткой совпадет с точкой привязки, указанной в `position`.
- `pen p=nullpen` — если перо `p` есть `nullpen`, то будет использовано перо, указанное в `Label`, которое по умолчанию является `currentpen`.
- `pen p=currentpen` — используется текущее перо.
- `filltype filltype=NoFill` — указывается тип заполнения бокса с меткой. (`NoFill` — бокс не заполняется) Подробности далее в примерах.

Текст метки `Label` можно подвергнуть преобразованиям, таким как гомотетия, косой сдвиг, поворот или, например композиции сдвига и некоторого аффинного преобразования.

Аргумент `embed` определяет, как метку следует преобразовать при наложении на рисунок:

`Shift`	только сдвиг;
`Rotate`	только сдвиг и поворот (по умолчанию);
`Rotate(pair z)`	поворот на вектор z;
`Slant`	только сдвиг, поворот, косой сдвиг и отражение;
`Scale`	сдвиг, поворот, косой сдвиг, отражение и гомотетия;

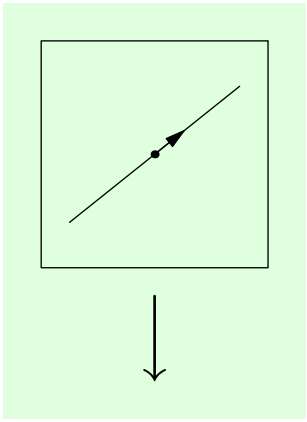
Переходим к примерам, продвигаясь от простого к сложному. В первых примерах покажем, как присоединить к несложной картинке какие-либо буквенные обозначения.

Пример 144

Для понимания работы с метками рассмотрим, например, рисунок из книги:
Д.В.Алексеевский, А.М.Виноградов, В.В.Лычагин **Основные идеи и понятия дифференциальной геометрии**.

Приводим код `Asymptote` этого рисунка:

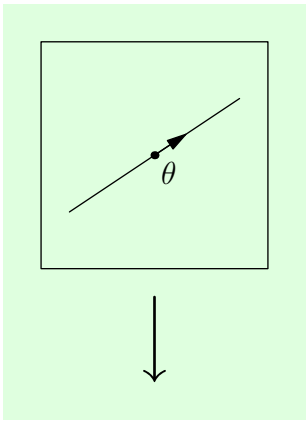
```
unitsize(1cm);
real a=1.5;
pair [] z;z[0]=(0,0);z[1]=(-a,-a);
z[2]=(-a,a);z[3]=(a,a);z[4]=(a,-a);
z[5]=(-.75a,-.6a);z[6]=(.75a,.6a);
z[7]=(0,-1.25a);z[8]=(0,-2a);
z[9]=z[0]+.5*dir(z[6]-z[5]);
z[10]=.5(z[7]+z[8]);z[11]=.5(z[1]+z[2]);
z[12]=.5(z[2]+z[3]);z[13]=.5(z[3]+z[4]);
path q=z[1]--z[2]--z[3]--z[4]--cycle;
path l=z[5]--z[6];path s=z[7]--z[8];
path st=z[0]--z[9];
draw(q); draw(l);
draw(s,linewidth(bp),Arrow(TeXHead,2bp));
draw(st,Arrow);dot(z[0]);
```



Теперь к этой картинке будем последовательно присоединять буквенные метки. Но для экономии места мы не будем далее показывать код самого рисунка, а будем указывать только код добавляемых меток.

Начнём с самой простой метки, буквы θ, которую присоединим к центру квадрата, точке `z[0]`.

Пример 145



Добавим к предыдущему коду строку с `label`.

```
label("$\theta$",z[0],SE);
```

После имени команды `label` в круглых скобках указаны три аргумента:

- первым аргументом является собственно строка с меткой "θ", она имеет тип `string`. Метка, которая будет изображена — это всё, что стоит внутри двойных кавычек.Здесь это греческая буква θ.
- второй аргумент — `pair position=z[0]` (точка привязки `z[0]`).
- третий аргумент, `align=SE` или просто `SE`, служит для позиционирования буквы θ относительно точки привязки. В данном случае выбраны компасные обозначения.

Замечания:

1. В примере 145 приведен усечённый список аргументов команды `label`. Полный список насчитывает шесть аргументов:

   ```
   void label(picture pic=currentpicture, Label L, pair position,
              align align=NoAlign, pen p=currentpen, filltype filltype=NoFill)
   ```

   Из этих шести аргументов в примере 145 три аргумента команды `label`, а именно:

   `picture pic=currentpicture`, `pen p=currentpen` и `filltype filltype=NoFill)` используются по умолчанию, а аргументы `Label L="`θ`"`, `pair position=z[0]` и `align=SE` указаны явно.

2. Обратите внимание, что вместо `Label L="θ"` пишем просто `"θ"`, вместо `pair position=z[0]` пишем `z[0]`, а вместо `align=SE` указываем просто `SE`.

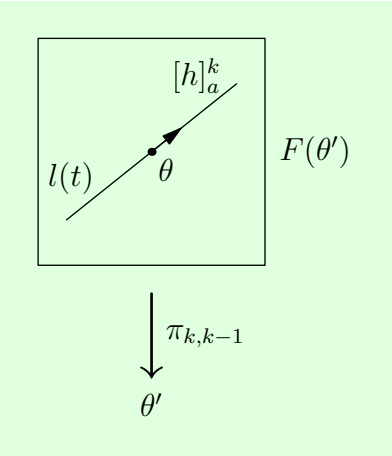
Пример 146

Добавим к нашему рисунку ещё две метки, "θ'" и "$\pi_{k,k-1}$". Действуем аналогично.

```
label("$\theta^{\prime}$",z[8],1.5S);
label("$\pi_{k,k-1}$",z[10],1.5E);
```

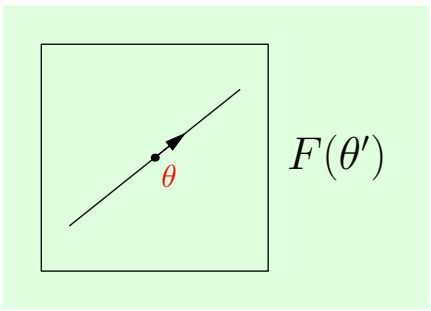
Отметим ещё один новый момент: `align=1.5E`. Числовой множитель позволяет нам изменять расстояние между меткой и точкой привязки. Затем добавим последние три метки.

```
label("$l(t)$",z[11],1.5SE);
label("$[h]^{k}_{a}$",z[12],3SE);
label("$F(\theta^{\prime})$",z[13],1.5E);
```



Приступим к изучению ещё двух аргументов: `pen` и `filltype`. Что касается пера, которым рисуется метка, то мы можем, например, изменить цвет пера, используя цветовые пространства RGB, CMYK и другие. Приведём пример.

Пример 147

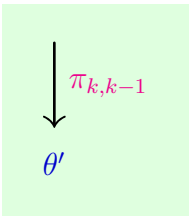


Определим вместо пера по умолчанию перо с нужными нам свойствами. Так, например, определим перо красного цвета и вставим его как аргумент команды `label`.

```
pen p=red;
label("$\theta$",z[0],SE,p);
```

Определим перо с размером кегля шрифта `16pt`.

```
pen r=fontsize(16pt);
string f="$F(\theta^{\prime})$";
label(f,z[13],1.5E,r);
```



Для экономии места покажем только нужную часть рисунка. Определим перо из цветового пространства RGB:

```
pen q1=rgb(0,0,.8);
label("$\theta^{\prime}$",z[8],1.5S,q1);
```

Определим перо из цветового пространства CMYK:

```
pen q2=cmyk(0, 1, 0, 0);
label("$\pi_{k,k-1}$",z[10],1.5E,q2);
```

В следующем примере показывается, что можно задавать перо сразу с несколькими свойствами. Например, можно увеличить кегль шрифта и одновременно изменить цвет пера.

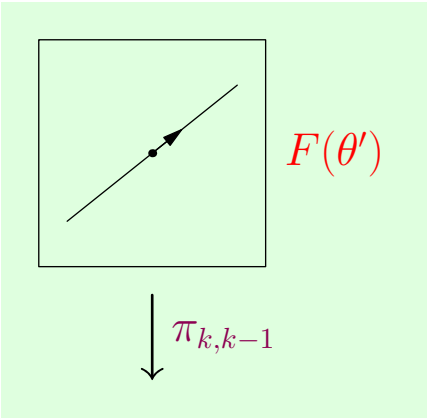
Пример 148

Скомбинировать можно, например, с помощью оператора (+). Приводим код `Asymptote`:

```
pen r=fontsize(16pt)+red;
string f="$F(\theta^{\prime})$";
label(f,z[13],1.5E,r);
```

Ещё перо из цветового пространства CMYK.

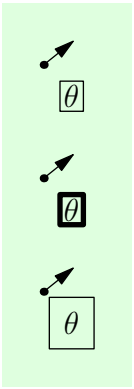
```
pen t=fontsize(16pt)+cmyk(0, 1, 0, .5);
string f="$\pi_{k,k-1}$";
label(f,z[10],1.5E,t);
```



Рассмотрим, наконец, аргумент `filltype`, который определяет способ заполнения бокса, содержащего метку. Аргумент `filltype` может принимать следующие значения:

- `filltype filltype=NoFill` — употребляется по умолчанию. Бокс с меткой не заполняется, рисуется только сама метка. На всех предыдущих картинках было использовано именно `filltype=NoFill`!
- `filltype filltype=Draw()` — позволяет начертить рамку по границе бокса с меткой. Внутри круглых скобок можно указать нужные параметры пера, которым будет начерчена рамка. Если круглые скобки пусты, рамка чертится пером по умолчанию.

Пример 149



Начертим рамку пером по умолчанию (currentpen):

```
string s0="$\theta$";
label(s0,z[0],3SE,filltype=Draw());
```

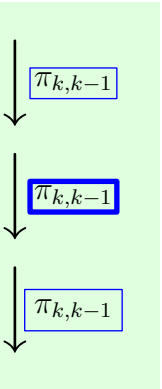
Увеличим толщину пера, рисующего рамку:

```
label(s0,z[0],3NW,filltype=Draw(linewidth(2bp)));
```

Увеличим размер бокса, раздвинув границы:

```
label(s0,z[0],3SE,filltype=Draw(4bp));
```

Пример 150



Изменим цвет пера на синий и начертим рамку.

```
string s2="$\pi_{k,k-1}$";
label(s2,z[10],2E,filltype=Draw(blue));
```

Начертим рамку синим пером толщиной 2bp.

```
label(s2,z[10],2E,filltype=Draw(2bp+blue));
```

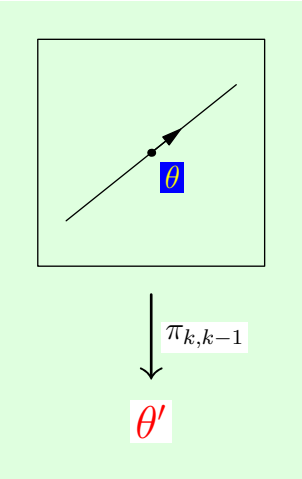
Заменим плюс на запятую:

```
label(s2,z[10],2E,filltype=Draw(2bp,blue));
```

Отыщите различие!

- `filltype filltype=Fill()` — бокс заполняется пером, указанным в круглых скобках.

Пример 151



Заполним бокс пером синего цвета, а саму метку — желтым пером.

```
string s0="$\theta$";
label(s0,z[0],2SE,yellow,filltype=Fill(blue));
```

Заполним бокс пером белого цвета, а саму метку — пером по умолчанию.

```
string s1="$\pi_{k,k-1}$";
label(s1,z[10],1.5E,filltype=Fill(white));
```

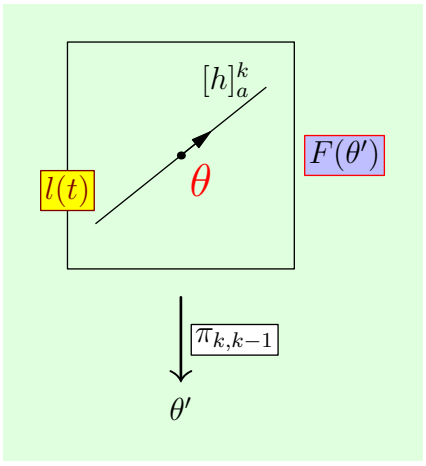
Заполним бокс пером белого цвета, а саму метку — указанным пером p.

```
pen p=fontsize(16pt)+red;
string s1="$\theta^{\prime}$";
label(s1,z[8],2S,p,filltype=Fill(white));
```

- `filltype filltype=FillDraw(pen fillpen, pen drawpen)` — бокс, содержащий метку, заполняется пером `pen fillpen`, а пером `pen drawpen` чертится рамка.

Пример 152

```
pen p=fontsize(16pt)+red;
label("$\theta$",z[0],SE,p);
label("$\theta^{\prime}$",z[8],1.5S);
label("$[h]^{k}_{a}$",z[12],3SE);
// ====================================
string f="$\pi_{k,k-1}$";
label(f,z[10],1.5E,filltype=
                  FillDraw(white,black));
// ====================================
string g="$F(\theta^{\prime})$";
label(g,z[13],1.5E,filltype=
                 FillDraw(paleblue,red));
// ====================================
label("$l(t)$",z[11],2S,brown,
        filltype=FillDraw(yellow,brown));
// ====================================
shipout(bbox(.5cm,Fill(palegreen+white)));
```



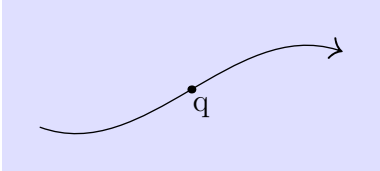
Для присоединения метки к пути используют команду:

```
void label(picture pic=currentpicture, Label L, path g, align align=NoAlign,
           pen p=nullpen, filltype filltype=NoFill);
```

Пример 153

Пусть у нас есть криволинейный путь q. Пометим путь меткой — именем пути. *По умолчанию метка будет находиться в середине пути.* Стрелкой укажем направление возрастания параметра пути.

```
unitsize(1cm);
path q=(-2,0){dir(-20)}..{dir(-20)}(2,1);
draw(q,Arrow(TeXHead,2bp));
label("q",q);//Зададим метку
dot(point(q,.5));// Отметим середину пути точкой.
```



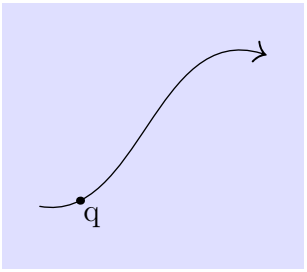
Как видно на рисунке, по умолчанию метка располагается справа от пути, если смотреть в направлении возрастания параметра. Иное положение метки (в смысле `point(path p, real t)`) может быть указано заданием действительного значения в конструкции метки.

Аргумент `Relative(real)` определяет место точки привязки метки по отношению к общей длине (arclength) пути. Есть предопределённые сокращения:

```
position BeginPoint=Relative(0);
position MidPoint=Relative(0.5);
position EndPoint=Relative(1);
```

Проиллюстрируем сказанное двумя примерами:

Пример 154



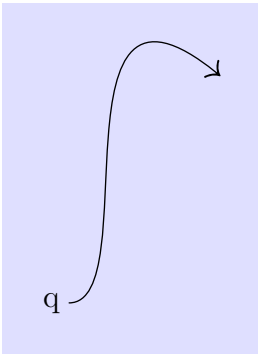
Укажем точку `point(q,.15)`, которая находится на расстоянии $0.15 arclength$ от начала пути. Чтобы «привязать» метку пути к данной точке, используем код:

```
unitsize(1cm);
path q=(-2,0){dir(0)}..{dir(-20)}(1,2.5);
draw(q,Arrow(TeXHead,2bp));
label(Label("q",Relative(.15)),q);
dot(point(q,.15));
```

Пример 155

Чтобы поставить метку q в начале пути, используем код с предопределённым сокращением `BeginPoint`, указанным выше :

```
unitsize(1cm);
path q=(-1,0){dir(30)}..{dir(-20)}(1,2);
draw(q,Arrow(TeXHead,2bp));
label(Label("q",BeginPoint),q);
```

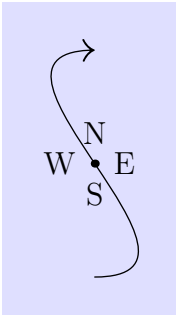


Метки путей можно позиционировать относительно точки привязки с помощью аргумента `align`, указав направление на метку (если смотреть на метку из точки привязки).

Допустим, что мы используем точку привязки в середине пути и позиционируем метку, используя «компасные» обозначения.

Далее примерах 156 и 157 показывается разница между двумя способами употребления аргумента `align`.

Пример 156

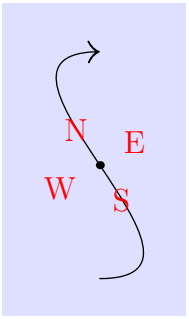


При первом способе направления N и S равносильны направлениям «вверх» и «вниз».

```
unitsize(1cm);
path q=(0,0){dir(0)}..{dir(0)}(0,3);
draw(q,Arrow(TeXHead,2bp));
dot(relpoint(q,.5));
label("N",q,2*N);
label("S",q,2*S);
label("W",q,2*W);
label("E",q,2*E);
```

Пример 157

При втором способе (метки красного цвета) линия север-юг направлена по той касательной к пути, которая проходит через точку привязки.




```
unitsize(1cm);
path q=(0,0){dir(0)}..{dir(0)}(0,3);
draw(q,Arrow(TeXHead,2bp));
dot(relpoint(q,.5));
label("N",q,3*Relative(N),red);
label("S",q,3*Relative(S),red);
label("W",q,3*Relative(W),red);
label("E",q,3*Relative(E),red);
```

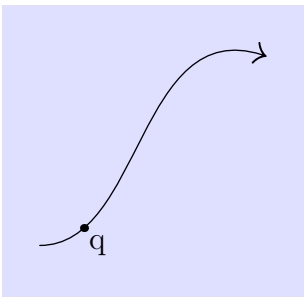
Заметим, что при втором способе употребления аргумента `align` направление на «север» соответствует возрастанию параметра пути.

Пример 158

Расположение метки по умолчанию, как, скажем, это сделано в примере 153, соответствует значению `align=Relative(E)`.

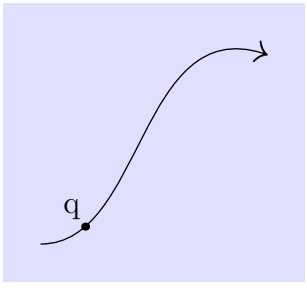
```
unitsize(1cm);
path q=(-2,0){dir(0)}..{dir(-20)}(1,2.5);
draw(q,Arrow(TeXHead,2bp));
label(Label("q",Relative(.15),Relative(E)),q);
dot(relpoint(q,.15));
```

Вместо `align=Relative(E)` используют сокращение `RightSide`.



Пример 159

Расположение метки слева от пути по направлению возрастания параметра можно получить, используя значение `align=Relative(W)`.



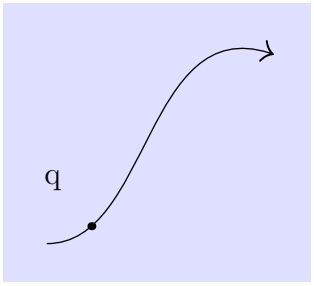

```
unitsize(1cm);
path q=(-2,0){dir(0)}..{dir(-20)}(1,2.5);
draw(q,Arrow(TeXHead,2bp));
label(Label("q",Relative(.15),Relative(W)),q);
dot(relpoint(q,.15));
```

Вместо `align=Relative(W)` используют сокращение `LeftSide`.

Умножение `LeftSide` и `RightSide` слева на действительный масштабирующий коэффициент будет отодвигать метку дальше от пути или приближать метку ближе к пути. Вот пример:

Пример 160

```
unitsize(1cm);
path q=(-2,0){dir(0)}..{dir(-20)}(1,2.5);
draw(q,Arrow(TeXHead,2bp));
label(Label("q",Relative(.15),5*Relative(W)),q);
dot(relpoint(q,.15));
```



Более гибкий способ указания направления на метку — использовать функцию `Relative(pair)`.

Попробуем разяснить это. Представим локальную декартову систему координат (правую) с центром в точке привязки, осью y, направленой по касательной к пути в сторону увеличения параметра. Тогда радиус вектор `pair` в этой локальной системе координат указывает направление на метку.

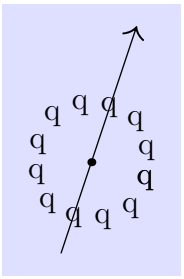
Этот способ является наиболее универсальным, наиболее общим, а все ранее приведённые примеры служат частными случаями применения этого способа.

Заметим, что вместо `position=Relative(real)` в коде можно писать просто `Relative(real)`, а вместо `align=Relative(pair)` писать `Relative(pair)`.

Вот два примера, из которых видно, насколько гибко можно использовать функцию `Relative(pair)`.

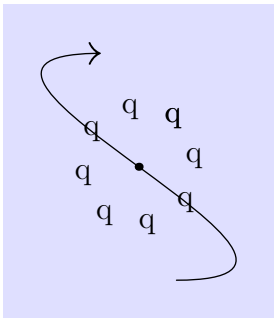
Пример 161

```
unitsize(1cm);
path q=(-1,0)--(0,3);
draw(q,Arrow(TeXHead,2bp));
for(int i=0;i<=12;++i){
label(Label("q",Relative(.4),5*Relative(dir(i*30))),q);
}
dot(relpoint(q,.4));
```



Пример 162

```
unitsize(1cm);
path q=(.5,0){dir(0)}..{dir(0)}(-.5,3);
draw(q,Arrow(TeXHead,2bp));
for(int i=0;i<=8;++i){
label(Label("q",Relative(.5),5*Relative(dir(i*45))),q);
}
dot(relpoint(q,.5));
```



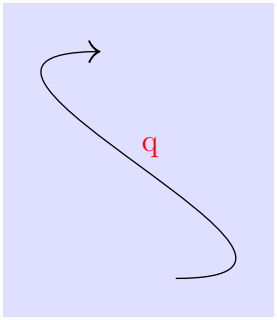
Сокращения `LeftSide, Center` и `RightSide`, которые приведены выше, заменяют соответственно `Relative(W)=Relative(-1,0)`, `Relative((0,0))` и `Relative(E)=Relative(1,0)`.

Нам осталось показать, как можно применять аргументы `pen` и `filltype`. Если мы используем их по умолчанию, это соответствует значениям

`pen p=nullpen` и `filltype filltype=NoFill`

При этом аргументы просто отсутствуют, их нет внутри круглых скобок команды `label`.

Пример 163



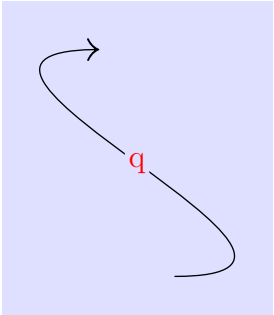
Чтобы вместо пера по умолчанию использовать другое, например перо красного цвета, достаточно после аргумента путь указать перо.

```
unitsize(1cm);
path q=(.5,0){dir(0)}..{dir(0)}(-.5,3);
draw(q,Arrow(TeXHead,2bp));
pen p=red;
label("q",q,p);// или label("q",q,red);
```

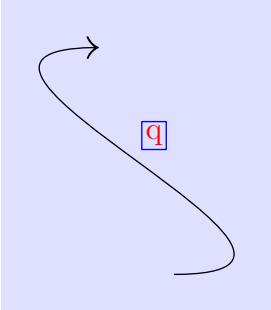
Пример 164

Для того, чтобы вставить метку, разорвав путь, нужен аргумент `filltype=UnFill`.

```
unitsize(1cm);
path q=(.5,0){dir(0)}..{dir(0)}(-.5,3);
draw(q,Arrow(TeXHead,2bp));
label(Label("q",Relative((0,0)),red,UnFill),q);
```



Пример 165



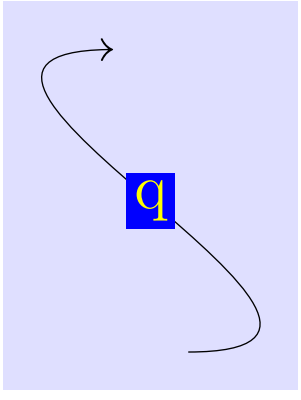
Метку можно поместить в рамку, имеющую форму прямоугольника. Вот соответствующий код:

```
unitsize(1cm);
path q=(.5,0){dir(0)}..{dir(0)}(-.5,3);
draw(q,Arrow(TeXHead,2bp));
label(Label("q",2*Relative(E),red,Draw(blue)),q);
```

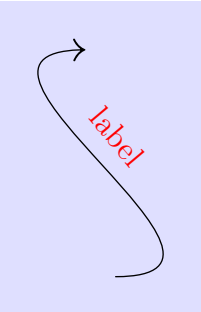
Пример 166

Можно увеличить метку и поместить её в закрашенный бокс:

```
unitsize(1cm);
path q=(.5,0){dir(0)}..{dir(0)}(-.5,3);
draw(q,Arrow(TeXHead,2bp));
label(scale(2)*Label("q",Center,yellow,Fill(blue)),q);
```



Пример 167



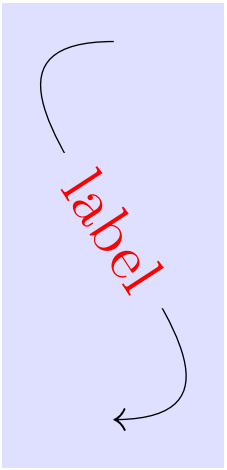
В этом примере метка поворачивается вокруг точки привязки и сдвигается так, чтобы быть параллельной пути.

```
unitsize(1cm);
path q=(.5,0){dir(0)}..{dir(0)}(-.5,3);
draw(q,Arrow(TeXHead,2bp));
label(shift(5,0)*rotate(-37)*Label("label",Relative(.55),red),q);
```

Пример 168

Разрываем путь и вставляем вдоль него метку, увеличенную в два раза.

```
unitsize(1cm);
path q=(0,0){dir(0)}..{dir(0)}(0,5);
path s=reverse(q);
draw(s,Arrow(TeXHead,2bp));
label(rotate(-58)*scale(2)*Label("label",Center,red,UnFill),s);
```



Ещё два примера, которые приоткроют некоторые новые возможности `Asymptote`. Они используют функции `string graphic` и `string minipage`.

1. Функция `string graphic`.

Функция `string graphic(string name, string options= "")` возвращает строку, которая может использоваться для включения в `asy`-файл encapsulated PostScript (EPS). Таким образом мы можем создавать с помощью `Asymptote` картинки, комбинируя программный код `Asymptote` и уже заранее созданные `eps`-картинки. Прокомментируем аргументы функции: здесь `name` — имя файла для включения и `options` есть строка, содержащая список разделенных запятыми опциональных параметров:

bounding box (`bb=llx lly urx ury`), ширина (`width=value`), высота (`height=value`), вращение (`angle=value`), масштабирования (`scale=factor`), обрезка (`clip=bool`) и draft режим (`draft=bool`).

Функция `layer`() может быть использована, чтобы вставить нарисованные объекты поверх включенного изображения.

На самом деле функция `string graphic` позволяет включать в `asy`-файл не только EPS, но и PDF, но при этом используется различная последовательность компилирования. В первом случае последовательность компилирования такая:

latex → asymptote → latex, а во втором — pdflatex → asymptote → pdflatex.

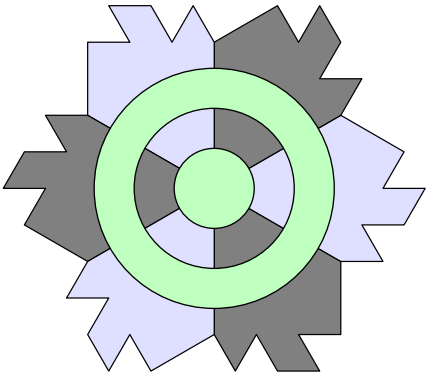
Пример 169

В первом примере мы имеем `pdf`-файл с именем `d6.pdf` и хотим наложить на него фигуру, представляющую собой концентрические круг и кольцо. Приведем код и полученное изображение. Этот код рисует круг и кольцо и вставляет готовое изображение многоугольной фигуры.

Замечания:

1) Обратите внимание на первую строку кода.

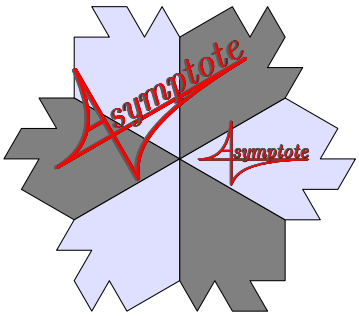
2) Компиляция по схеме pdflatex → asymptote → pdflatex.



```
settings.outformat="pdf";
frame f,g;
label(f,graphic("d6.pdf","width=6cm"),(0,0));
layer(g);
path K1=scale(45)*unitcircle;
path K2=scale(30)*unitcircle;
path K3=scale(15)*unitcircle;
filldraw(g,K1^^K2^^K3,palegreen+evenodd);
add(f);
add(g);
```

Пример 170

Особенность этого примера состоит в том, что код просто комбинирует заранее созданные `pdf`-картинки.




```
settings.outformat="pdf";
frame f,g,h;
label(f,graphic("d6.pdf","width=5cm"),(0,0));
label(g,graphic("Alogo.pdf","width=3cm"),(0,0));
label(h,graphic("Alogo.pdf","width=1.5cm"),(0,0));
add(f);
add(rotate(30)*shift(0,25)*g);
add(shift(28,2.2)*h);
```

2. Функция `string minipage`.

Функция `string minipage(string s, width=100pt)` используется для форматирования строки `s` указанной ширины `width`.

В пособии уже есть пример использования этой функции смотри Пример 95. Ещё один Пример 203 в подразделе 2.7.2.

2.6 Функции

С помощью функций мы можем сделать структуру нашей программы более соответствующей принципам структурного программирования.

Функция — это группа инструкций, которая имеет формат вида:

```
type name ( parameter1, parameter2, ...) { statements }
```

где:

- *type* — это тип данных, возвращаемый функцией.
- *name* — идентификатор, по которому вызвается функция.
- *parameters* (параметры): каждый параметр состоит из данных: спецификатор типа, затем идентификатор, как в любом обычном объявлении переменной (например: int x) и которая выступает в функции, как обычная локальная переменная. Они позволяют передавать аргументы функции во время вызова. Различные параметры разделяются запятыми.
- *statements* (инструкции) составляют так называемое тело функции. Этот блок инструкций заключают в фигурные скобки { }.

Полный список функций `Asymptote` можно найти по адресу: http://gmaths.net/asy/index/

Однако `Asymptote` позволяет создавать и новые, собственные функции. Перейдем к конкретным примерам создания функций.

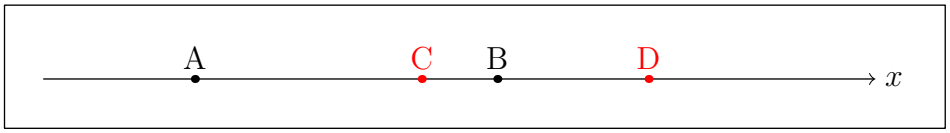
2.6.1 Примеры функций, созданных пользователем

Пример 171

Пусть A, B и C — точки на оси абсцисс. В примере определяется функция `harmonic`, которая вычисляет координату точки D, которая является гармонически сопряженной с точкой C.

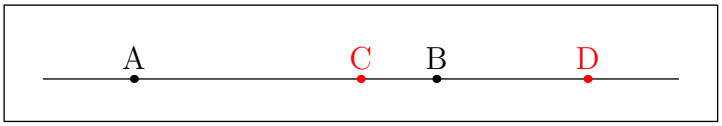
Затем на оси изображаются все четыре точки. Рассмотрим первый вариант кода:

```
unitsize(1cm);
//Задаем абсциссы точек
real a=-1.0; real b=3.0; real c=2.0;
// Определяем функцию harmonic
real harmonic(real a,real b,real c){
return (2*a*b-a*c-c*b)/(a+b-2*c);}
//Задаем исходные точки
pair A=(a,0); pair B=(b,0);pair C=(c,0);
path l=(-3,0)--(8,0);/* Задаем ось x и
затем чертим ее */
draw(Label("$x$",position=EndPoint),l,Arrow(TeXHead));
real d=harmonic(a,b,c);//Вычисляем координаты точки D
pair D=(d,0);/*Определяем точку D и затем изображаем
все четыре точки */
dot("A",A,N); dot("B",B,N);
dot("C",C,N,red); dot("D",D,N,red);
shipout(bbox(5mm));
```



Второй вариант кода использует пакет `geometry`:

```
import geometry;
unitsize(1cm);
real a=-1.0; real b=3.0; real c=2.0;
real harmonic(real a,real b,real c){
return (2*a*b-a*c-c*b)/(a+b-2*c);
}
point A=(a,0); point B=(b,0);
point C=(c,0);
line l=line(A,B);
linemargin=-10mm;
draw(l);
real d=harmonic(a,b,c);
point D=(d,0);
dot("A",A,N); dot("B",B,N); dot("C",C,N,red);
dot("D",D,N,red);
shipout(bbox(5mm));
```



Пример 172

Задаем три комплексных числа: z_0, z_1, z_2. С помощью `Asymptote` можно вычислить, например, произведение $z_0 z_1$ и изобразить соответствующую точку.

```
pair [] z={(2,1),(-1,1),(3,-1)};//задаем точки
```

Изображаем эти точки и соответствующие радиус-векторы.

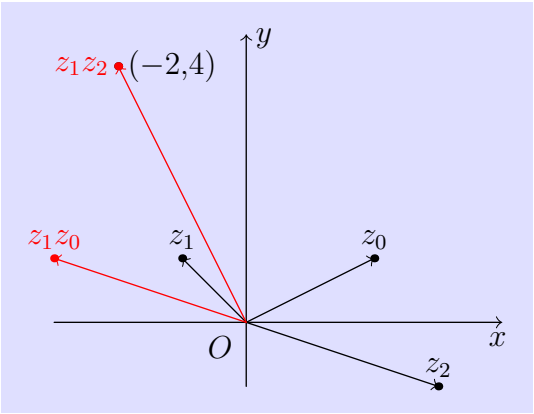
```
dot("$z_{0}$",z[0],N);
dot("$z_{1}$",z[1],N);
dot("$z_{2}$",z[2],N);
draw(O--z[0],Arrow(TeXHead));
draw(O--z[1],Arrow(TeXHead));
draw(O--z[2],Arrow(TeXHead));
```

Определим функцию, которая будет умножать комплексные числа.

```
//---------------
pair mc (pair a,pair b)
{
return (a.x*b.x-a.y*b.y, a.x*b.y+a.y*b.x);
}
//----------------
```

Изображаем результат умножения $z_1 z_0$ и $z_1 z_2$. Точки $z_1 z_0$ и $z_1 z_2$ и их радиус-векторы красного цвета.

```
draw(O--mc(z[0],z[1]),red,Arrow(TeXHead));
dot("$z_{1}z_{0}$",mc(z[0],z[1]),N,red);
dot("",mc(z[2],z[1]));
draw(O--mc(z[2],z[1]),red,Arrow(TeXHead));
dot("$z_{1}z_{2}$",mc(z[2],z[1]),W,red);
```



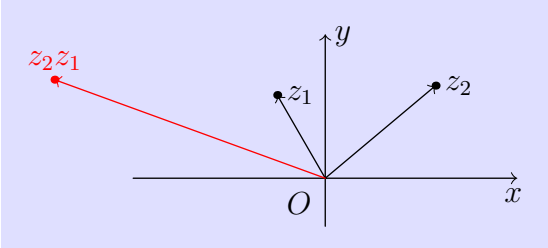
Пример 173

Для изображения произведения используем второй способ задания комплексных чисел.

```
/* задаем массив комплексных чисел (точек) */
real [] r={1, 2, 3};/* массив модулей этих чисел */
real [] dg={20, 120, 40};/* массив аргументов этих чисел */
pair [] z={r[0]*dir(dg[0]),r[1]*dir(dg[1]),r[2]*dir(dg[2])};
dot("$z_{1}$",z[1]);
dot("$z_{2}$",z[2]);
```

Для этого нужно определить совсем другую функцию:

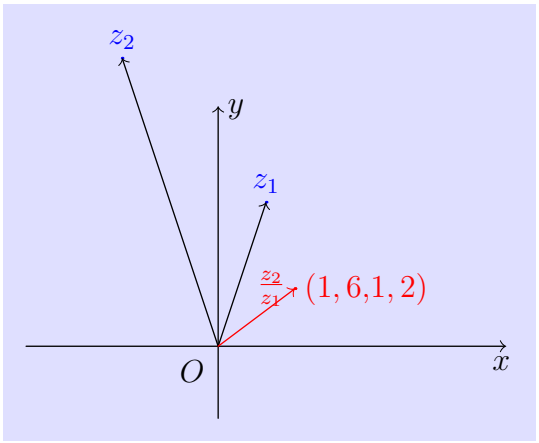
```
//---------------
pair cpm(pair a, pair b){
return length(a)*length(b)*dir(degrees(a)+degrees(b));
}
//---------------
draw(O--z[1],Arrow(TeXHead));
draw(O--z[2],Arrow(TeXHead));
draw(O--cpm(z[1],z[2]),red,Arrow(TeXHead));
dot("$z_{2}z_{1}$",cpm(z[1],z[2]),N,red);
```



Пример 174

Комплексные числа можно не только умножать, но и делить. Для визуализации деления можно применить функцию сопряжения `conj(pair z)`, а также `xpart(pair z)` и `ypart(pair z)`.

```
pair [] z={(3,1),(1,3),(-2,6)};
draw("$z_{2}$",z[2],N,bp+blue);
draw("$z_{1}$",z[1],N,bp+blue);
draw(O--z[2],Arrow(TeXHead));
draw(O--z[1],Arrow(TeXHead));
//---------------------------------
pair mc (pair a,pair b){
return (a.x*b.x-a.y*b.y, a.x*b.y+a.y*b.x);}
//---------------------------------
pair dc (pair a,pair b){
return (1/(b.x*b.x+b.y*b.y))*(xpart(mc(a,conj(b))), ypart(mc(a,conj(b))));}
//---------------------------------
pair  P=dc(z[2],z[1]);
draw("$\frac{z_{2}}{z_{1}}$",P,W,bp+red);
dot("",P,bp+red);
draw(O--P,red,Arrow(TeXHead));
```

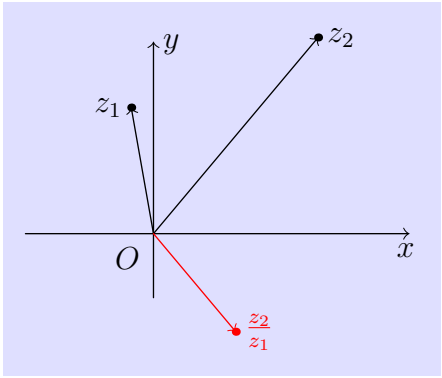


В случае использования полярных координат для задания комплексных чисел заменим функцию деления на соответствующую способу задания.

Пример 175

Задаем массив комплексных чисел (точек), то есть массив модулей и массив аргументов этих чисел:

```
real [] r={1, 2, 4};
real [] dg={20, 100, 50};
pair [] z={r[0]*dir(dg[0]),r[1]*dir(dg[1]),
                           r[2]*dir(dg[2])};
dot("$z_{1}$",z[1],W);
dot("$z_{2}$",z[2]);
pair dcp(pair a, pair b){ return(
length(a)/length(b))*dir(degrees(a)-degrees(b));
}
draw(O--z[1],Arrow(TeXHead));
draw(O--z[2],Arrow(TeXHead));
draw(O--dcp(z[2],z[1]),red,Arrow(TeXHead));
dot("$\frac{z_{2}}{z_{1}}$",dcp(z[2],z[1]),E,red);
```



Следующие три примера заимствованы из замечательной галереи примеров от Gaétan Marris

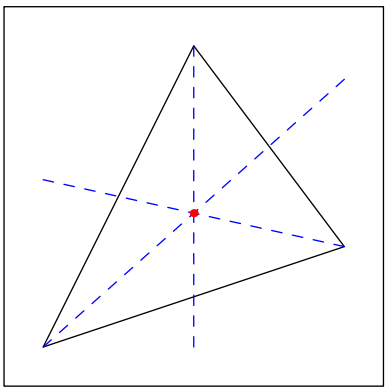
http://asy.marris.fr/asymptote

Пример 176

Пример иллюстрирует применение понятия центра тяжести для построения медиан треугольника. Мы определяем функцию, которая будет возвращать центр тяжести треугольника, заданного его вершинами.
Определяем функцию центр тяжести:

```
pair CentreGravite (pair Point1, pair Point2, pair Point3)
        {
          return (Point1+Point2+Point3)/3;
        }
```

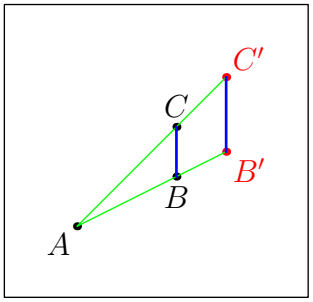
```
import math;
size(5cm,0);
// Задаем вершины треугольника
pair A=0, B=(3,1), C=(1.5,3);
// Чертим треугольник
path p=A--B--C--cycle; draw(p);
// Определяем центр тяжести треугольника
pair G=CentreGravite(A,B,C);
// Рисуем медианы, используя цикл
for (int i = 0; i <= 2; ++i)
{drawline(G,(point(p,i)+point(p,i+1))/2,
            .5bp+blue+dashed);}
dot(G,red);
// Выделяем центр тяжести красным цветом
```



Пример 177

В примере показывается получение образа точки при гомотетии. Для этого определяется функция `pair ImageParHom(pair M, real k, pair Centre)`, возвращающая изображение точки M при гомотетии с центром в точке `Centre` и коэффициентом k.

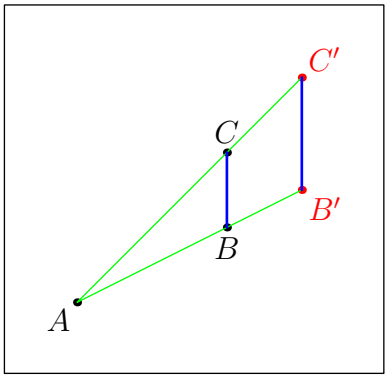
```
size(4cm,0);

pair ImageParHom(pair M, real k, pair Centre)
{
  return shift(k*(M-Centre))*Centre;
}
// Задаем три точки, изображем их и обозначаем
pair pA=(0,0),pB=(2,1),pC=(2,2);
dot("$A$",pA,SW);
dot("$B$",pB,S);
dot("$C$",pC,N);
// Определяем образы B' и C' для B и C
// при гомотетии с центром A и коэффициентом 1,5
pair imB=ImageParHom(pB,1.5,pA),
        imC=ImageParHom(pC,1.5,pA);
dot("$B'$",imB,SE,red);dot("$C'$",imC,NE,red);
// Изображаем и обозначаем их
draw(imB--pA--imC,green);
draw(pB--pC^^imB--imC,1bp+blue);
//Рисуем отрезок BC и его образ B'C'
```



Пример 178

Этот пример усовершенствует предыдущий, в нем гомотетия определяется как преобразование.

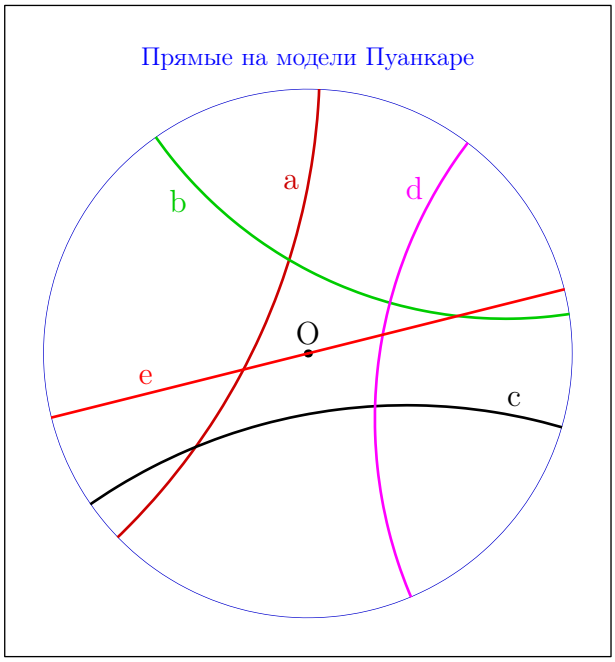
```
size(5cm,0);
// Определяем гомотетию как преобразование
transform Homothetie(real k, pair centre)
{
  return shift(centre)*scale(k)*shift(-centre);
}
// Задаем, изображаем и обозначаем три точки
pair pA=(0,0),pB=(2,1),pC=(2,2);
dot("$A$",pA,SW);
dot("$B$",pB,S);
dot("$C$",pC,N);
// Определяем h, гомотетию с центром pA
// и коэффициентом 1.5
transform h=Homothetie(1.5,pA);
// Определяем B'и C', образы B и C при  h
pair imB=h*pB, imC=h*pC;
// Изображаем и обозначаем B' и C'
dot("$B'$",imB,SE,red);dot("$C'$",imC,NE,red);
draw(imB--pA--imC,green);
// Чертим отрезок BC и его образ
draw(pB--pC^^imB--imC,1bp+blue);
```



Несколько примеров функций, связанных с моделью Пуанкаре планиметрии Лобачевского. Рассмотрим модель Пуанкаре в круге. На рисунке этот круг ограничен синей (базисной) окружностью. Точки плоскости Лобачевского моделируются внутренними точками круга. Прямые плоскости Лобачевского моделируются дугами окружностей, которые пересекают синюю окружность ортогонально. Диаметры также считаются прямыми(дугами окружностей бесконечного радиуса).

Пример 179

Определяемая в этом примере функция `lineLob(point M,point N)` типа `circle` позволяет, используя модель Пуанкаре в круге, нарисовать прямую плоскости Лобачевского по двум заданным точкам.



На рисунке изображены прямые: a, b, c, d и диаметр e. Приведем код к рисунку и снабдим его комментариями.

```
import geometry;// Подгружаем геометрический модуль
/* Две последующие строки позволяют создать метки на русском языке */
texpreamble("\usepackage[T2A]{fontenc}\usepackage[utf8]{inputenc}
\usepackage{mathtext}\usepackage[russian]{babel}");
size(8cm,0); // Определяем размер рисунка и массив точек
point O=origin; point A [];  A[0]=(-.6,-.5); A[1]=(-.1,.5);
 A[2]=(.4,-.8); A[3]=(.9,-.3); A[4]=(.4,.1); A[5]=(.8,.2);
dot("O",O,N); // Обозначаем центр круга Пуанкаре
/* Определяем окружность инверсии и рисуем ее */
inversion t=inversion(2,O);
circle Ct=circle(t);
draw(Ct, 0.8*blue);
/* Определяем функцию lineLob(point M,point N), используя стандартный
оператор circle(point A, point B, point C), рисующий окружность по трем
заданным точкам и определенную ранее инверсию */
//---------------------------
circle lineLob(point M,point N)
{
return circle(M,N,t*M);
}
//---------------------------
```

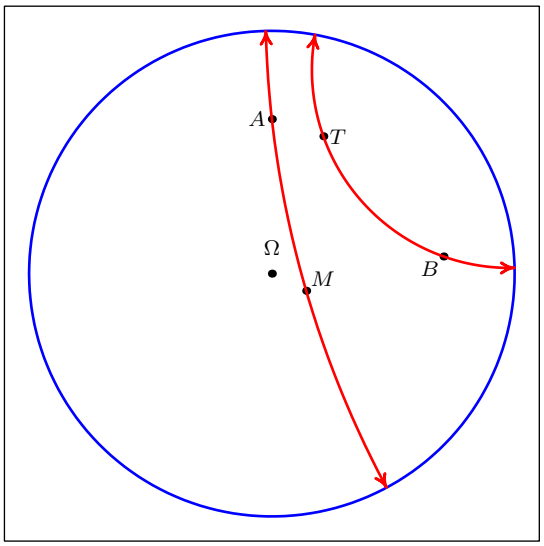
В следующем фрагменте кода определяется массив окружностей, иллюстрирующих прямые плоскости Лобачевского, затем рисуются эти прямые.

```
circle [] C; C[0]=lineLob(A[0],A[1]);C[1]=lineLob(A[1],A[5]);
C[2]=lineLob(A[0],A[3]);C[3]=lineLob(A[2],A[4]);C[4]=lineLob(A[4],A[5]);
draw(Label("a",position=Relative(.97),align=LeftSide),C[0],bp+.8*red);
draw(Label("b",position=Relative(.62)),C[1],bp+.8*red);
draw(Label("c",position=Relative(.22),align=RightSide),C[2],bp+.8*red);
draw(Label("d",position=Relative(.42)),C[3],bp+.8*red);
draw(Label("e",position=Relative(.8)),C[4],bp+.8*red);
path g=Ct; clip(currentpicture,g); /* Обрезаем рисунок, удаляя все, что
находится вне базисной окружности инверсии. */
label(scale(.8)*"Прямые на модели Пуанкаре",truepoint(N),2*N,blue);
shipout(bbox(5mm));
```

Приведем еще один вариант кода, который позволяет нарисовать прямую Лобачевского на модели Пуанкаре: он использует функцию `LobLine(point K,point L)` другого типа, типа `arc`.

```
import geometry;
size(250,0);

point pO=(0,0);
real k=2;
inversion inv=inversion(k,pO);

circle c0=circle(inv);
draw(c0,linewidth(bp)+blue);
dot("$\Omega$",pO,3*N,fontsize(8pt));

point M=(.2,-.1); point T=(.3,.8);
point B=(1,.1); point A=(0,.9);
dot("$A$",A,W,fontsize(8pt));
dot("$B$",B,SW,fontsize(8pt));

dot("$M$",M,NE,fontsize(8pt));
dot("$T$",T,E,fontsize(8pt));
//----------------------------
arc LobLine(point K,point L)
{
point iK=inv*K, iL=inv*L;

circle c1=circle(K,L,iK);

point[] uv=intersectionpoints(c1,c0);
point u=uv[1], v=uv[0];

return arc(c1, u, v);
}
//----------------------------
draw((path)LobLine(A,M),bp+red,ArcArrows(size=.7mm,arrowhead=HookHead));
draw((path)LobLine(T,B),bp+red,ArcArrows(size=.7mm,arrowhead=HookHead));
shipout(bbox(3mm));
```

Вторая функция имеет тот недостаток, что при некотором расположении исходных точек рисуется не внутренняя дуга, а дополнительная, расположенная вне базисной окружности.[2]



В следующем примере определим функцию, которая позволит построить на модели Пуанкаре прямую, проходящую через данную точку и параллельную данной прямой в смысле Лобачевского.

Известно, что таких параллельных прямых две, поэтому и функций будет также две. Функция `ParLobR` определяет прямую, параллельную данной вправо, а `ParLobL` — параллельную влево.

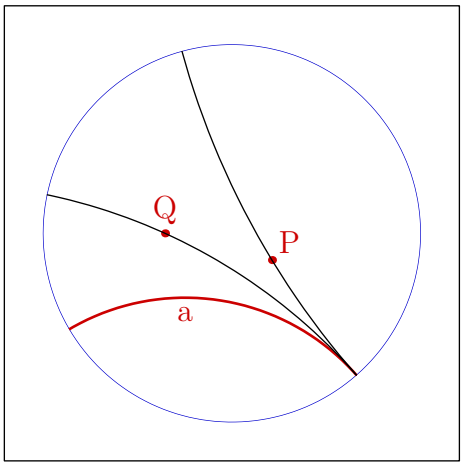
Отметим также, что эти функции используют функцию `lineLob`, определенную ранее в предыдущем примере.

Вот как выглядит определение функции `ParLobR`:

```
circle ParLobR(point P, circle C)
{
point [] T=intersectionpoints(C,Ct);
return lineLob(P,T[0]);
}
```

Функция зависит от двух аргументов: точки, через которую должна проходить параллельная прямая, и окружности, которая моделирует прямую Лобачевского. Рассмотрим пример подробнее.

Пример 180

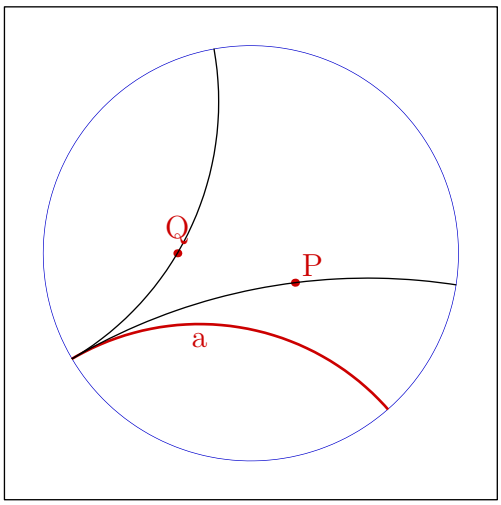


[2] Для такой пары точек дефект можно исправить, причем различными способами:
1) поменять местами аргументы u и v функции `arc(c1, u, v)`.
2) применить оператор `arc complementary(arc a)`

Код к рисунку выглядит следующим образом:

```
/* Начальный код не отличается от кода предыдущего примера.
Определяются точки P и Q, прямая a=AB и так же, как в
предыдущем примере, необходимая для дальнейшего функция
LobLine(point M,point N) */
import geometry;
size(8cm,0);
point O=origin, A=(-.6,-.5), B=(-.1,-.5), P=(.3,-.2), Q=(-.5,0);
inversion t=inversion(2,O);
circle Ct=circle(t);
draw(Ct, 0.8*blue);
point A1=t*A, B1=t*B;
// определяем вспомогательную функцию
circle lineLob(point M,point N)
{
return circle(M,N,t*M);
}
// Рисуем исходные данные
circle C= lineLob(A,B);
draw(Label("a",position=Relative(.25),align=LeftSide),C,bp+.8*red);
dot("P",P,NE,.8*red);
dot("Q",Q,N,.8*red);
// Определяем новую функцию
circle ParLobR(point P, circle C)
{
point [] T=intersectionpoints(C,Ct);
return lineLob(P,T[0]);
}
// Определяем и затем рисуем параллельные
circle C1=ParLobR(P,C);
circle C2=ParLobR(Q,C);
draw(C1);
draw(C2);
// Обрезаем рисунок как в предыдущем примере
path g=Ct;
clip(currentpicture,g);
shipout(bbox(5mm));
```

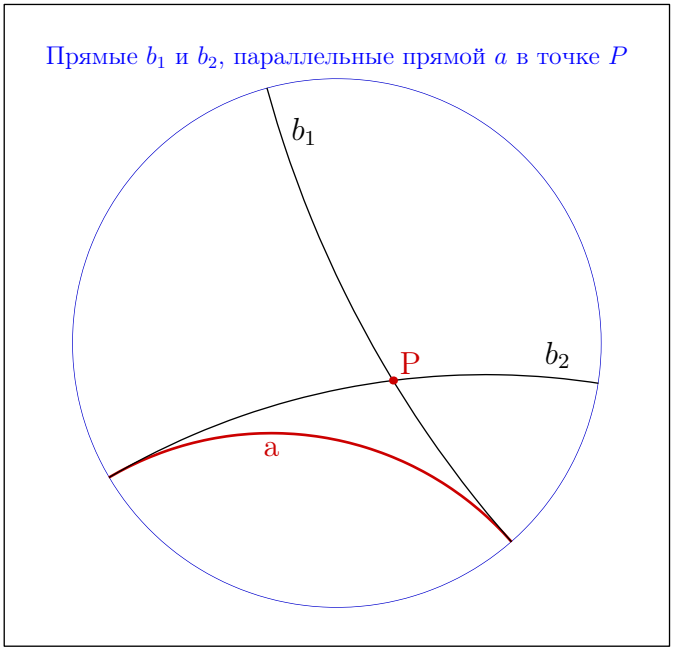
Пример 181



```
import geometry;
size(7.5cm,0);
point O=origin, A=(-.6,-.5), B=(-.1,-.5), P=(.3,-.2), Q=(-.5,0);
inversion t=inversion(2,O);
circle Ct=circle(t);
draw(Ct, 0.8*blue);
point A1=t*A, B1=t*B;
//----------------------
circle lineLob(point M,point N)
{
return circle(M,N,t*M);
}
//----------------------
circle C= lineLob(A,B);
draw(Label("a",position=Relative(.25),align=LeftSide),C,bp+.8*red);
dot("P",P,NE,.8*red);
dot("Q",Q,N,.8*red);
//----------------------
circle ParLobL(point P, circle C)
{
point [] T=intersectionpoints(C,Ct);
return lineLob(P,T[0]);
}
//----------------------
circle C1=ParLobL(P,C);
circle C2=ParLobL(Q,C);
draw(C1);
draw(C2);
path g=Ct;
clip(currentpicture,g);
shipout(bbox(5mm));
```

Пример 182

```
import geometry;
texpreamble("\usepackage[T2A]{fontenc}\usepackage[utf8]{inputenc}
\usepackage{mathtext}\usepackage[russian]{babel}");
size(8cm,0);
point O=origin, A=(-.6,-.5), B=(-.1,-.5), P=(.3,-.2);
inversion t=inversion(2,O);
circle Ct=circle(t);
draw(Ct, 0.8*blue);
point A1=t*A, B1=t*B;
circle lineLob(point M,point N)
{
return circle(M,N,t*M);
}
circle C= lineLob(A,B);
draw(Label("a",position=Relative(.25),align=LeftSide),C,bp+.8*red);
point [] T=intersectionpoints(C,Ct);
dot("P",P,NE,.8*red);
circle C1=lineLob(P,T[0]);
circle C2=lineLob(P,T[1]);
draw(C1);
draw(C2);
path g=Ct;
clip(currentpicture,g);
shipout(bbox(5mm));
```

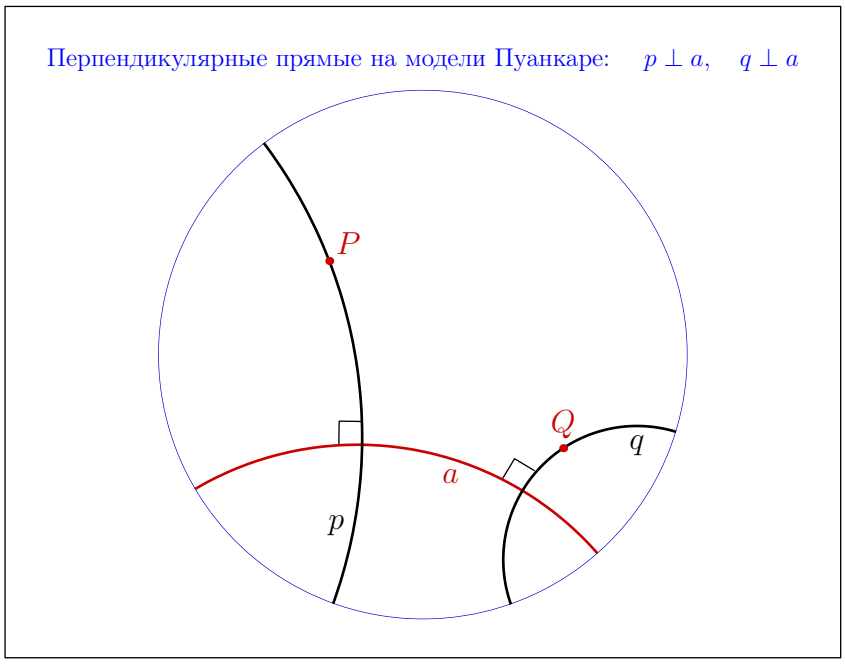


В следующем примере определяется функция `Perp(point P, point A, point B)` типа `circle`, позволяющая рисовать на модели Пуанкаре прямую, проходящую через заданную точку и перпендикулярную прямой, заданной двумя точками.

Пример 183

```
import geometry;
import markers;
texpreamble("\usepackage[T2A]{fontenc}\usepackage[utf8]{inputenc}
\usepackage{mathtext}\usepackage[russian]{babel}");
size(8cm,0);
point O=origin, A=(-.6,-.5), B=(-.1,-.5), P=(-.5,.5), Q=(.75,-.5);
inversion t=inversion(2,O);
circle Ct=circle(t);
draw(Ct, 0.8*blue);
point A1=t*A, B1=t*B;
//-------------------
circle lineLob(point M,point N)
{
return circle(M,N,t*M);
}
//-------------------
circle C= lineLob(A,B);
draw(Label("$a$",position=Relative(.2),align=LeftSide),C,bp+.8*red);
point [] T=intersectionpoints(C,Ct);
triangle t=triangle(A,A1,B);
point S=circumcenter(t);
path p=A--S;
real R=arclength(p); real k=R*R;
inversion inv=inversion(S,k);
//-------------------
circle Perp(point P, point A, point B)
{
return lineLob(P,inv*P);
}
//-------------------
circle Cq=Perp(Q,A,B); circle Cp=Perp(P,A,B);
draw(Cp);
```

```
draw(Label("$p$",position=Relative(.97),align=LeftSide),Cp,linewidth(bp));
draw(Cq);
draw(Label("$q$",position=Relative(.25),align=LeftSide),Cq,linewidth(bp));
dot("$P$",P,NE,.8*red);
dot("$Q$",Q,N,.8*red);
point [] H=intersectionpoints(C,Cp);
point [] G=intersectionpoints(C,Cq);
//dot("H", H[1]);
//dot("G", G[1]);
line tg1=tangent(Cp,H[1]);line tg2=tangent(C,H[1]);
perpendicularmark(tg1,tg2);
line tg3=tangent(C,G[1]);line tg4=tangent(Cq,G[1]);
perpendicularmark(reverse(tg4),tg3);
path g=Ct;
clip(currentpicture,g);
label(scale(.8)*"Перпендикулярные прямые на модели Пуанкаре:
$\quad p\perp a,\quad q\perp a$",truepoint(N),2*N,blue);
//label(scale(.8)*"$p\perp a,\quad q\perp a$",truepoint(N),2*N,blue);
shipout(bbox(5mm));
```



Заметим, что при определении функции `Perp(point P, point A, point B)` была использована функция `LobLine(point M,point N)`, определенная в в коде раньше.

Код в следующем примере позволяет нарисовать пучок расходящихся прямых, а именно пучок прямых, пересекающих данную прямую перпендикулярно.

Пример 184

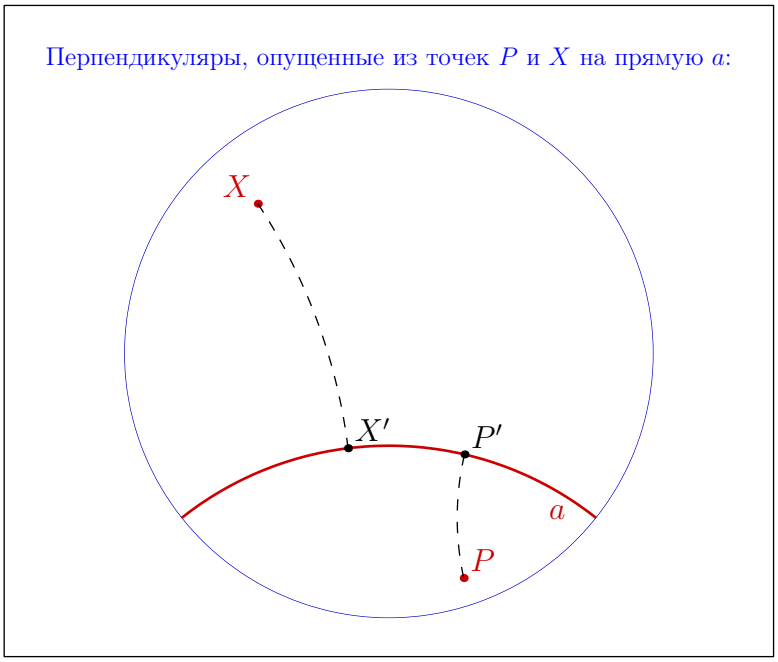
```
import geometry;
texpreamble("\usepackage[T2A]{fontenc}\usepackage[utf8]{inputenc}
\usepackage{mathtext}\usepackage[russian]{babel}");
size(6cm,0);
point O=origin; point A=(-1,-.8); point B=(1,-.8);
point[] P;
P[0]=(-1.6,-1); P[1]=(-1.1,-1); P[2]=(-.6,-1); P[3]=(-.1,0);
inversion t=inversion(2,O);
circle Ct=circle(t);
draw(Ct, 0.8*blue);
```

```
point A1=t*A, B1=t*B;
//--------------------
circle LobLine(point M,point N)
{
return circle(M,N,t*M);
}
//--------------------
circle C=LobLine(A,B);
draw(Label("$a$",position=Relative(.15),align=LeftSide),C,bp+.8*red);
triangle tri=triangle(A,A1,B);
point S=circumcenter(tri);
dot("$S$",S,red);
path p=A--S;
real R=arclength(p);
real k=R*R;
inversion inv=inversion(S,k);
//------------------
circle Perp(point Q, point A, point B)
{
return lineLob(Q,inv*Q);
}
//------------------
for (real d=.001; d<=1; d+=.1){
 draw(Perp(relpoint(shift(0,-.05)*C,d),A,B));
}
path g=Ct;
clip(currentpicture,g);
label(scale(.8)*("Пучок прямых, перпендикулярных прямой $a$:"),
truepoint(N),2*N,blue);
shipout(bbox(5mm));
```

В следующем примере перпендикуляры к заданной прямой рассматриваются не как прямые, а как сегменты прямой.

Пример 185

```
import geometry;
texpreamble("\usepackage[T2A]{fontenc}\usepackage[utf8]{inputenc}
\usepackage{mathtext}\usepackage[russian]{babel}");
size(8cm,0);
point O=origin; point A=(-1,-.8); point B=(1,-.8);
point P=(-.7,.8);
dot("$P$",P,NE,.8*red);
inversion t=inversion(2,O);
circle Ct=circle(t);
draw(Ct, 0.8*blue);
point A1=t*A, B1=t*B;
//-------------------
circle lineLob(point M,point N)
{
return circle(M,N,t*M);
}
//-------------------
circle C=lineLob(A,B);
draw(Label("$a$",position=Relative(.15),align=LeftSide),
C,bp+.8*red);
triangle tri=triangle(A,A1,B);
point S=circumcenter(tri);
dot("$S$",S,red);
path p=A--S;
real R=arclength(p);
real k=R*R;
inversion inv=inversion(S,k);
//------------------
circle Perp(point Q, point A, point B)
{
return lineLob(Q,inv*Q);
}
//------------------
circle perp=Perp(P,A,B);
point [] T=intersectionpoints(perp,C);
dot("$P^{\prime}$",T[1],NE);
arc TP=arc(perp,T[1],P);
draw(TP);
path g=Ct;
clip(currentpicture,g);
label(scale(.8)*("Перпендикуляр, опущенный из точки $P$ на прямую $a$:"),
truepoint(N),2*N,blue);
shipout(bbox(5mm));
```



Пример 186

Однако, перпендикуляры к заданной прямой из заданной точки можно не только опускать, но и восстанавливать. На рисунке изображен перпендикуляр p, проходящий через точку $P \in a$.

Для изображения этой ситуации на модели Пуанкаре требуется несколько иной алгоритм.

Код и комментарии к нему приведены ниже. В первом блоке подгружается геометрический пакет, определяется размер рисунка и пара срок кода для создания меток на русском языке.

```
import geometry;
size(8cm,0);
texpreamble("\usepackage[T2A]{fontenc}\usepackage[utf8]{inputenc}
\usepackage{mathtext}\usepackage[russian]{babel}");
```

```
point O=origin; point A=(-.2,-.8); point B=(.4,-.8);//Задаем точки
```

В следующих трех строках кода определяется инверсия, базисная окружность которой является базисной окружностью в модели Пуанкаре. В четвертой строке определяются образы заданных точек при инверсии.

```
inversion t=inversion(2,O);
circle Ct=circle(t);
draw(Ct, 0.8*blue);
point A1=t*A, B1=t*B;
```

Дальше мы определим функцию `lineLob(point M,point N)`, для чего используем стандартный оператор `circle(point A, point B, point C)`, рисующий окружность по трем заданным точкам и определенную ранее инверсию. С помощью этой функции рисуется окружность, моделирующая прямую плоскости Лобачевского.

```
circle lineLob(point M,point N)
{
return circle(M,N,t*M);
}
circle C=lineLob(A,B);// окружность C - прямая на плоскости Лобачевского
```

Далее задается точка P на заданной прямой и её образ при инверсии.

```
point P=relpoint(C,.24);
point P1=t*P;
```

В следующем блоке описывается построение перпендикуляра к заданной прямой, который проходит через заданную точку P.

```
dot("P",P,NW);
//-------------------------
triangle tri=triangle(A,A1,B);
point S=circumcenter(tri);
dot("$S$",S,red);
//-------------------------
line l1=line(S,P);
line l2=perpendicular(P,l1);
line m1=line(S,P1);
line m2=perpendicular(P1,m1);
point Z=intersectionpoint(l2,m2);
dot(Z);
//--------------------------
path PZ=P--Z;
real R1=arclength(PZ);
//----------------------
circle Cp=circle(Z,R1); //
//-------------------
draw(Label("$a$",position=Relative(.35),align=LeftSide),C,bp+.8*red);
//-----------------
draw(Label("$p$",position=Relative(.27),align=LeftSide),Cp,bp+.5*green);
```

Наконец, обрезаем рисунок по границе круга Пуанкаре.

```
path g=Ct;
clip(currentpicture,g);
shipout(bbox(5mm));
```

Пример 187

В этом примере определяется функция `commonPerp` двух аргументов, с помощью которой рисуется общий перпендикуляр расходящихся прямых.

Аргументы функции — это две окружности, "прямые" модели Пуанкаре.

Прокомментируем код. Первый блок кода стандартный, в нем определяются и рисуются заданные "прямые" модели Пуанкаре.

```
import geometry;
texpreamble("\usepackage[T2A]{fontenc}\usepackage[utf8]{inputenc}
\usepackage{mathtext}\usepackage[russian]{babel}");
size(8cm,0);
point O=origin; point U=(-1,-1.5), V=(.8,-1), I=(-.1,.5), J=(.8,.3);
inversion t=inversion(2,O);
circle Ct=circle(t);
draw(Ct, 0.8*blue);
//--------------------------
circle lineLob(point M,point N)
{
return circle(M,N,t*M);
}
//---------------------------
circle C1=lineLob(U,V);
circle C2=lineLob(I,J);
draw(Label("a",position=Relative(.15)),C1,bp+.8*red);
draw(Label("b",position=Relative(.62)),C2,bp+.8*red);
```

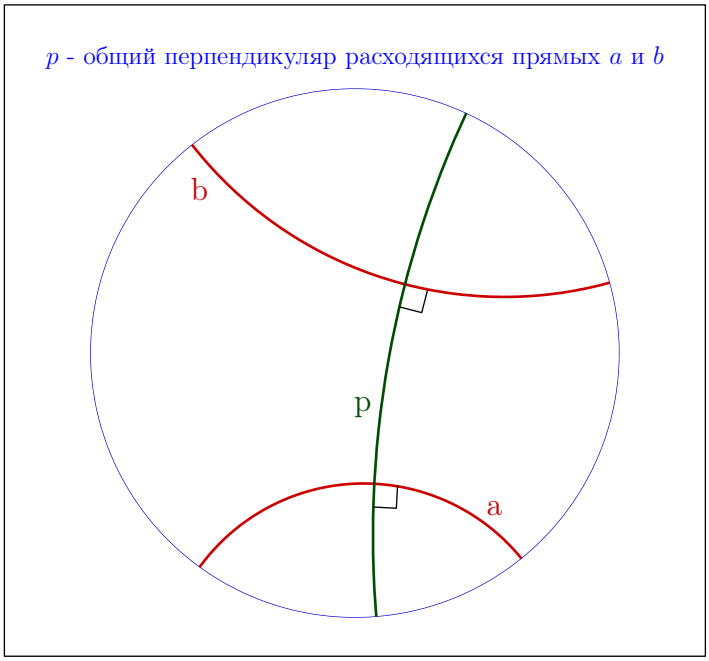
Далее определяем общий перпендикуляр и рисуем его:

```
//-----------------
circle commonPerp(circle C1,circle C2)
{
circle C1=lineLob(U,V);
circle C2=lineLob(I,J);
return circle(inversion(C1,C2,Ct));
}
//-----------------
circle C12=commonPerp(C1,C2);
draw(Label("p",position=Relative(.48),align=RightSide),C12,bp+.3*green);
//------------------
```

Наконец, создаем меткм прямых углов и обрезаем все вне круга Пуанкаре.

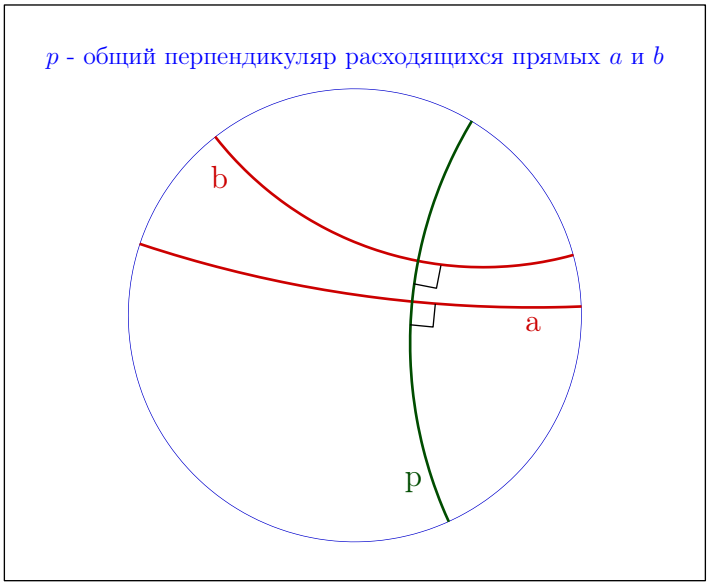
```
point [] H=intersectionpoints(C1,C12);
point [] G=intersectionpoints(C2,C12);
//dot("H", H[1]);
//dot("G", G[1]);
line tg1=tangent(C12,H[1]);line tg2=tangent(C1,H[1]);
perpendicularmark(tg1,tg2);
line tg3=tangent(C12,G[0]);line tg4=tangent(C2,G[0]);
perpendicularmark(tg3,tg4);
//------------------
path g=Ct;
clip(currentpicture,g);
label(scale(.8)*"$p$ - общий перпендикуляр расходящихся прямых $a$ и $b$",
                                                truepoint(N),2*N,blue);
shipout(bbox(5mm));
```

Рисунок готов!



Пример 188

Ещё пример общего перпендикуляра с другими прямыми a и b. Код к рисунку мы не приводим, предлагаем читателю написать его самостоятельно.



Пример 189

Следующий код описывает построение *эквидистанты* прямой, которую называют еще *линией равных расстояний*. Прокомментируем код, разбив его на блоки для удобства.

Первый блок стандартный, как в предыдущих рисунках.

```
import geometry;
texpreamble("\usepackage[T2A]{fontenc}\usepackage[utf8]{inputenc}
\usepackage{mathtext}\usepackage[russian]{babel}");
size(8cm,0);
```

Во втором блоке задаются точки. Точки A и B задают прямую, а точка D определяет эквидистанту.

```
point O=origin; point A=(-1,-.8); point B=(1,-.8);
point D=(.8,0);
```

В следующем блоке стандартно с помощью инверсии определяется круг Пуанкаре.

```
inversion t=inversion(2,O);
circle Ct=circle(t);
draw(Ct, 0.8*blue);
point A1=t*A, B1=t*B, D1=t*D;// Образы точек при инверсии
```

Определяется функция, задающая "прямые Лобачевского".

```
//-------------------
circle lineLob(point M,point N)
{
return circle(M,N,t*M);
}
//-------------------
```

Рисуется и обозначается прямая a.

```
circle C=lineLob(A,B);
draw(Label("$a$",position=Relative(.16),align=LeftSide),C,bp+.8*red);
```

Следующий блок очень важный. В нем определяется вторая инверсия (`inv`), инверсия относительно окружности, на которой лежит дуга ("прямая") a. Эта инверсия на модели Пуанкаре играет роль осевой симметрии. Осью является a, а симметричные точки — это соответствующие точки при этой инверсии.

Определяется центр S окружности инверсии, ее радиус R и коэффициент k. Затем стандартно определяется сама инверсия.

```
triangle tri=triangle(A,A1,B);
point S=circumcenter(tri);
path p=A--S;
real R=arclength(p);
real k=R*R;
inversion inv=inversion(S,k);
point D2=inv*D;// определяется образ точки D при инверсии inv
```

Есть ещё один вариант кода этого блока:

```
circle umc=circumcircle(A,A1,B);
point S=umc.C;
//dot(umc.C);
path radius=A--S;
real R=arclength(radius);
real k=R*R;
inversion inv=inversion(S,k);
```

Определяется функция, опускающая перпендикуляр из D на a.

```
//-------------------
circle Perp(point Q)
{
return lineLob(Q,inv*Q);
}
//-------------------
```

В следующем блоке определяются и рисуются эквидистанты $Eq1$ и $Eq2$

```
point [] Z=intersectionpoints(Ct,C);
circle Eq1=circle(Z[0],Z[1],D);
draw(Label("$\varepsilon_{1}$",position=Relative(.3),align=RightSide),
                                                    Eq1,linewidth(bp));
circle Eq2=circle(Z[0],Z[1],D2);
draw(Label("$\varepsilon_{2}$",position=Relative(.75),align=RightSide),
                                                     Eq2,linewidth(bp));
```

Очередной блок. Определяются и рисуются "симметричные" точки D и D'.

```
circle q=Perp(D);
arc  d=arc(q,D,D2);
draw(Label("$h$",position=Relative(.3),align=RightSide),d,dashed);
draw(Label("$h$",position=Relative(.8),align=LeftSide),d,dashed);
dot("$D$",D,NE,.8*red);
dot("$D^{\,\prime}$",D2,SE,.8*red);
```

Далее создается и рисуется маркер прямого угла.

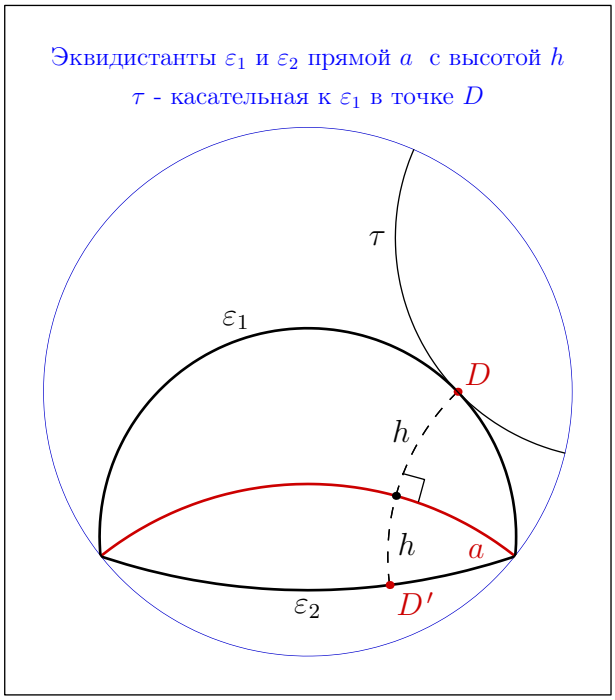
```
//------------------
point [] H=intersectionpoints(C,q);
dot(H[1]);
line tg2=tangent(q,H[1]);
line tg1=tangent(C,H[1]);
perpendicularmark(reverse(tg1),tg2);
//------------------------
```

Наконец, в предпоследнем, тоже существенном блоке, рисуется "прямая" Лобачевского, которая является касательной к эквидистанте в точке D.

```
segment seg=segment(D,D1);
line b=bisector(seg);
line f=line(D,Eq1.C);
point m=intersectionpoint(b,f);
circle tg=circle(m,arclength(m--D));
draw(Label("$\tau$",position=Relative(.5),align=RightSide),tg);
```

В последнем, стандартном блоке создаются метки с надписями и удаляется все вне круга Пуанкаре.

```
//-------------------------
path g=Ct;
clip(currentpicture,g);
label(scale(.8)*("$\tau$ - касательная к $\varepsilon_{1}$ в точке $D$"),
                                                  truepoint(N),2*N,blue);
label(scale(.8)*("Эквидистанты $\varepsilon_{1}$ и $\varepsilon_{2}$
                    прямой $a$ \  с высотой $h$"),truepoint(N),2*N,blue);
shipout(bbox(5mm));
```



Пример 190

В примере иллюстрируются орициклы пучка прямых, параллельных в одном и том же направлении. Орициклы выделены красным цветом, а параллельные прямые — черным.

Далее представлен код, создающий рисунок.

```
import geometry;
texpreamble("\usepackage[T2A]{fontenc}\usepackage[utf8]{inputenc}
\usepackage{mathtext}\usepackage[russian]{babel}");
size(8.5cm);
point O=origin, A=(-.6,-.5), B=(-.1,-.5), P=(.3,-.2), Q=(-.5,0);
point P1=(0,-1.5), Q1=(-.7,.6);
inversion t=inversion(2,O);
circle Ct=circle(t);
draw(Ct, 0.8*blue);
point A1=t*A, B1=t*B;
//----------------------
circle lineLob(point M,point N)
{
return circle(M,N,t*M);
}
//----------------------
circle C= lineLob(A,B);
draw(C);
//----------------------
circle ParLobL(point P, circle C)
{
point [] T=intersectionpoints(C,Ct);
return lineLob(P,T[1]);
}
//----------------------
point [] T=intersectionpoints(C,Ct);
line l=line(T[1],O);
draw(l);
circle C1=ParLobL(P,C);
circle C4=ParLobL(P1,C);
circle C2=ParLobL(Q,C);
circle C5=ParLobL(Q1,C);
draw(C1);draw(C4);
draw(C2);draw(C5);
point X=midpoint(T[1]--O);
circle C3=circle(X,.5*sqrt(2));
transform Homothetie(real k, pair centre)
{
  return shift(centre)*scale(k)*shift(-centre);
}
pair centre=T[1];
for(real k=.25; k<1.6; k+=.25){
transform h=Homothetie(k,T[1]);
draw(h*C3,0.8*red);
}
path g=Ct;
clip(currentpicture,g);
label(scale(.8)*("Орициклы пучка параллельных прямых"),truepoint(N),2*N,blue);
shipout(bbox(5mm));
```

2.7 Холсты и картинки

Иногда наш рисунок нужно составить из нескольких других. Для более удобного управления созданием таких изображений в `Asymptote` существуют такие типы данных, как `frame` и `picture`. Термин `frame` мы переведем как основа (холст), а `picture` как картинка, рисунок.

Типы `frame` и `picture` позволяют сделать код программы более понятным для восприятия и более легким для изменения.

Коротко охарактеризуем `frame` и `picture` следующим образом:

- Тип `frame` предназначен для рисования в `Postscript`-координатах.
- Тип `picture` — более развитая структура, предназначенная для рисования с единицей длины, заданной пользователем.

В этом разделе на ряде простых примеров мы более подробно рассмотрим использование типов `frame` и `picture`.

2.7.1 Тип `frame`

Тип `frame` используют для рисования в `Postscript`-координатах. Согласно общим правилам `Asymptote` мы должны сначала объявить `frame`, затем наполнить его графическими объектами, а потом добавить командой `add()`. После компиляции можно просмотреть рисунок.

Пример 191

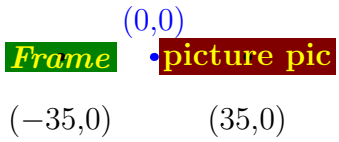
В этом примере раскрывается один из способов употребления `frame`. Рассмотрим код и соответствующую картинку:

```
frame f; // определяем frame, используя переменную  f
label(f,"$\textbf{frame \& picture}$",yellow,Fill(.5blue));
// В качестве содержимого f задаем метку с желтым текстом на синем фоне
add(f); // Команда add рисует содержимое f
```

Пример 192

Определим два `frame` и в каждом свое содержимое. Чтобы они не накладывались один на другой, мы сдвинем один `frame` влево, а другой вправо.

```
frame g,h;
label(g,"$\textbf{\textit{Frame}}$",yellow,Fill(.5green));
label(h,"\textbf{picture pic}",yellow,Fill(.5red));
add(shift(35,0)*h);
add(shift(-35,0)*g);
```

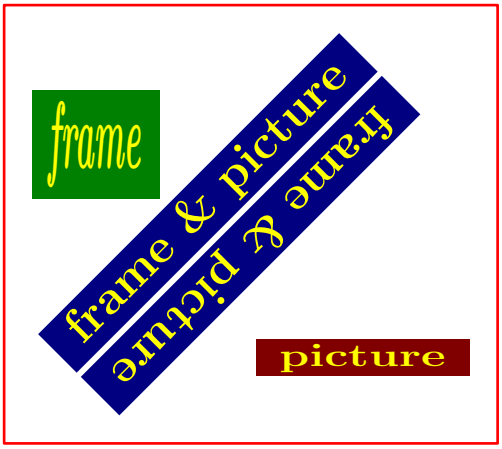


При сдвигах, указанных в двух последних строках кода, центры боксов, содержащих `frame g` и `frame h` (они указаны чёрными точками), расположены на 35 bp левее и правее точки с `postscript`-координатами `(0,0)`.

Пример 193

А вот пример картинки с использованием трех `frame`. Заметим, что боксы с `frame` можно не только сдвигать, но и масштабировать и поворачивать.

```
frame f,g,h;
label(f,"$\textbf{ frame \& picture }$",yellow,Fill(.5blue));
label(g,"\textbf{ \textit{frame} }",yellow,Fill(.5green));
label(h,"\textbf{ picture }",yellow,Fill(.5red));
add(shift(50,-50)*xscale(1.5)*h);
add(shift(-50,30)*yscale(3)*g);
add(shift(-8,8)*rotate(45)*scale(1.5)*f);
transform t; t=rotate(225)*scale(1.5);
add(shift(8,-8)*t*f);
```



Ещё раз заметим, что рисование в `frame` использует postscript координаты !!!
Единица длины `1 bp` (`big point`) в `postscript` равна `1/72 inch` (дюйма) $\approx$ `0.353 mm`.

Для добавления "`frame`" (или "`picture`"), используется команда `add`. Рассмотрим более подробно синтаксис команды `add`. Возможны два варианта синтаксиса:

```
1. void add(picture dest=currentpicture, frame src, pair position=0,
            bool group=true, filltype filltype=NoFill, bool above=true);

2. void add(picture dest=currentpicture, frame src, pair position, pair align,
            bool group=true, filltype filltype=NoFill, bool above=true);
```

Иллюстрируем первый вариант синтаксиса. По умолчанию `frame` добавляется к текущей картинке в позиции (точке) (0,0), как, например, это сделано в примере 2.7.1. Вот ещё несколько примеров.

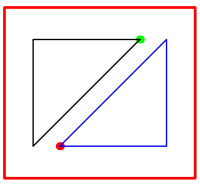
Пример 194

```
frame f;
path triangle=(20,0)--(40,40)--(40,0)--(20,0);
draw(f,triangle); // Рисуем треугольник в f
add(f,(0,0));
// f добавляется в позиции (0,0)
shipout(bbox(10)); // Рисуем  рамку.
```

Пример 195

Иллюстрируем второй вариант синтаксиса:

```
dot((0,0),3bp+red);// Отметим красным цветом начальную точку.
dot((30,40),3bp+green);// Отметим зеленым цветом точку (30,40).
frame f,g; // Задаем два frame.
path tr=(0,0)--(40,40)--(40,0)--(0,0);// tr - путь, задающий контур треугольника.
draw(g,tr,blue);// Рисуем треугольник в g.
add(g);// Добавляем g к пустой картинке.
draw(f,rotate(180)*tr); // Рисуем в f треугольник tr,
//повернув его на 180 градусов против часовой стрелки.
add(f,(30,40)); // f добавляется к текущей картинке в позиции (30,40)
shipout(bbox(10,red+bp)); // Рисуем красную рамку.
```



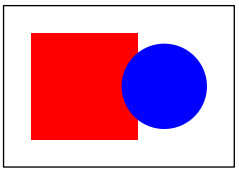
Этот же эффект можно получить кодом

```
dot((0,0),3bp+red);// Отметим красным цветом начальную точку.
dot((30,40),3bp+green);// Отметим зеленым цветом точку (30,40).
frame f,g; // Задаем два frame.
path tr=(0,0)--(40,40)--(40,0)--(0,0);// tr - путь, задающий контур треугольника.
draw(g,tr,blue);// Рисуем треугольник в g.
add(g);// Добавляем g к пустой картинке.
draw(f,shift(30,40)*rotate(180)*tr); // Рисуем в f треугольник tr, повернув его
// на 180 градусов против часовой стрелки и сдвинув на вектор (30,40).
add(f); // f добавляется к текущей картинке)
shipout(bbox(10,red+bp)); // Рисуем красную рамку.
```

Пример 196

Определяем `frame f` и `frame g`.
Рисуем красный заполненный квадрат в `f`.
Рисуем синий круг в `g`.
Мы добавляем квадрат в `f` к текущей картинке в точке $(0, 0)$.
Мы добавляем круг в `g` к текущей картинке в точке $(50, 20)$.

```
frame f;frame g;
fill(f,(0,0)--(40,0)--(40,40)--(0,40)--cycle,red);
fill(g,scale(16)*unitcircle,2bp+blue);
add(f,(0,0));
add(g,(50,20));
shipout(bbox(10,black));
```



Замечание: Обратите внимание, как порядок добавления `frames` отражается на изображении. Что будет, если поменять порядок команд `add(f,(0,0));` и `add(g,(50,20));`?

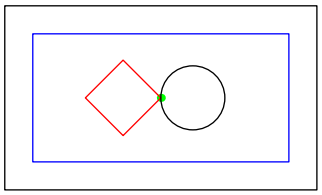
Пример 197

В этом примере зададим пользовательскую единицу `12bp`. Это можно сделать двумя способами: 1) Задать в коде `unitsize(12);`
2) Задать в коде `real u=12;` и использовать параметр `u` в виде множителя.

Первый способ. Рисуем прямоугольник синего цвета в пользовательских единицах (`12bp`). Выделим центр прямоугольника, точку $(4, 2)$ зеленым цветом.

Задаем `frame f`. Далее определяем в обычных `postscript`-единицах (`1bp`) два пути: единичную окружность и квадрат со стороной `20` единиц. Рисуем в `f` квадрат красного цветом, повернув его на 135° и единичную окружность, увеличив ее в 12 раз и сдвинув на вектор $(12, 0)$ (Еще раз подчеркнем, что единица 1bp!). Добавляем содержимое f в текущую картинку (Текущая картинка использует единицу пользователя!) в точке $(4, 2)$.

```
unitsize(12);
draw((0,0)--(8,0)--(8,4)--(0,4)--cycle,blue);
dot((4,2),green);
frame f;
path c=unitcircle;
path sq=(0,0)--(20,0)--(20,20)--(0,20)--cycle;
draw(f,rotate(135)*sq,red); draw(f,shift(12,0)*scale(12)*c);
add(f,(4,2));
shipout(bbox(10,black));
```



Для неопытного пользователя может быть трудно различить в тексте кода пользовательские координаты и обычные координаты в `postscript`-единицах. Избежать этого можно, используя второй способ.

Второй способ. Наличие в координатах параметра `u` говорит о том, что координаты пользовательские. Приводим соответствующий код:

```
real u=12;
draw((0,0)--(u*8,0)--(u*8,u*4)--(0,u*4)--cycle,blue);
dot((u*4,u*2),green);
frame f;
path c=unitcircle;
path sq=(0,0)--(20,0)--(20,20)--(0,20)--cycle;
draw(f,rotate(135)*sq,red); draw(f,shift(12,0)*scale(12)*c);
add(f,(u*4,u*2));
shipout(bbox(10,black));
```

Пример 198

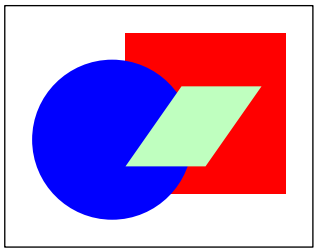
В третьем примере мы уже разбирали влияние порядка добавления `frames` на результирующую картинку. В данном примере мы проиллюстрируем некоторые варианты кода, когда число `frames` более двух.

Допустим, нам нужно изобразить три цветные формы: красный заполненный квадрат, салатный заполненный параллелограмм и синий круг. Кроме того, нам нужно, чтобы синий круг был расположен поверх красного квадрата, а параллелограмм располагался поверх обоих.

Можно просто употребить команды `add()` в следующем порядке: `add(f1); add(f3); add(f2);`

Мы приведем альтернативный код, использующий команду `prepend(f2,f3);`

```
real s=.7;     // Действительная константа для рисования параллелограмма.
frame f1, f2, f3;  // Определяем  три различных frame.
fill(f1,scale(60)*unitsquare,red);  // Рисуем красный заполненный
                                    // квадрат и помещаем его в f1.
fill(f2,shift(0,10)*slant(s)*scale(30)*unitsquare,palegreen); // Рисуем
            // салатный заполненный параллелограмм и помещаем его в f2.
fill(f3,shift(-5,20)*scale(30)*unitcircle,blue);// Рисуем синий круг и
                                                // помещаем его в f3.
add(f1);    // Добавляем f1 к текущей картинке.
prepend(f2,f3); //Добавляем f3 к f2, причем под (ниже) f2.
add(f2);    //Добавляем f2 поверх текущей картинки.
shipout(bbox(10,black));   // Рисуем рамку.
```



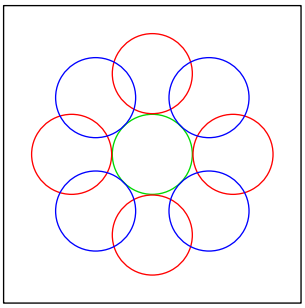
Этот же результат можно получить, используя код:

```
real s=.7;
frame f1, f2, f3;
fill(f1,scale(60)*unitsquare,red);
fill(f2,shift(0,10)*slant(s)*scale(30)*unitsquare,palegreen);
fill(f3,shift(-5,20)*scale(30)*unitcircle,blue);
prepend(f3,f1);
add(f3);
add(f2);
shipout(bbox(10,black));
```

Пример 199

В примере показано, как можно изменить (переопределить) `frame`, изменив его выравнивание относительно точки привязки. Используем функцию `align()`. Первым аргументом функции является имя `frame`, вторым аргументом указатель направления, как на компасе.

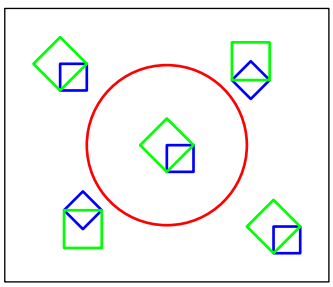
```
frame h, f,fN,fS,fW,fE;
draw(f,scale(20)*unitcircle,red);
draw(h,scale(20)*unitcircle,.85green);
fN=align(f,20*N);//или иначе: fN=align(f,20*dir(90));
fS=align(f,20*S);//или иначе: fS=align(f,20*dir(270));
fE=align(f,20*E);//или иначе: fE=align(f,20*dir(0));
fW=align(f,20*W);//или иначе: fW=align(f,20*dir(180));
add(h); add(fN); add(fS);add(fW);add(fE);
frame g,gNE,gSE,gNW,gSW;
real s=8*sqrt(2);
draw(g,scale(20)*unitcircle,blue);
gNE=align(g,s*NE);//или иначе: gNE=align(g,s*dir(45));
gSE=align(g,s*SE);//или иначе: gSE=align(g,s*dir(135));
gNW=align(g,s*NW);//или иначе: gNW=align(g,s*dir(-45));
gSW=align(g,s*SW);//или иначе: gSW=align(g,s*dir(-135));
add(gNE); add(gSE);add(gNW); add(gSW);
shipout(bbox(10,black));
```



Таким образом функция `align` переопределяет способ выравнивания `frame`, сохраняя его содержимое. В следующем примере показано, что можно, помимо этого, указывать в команде `add` также и точку привязки.

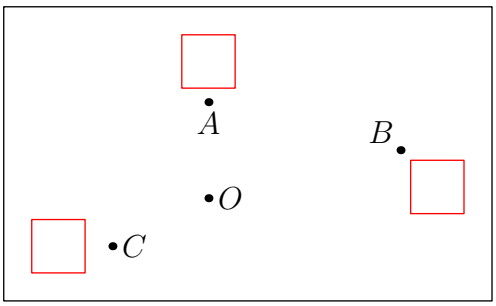
Пример 200

```
real u=20,s=sqrt(2);// вводим две действительные постоянные
frame f,g;  // определяем frames
path a=scale(.5*u)*unitsquare; // определяем  квадрат
path b=scale(1.5*u)*unitcircle;// определяем окружность
path c=rotate(45)*scale(10*s)*unitsquare;// определяем второй квадрат
draw(f,b,bp+red); add(f); // рисуем окружность и добавляем ее
draw(g,a,bp+blue); draw(g,c,bp+green); // Рисуем синий и зеленый квадраты
add(shift(0,-10)*g);// Добавляем квадраты,
                    // совмещая центр зеленого квадрата с центром круга
frame f1=align(g,u*N);// переопределяем g с новым выравниванием
frame h=rotate(135)*f1;// определяем h
add(g,(-2u,0),u*N);add(g,(2u,0),u*S);//или add(f1,(-2u,0));add(f1,(2u,0),u*S);
// Добавляем квадраты по главной диагонали
add(rotate(-45)*f1,.7u*NE);// Добавляем f1 в правый верхний угол
                           //с поворотом и новым выравниванием
add(h,.7u*SW);// Добавляем h в левый нижний угол
shipout(bbox(10,black));
```



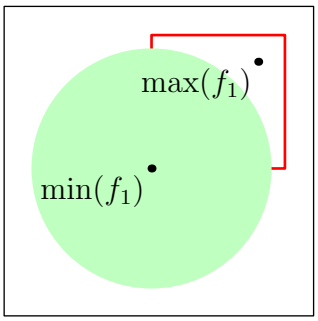
Пример 201

```
unitsize(1cm);// определяем пользовательскую единицу
frame f,fN,fSE;// определяем три frame
draw(f,scale(20)*unitsquare,red); // рисуем красный квадрат в f
fN=align(f,N); // fN : позиция f в "direction" N (nord)
fSE=align(f,SE); // fSE : позиция f в "direction" SE (sud-est)
f=align(f,(-10,0)); // переопределяем f : f в "direction" вектора (-10,0)
// Задаем 4 точки, определяющие расположение фигуры.
pair pO=(0,0), pA=(0,1), pB=(2,.5), pC=(-1,-.5);
// координаты точек даны в пользовательских единицах (1cm)
dot(pO); dot("$A$",pA,S);dot("$B$",pB,NW);dot("$C$",pC);// рисуем точки
add(fN,pA); // добавляем fN к текущей картинке в точке pA
add(fSE,pB); // добавляем fSE к текущей картинке в точке pB
add(f,pC); // добавляем f (переопределенный) текущей картинке в точке pC
shipout(bbox(10,black));
```



Пример 202

```
draw(scale(100)*unitsquare,1bp+green); // чертим зеленый квадрат.
// он становится текущей картинкой.
frame f1,f2;
fill(f1,scale(80)*unitsquare,red); // заполняем красный квадрат в f1
fill(f2,scale(75)*unitcircle,paleblue); // заполняем сиреневый круг в f2
add(f1); // добавляем f1 к текущей картинке
add(f2); // добавляем f2 к текущей картинке
erase(f1); // очищаем frame f1, красный квадрат исчезает
// Можно попытаться повторно добавить f1, но попытка тщетна.
add(f1);
// Однако старые "размеры" f1 сохраняются.
dot("min(f1)",min(f1),SW);
dot("max(f1)",max(f1),SW);
shipout(bbox(10,black));
```

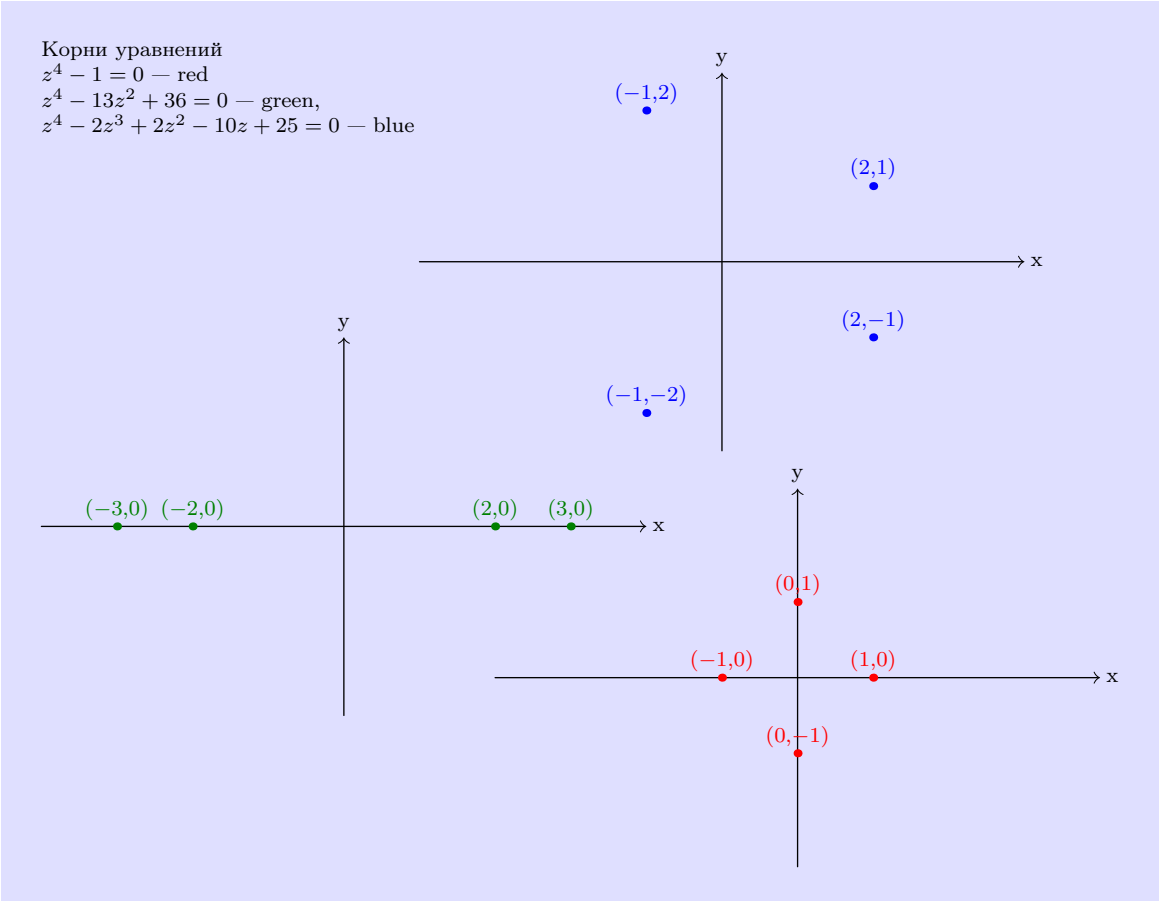


2.7.2 Тип `picture`

В первой части пособия рассматривались два примера, использующих тип данных `picture`. Продолжим ряд примеров, расширяющих наши представления об особенностях использования `picture`.

Пример 203

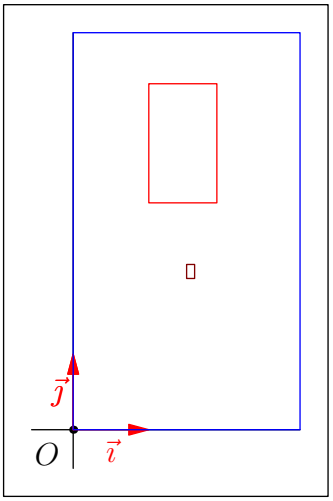
```
texpreamble("\usepackage[T2A]{fontenc}\usepackage[utf8]{inputenc}
\usepackage[russian]{babel}\usepackage{amsfonts}");
unitsize(1cm);
import math;
picture f,g,h;
path x=(-4,0)--(4,0); path y=(0,-2.5)--(0,2.5);
draw(f,Label("x",position=EndPoint),x,Arrow(TeXHead));
draw(f,Label("y",position=EndPoint),y,Arrow(TeXHead));
draw(g,Label("x",position=EndPoint),x,Arrow(TeXHead));
draw(g,Label("y",position=EndPoint),y,Arrow(TeXHead));
draw(h,Label("x",position=EndPoint),x,Arrow(TeXHead));
draw(h,Label("y",position=EndPoint),y,Arrow(TeXHead));
pair [] z=quarticroots(1 , -2, 2, -10 , 25);
pair [] w=quarticroots(1 , 0, -13, 0 , 36);
pair [] v=quarticroots(1 , 0, 0, 0 , -1);
for(int k=0; k<=3;++k){dot(f,"",z[k],N,3bp+blue);
dot(g,"",w[k],N,3bp+.5green);dot(h,"",v[k],N,3bp+red);}
add(shift(2,0)*f); add(shift(-3,-4)*g); add(shift(4,-6)*h);
label(minipage("Корни уравнений\\
$z^4 -1 = 0$ --- red\\
$z^4 -13z^2 +36 = 0$ --- green,\\
$z^4 -2z^3 +2z^2 -10z +25 = 0$ --- blue ",width=8cm),(-4,2.5), S);
```



Пример 204

Первый пример показывает различные варианты использования команды `add(...)`. Например, мы можем указывать точку привязки `picture`. (Смотри синтаксис команды `add(...)` в начале раздела).

Однако указание точки привязки может приводить к неожиданным результатам: изображенные фигуры различаются не только по их позициям, но и по размерам!




```
import geometry;
unitsize(1cm);
show(currentcoordsys);
path p=yscale(1.75)*scale(3)*unitsquare;
picture obj1, obj2, obj3;
draw(obj1,p,blue);
add(obj1); //эквивалентно: currentpicture.add(obj1);
draw(obj2,p,brown);
add(obj2,(1.5,2));
/* См. определения и варианты синтаксиса команды
add(...) в файле plain_picture.asy.*/
draw(obj3,shift(1,3)*scale(.3)*p,red);
add(obj1,obj3);
add(obj1); // Получаем красный прямоугольник
shipout(bbox(10));
```

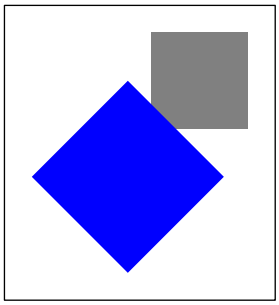
В приведенном примере в качестве фигуры выступает прямоугольник, границей которого является путь `p`. Сначала мы используем единицу `1cm`, которая действует на `currentpicture`. Добавляя к `currentpicture` без указания точки привязки путь `p` на `picture obj1`, получим большой синий прямоугольник со сторонами `3 cm` по горизонтали и `5.25 cm` по вертикали.

Рисуя тот же путь на `picture obj2` и добавляя его в точке `(1.5,2)`, получаем маленький коричневый прямоугольник, подобный синему, но в `postscript`-единицах, со сторонами `3 bp` по горизонтали и `5.25 bp` по вертикали.

Наконец, нарисуем прямоугольник на `picture obj3`, уменьшив и сдвинув его, и добавим к `picture obj1`, а затем добавим `picture obj1` к `currentpicture`.

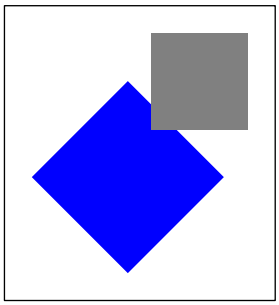
Пример 205

Так же, как и в работе с `frame`, очерёдность добавления имеет значение. Это особенно заметно для заполненных фигур.



```
unitsize(36);
picture pic1, pic2;
fill(pic1,shift(.25,.5)*unitsquare,gray);
// Рисуем серый заполненный квадрат на pic1
path p=(1,0)--(0,1)--(-1,0)--(0,-1)--cycle;
fill(pic2,p,blue); // синий заполненный
//квадрат на pic2
add(pic1,pic2);// Накладываем pic2 на pic1
add(pic1);// Добавляем pic1 к currentpicture
shipout(bbox(10));
```

Поменяем порядок (очерёдность) добавления pic1 и pic2.



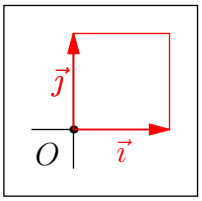
```
unitsize(36);
picture pic1, pic2;
fill(pic1,shift(.25,.5)*unitsquare,gray);
// Рисуем серый заполненный квадрат на pic1
path p=(1,0)--(0,1)--(-1,0)--(0,-1)--cycle;
fill(pic2,p,blue); // синий заполненный
//квадрат на pic2
add(pic2,pic1);// Накладываем pic1 на pic2
add(pic2);// Добавляем pic2 к currentpicture
shipout(bbox(10));
```

Поскольку при манипуляциях с `picture` можно использовать пользовательские единицы, а они могут быть разные в каждом `picture`, возникает задача, как избежать эффекта первого примера, когда указание точки привязки приводит к переходу на `postscript`-единицы.

Следующий пример содержит код, который позволяет обойти возникающие трудности.

Пример 206

Сначала приведем код, который на `pic1` рисует красный квадрат со стороной, равной `unitsize`, то есть `36 bp`.




```
import geometry;
unitsize(36);
show(currentcoordsys);
picture pic1;
draw(pic1,unitsquare,red);
add(pic1);
shipout(bbox(10));
```

Введем `pic2` с новой единицей `unitsize(pic2,18)`. В новых единицах рисуем квадрат с диагональю `36 bp`. Допустим, что нам нужно добавить его с центром в точке `(.5,0)`.

Команда `add(pic2,(.5,0));` не дает нужного результата. Помещаем `pic2` в `frame` с именем `f`:

```
frame f=pic2.fit();
```

Затем добавляем обычным образом:

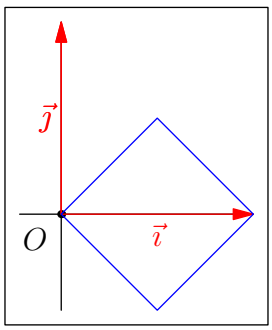
```
add(f,(.5,0));
```

Это равносильно следующему фрагменту кода:

```
frame f; // Определяем frame f
f=pic2.fit(); // Помещаем pic2 в frame
add(f,(.5,0)); // Добавляем f в точке (.5,0)
```

Приводим код с комментариями:

```
import geometry;
unitsize(36);// единица для currentpicture
show(currentcoordsys);
picture pic2;
unitsize(pic2,18); //единица для pic2
path p=(1,0)--(0,1)--(-1,0)--(0,-1)--cycle;
draw(pic2,p,blue); // Рисуем синий квадрат
// с центром в начале координат и диагональю
// на оси x
frame f=pic2.fit(); // Помещаем pic2 в frame
// с именем f
add(f,(.5,0)); // Добавляем f в точке (.5,0)
shipout(bbox(10));
```



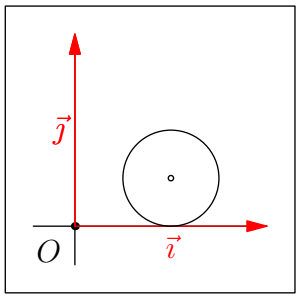
Определим третью картинку — `pic3` с новой единицей `unitsize(pic3,9)`. На ней рисуем единичную окружность, то есть радиуса `9 bp`. Команда

```
add(pic3,(.5,.25));
```

дает в результате маленькую окружность с центром `(.5,.25)` и радиусом `1 bp`. Для получения нужного результата опять применяем прием:

```
frame g=pic3.fit();
add(g,(.5,.25));
```

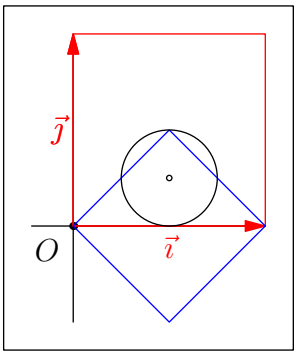
Полностью код выглядит так:




```
import geometry;
unitsize(36);
show(currentcoordsys);
picture  pic3; unitsize(pic3,9);
draw(pic3,unitcircle);
add(pic3,(.5,.25));
frame g=pic3.fit();
add(g,(.5,.25));
shipout(bbox(10));
```

Мы можем поместить все изображенное в этом примере на одном чертеже. Приведем код с комментариями.

```
import geometry;
unitsize(36);
show(currentcoordsys);
picture pic1, pic2, pic3;
unitsize(pic2,18);unitsize(pic3,9);
path p=(1,0)--(0,1)--(-1,0)--(0,-1)--cycle;
draw(pic2,p,blue); // синий квадрат на pic2
draw(pic1,unitsquare,red);
add(pic1); // добавим pic1 (красный  квадрат)
// к currentpicture
draw(pic3,unitcircle);
add(pic3,(.5,.25)); // добавим pic3 к
//currentpicture в точке (.5,.25); это дает
//окружность с центром в точке (.5,.25) и радиусом 1bp.
frame f=pic2.fit(); // вставляем pic2 в f
// (unitsize(pic2,18)!)
add(f,(.5,0)); // добавим f в currentpicture в
//точке (.5,0)
frame g=pic3.fit(); // вставляем pic3 в g
//(unitsize(pic3,9)!)
add(g,(.5,.25));
shipout(bbox(10));
```



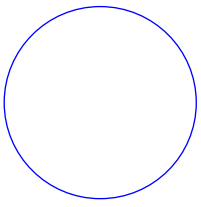
Пример 207

Пусть имеется 5 картинок (`picture`): `currentpicture, pic18, pic9, pic3, pic1`.

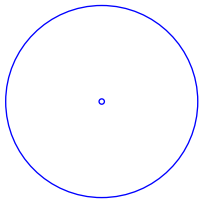
Единица длины `currentpicture - 36bp, pic18 - 18bp, pic9 - 9bp, pic3 - 3bp` и, наконец, единица длины `pic1` по умолчанию `1bp`.

Нарисуем на `pic1` синюю окружность радиуса `1 (1bp)`.

```
unitsize(36); // единица длины currentpicture: 36bp
picture pic1; // единица длины pic1: по умолчанию 1bp
draw(pic1,unitcircle,blue);// синяя окружность
                           // радиуса 1 (1bp) на pic1
add(pic1); // добавляем pic1 к currentpicture
```

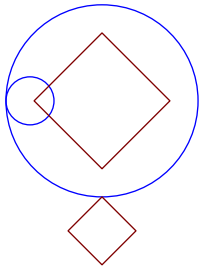


Получаем синюю окружность радиуса `36bp`, поскольку при добавлении картинки `B` к картинке `A` команда `add(A,B)` использует единицу картинки `A`, то есть той, к которой добавляется другая картинка. Чтобы добавить `pic1` с учетом его `unitsize (1bp)`, нужен следующий прием:



```
unitsize(36); // единица длины currentpicture: 36bp
picture pic1; // единица длины pic1: по умолчанию 1bp
draw(pic1,unitcircle,blue); // синяя окружность
                            //радиуса 1 (1bp) на pic1
add(pic1); // добавляем к  currentpicture, получаем
           // синюю окружность радиуса (1x36 bp)
add(pic1.fit()); // pic1.fit() - frame, учитывающий
                 // unitsize (1bp) на pic1. Добавляем
                 // его к currentpicture, получим синюю
                 // окружность радиуса 1x1 = 1bp
```

Замечание. Команду `add(pic1.fit());` можно использовать даже с указанием координат точки привязки, которые даются в единицах `picture`, к которой добавляется картинка. Приведем пример.



```
unitsize(36);
picture pic1; unitsize(pic1,9);
picture pic2; unitsize(pic2,18);
draw(pic1,unitcircle,blue);
draw(pic2,unitsquare,.5red);
add(pic1);// большая синяя окружность
add(pic1.fit(),(-.75,0));// маленькая синяя окружность
real s=sqrt(2)/2;
add(shift(0,s)*rotate(-135)*pic2); // большой квадрат
add(rotate(45)*pic2.fit(),(0,-1-s));// маленький квадрат
```

Пример 208

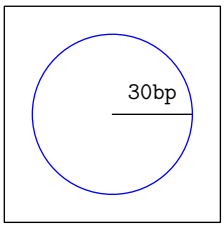
Ещё пример для лучшего понимания размеров. Есть картинки (`currentpicture, pic20, pic10, pic5, pic1`) с 5 разными единицами.

```
unitsize(30); // единица длины currentpicture: 30bp
picture pic20; unitsize(pic20,20); // единица длины pic2: 20bp
picture pic10; unitsize(pic10,10); // единица длины pic2: 10bp
picture pic5; unitsize(pic5,5); // единица длины pic2: 5bp
picture pic1; // единица длины pic1: по умолчанию 1bp
```

Нарисуем на `pic1` синюю окружность радиуса `1 (1 bp)` командой `draw(pic1,unitcircle, blue);`

Командой `add(pic1);` добавляем `pic1` к `currentpicture`. На картинке слева мы увидим синюю окружность радиуса (`1x30 bp`).

`pic1.fit()` — `frame`, который учитывает `unitsize` на `pic1`. Командой `add(pic1.fit());` добавляем `pic1.fit()` к `currentpicture`. На картинке справа видим результат: синюю окружность радиуса `1x1 = 1bp`.



Рисуем на `pic5` красную окружность радиуса `2(2x5bp)`, допечатывая в код команду:

```
draw(pic5,scale(2)*unitcircle,red);
```

Добавим картинку `pic5` к `pic10` с помощью:

```
add(pic10,pic5);
```

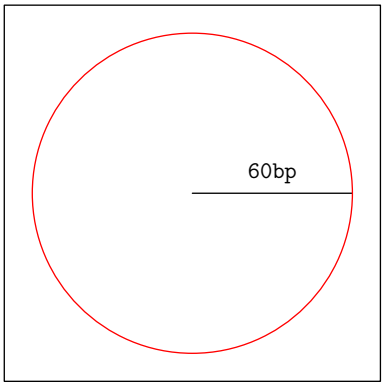
Затем добавляем картинку `pic10` к `pic20`:

```
add(pic20,pic10);
```

Окончательно, команда

```
add(pic20);
```

добавляет к `currentpicture` красную окружность радиуса `2(2x30 bp)`. Рисунок слева.



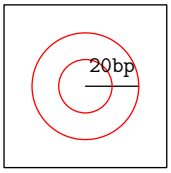


Рисунок справа получен кодом:

```
unitsize(30);
path p=(0,0)--(1/6,0)--(1/3,0)--(2/3,0);
draw(p);
picture pic10; unitsize(pic10,10);
picture pic5; unitsize(pic5,5);
draw(pic5,scale(2)*unitcircle,red);
add(pic5.fit());
add(pic10,pic5);
add(pic10.fit());
```

Команда

```
draw(pic5,scale(2)*unitcircle,red);
```

рисует на `pic5` красную окружность радиуса `2 (2x5bp)`. Команда `add(pic5.fit());` дает нам маленькую красную окружность радиуса `10bp`. Следующая за ней добавляет картинку `pic5` к `pic10`, наконец, последняя добавляет `pic10` к `currentpicture` с учетом `unitsize(pic10,10)`. Мы видим на рисунке справа большую красную окружность радиуса `2 (2x10bp)`.

Ранее в примерах использовалась команда `prepend()`, с помощью которой можно было устанавливать очерёдность наложения изображений.

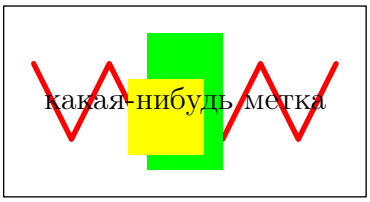
Ещё в `Asymptote` для манипуляций с последовательностью наложения изображений используют команду `layer()`.

Заметим, что эта команда разбивает код на "слои", в каждом из которых действуют установки по умолчанию. Но объекты верхнего "слоя" всегда рисуются поверх объектов нижнего.

Пример 209

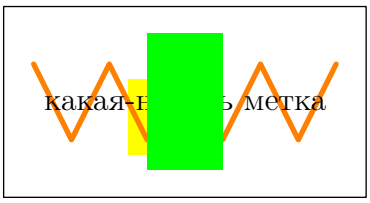
Рассмотрим картинку, на которой изображены четыре объекта: зигзагообразная ломаная красного цвета, зеленый прямоугольник, желтый квадрат и какая-нибудь метка. По умолчанию метка всегда рисуется поверх других объектов, где бы команда `label` не находилась в коде. Последовательность наложения остальных объектов устанавливается очередностью их появления в коде.

```
texpreamble("\usepackage[T2A]{fontenc}
\usepackage[utf8]{inputenc}\usepackage[russian]{babel}");
picture pic1;
unitsize(pic1,1cm);
label(pic1,"какая-нибудь метка",(5,1),W);
draw(pic1,(1,1.5)--(1.5,.5)--(2,1.5)--(2.5,.5)--(3,1.5)
--(3.5,.5)--(4,1.5)--(4.5,.5)--(5,1.5),2bp+red);
fill(pic1,box((2.5,.1),(3.5,1.9)),green);
fill(pic1,shift(2.25,.3)*unitsquare,yellow);
add(pic1.fit());
shipout(bbox(10,black));
```



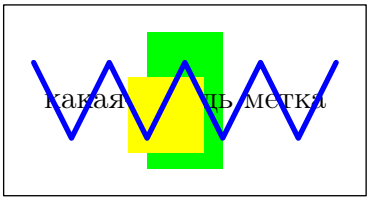
Для того, чтобы зеленый прямоугольник был нарисован поверх всех других объектов, даже поверх метки, разделим код командой `layer(pic2);`

```
texpreamble("\usepackage[T2A]{fontenc}
\usepackage[utf8]{inputenc}\usepackage[russian]{babel}");
picture pic2;
unitsize(pic2,1cm);
fill(pic2,shift(2.25,0.3)*unitsquare,yellow);
label(pic2,"какая-нибудь метка",(5,1),W);
draw(pic2,(1,1.5)--(1.5,.5)--(2,1.5)--(2.5,.5)--(3,1.5)
--(3.5,.5)--(4,1.5)--(4.5,.5)--(5,1.5),2bp+red+yellow);
layer(pic2);
fill(pic2,box((2.5,.1),(3.5,1.9)),green);
add(pic2.fit());
shipout(bbox(10,black));
```

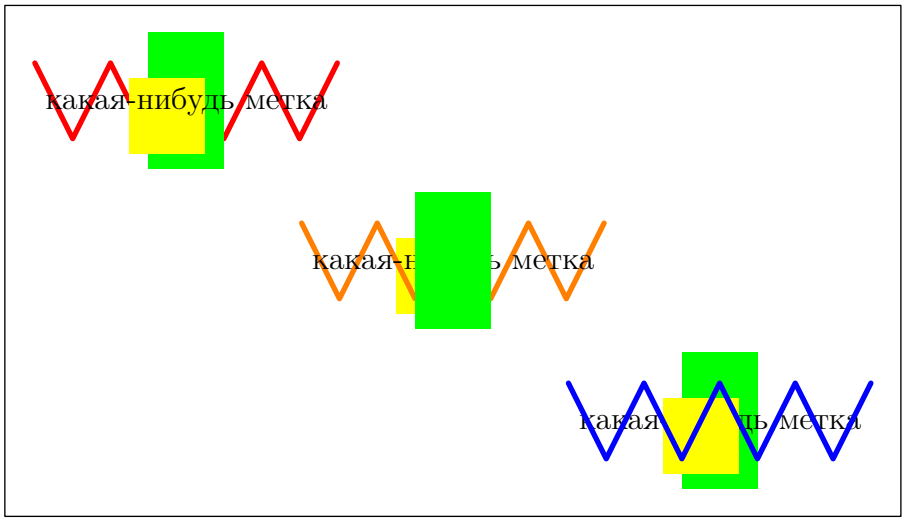


На следующем рисунке командой `layer(pic3);` разделяются два "слоя": в нижнем расположен зеленый прямоугольник и метка, а в верхнем — желтый квадрат, а поверх него синяя зигзагообразная ломаная.

```
texpreamble("\usepackage[T2A]{fontenc}
\usepackage[utf8]{inputenc}\usepackage[russian]{babel}");
picture pic3;
unitsize(pic3,1cm);
label(pic3,"какая-нибудь метка",(5,1),W);
fill(pic3,box((2.5,.1),(3.5,1.9)),green);
layer(pic3);
fill(pic3,shift(2.25,0.3)*unitsquare,yellow);
draw(pic3,(1,1.5)--(1.5,.5)--(2,1.5)--(2.5,.5)--(3,1.5)
--(3.5,.5)--(4,1.5)--(4.5,.5)--(5,1.5),2bp+blue);
add(pic3.fit());
shipout(bbox(10,black));
```



А вот как можно расположить представленные три картинки как три части единой картины. Для этого размещаем `pic1, pic2` и `pic3` на трех полотнах `frame f1`, `frame f2` и `frame f3`, затем добавляем их по нашему усмотрению. Например



Для этого используем следующий код:

```
texpreamble("\usepackage[T2A]{fontenc}
\usepackage[utf8]{inputenc}\usepackage[russian]{babel}");
//-------------------------------
picture pic1;
unitsize(pic1,1cm);
path p1=(1,1.5)--(1.5,.5)--(2,1.5)--(2.5,.5)--
                                    (3,1.5)--(3.5,.5)--(4,1.5)--(4.5,.5)--(5,1.5);
label(pic1,"какая-нибудь метка",(5,1),W);
draw(pic1, p1, 2bp+red);
fill(pic1,box((2.5,.1),(3.5,1.9)),green);
fill(pic1,shift(2.25,.3)*unitsquare,yellow);
//-------------------------------
picture pic2;
unitsize(pic2,1cm);
fill(pic2,shift(2.25,0.3)*unitsquare,yellow);
label(pic2,"какая-нибудь метка",(5,1),W);
draw(pic2, p1, 2bp+red+yellow);
layer(pic2);
fill(pic2,box((2.5,.1),(3.5,1.9)),green);
//-------------------------------
picture pic3;
unitsize(pic3,1cm);
label(pic3,"какая-нибудь метка",(5,1),W);
fill(pic3,box((2.5,.1),(3.5,1.9)),green);
layer(pic3);
fill(pic3,shift(2.25,0.3)*unitsquare,yellow);
draw(pic3, p1,2bp+blue);
//-------------------------------
frame f1=pic1.fit(), f2=pic2.fit(), f3=pic3.fit();
add(f1);
add(f2,(100,-60));
add(f3,(200,-120));
//-------------------------------
shipout(bbox(10,black));
```

Глава 3

Краткий обзор модулей

В настоящее время `Asymptote` состоит из следующих модулей:

plain, simplex, math, interpolate, geometry, trembling, stats, patterns, markers, tree, binarytree, drawtree, syzygy, feynman, roundedpath, animation, embed, slide, MetaPost, unicode, latin1, babel, labelpath, labelpath3, annotate, CAD, graph, palette, three, obj, graph3, grid3, solids, tube, flowchart, contour, contour3, ode.

Приведём очень краткий обзор модулей и лишь для некоторых из них дадим более развёрнутое описание.

1. Модуль *plain* — по умолчанию базовый файл `Asymptote`, который определяет основные части языка (как, например, структуру `picture`). Подключение (`private import plain;`) происходит неявно до трансляции файла и перед первой командой, заданной в интерактивном режиме. Это означает, что типы и функции, определенные в `plain`, доступны почти во всех кодах `Asymptote`.

2. Модуль *simplex* решает задачи линейного программирования двух переменных с помощью симплекс-метода. Используется в модуле `plain` для автоматической установки размеров рисунка.

3. Модуль *math* расширяет возможности `Asymptote` полезными математическими функциями.

4. Модуль *interpolate* позволяет использовать интерполяции Лагранжа, Эрмита и стандартную кубическую сплайн-интерполяцию в `Asymptote`.

5. Модуль *geometry*, написанный Филиппом Ивальди, предоставляет обширный набор геометрических процедур. Автор написал отличную книжку: *Philippe Ivaldi* **Géométrie euclidienne avec asymptote**, которую можно найти по адресу

   http://www.piprime.fr/files/res/geometry_fr.pdf

   Русский перевод книжки можно найти по адресу

   http://www.mif.vspu.ru/books/geometry_new_ru.pdf

   Большой набор примеров: http://www.piprime.fr/files/asymptote/geometry/

   Предметные указатели к модулю: index

6. Модуль *trembling*, также написанный Филиппом Ивальди, позволяет рисовать волнистыми линиями, как будто нарисованными от руки. Примеры размещены на страничке

   http://www.piprime.fr/files/asymptote/trembling/

7. Модуль *stats* реализует генерацию гауссовских случайных чисел и набор статистических процедур, включая гистограммы и (метод наименьших квадратов) `leastsquares`.

8. Модуль *patterns* реализует `PostScript`-модели штриховки и включает несколько удобных процедур генерации шаблонов штриховки.

9. Модуль *markers* реализует специализированные программы для маркировки путей и углов. Примеры предопределенных маркеров можно посмотреть в пособии *Christophe Grospellier* **Asymptote. Démarrage «rapide»**
   (Смотри http://www.cgmaths.fr/cgFiles/Dem_Rapide.pdf) и

   http://asy.marris.fr/asymptote/

   Предусмотрены процедуры создания новых маркеров.

10. Модуль *tree* реализует пример динамического двоичного дерева поиска.

11. Модуль *binarytree* может быть использован для рисования произвольных бинарных деревьев и включает процедуру для особого случая двоичного дерева поиска.

12. Модуль *drawtree* просто рисует деревья.

13. Модуль *syzygy* автоматизирует процесс рисования кос, отношений и узлов.

14. Модуль *feynman* полезен для рисования диаграмм Фейнмана. Автор Martin Wiebusch.

15. Модуль *roundedpath* полезен для округления острых углов путей. Автор Стефан Кнорр.

16. Модуль *animation* позволяет создавать анимации.

    Эти анимации используют программу `ImageMagick convert` для объединения нескольких изображений в GIF или MPEG фильмы.

    Родственный модуль **`animate`**, производный от модуля *animation*, генерирует портативные интерактивные PDF фильмы высокого качества с дополнительными элементами управления. Это требует установки пакета

    `http://www.ctan.org/tex-archive/macros/latex/contrib/animate/animate.sty`

    (версия 2007/11/30 или более поздняя версия) в новый каталог `animate` в локальном каталоге LaTeX(для примера, в `/usr/local/ share/texmf/tex/latex/animate`). В UNIX системах затем нужно выполнить команду `texhash`.

17. Модуль *embed* предоставляет интерфейс для LaTeX для встраивания фильмов, звука и 3D объектов в формате PDF. Подробности можно прочитать в [9], глава 8 [Base modules], стр.94.

18. Модуль *slide* предоставляет простое, но качественное средство для изготовления слайдов презентаций, в том числе переносимых и встроенных PDF анимаций.

19. Модуль *MetaPost* предоставляет пользователям несколько полезных процедур для преобразования старого `MetaPost`-кода в код `Asymptote`.

    В отличие от `MetaPost` `Asymptote` не решает неявно заданных линейных уравнений и, следовательно, не имеете команды `whatever`. Процедура `extension` (см. [9], [extension], стр. 34) обеспечивает полезную замену для `whatever`: нахождение точки пересечения линий, проходящих через p, q и P, Q. Менее употребительная замена `whatever` заключается в использовании встроенного оператора `solve`.

20. Модуль *unicode*. Команда `import unicode` в начале файла предписывает `LaTeX`'у принять стандартизированные международные символы `unicode` (UTF-8). Для использования кириллических шрифтов вам необходимо будет изменить кодировку шрифтов:

    ```
    import unicode;
    texpreamble("\usepackage{mathtext}\usepackage[russian]{babel}");
    defaultpen(font("T2A","cmr","m","n"));
    ```

    При установке **Ubuntu 12.04** приведенный способ не дает нужного результата. Однако можно использовать следующий код:

    ```
    texpreamble("\usepackage[T2A]{fontenc}\usepackage[utf8]{inputenc}
    \usepackage{mathtext}\usepackage[russian]{babel}");
    ```

    Два замечания:

    1) Подгружение пакета `\usepackage{mathtext}` не обязательно. Так же не обязательна строка третья `defaultpen(font("T2A "cmr "m "n");`

    2) В математической моде (например, между знаками $$) русский текст не воспроизводится.

21. Модуль *latin1*. Если у вас в `LaTeX` нет поддержки `unicode`, вы можете включить поддержку Западноевропейских языков (ISO 8859-1) при импорте модуля `latin1`. Этот модуль может быть использован в качестве шаблона для оказания поддержки других ISO 8859 алфавитов.

22. Модуль *babel* внедряет LaTeX-пакет `babel` в `Asymptote`. Например

```
import babel;
babel("german");
```

23. Модуль *labelpath* использует макрос `pstextpath` из пакета `PSTricks`, чтобы подогнать метки вдоль пути с помощью команды:

```
void labelpath(picture pic=currentpicture, Label L, path g,
string justify=Centered, pen p=currentpen);
```

Здесь `justify` есть `LeftJustified, Centered`, или `RightJustified`. Компонента x в преобразовании `shift` применительно к метке интерпретируется как сдвиг вдоль кривой, а компонента y интерпретируется как сдвиг в сторону от кривой. Все другие преобразования метки игнорируются. Этот пакет требует latex или tex-движка и наследует ограничения `\pstextpath` макро пакета `PSTricks`.

24. Модуль *labelpath3* написанный Jens Schwaiger, реализует 3D версию модуля *labelpath*, но не требует пакета `PSTricks`.

25. Модуль *annotate* поддерживает PDF аннотации для просмотра с помощью `Adobe Reader`, используя

```
void annotate(picture pic=currentpicture, string title, string text,
pair position);
```

В настоящее время аннотации внедряются только с помощью latex (по умолчанию) и tex-движков.

26. Модуль *CAD* содержит основные определения перьев и функции измерений для простого 2D CAD черчения в соответствии с DIN 15. Это документировано отдельно в файле `CAD.pdf`. Автор Mark Henning.

27. Модуль *graph* реализует двумерные линейные и логарифмические графики, включает автоматическое масштабирование и выбор отметок (с возможностью для переопределения вручную). График представляет собой `guide` (это можно сделать с помощью команды `draw` с дополнительной легендой)

28. Модуль *palette* `Asymptote` может также генерировать плотность цвета изображения и палитры. Палитры предопределенны в файле `palette.asy`:

29. Модуль *three* полностью расширяет понятие `guide` и `path` в `Asymptote` в трех измерениях. Он вводит новые типы `guide3, path3` и `surface`(поверхность). `guide3` в трех измерениях заданы с таким же синтаксисом, как и в двух измерениях, за исключением того, что вместо пары `(x,y)` для узлов и спецификаторов направлений используются тройки `(x,y,z)`.

30. Модуль *obj* позволяет построение поверхностей от простых `obj`-файлов.

31. Модуль *graph3* внедряет трехмерные версии функций из `graph.asy`. Имеет специальные процедуры для черчения осей в трех измерениях, поверхностей и векторных полей.

32. Модуль *grid3* написан Philippe Ivaldi, может использоваться для черчения 3D сеток. Примеры:
http://www.piprime.fr/developpeur/asymptote/examples-asy3d/grid3-asy/:

33. Модуль *solids*. Этот стереометрический пакет определяет структуру `revolution`, которую можно использовать для закраски и рисования поверхности вращения.

34. Модуль *tube* для изображения трубчатых поверхностей, представленных в файле
`three_arrows.asy` для представления сечений, преобразований цвета и преобразований вращения.

35. Модуль *flowchart* содержит программы для рисования блок-схем. Первичная структура — блок, он представляет собой единый блок на блок-схеме. Модуль содержит процедуры, позволяющие создавать блоки различной формы и процедуры их связи и позиционирования.

36. Модуль *contour* строит контурные линии. Для создания контуров, в соответствии со значениями из действительного массива `real[] c` для функции `f` от двух переменных, определенных в `box(a,b)`, используя процедуру

```
guide[][] contour(real f(real, real), pair a, pair b,
                  real[] c, int nx=ngraph, int ny=nx,
                  interpolate join=operator --, int subsample=1);
```

37. Модуль *contour3* чертит поверхности, описанные в нуль-пространства вещественнозначных функций `(x, y, z)` или `real[][][]` матриц. Его использование проиллюстрировано в примере файла `magnetic.asy`.

38. Модуль *slopefield* чертит поле касательных для дифференциального уравнения

$$dy/dx = f(x, y)(\text{ или } dy/dx = f(x))$$

39. Модуль *ode* реализует ряд схем явного численного интегрирования для обыкновенных дифференциальных уравнений.

Заключение

Автор не ставил задачи изложить тонкости программирования на `Asymptote` прежде всего потому, что сам не является профессиональным программистом и не обладает большим опытом работы с векторной графикой, а также потому, что опытные пользователи и программисты вполне способны самостоятельно разобраться с теми материалами, которые можно найти на головном сайте `Asymptote`:

http://asymptote.sourceforge.net.

При вёрстке настоящей версии пособия использовался дистрибутив Linux **Ubuntu 14.04 LTS**, LaTeX-дистрибутив TeX Live 2013, содержащий пакет `Asymptote` и LaTeX-редактор **Texmaker 4.1**.

Однако интерпретатор `Asymptote` — многоплатформенный, поэтому пользователи **Windows** могут установить не TeX Live, а MikTeX, использовать вместо LaTeX-редактора `Texmaker` редактор `TeXnicCenter` или другие, а для работы с `PostScript`-графикой должны дополнительно установить свободные программы `Ghostscript` и `GSview`. Для работы с `pdf`-графикой можно установить, например, `Adobe Reader`.

С благодарностью принимаются замечания и любая конструктивная критика.

Крячков Юрий Геннадьевич
e-mail jugkr45@yandex.ru

Литература

[1] *Гуссенс М., Ратц С., Миттельбах Ф.* Путеводитель по пакету LaTeX и его графическим расширениям. Иллюстрирование документов при помощи TeX'а и PostScript'а. — М.: Мир: Бином Л3, 2002.

[2] *Львовский С.М.* Набор и верстка в системе LaTeX, 3-е издание, испр. и доп. — М.: МЦНМО, 2003.

[3] *И. Котельников, П. Чеботаев* Издательская система LaTeX 2_ε, Сибирский хронограф, Новосибирск, 1998.

[4] *Котельников И.А., Чеботаев П.З.* LaTeX по-русски.— 3-е издание, перераб. и доп.— Новосибирск: Сибирский хронограф, 2004.

[5] *Балдин Е.М.* Компьютерная типография LaTeX, СПб.: БХВ-Петербург, 2008.

[6] *Кнут Д.Э.* Все про METAFONT. М.: Вильямс, 2003

[7] *John D. Hobby* METAPOST. Руководство пользователя, перевод с англ. В.Лидского, 2008

[8] *Балдин Е.М.* Создание иллюстраций в MetaPost, Linux Format, N 6-10, 2006

[9] *Andy Hammerlindl, John Bowman, Tom Prince* Asymptote. Vector Graphics Language