GD.pm - Интерфейс к графической библиотеке Gd

Название

GD.pm - Интерфейс к графической библиотеке Gd

Поддерживаемые платформы

- Linux
- Windows

Обзор

```
use GD;


# создаём новое изображение
$im = new GD::Image(100,100);

# назначаем некоторые цвета
$white = $im->colorAllocate(255,255,255);
$black = $im->colorAllocate(0,0,0);
$red = $im->colorAllocate(255,0,0);
$blue = $im->colorAllocate(0,0,255);

# устанавлеваем свойства заднего фона: transparent и interlaced
$im->transparent($white);
$im->interlaced('true');

#делаем чёрную рамку вокруг изображения
```

```
$im->rectangle(0,0,99,99,$black);

# Рисуем синий овал.
$im->arc(50,50,95,75,0,360,$blue);

# Заполняем его красным
$im->fill(50,50,$red);

# удостоверимся, что метод вывода - бинарный
binmode STDOUT;

# конвертируем изображение в формат PNG и выводим
print $im->png;
```

Описание

GD.pm это интерфейс к графической библиотеке gd Томаса Бутелла (Thomas Boutell) (см. ниже). GD позволяет Вам создавать различные изображения, используя множество графических методов, и сохранять/выводить их в формате PNG

В GD присутствует три следующих класса:

GD::Image
: Содержит данные о изображении и поддерживает основные графические функции.

GD::Font
: Создан для поддержки работы с текстом.

GD::Polygon
: Создан для работы с геометрическими фигурами.

Пример:

```
use GD;


# создаём новое изображение
$im = new GD::Image(100,100);

# назначаем некоторые цвета
$white = $im->colorAllocate(255,255,255);
$black = $im->colorAllocate(0,0,0);
$red = $im->colorAllocate(255,0,0);
$blue = $im->colorAllocate(0,0,255);

# устанавлеваем свой заднего фона: transparent и interlaced
$im->transparent($white);
$im->interlaced('true');

#делаем чёрную рамку вокруг изображения
$im->rectangle(0,0,99,99,$black);

# Рисуем синий овал.
$im->arc(50,50,95,75,0,360,$blue);

# Заполняем его красным
```

```
$im->fill(50,50,$red);

# удостоверимся, что метод вывода - бинарный
binmode STDOUT;

# конвертируем изображение в формат PNG и выводим
print $im->png;
```

Примечания:

1. Для создания новой, пустой картинки запросите метод `new()` у GD::Image, передав при этом ширину и высоту изображения, которое хотите создать. Вам будет возвращён объект-изображение. Другие методы позволят Вам создать новую катртинку из существующего PNG, GD or XBM файла.
2. Далее обычно добывляют цвета в цветовую таблицу изображения. Цвета добавляются при помощи вызова метода `colorAllocate()`. Три параметра, необходимые для инициализации цвета, это значения красного, зелёного и синего цветов, определённых для инициализируемого цвета. Метод вернёт индекс цвета в цветовой таблице изображения. Этот индекс понадобится Вам в дальнейшем.
3. Перешли к рисованию. Описание основных методов осуществлено ниже. В этом примере мы нарисовали некоторую строку, нарисовали овал и т.д.
4. Геометрические фигуры (далее просто, фигуры) создаются при помощи метода `new()` класса GD::Polygon. Вы можете добавлять точки к фигуре, используя метод `addPt()`. Далее фигура может быть передана изображению для дальнейшего использования.
5. После окончания прорисовки изображения Вы можете конвертировать его в формат PNG, использовав при этом метод `png()`. Данный метод вернёт Вам достаточно длинный скаляр, содержащий бинарные данные изображения. Обычно данный метод используется для вывода изображения на экран или сохранения в файл. перед выводом удостоверьтесь, что режим вывода - бинарный, вызвав метод `binmode()`.

Методы

Создание и сохранение изображений

new

`GD::Image->new(width,height)` *class method*

Для создания нового, пустого изображения обратитесь к методу `new()` класса GD::Image. Например:

```
$myImage = new GD::Image(100,100) || die;
```

Данный код приведёт к созданию изображения 100 x 100 пикселов. Если Вы не введёте параметры, то будет создано стандартное

изображение 64 x 64. Если что-то приведёт к ошибке (например, нехватка памяти), то метод вернёт значение undef.

newFromPng

GD::Image->newFromPng(FILEHANDLE) *class method*

Вызов данного метода создаст новое изображение из картинки, созданной и сохранённой ранее в формате PNG. Для этого должен быть предварительно открыт файл с картинкой, используя описатель FILEHANDLE. Если всё пройдёт успешно, то Вы получите картинку. В противном случае, который обычно происходит, если файл "на другом конце" описателя не PNG, будет возвращено undef. Отмечу, что метод автоматически вызывает binmode(FILEHANDLE) для Вас на платформах, где это необходимо.

Пример:

```
open (PNG,"barnswallow.png") || die;
$myImage = newFromPng GD::Image(PNG) || die;
close PNG;
```

newFromXbm

GD::Image->newFromXbm(FILEHANDLE) *class method*

Метод работает аналогично newFromPng, но расценивает содержимое, как файл X Bitmap (black & white):

```
open (XBM,"coredump.xbm") || die;
$myImage = newFromXbm GD::Image(XBM) || die;
close XBM;
```

newFromXpm

GD::Image->newFromXpm($filename) *class method*

Метод создаёт новый объект GD::Image из файла **filename**. Это отличие от предыдущих функций newFrom() связано с противоречивостью в основе библиотеки gd.

```
$myImage = newFromXpm GD::Image('earth.xpm') || die;
```

Данная функция доступна только, если libgd откомпилировано с поддержкой XPM.

newFromGd

GD::Image->newFromGd(FILEHANDLE) *class method*

Данная функция работает аналогично newFromPng, однако содержание воспринимает, как файл GD. GD - это формат Тома Бутелла (Tom Boutell) для сохранения на дисках, предназначенный для редких случаев, когда Вам необходимо считать и записать на диск быстро. Этот формат не предназначен для обыкновенного использования, как PNG или JPEG, т.к. не имеет графического сжатия и файлы могут

достигать достаточно **БОЛЬШИХ** размеров.

```
open (GDF,"godzilla.gd") || die;
$myImage = newFromGd GD::Image(GDF) || die;
close GDF;
```

newFromGd2

GD::Image->newFromGd2(FILEHANDLE) *class method*

Данная функция работает аналогично newFromgd(), однако использует новый сжатый формат GD2.

newFromGd2Part

GD::Image->newFromGd2Part(FILEHANDLE,srcX,srcY,width,height) *class method*

Данный метод позволяет получить часть изображения формата GD версии 2. В дополнение к filehandle, вводятся координаты верхнего левого угла части, а также ширина и высота куска. Например:

```
open (GDF,"godzilla.gd2") || die;
$myImage = GD::Image->newFromGd2Part(GDF,10,20,100,100) || die;
close GDF;
```

Читаем квадрат 100x100 картинки, начиная с позиции (10,20).

png

$image->png *object method*

Данные метод возвращает картинку в формате PNG. Пример:

```
$png_data = $myImage->png;
open (DISPLAY,"| display -") || die;
binmode DISPLAY;
print DISPLAY $png_data;
close DISPLAY;
```

Используйте: binmode(). Это важный момент для переносимости для DOS-платформ.

gd

$image->gd *object method*

Возвращает картинку в формате GD. Например:

```
binmode MYOUTFILE;
print MYOUTFILE $myImage->gd;
```

gd2

$image->gd2 *object method*

Аналогично gd(), только возвращает данные в формате сжатого GD2 format.

Управление цветом

colorAllocate

$image->colorAllocate(red,green,blue) *object method*

Данный метод назначает цвет, определённый каналами красного, зелёного и синего цветов. Возвращается индекс цвета в цветовой таблице. Первый цвет, определённый таким образом, становится фоновым цветом изображения. (255,255,255) - белый. (0,0,0) - чёрный (255,0,0) - довольно насыщенный и, если можно так сказать, глубокий красный. (127,127,127) - 50% серый.

Если цвет не назначен, то функция вернёт -1.

Например:

```
$white = $myImage->colorAllocate(0,0,0); #фон
$black = $myImage->colorAllocate(255,255,255);
$peachpuff = $myImage->colorAllocate(255,218,185);
```

colorDeallocate

$image->colorDeallocate(colorIndex) *object method*

Данный метод удаляет уже определённый индекс цвета, но выполняется это только при последующем вызове функции colorAllocate. Вы можете использовать данный метод несколько раз для ликвидации множества индексов цветов.

Например:

```
$myImage->colorDeallocate($peachpuff);
$peachy = $myImage->colorAllocate(255,210,185);
```

colorClosest

$image->colorClosest(red,green,blue) *object method*

Данный метод возвращает индекс цвета, наиболее "близкого" к заданному из уже созданных. Если ещё не было назначено цветов, то функция вернёт -1.

Например:

```
$apricot = $myImage->colorClosest(255,200,180);
```

colorExact

$image->colorExact(red,green,blue) *object method*

Данный метод возвращает индекс цвета, если цвет с точно такими же RGB компонентами был ранее назначен, в противном случае будет возвращена -1.

```
$rosey = $myImage->colorExact(255,100,80);
```

```
warn "Everything's coming up roses.\n" if $rosey >= 0;
```

colorResolve

`$image->colorResolve(red,green,blue)` *object method*

Данный метод вернёт индекс цвета, если цвет с такими RGB компонетами существует, в противном случае данный цвет будет назначен и его индекс возвращён.

```
$rosey = $myImage->colorResolve(255,100,80);
warn "Everything's coming up roses.\n" if $rosey >= 0;
```

colorsTotal

`$image->colorsTotal()` *object method*

Возвращает количество назначенных цветов.

```
$maxColors = $myImage->colorsTotal;
```

getPixel

`$image->getPixel(x,y)` *object method*

Данный метод возвращает индекс цвета в заданной точке изображения. Обычно данный метод используется вместе с `rgb()` для "разбития" полученного цвета на каналы.

Example:

```
$index = $myImage->getPixel(20,100);
($r,$g,$b) = $myImage->rgb($index);
```

rgb

`$image->rgb(colorIndex)` *object method*

Данный метод используется для определения RGB каналов (компонетов) цвета по индексу данного цвета.

Пример:

```
@RGB = $myImage->rgb($peachy);
```

transparent

`$image->transparent(colorIndex)` *object method*

Данный метод делает все пикселы, окрашенные в цвет заданного индекса прозрачными. Данный метод полезен для создания дополнительных фигур, также, собственно, как и в создании прозрачных фонов в PNG картинках для отображения в Интернете. Одновременно только один цвет может иметь это свойство. Для отключения прозрачности необходимо и достаточно в качестве параметра ввести -1.

Если Вы вызовите данный метод без параметров, то будет возвращён индекс текущего прозрачного цвета либо -1, если такой цвет не

определён.

Пример:

```
open(PNG,"test.png");
$im = newFromPng GD::Image(PNG);
$white = $im->colorClosest(255,255,255); #найти белый цвет
$im->transparent($white);
binmode STDOUT;
print $im->png;
```

Специфические цвета

GD реализует возможность использовать некоторые специальные цвета, которые могут быть использованы для достижения специфических эффектов.

setBrush
gdBrushed

`$image->setBrush( )` и `GD::gdBrushed`

Вы можете рисовать линии и другие фигуры, используя кисти-шаблоны. Под кистями понимаются изображения, которые Вы можете создавать и которыми Вы можете манипулировать как Вам угодно.

Пример:

```
# создаём кисть
$diagonal_brush = new GD::Image(5,5);
$white = $diagonal_brush->allocateColor(255,255,255);
$black = $diagonal_brush->allocateColor(0,0,0);
$diagonal_brush->transparent($white);
$diagonal_brush->line(0,4,4,0,$black);

# устанавливаем кисть
$myImage->setBrush($diagonal_brush);


# рисуем круг, используя установленную выше кисть
$myImage->arc(50,50,25,25,0,360,gdBrushed);
```

setStyle
gdStyled

`$image->setStyle(@colors)` and `GD::gdStyled`

Стилизированные линии состоят из произвольных серий повтроряющихся цветов. Они полезны для создания точечных и пунктирных линий. Для создания такой линии нужно использовать `setStyle` для назначения повторяющихся серий цветов. Допускаются массивы содержащие один или более индексов цветов, то есть массив не должен быть пустым. Далее рисуйте, используя добавленный `gdStyled` цвет. Другой дополнительный цвет, `gdTransparent` может быть использован для вставки прозрачных мест, при помощи следующего примера попытаемся проиллюстрировать это:

Пример:

```
# Устанавливаем стиль, содержащий 4 пиксела жёлтого,
# 4 пиксела синегои 2 прозрачных пиксела
$myImage->setStyle($yellow,$yellow,$yellow,$yellow,
                   $blue,$blue,$blue,$blue,
                   gdTransparent,gdTransparent);
$myImage->arc(50,50,25,25,0,360,gdStyled);
```

Если Вы хотите комбинировать gdStyled и gdBrushed, Вы можете назначить gdStyledBrushed. В этом случае, пикселы текущей кисти-шаблона заменяются каким-либо цветом, определённым в setStyle() и отличным от gdTransparent и 0.

gdTiled

Используется для создания мозаичных, составных изображений.

gdStyled

Используется для создания точечных и пунктирных линий. Такая линия создана при помощи команды setStyledи может состоять и различных повторяющихся комбинаций цветов.

Команды прорисовки

setPixel

$image->setPixel(x,y,color) *object method*

Данный метод устанавливает цвет пиксела с заданными координатами в соответствии с индексом, переданным в качестве третьего параметра. Система координат начинается с пиксела, находящегося в левом вернем углу изображения(координаты увеличиваются по мере перемещения птксела вниз и вправо). Вы можете использовать "нормальный" цвет либо цвет, установленный методами: gdBrushed, gdStyled и gdStyledBrushed.

Example:

```
# Предполагается, что $peach уже назначен
$myImage->setPixel(50,50,$peach);
```

line

$image->line(x1,y1,x2,y2,color) *object method*

Данный метод рисует линию от пиксела с координатами (x1,y1) до пиксела с координатами (x2,y2), индекс цвета определятся параметром color. Вы можете использовать "нормальный" цвет либо цвет, установленный методами: gdBrushed, gdStyled и gdStyledBrushed.

Example:

```
# Рисуем диагональ, используя
```

```
        # уже установленный шаблон
        $myImage->line(0,0,150,150,gdBrushed);
```

dashedLine

$image->dashedLine(x1,y1,x2,y2,color) *object method*

Данный метод рисует пунктирную линию заданного цвета между точками с координатами (x1,y1) и (x2,y2) Более мощным способом для рисования таких линий является использование метода setStyle(), который описан выше, и рисовании со специальным цветом gdStyled.

Пример:

```
        $myImage->dashedLine(0,0,150,150,$blue);
```

rectangle

GD::Image::rectangle(x1,y1,x2,y2,color) *object method*

Данный метод рисует прямоугольник заданным цветом. (x1,y1) и (x2,y2) - это координаты верхнего левого и нижнего правого углов этого прямоугольника соответственно. Вы можете использовать "нормальный" цвет либо цвет, установленный методами: gdBrushed, gdStyled и gdStyledBrushed.

Пример:

```
        $myImage->rectangle(10,10,100,100,$rose);
```

filledRectangle

$image->filledRectangle(x1,y1,x2,y2,color) *object method*

Рисует прямоугольник, аналогично предыдущему методу, но отличие в том, что данный метод не обрамляет, а заливает прямоугольник заданным цветом. Вы можете использовать "нормальный" цвет либо цвет, установленный методами: gdBrushed, gdStyled и gdStyledBrushed.

Пример:

```
        # читаем и устанавливаем заливочный шаблон
        open(PNG,"happyface.png") || die;
        $tile = newFromPng GD::Image(PNG);
        $myImage->setTile($tile);

        # рисуем залитый прямоугольник, используя уже установленный шаблон
        $myImage->filledRectangle(10,10,150,200,gdTiled);
```

polygon

$image->polygon(polygon,color) *object method*

Данный метод рисует многоугольник заданным цветом. Но для этого многоульник должен быть предварительно создан (см. ниже). Он должен иметь как минимум три вершины. Если последняя вершина "не замкнёт" многоугольник, то данный метод сделает это за Вас. Вы можете использовать "нормальный" цвет либо цвет, установленный

методами: gdBrushed, gdStyled и gdStyledBrushed.

Пример:

```
# Создаём новый многоугольник
$poly = new GD::Polygon;
# Добавляем вершины с заданными координатами:
$poly->addPt(50,0);
$poly->addPt(99,99);
$poly->addPt(0,99);
# Используем "наш" метод
$myImage->polygon($poly,$blue);
```

filledPolygon

`$image->filledPolygon(poly,color)` *object method*

Работает аналогично предыдущему методы, исключение - то, что многоугольник заливается заданным цветом. Вы можете использовать "нормальный" цвет либо цвет, установленный методами: gdBrushed, gdStyled и gdStyledBrushed.

Example:

```
# создаём многоугольник
$poly = new GD::Polygon;
$poly->addPt(50,0);
$poly->addPt(99,99);
$poly->addPt(0,99);

# рисуем многоугольник, используя заданный цвет
$myImage->filledPolygon($poly,$peachpuff);
```

arc

`$image->arc(cx,cy,width,height,start,end,color)` *object method*

Рисует дуги и эллипсы. (cx,cy) - координаты центра дуги. (width,height) определяют ширину и высоту. Если Вы хотите нарисовать часть эллипса, то Вы должны указать начальную и конечную позиции, задаваемые параметрами start и end, они задаются в градусах от 0 до 360. Ноль - в вершине эллипса, угол увеличивается по часовой стрелке. Для того чтобы нарисовать полный эллипс, укажите параметры start и end: 0 и 360 соответственно. Для рисования окружности параметры width и height должны быть одинаковыми.

Вы можете использовать "нормальный" цвет либо цвет, установленный методами: gdBrushed, gdStyled и gdStyledBrushed.

Пример:

```
# Рисуем полукруг с центром в точке с координатами 100,100.
$myImage->arc(100,100,50,50,0,180,$blue);
```

fill

`$image->fill(x,y,color)` *object method*

Данный метод заполняет область изображения, ограниченную цветами, отличными от заданного в качестве параметра. Данный метод работает подобно "paintbucket" в графических редакторах. Вы можете использовать "нормальный" цвет либо цвет, установленный методами: gdBrushed, gdStyled и gdStyledBrushed.

Пример:

```
# Рисуем прямоугольник, далее делаем его внутреннюю область синей
$myImage->rectangle(10,10,100,100,$black);
$myImage->fill(50,50,$blue);
```

$image->fillToBorder(x,y,bordercolor,color) ***object method***

Работает аналогично fill, отличие в том, что заливает до тех пор, пока не встретит цвет bordercolor.

Пример:

```
# Работает аналогично предыдущему примеру
$myImage->rectangle(10,10,100,100,$black);
$myImage->fillToBorder(50,50,$black,$blue);
```

Операции копирования

Предоставлятся два типа методов для копирования прямоугольных частей изображений. Первый позволяет копировать область без изменения её размеров. Другой метод позволяет сжимать/растягивать область при операции копирования.

Важно знать, что для обоих этих методов характерно то, что они попытаются расширить таблицу цветов изображения-получателя цветами, взятыми из источника. Если таблица цветов уже заполнена, то метод попытается определить наиболее соответствующий цвет среди уже имеющихся, то есть результат может быть отличен от оригинала.

copy

$image->copy(sourceImage,dstX,dstY,srcX,srcY,width,height) *object method*

Это наипростейший из нескольких типов операций копирования. Он копирует заданный участок изображения-источника и вставляет в изображение-получатель. (srcX,srcY) - координаты верхнего левого угла прямоугольного участка в изображении-источнике sourceImage, а (width,height) - ширина и высота этого участка соответственно. (dstX,dstY) - координаты, куда будет помещён штамп в изображении-получателе. Вы можете использовать одно и тоже изображение в качестве источника и получателя, однако копируемая и накладываемая части не должны пересекаться, в противном случае будут возникать различные некорректные картинки.

Пример:

```
$myImage = new GD::Image(100,100);
... рисуем что-либо ...
$srcImage = new GD::Image(50,50);
... ещё рисуем ...
# копируем участок 25x25 пикселов из $srcImage в
# прямоугольник, "начинающийся" в (10,10) изображения $myImage
$myImage->copy($srcImage,10,10,0,0,25,25);
```

clone

`$image->clone()` *object method*

Делает копию изображения и возвращает её, как новый объект. Новое изображение будет выглядеть идентично старому изображение, но может отличаться размером цветовой таблицы и другими несущественными деталями.

Пример:

```
$myImage = new GD::Image(100,100);
... рисуем ...
$copy = $myImage->clone;
```

`$image->copyMerge(sourceImage,dstX,dstY,srcX,srcY,width,height,percent)` *object method*

Данный метод копирует заданный прямоугольник из sourceImage и накладывает на изображение image (на прямоугольник с такими же размерами, верхний левый угол которого расположен в точке (dstX,dstY)), делая слияние в соответствии с процентным показателем (целое от 0 до 100), то есть если percent будет равняться 100, то действия данного метода будут аналогичны `copy()` - замена пикселей изображения-получателя на пикселы изображения-источника.

Пример:

```
$myImage = new GD::Image(100,100);
... рисуем ...
$redImage = new GD::Image(50,50);
... ещё рисуем ...
# копируем прямоугольник 25x25 из $srcImage и накладываем
# на прямоугольник с верхней левой вершиной (10,10) в $myImage, про
$myImage->copyMerge($srcImage,10,10,0,0,25,25,50);
```

`$image->copyMergeGray(sourceImage,dstX,dstY,srcX,srcY,width,height,percent)` *object method*

Аналогично `copyMerge()`, исключение в том, что картинка-получатель конвертируется в чёрно-белое перед слиянием.

copyResized

`$image->copyResized(sourceImage,dstX,dstY,srcX,srcY,destW,destH,srcW,srcH)` *object method*

Метод аналогичен copy(), различие в том, что он позволяет устанавливать произвольные значения прямоугольника-источника и прямоугольника-получателя. Длина и высота источника и получателя определяются в зависимости от (srcW,srcH) и (destW,destH) соответственно. copyResized() будет растягивать или уменьшать картинку для достижения установленных размеров.

Пример:

```
        $myImage = new GD::Image(100,100);
        ... рисуем ...
        $srcImage = new GD::Image(50,50);
        ... рисуем ...
        # копируем прямоугольник 25x25 из $srcImage и вставляем
        # в больший прямоугольник с главной вершиной (верхняя левая) (10,10
  # изображения $myImage
        $myImage->copyResized($srcImage,10,10,0,0,50,50,25,25);
```

Прорисовка текста

Gd позволяет Вам рисовать символы и строки обычным образом и повёрнутым на 90 градусов. За данные операции отвечает объект GD::Font, описанный ниже. Существует несколько встроенных шрифтов, они доступны в переменных gdGiantFont, gdLargeFont, gdMediumBoldFont, gdSmallFont and gdTinyFont. В настоящее время нет возможности создавать свои собственные шрифты.

string

$image->string(font,x,y,string,color) *Object Method*

Данный метод рисует строку, начиная с позиции (x,y) заданным шрифтом font и цветом color. В качестве шрифта должно быть установлено одно из следующих значений: gdSmallFont, gdMediumBoldFont, gdTinyFont, gdLargeFont and gdGiantFont.

Пример:

```
        $myImage->string(gdSmallFont,2,10,"Peachy Keen",$peach);
```

stringUp

$image->stringUp(font,x,y,string,color) *Object Method*

Работает аналогично предыдущему методу, однако рисует текст повёрнут на 90 градусов по часовой стрелке.

char
charUp

$image->char(font,x,y,char,color) *Object Method*
$image->charUp(font,x,y,char,color) *Object Method*

Эти методы прорисовывают символ с позиции (x,y) определённым шрифтом и цветом. Эти методы пришли из интерфейса для языка C,

где существует разница между символами и строками. Perl не чувствителен для таких тонких различий.

stringTTF

`@bounds = $image->stringTTF(fgcolor,fontname,ptsize,angle,x,y,string)` *Object Method* `@bounds = GD::Image->stringTTF(fgcolor,fontname,ptsize,angle,x,y,string)` *Class Method*

Этот метод использует TrueType для рисования маштабированной с устранёнными неровностями (сглаженной) строки, используя векторный шрифт TrueType, выбранный Вами. Также необходимо, чтобы libgd был откомпилирован с поддержкой TrueType, и, естественно, выбранный шрифт должен быть установлен в Вашей системе.

Аргументы означают следующее:

```
fgcolor    Индекс цвета шрифта
fontname   Абсолютный или относительный путь к файлу (.ttf) шрифта
ptsize     Желательный размер точки
angle      Угол поворота в радианах
x,y        Координаты X и Y, где будет начата прорисовка строки
string     Собственно сама строка
```

Если всё пройдёт успешно, то метод вернёт список из восьми элементов, предоставляя границы введённой строки:

```
@bounds[0,1]  Нижний левый угол (x,y)
@bounds[2,3]  Нижний правый угол (x,y)
@bounds[4,5]  Верхний правый угол (x,y)
@bounds[6,7]  Верхний левый угол (x,y)
```

В случае ошибки, такой как отсутствие шрифта либо поддержки TTF, метод вернёт пустой список и запишет в переменную $@ сообщение об ошибке.

Вы также можете вызвать данный метод из классового имени GD::Image, в случае если Вам не надо делать никаких прорисовок, Вам будет просто возвращён список элементов, описанных выше. Вы можете использовать это для "чертёжных" операций (набросок) You can use this to perform layout перед рисованием.

Разнообразные методы

interlaced

`$image->interlaced( )` `$image->interlaced(1)` *Object method*

Interlace создаёт эффект подъёмных жалюзей для некоторых вьюверов (понимаю под этим графический пакет, возможно, встроенный в броузер, позволяющий просматривать изображения). Если передать параметр "правда", то эффект будет активизирован,

если передать "неправда", то отключен. Для определения текущего параметра данного эффекта, вызовите данный метод без параметров.

getBounds

`$image->getBounds( )` *Object method*

Данный метод возвращает список из двух элементов, сответствующих размерам изобраения.

compare

`$image1->compare($image2)`

Сравнивает два изображения и, если различия найдены, то возвращает различия. Возвращённое значение должно быть логически "проANDировано" с одной или несколькими константами для определения различий. Существую следующие константы:

```
GD_CMP_IMAGE            Два изображения выглядят различными
GD_CMP_NUM_COLORS       Изображения имеют различное количество цветов
GD_CMP_COLOR            Цветовые палитры изображений различны
GD_CMP_SIZE_X           Горизонтальные размеры изображений различны
GD_CMP_SIZE_Y           Вертикальные размеры изображений различны
GD_CMP_TRANSPARENT      Изображения имеют различия в прозрачности
GD_CMP_BACKGROUND       Изображения имеют различные цвета фона
GD_CMP_INTERLACE        Изображения имею различия в параметре interlace
```

Наиболее важной константой есть GD_CMP_IMAGE, которая сообщает о различности двух изображений, игнорируя различия в количестве цветов и в цветовой палитре, а также в других невидимых свойствах. Константы не импортируются по умолчанию, они должны быть импортированы индивидуально либо при помощи тэга :cmp. Например:

```
use GD qw(:DEFAULT :cmp);
# Берём откуда-то $image1
# Берём откуда-то $image2
if ($image1->compare($image2) & GD_CMP_IMAGE) {
   warn "images differ!";
}
```

Методы рисования многоулольников

Существуют некоторые методы создания и изменения простейших многоугольников. Эти методы не являются частью библиотеки Gd, но они довольно удобны.

new

`GD::Polygon->new` *class method*

Создаёт пустой объект многоугольника без вершин.

```
$poly = new GD::Polygon;
```

addPt

$poly->addPt(x,y) *object method*

Добавляет вершину (x,y) к многоугольнику.

```
$poly->addPt(0,0);
$poly->addPt(0,50);
$poly->addPt(25,25);
$myImage->fillPoly($poly,$blue);
```

getPt

$poly->getPt(index) *object method*

Возвращает координаты вершины, соответствующей переданному индексу.

```
($x,$y) = $poly->getPt(2);
```

setPt

$poly->setPt(index,x,y) *object method*

Изменяет значения координат вершины с заданным индексом. Возникнет ошибка, если Вы попробуете изменить координаты несуществующей вершины.

```
$poly->setPt(2,100,100);
```

deletePt

$poly->deletePt(index) *object method*

Удаляет вершину с заданным индексом, возвращая значения её координат.

```
($x,$y) = $poly->deletePt(1);
```

toPt

$poly->toPt(dx,dy) *object method*

Добавляет вершину, испозуя координаты текущей вершины и относительные координаты (dx,dy). То есть, если последней вершиной, которую Вы добавили, является вершина с координатами (25,25), то следствием работы данного метода с параметрами (0,10) будет создание вершины с координатами (25,35) Если это первая вершина, то работает аналогично addPt().

```
$poly->addPt(0,0);
$poly->toPt(0,50);
$poly->toPt(25,-25);
$myImage->fillPoly($poly,$blue);
```

length

$poly->length *object method*

Возвращает количество вершин многоугольника.

```
$points = $poly->length;
```

vertices

`$poly->vertices` *object method*

Возвращает список всех вершин объекта. Каждый член списка является ссылкой на массив (x,y), содержащий координаты вершины.

```
@vertices = $poly->vertices;
foreach $v (@vertices)
   print join(",",@$v),"\n";
}
```

bounds

`$poly->bounds` *object method*

Возвращает наименьший прямугольник, который полностью покрывает данный многоугольник. Возвращаемое значение является массивом, содержащим параметры многоугольника (left,top,right,bottom).

```
($left,$top,$right,$bottom) = $poly->bounds;
```

offset

`$poly->offset(dx,dy)` *object method*

Перемещает все вершины многоугольника на заданное количество пикселов (dh) по горизонтали и (dy) по вертикали. Положительные значения соответствуют перемещению вправо и вверх.

```
$poly->offset(10,30);
```

scale

`$poly->scale(sx,sy)` *object method*

Маштабирует координаты вершин в соответствии с заданными параметрами. Например `scale(2,2)` сделает многоугольник в два раза больше. Для лучших результатов переместите центр многоугольника в точку (0,0) перед маштабированием, а после переместите центр обратно.

transform

`$poly->transform(sx,rx,sy,ry,tx,ty)` *object method*

Пропускает многоугольник через трансформационную матрицу, где sx и sy - это факторы маштабирования координат (подобно scale), rx и ry - фокторы поворота, а tx и ty - факторы перемещения многоугольника (подобно offset). Посмотрите "Adobe PostScript Reference", страница 154 для полного описания данного эффекта либо поэкспериментируйте.

Утилиты шрифтов

gdSmallFont

`GD::Font->Small` *constant*

Это основной маленький шрифт, "взятый" из хорошо известного "6x12".

gdLargeFont

`GD::Font->Large` *constant*

Основной большой шрифт, взятый из шрифта 8x16.

gdMediumBoldFont

`GD::Font->MediumBold` *constant*

Это средний жирный шрифт, по размеру находится между шрифтами small и large (7x13 font).

gdTinyFont

`GD::Font->Tiny` *constant*

Крохотный, практически нечитабельный шрифт, размером 5x8 пикселей.

gdGiantFont

`GD::Font->Giant` *constant*

Это жирный шрифт 9x15, конвертированный Яном Паздзиором (Jan Pazdziora) из шрифта Sans Serif X11.

nchars

`$font->nchars` *object method*

Возвращает кол-во символов шрифта.

```
print "The large font contains ",gdLargeFont->nchars," characters\n
```

offset

`$font->offset` *object method*

Возвращает значение ASCII первого символа шрифта

width
height

`$font->width` `GD::Font::height` *object methods*

Возвращает размеры символов шрифта в пикселах.

```
($w,$h) = (gdLargeFont->width,gdLargeFont->height);
```

Получение версии GD для языка C

libgd, версия gd для языка C, может быть найдена по алресу http://www.boutell.com/gd/. Описание и информация по установке может быть найдена на этом сайте.

Copyrights

Последняя версия интерфейса доступна на

http://stein.cshl.org/WWW/software/GD