PHP: работа с изображениями

Было бы довольно расточительно хранить и передавать все рисунки в обыкновенном растровом формате (наподобие BMP), тем более что современные алгоритмы сжатия позволяют упаковывать такого рода данные в сотни и более раз эффективнее. Чаще всего для хранения изображений в Web используются три формата сжатия с перечисленными ниже свойствами.

- JPEG. Идеален для фотографий, но сжатие изображения происходит с потерями качества, так что этот формат совершенно не подходит для хранения различных диаграмм и графиков.
- GIF. Позволяет достичь довольно хорошего соотношения «размер/качество», в то же время не искажая изображение; применяется в основном для хранения небольших точечных рисунков и диаграмм.
- PNG. Сочетает в себе хорошие стороны как JPEG, так и GIF, но даже сейчас он все еще не очень распространен — скорее всего, по историческим причинам, из-за нежелания отказываться от GIF и т. д.

Зачем может понадобиться в Web-программировании работа с изображениями? Разве это не работа дизайнера?

В большинстве случаев это действительно так. Однако есть и исключения, например, графические счетчики (автоматически создаваемые картинки с отображаемым поверх числом, которое увеличивается при каждом «заходе» пользователя на страницу), или же графики, которые пользователь может строить в реальном времени — скажем, диаграммы сбыта продукции или снижения цен на комплектующие. Все эти приложения требуют как минимум умения генерировать изображения «на лет», причем с довольно большой скоростью. Чтобы этого добиться на PHP, можно применить два способа: задействовать какую-нибудь внешнюю утилиту для формирования изображения или воспользоваться встроенными функциями PHP для работы с графикой. Оба способа имеют как достоинства, так и недостатки, но, пожалуй, недостатков меньше у второго метода. Его и рассмотрим.

Всюду ниже слово int означает, что ожидается значение целого типа, string – что ожидается значение строкового типа и т.п.

Универсальная функция getimagesize()

Среди наиболее распространенных операций можно особо выделить одну — определение размера рисунка. Разработчики PHP встроили в него функцию, которая работает практически со всеми распространенными форматами изображений, в том числе с GIF, JPEG и PNG.

list **getimagesize**(string $filename)

Эта функция предназначена для быстрого определения в сценарии размеров (в пикселах) и формата рисунка, имя файла которого ей передано. Она возвращает список из следующих элементов.

- Нулевой элемент (с ключом 0) хранит ширину картинки в пикселах.
- Первый элемент (с ключом 1) содержит высоту изображения.
- Ячейка массива с ключом 2 определяется форматом изображения: 1 = GIF, 2 = JPG, 3=PNG, 4=SWF, 5=PSD, 6=BMP, 7=TIFF (на Intel-процессорах), 8=TIFF (на процессорах Motorola), 9 = JPC, 10 = JP2, 11 = JPX, 12 = JB2, 13 = SWC, 14 = IFF, 15=WBMP, 16=XBM. Можно также использовать и константы вида IMAGETYPE_XXX, встроенные в PHP, где XXX соответствует названию формата (только что мы перечислили все поддерживаемые форматы).
- Элемент, имеющий ключ 3, содержит строку примерно следующего вида: "height=sx width=sy", где sx и sy — соответственно ширина и высота изображения. Указанный элемент задумывался для того, чтобы облегчить вставку данных о размере изобра- жения в тег <img>, который может быть сгенерирован сценарием.
- Ячейка с индексом 4 содержит число битов, используемых для хранения информации о каждом пикселе изображения.
- Элемент с ключом 5 содержит количество цветовых каналов, задействованных в изображении. Для JPEG-картинок в формате RGB он будет равен 3, а в формате CMYK — 4.
- Элемент со строковым ключом mime хранит MIME-тип изображения (например, image/gif, image/jpeg и т. д.). Его очень удобно использовать при выводе заголовка Content-type, определяющего тип изображения.

В случае ошибки функция возвращает false и генерирует предупреждение.

Следует передавать в качестве $filename файловый путь (относительный или абсолютный), но не URL.

Пусть в текущем каталоге у нас есть набор картинок.

В первом листинге приведен скрипт random, который выводит случайную картинку из текущего каталога. При этом не важно, имеет ли изображение формат GIF или JPEG — нужный заголовок Content-type определяется автоматически. Попробуйте понажимать кнопку Обновить в браузере, чтобы наблюдать эффект смены картинок.

Тексты программ: https://github.com/GossJS/php7/tree/master/gd

Так выглядит работа этого сценария:
http://kodaktor.ru:9876/php/random.php

Вот его подсвеченный код, доступный для копирования:

```php
<?php ## Автоопределение MIME-типа изображения.
  // Выбираем случайное изображение любого формата.
  $fnames = glob("*.{gif,jpg,png}", GLOB_BRACE);
  $fname = $fnames[mt_rand(0, count($fnames)-1)];
  // Определяем формат.
  $size = getimagesize($fname);
  // Выводим изображение.
  header("Content-type: {$size['mime']}");
  echo file_get_contents($fname);
```

Работа с изображениями и библиотека GD

Рассмотрим создание рисунков сценарием «на лету». Например, это очень может пригодиться при создании сценариев-счетчиков, графиков, картинок-заголовков и многого другого.

Для деятельности такого рода существует ряд библиотек и утилит. Наиболее распространены библиотеки ImageMagick (http://www.imagemagick.org) и **GD** (http://www.boutell.com/gd).

Рассмотрим встроенную в PHP библиотеку под названием GD — точнее, ее вторую версию, GD2. Она содержит в себе множество функций (таких как рисование линий, растяжение/сжатие изображения, заливка до границы, вывод текста ит.д.), которые могут использовать программы, поддерживающие работу с данной библиотекой. PHP (с включенной поддержкой GD) как раз и является такой программой.

Для установки расширения в операционной системе Windows необходимо отредактировать конфигурационный файл php.ini, раскомментировав строку

extension=php_gd2.dll

В случае Ubuntu, установить расширение можно при помощи команды

$ sudo apt-get install php5-gd

или

$ sudo apt-get install php-gd

(без этого будет выдаваться ошибка типа функция imageCreateFromPng не найдена)

программа начнёт работать сразу после установки, нет нужды перезапускать PHP

Для Mac OS X можно воспользоваться менеджером пакетов Homebrew, указав директиву --with-gd при установке

$ brew install php70 --with-gd

либо отдельно установив расширение при помощи команды

Пример создания изображения

Начнем сразу с примера сценария (button.php), который представляет собой не HTML-страницу в обычном смысле, а рисунок PNG. То есть URL этого сценария можно поместить в тег:

<img src="button.php?Hello+world!">

Как только будет загружена страница, содержащая указанный тег, сценарий запустится и отобразит надпись Hello world! на фоне рисунка, находящегося в файле button.gif. Полученная картинка нигде не будет храниться — она создается «на лету».

http://kodaktor.ru:9876/php/button.php?hello+world

```php
<?php ## Создание картинки «на лету».
  // Получаем строку, которую нам передали в параметрах
  $string = $_SERVER['QUERY_STRING'] ?? 'Hello, world!';
  // Загружаем рисунок фона с диска.
  $im = imageCreateFromGif('button.gif'); // http://kodaktor.ru/button.gif
  // Создаем в палитре новый цвет - черный.
  $color = imageColorAllocate($im, 0, 0, 0);
  // Вычисляем размеры текста, который будет выведен.
  $px = (imageSX($im)-6.5*strlen($string))/2;
  // Выводим строку поверх того, что было в загруженном изображении.
  imageString($im, 3, $px, 1, $string, $color);
  // Сообщаем о том, что далее следует рисунок PNG.
  header('Content-type: image/png');
  // Теперь - самое главное: отправляем данные картинки в
  // стандартный выходной поток, т. е. в браузер.
  imagePng($im);
  // В конце освобождаем память, занятую картинкой.
  imageDestroy($im);
```

Итак, мы получили возможность «на лету» создавать стандартные кнопки с разными надписями, имея только «шаблон» кнопки.

Обратите внимание на важный момент: мы считали изображение в формате GIF (функция imageCreateFromGif()), но вывели в браузер — уже в виде PNG (imagePng()).

Создание изображения

Рассмотрим, как работать с изображениями в GD. Для начала нужно создать пустую картинку (при помощи функции imageCreate()) или же загруженную с диска (функции imageCreateFromPng(), imageCreateFromJpeg() или imageCreateFromGif()).

resource **imageCreate**(int $x, int $y)

Создает «пустую» палитровую (palette-based, т. е. с фиксированным набором возможных цветов) картинку размером $x на $y точек и возвращает ее идентификатор. После того как картинка создана, вся работа с ней осуществляется именно через этот идентификатор, по аналогии с тем, как мы работаем с файлом через его дескриптор.

resource **imageCreateTrueColor**(int $x, int $y)

Данная функция отличается от предыдущей только тем, что она создает полноцветные изображения. В таких изображениях число цветов не ограничено палитрой, и можно использовать точки любых оттенков. Обычно imageCreateTrueColor() применяют для создания JPEG-изображений, а также для более аккуратного манипулирования картинками (например, при их сжатии или растяжении), чтобы не «потерять цвета».

Загрузка изображения

resource **imageCreateFromPng**(string $filename)

resource **imageCreateFromJpeg**(string $filename)

resource **imageCreateFromGif**(string $filename)

Эти функции загружают изображения из файла в память и возвращают их идентификаторы. Как и после вызова функции imageCreate(), дальнейшая работа с картинкой возможна только через этот идентификатор. При загрузке с диска изображение распаковывается и хранится в памяти уже в неупакованном формате, для того чтобы можно было максимально быстро производить с ним различные операции, например масштабирование, рисование линий и т. д. При сохранении на диск или выводе в браузер функцией imagePng() (или, соответственно, imageJpeg() и imageGif()) картинка автоматически упаковывается.

ЗАМЕЧАНИЕ

Функция imageCreateFromJpeg() всегда формирует полноцветное изображение. Функция же imageCreateFromPng() создает в памяти палитровую картинку только в том случае, если PNG-файл, указанный в параметрах, содержал палитру (PNG-изображения бывают как палитровыми, так и полноцветными).

Интересно, что функции imageCreateFrom*() могут работать не только с именами файлов, но также и с URL (в случае, если в настройках файла php.ini разрешен режим allow_url_fopen).

Определение параметров изображения

Как только картинка создана и получен ее идентификатор, библиотеке GD становится совершенно все равно, какой формат она (картинка) имеет и каким путем ее создали. То есть все остальные действия с картинкой происходят через ее идентификатор вне зависимости от формата, и это логично — ведь в памяти изображение все равно хранится в распакованном виде (наподобие BMP), а значит, информация о ее формате нигде не используется. Так что вполне возможно открыть PNG-изображение с помощью функции imageCreateFromPng() и сохранить ее на диск функцией imageJpeg(), уже в другом формате. В дальнейшем можно в любой момент времени определить размер загруженной картинки, воспользовавшись функциями imageSX() и imageSY().

int **imageSX**(int $im)

Функция возвращает горизонтальный размер изображения, заданного своим идентифи- катором, в пикселах.

int **imageSY**(int $im)

Возвращает высоту картинки в пикселах.

int **imageColorsTotal**(int $im)

Эту функцию имеет смысл применять только в том случае, если вы работаете с изображениями, «привязанными» к конкретной палитре, например с файлами GIF или PNG. Она возвращает текущее количество цветов в палитре. Каждый вызов imageColorAllocate() увеличивает размер палитры. В то же время известно, что если при небольшом размере палитры GIF- и PNG-картинка сжимается очень хорошо, то при переходе через степень двойки (например, от 16 к 17 цветам) эффективность сжатия заметно падает, что ведет к увеличению размера (так уж устроены форматы). Если мы не хотим этого допустить и собираемся вызывать imageColorAllocate() только до предела 16 цветов, а затем перейти на использование imageColorClosest(), может пригодиться рассматриваемая функция.

ЗАМЕЧАНИЕ

Примечательно, что для полноцветных изображений функция imageColorsTotal() всегда возвращает 0. Например, если вы создали картинку вызовом imageCreateFromJpeg() или imageCreateTrueColor(), то узнать размер ее палитры вам не удастся — её нет.

bool **imageIsTrueColor**(resource $im)

Функция позволяет определить, является ли изображение с идентификатором $im полноцветным или же палитровым. В первом случае возвращается true, во втором — false.

Рассмотрим функцию, которая выводит изображение в браузер пользователя. Эту же функцию можно применять и для сохранения рисунка в файл.

int **imagePng**(resource $im [,string $filename] [,int $quality])
int **imageJpeg**(resource $im [,string $filename] [,int $quality])
int i**mageGif**(resource $im [,string $filename])

Перечисленные функции сохраняют изображение, заданное своим идентификатором и находящееся в памяти, на диск или же выводят его в браузер. Разумеется, вначале изображение должно быть загружено или создано при помощи функции imageCreate() (или imageCreateTrueColor()), т. е. мы должны знать его идентификатор $im.

Если аргумент $filename опущен (или равен пустой строке "" или NULL), то сжатые данные в соответствующем формате отправляются прямо в стандартный выходной поток, т. е. в браузер. Нужный заголовок Content-type при этом не выводится, ввиду чего нужно указывать его вручную при помощи header(), как это было показано в примере из листинга 1.

ВНИМАНИЕ!

Некоторые браузеры не требуют вывода правильного Content-type, а определяют, что перед ними рисунок, по нескольким первым байтам присланных данных. На это нельзя полагаться, т.к. все еще существуют браузеры, которые этого делать не умеют. Кроме того, такая техника идет вразрез со стандартами HTTP.

Фактически вы должны вызвать одну из двух команд, в зависимости от типа изображения:

header('Content-type: image/png'); // для PNG
header('Content-type: image/jpeg'); // для JPEG

Рекомендуется их вызывать не в начале сценария, а непосредственно перед вызовом imagePng() или imageJpeg(), поскольку иначе никак не получится увидеть сообщения об ошибках и предупреждения, которые, возможно, будут сгенерированы программой.

Необязательный параметр $quality, который может присутствовать для JPEG-изображений, указывает качество сжатия. Чем лучше качество (чем больше $quality), тем большим получается размер изображения, но тем качественнее оно выглядит. Диапазон изменения параметра — от 0 (худшее качество, маленький размер) до 100 (лучшее качество, но большой размер).

Преобразование изображения в палитровое

Иногда бывает, что загруженная с диска картинка имеет полноцветный формат (JPEG или PNG), а нам в программе нужно работать с ней, как с палитровой, причем размер палитры указывается явно. Для осуществления преобразований можно применять описанную далее функцию.

void **imagetruecolortopalette**(resource $im, bool $dither, int $ncolors)

Функция принимает своим первым параметром идентификатор некоторого загруженного ранее (или созданного) полноцветного изображения и производит его конверта- цию в палитровое представление. Число цветов в палитре задается параметром $ncolors. Если аргумент $dither равен true, то PHP и GD будут стараться не просто подобрать близкие цвета в палитре для каждой точки, но и имитировать «смешение» цветов путем заливки области подходящими оттенками из палитры. Подобные изображения выглядят «шероховато», но обычно их «огрехи», связанные с неточной передачей цвета палитрой, меньше бросаются в глаза.

Работа с цветом в формате RGB

Наверное, нужно что-нибудь нарисовать на пустой или только что за- груженной картинке. Но чтобы рисовать, нужно определиться, каким цветом это делать. Проще всего указать цвет заданием тройки RGB-значений (от англ. red, green, blue — красный, зеленый, синий) — это три значения от 0 до 255, определяющие содержание красной, зеленой и синей составляющих в нужном нам цвете. Число 0 обозначает нулевую яркость соответствующего компонента, а 255 — максимальную интенсивность. Например, (0, 0, 0) задает черный цвет, (255, 255, 255) — белый, (255, 0, 0) — ярко-красный, (255, 255, 0) — желтый и т. д.

Библиотека GD не работает с такими тройками напрямую. Она требует, чтобы перед использованием RGB-цвета был получен его специальный идентификатор. Дальше вся работа опять же осуществляется через этот идентификатор.

Создание нового цвета

int **imageColorAllocate**(int $im, int $red, int $green, int $blue)

Функция возвращает идентификатор цвета, связанного с соответствующей тройкой RGB. Обратите внимание, что первым параметром функция требует идентификатор изображения, загруженного в память или созданного до этого. Практически каждый цвет, который планируется в дальнейшем использовать, должен быть получен (опреде- лен) при помощи вызова этой функции. Почему «практически» — станет ясно после рассмотрения функции imageColorClosest().

Текстовое представление цвета

Как видно, цвет в функции imageColorAllocate() указывается в виде трех различных параметров. То же самое касается и остальных «цветных» функций: они все принимают минимум по три параметра. Однако в реальной жизни цвет часто задается в виде строки

'RRGGBB', где RR — шестнадцатеричное представление красного цвета, GG — зеленого и BB — синего. Например, строки #RRGGBB используются в HTML-атрибутах color, bgcolor и т. д. Да и хранить одну строку проще, чем три числа. В связи с этим приведём код, осуществляющий преобразования строкового представления цвета в числовое:

```
$txtcolor = 'FFFF00';
sscanf($txtcolor, "%2x%2x%2x", $red, $green, $blue);
$color = imageColorAllocate($ime, $red, $green, $blue);
```

Мы используем здесь функцию sscanf(). Для выполнения обратного преобразования воспользуйтесь следующим кодом:

```
$txtcolor = sprintf("%2x%2x%2x", $red, $green, $blue);
```

Получение ближайшего в палитре цвета

Зачем нужна такая технология работы с цветами через промежуточное звено — идентификатор цвета? Дело в том, что некоторые форматы изображений (такие как GIF и частично PNG) не поддерживают любое количество различных цветов в изображении. (как если бы художнику дали палитру с фиксированным набором различных красок и запретили их смешивать при рисовании, заставляя использовать только «чистые» цвета.) А именно, в GIF-формате количество одновременно присутствующих цветов ограничено числом 256, причем чем меньше цветов используется в рисунке, тем лучше он «сжимается» и тем меньший размер имеет файл. Тот набор цветов, который реально использован в рисунке, называется его палитрой.

Представим себе, что произойдет, если все 256 цветов уже «заняты» и вызывается функция imageColorAllocate(). В этом случае она обнаружит, что палитра заполнена полностью, и найдет среди занятых цветов тот, который ближе всего находится к запрошенному — будет возвращен именно его идентификатор. Если же «свободные места» в палитре есть, то они и будут использованы этой функцией (конечно, если в палитре вдруг не найдется точно такой же цвет, как запрошенный — обычно дублирование одинаковых цветов всячески избегается).

ПРИМЕЧАНИЕ

При работе с полноцветными изображениями никакой палитры, конечно, нет. Поэтому новые цвета можно создавать практически до бесконечности.

int imageColorClosest(int $im, int $red, int $green, int $blue)

Зачем нужна функция imageColorClosest()? Вместо того чтобы пытаться выискать свободное место в палитре цветов, она просто возвращает идентификатор цвета, уже существующего в рисунке и находящегося ближе всего к затребованному. Таким образом, новый цвет в палитру не добавляется. Если палитра невелика, то функция может вернуть не совсем тот цвет, который вы ожидаете. Например, в палитре из трех цветов «красный-зеленый-синий» на запрос желтого цвета будет, скорее всего, возвращен идентификатор зеленого — он «ближе всего» с точки зрения GD соответствует понятию «желтый».

Эффект прозрачности

Функцию imageColorClosest() можно и нужно использовать, если мы не хотим допустить разрастания палитры и уверены, что требуемый цвет в ней уже присутствует. Однако есть и другое, гораздо более важное, ее применение — определение эффекта прозрачности для изображения. «Прозрачный» цвет рисунка — это просто те точки, которые в браузер не выводятся. Таким образом, через них «просвечивает» фон. Прозрачный цвет у картинки всегда один, и задается он при помощи функции imageColorTransparent().

int **imageColorTransparent**(int $im [,$int col])

Функция указывает GD, что соответствующий цвет $col (заданный своим идентификатором) в изображении $im должен обозначиться как прозрачный. Возвращает она идентификатор нового цвета или текущего цвета, если ничего не изменилось. Если аргумент $col не задан и в изображении нет прозрачных цветов, функция вернет -1.

Например, мы нарисовали при помощи GD что-то на кислотно-зеленом фоне и хотим, чтобы этот фон как раз и был «прозрачным»

В этом случае потребуются такие команды:

```
$tc = imageColorClosest($im, 0, 255, 0);
imageColorTransparent($im, $tc);
```

Обратите внимание на то, что применение функции imageColorAllocate() здесь совершенно бессмысленно, потому что нам нужно сделать прозрачным именно тот цвет, который уже присутствует в изображении, а не новый, только что созданный.

ВНИМАНИЕ!

Задание прозрачного цвета поддерживают только палитровые изображения, но не полноцветные. Например, картинка, созданная при помощи imageCreateFromJpeg() или imageCreateTrueColor(), не может его содержать.

Получение RGB-составляющих

array **imageColorsForIndex**(int $im, int $index)

Функция возвращает ассоциативный массив с ключами red, green и blue (именно в таком порядке), которым соответствуют значения, равные величинам компонентов RGB в идентификаторе цвета $index. Впрочем, мы можем и не обращать особого внимания на ключи и преобразовать возвращенное значение как список:

```
$c = imageColorAt($i,0,0);
list($r, $g, $b) = array_values(imageColorsForIndex($i, $c));
echo "R=$r, g=$g, b=$b";
```

Эта функция ведет себя противоположно по отношению к imageCollorAllocate() или imageColorClosest().

Использование полупрозрачных цветов

Полупрозрачным называют цвет в изображении, сквозь который «просвечивают» другие точки (как сквозь цветное стекло). Например, загрузив некоторое изображение и нарисовав на нем что-нибудь полупрозрачным цветом, вы получите эффект «просвечивания»: имеющиеся точки изображения «смешаются» с новым цветом, и результат будет записан в память, как обычно.

ПРИМЕЧАНИЕ

Еще иногда употребляют термин «alpha-канал», что означает место в графическом файле, отведенное на хранение информации о полупрозрачности.

С точки зрения библиотеки GD полупрозрачные цвета ничем не отличаются от обычных, но создавать их нужно функциями imageColorAllocateAlpha(), imageColorClosestAlpha(), imageColorExactAlpha() и т. д. Перечисленные функции вызываются точно так же, как и их не-alpha-аналоги, однако при обращении необходимо указать еще один, пятый, параметр: степень прозрачности. Он изменяется от 0 (полная непрозрачность) до 127 (полная прозрачность).

Далее приведен скрипт, демонстрирующий применение полупрозрачных цве- тов.

Файл semitransp.php

```php
<?php ## Работа с полупрозрачными цветами.
  $size = 300;
  $im = imageCreateTrueColor($size, $size);
  $back = imageColorAllocate($im, 255, 255, 255);
  imageFilledRectangle($im, 0, 0, $size - 1, $size - 1, $back);
  // Создаем идентификаторы полупрозрачных цветов.
  $yellow = imageColorAllocateAlpha($im, 255, 255, 0, 75);
  $red    = imageColorAllocateAlpha($im, 255, 0, 0, 75);
  $blue   = imageColorAllocateAlpha($im, 0, 0, 255, 75);
  // Рисуем 3 пересекающихся круга.
  $radius  = 150;
  imageFilledEllipse($im, 100, 75,  $radius, $radius, $yellow);
  imageFilledEllipse($im, 120, 165, $radius, $radius, $red);
  imageFilledEllipse($im, 187, 125, $radius, $radius, $blue);
  // Выводим изображение в браузер.
  header('Content-type: image/png');
  imagePng($im);
```

Данный скрипт выводит три разноцветных пересекающихся круга, и в местах пересечения можно наблюдать эффектное смешение цветов.

Увидеть результат его работы можно по адресу
http://kodaktor.ru:9876/php/semitrans.php

Графические примитивы

Рассмотрим примеры функций для работы с картинками. Список таких функций не полон и постоянно расширяется вместе с развитием библиотеки GD. За полным списком функций обращайтесь к официальной документации по адресу: http://ru.php.net/manual/ru/ref.image.php.

Копирование изображений

int imageCopyResized(int $dst_im, int $src_im, int $dstX, int $dstY, int $srcX, int $srcY, int $dstW, int $dstH, int $srcW, int $srcH)

Эта функция одна из самых мощных и универсальных, с ее помощью можно копировать изображения (или их участки), перемещать и масштабировать их.

$dst_im задает идентификатор изображения, в который будет помещен результат работы функции. Это изображение должно уже быть создано или загружено и иметь надлежащие размеры. Соответственно, $src_im — идентификатор изображения, над которым проводится работа. Впрочем, $src_im и $dst_im могут и совпадать.

ВНИМАНИЕ!

Следите, чтобы изображение $dst_im было полноцветным, а не палитровым! В противном случае возможно искажение или даже потеря цветов при копировании. Полноцветные изо- бражения создаются, например, вызовом функции imageCreateTrueColor().

Параметры $srcX, $srcY, $srcW, $srcH (обратите внимание на то, что они следуют при вызове функции не подряд!) задают область внутри исходного изображения, над кото- рой будет осуществлена операция — соответственно, координаты ее верхнего левого угла, ширину и высоту.

Наконец, четверка $dstX, $dstY, $dstW, $dstH задает то место на изображении $dst_im, в которое будет «втиснут» указанный в предыдущей четверке прямоугольник. Если ширина или высота двух прямоугольников не совпадают, то картинка автоматически будет нужным образом растянута или сжата.

Таким образом, с помощью функции imageCopyResized() мы можем:

- копироватьизображения;
- копировать участки изображений;
- масштабировать участки изображений;
- копировать и масштабировать участки изображения в пределах одной картинки.

int **imageCopyResampled**(int $dst_im, int $src_im, int $dstX, int $dstY, int $srcX, int $srcY, int $dstW, int $dstH, int $srcW, int $srcH)

Данная функция очень похожа на imageCopyResized(), но у нее есть одно очень важное отличие: если при копировании производится изменение размеров изображения, биб- лиотека GD пытается провести сглаживание и интерполяцию точек. А именно, при увеличении картинки недостающие точки заполняются промежуточными цветами (функция imageCopyResized() этого не делает, а заполняет новые точки цветами распо- ложенных рядом точек).

Впрочем, качество интерполяции функции imageCopyResampled() все равно оставляет желать лучшего. Например, при большом увеличении легко наблюдать «эффект сту- пенчатости»: видно, что плавные цветовые переходы имеют место только по горизонтали, но не по вертикали. Таким образом, функция еще может использоваться для увеличения фотографий (JPEG-изображений), но увеличивать с ее помощью GIF-картинки не рекомендуется.

http://kodaktor.ru:9876/php/resample.php

Файл resample.php

```php
<?php ## Увеличение картинки со сглаживанием.
  $from = imageCreateFromJpeg('sample2.jpg');
  $to   = imageCreateTrueColor(2000, 2000);
  imageCopyResampled(
    $to, $from, 0, 0, 0, 0, imageSX($to), imageSY($to),
    imageSX($from), imageSY($from)
  );
  header('Content-type: image/jpeg');
  imageJpeg($to);
```

Исходная картинка: http://kodaktor.ru:9876/php/sample2.jpg

Это простейший сценарий, который увеличивает некоторую картинку до размера 2000×2000 точек и выводит результат в браузер. Ему необходимо хотя бы 16 Мбайт свободной оперативной памяти.

Прямоугольники

int **imageFilledRectangle**(int $im, int $x1, int $y1, int $x1, $y2, int $col)

Эта функция рисует закрашенный прямоугольник в изображении, заданном идентификатором $im, цветом $col (полученным, например, при помощи функции imageColorAllocate()). Пары ($x1, $y1) и ($x2, $y2) задают координаты левого верхнего и правого нижнего углов соответственно (отсчет, как обычно, начинается с левого верхнего угла и идет слева направо и сверху вниз).

Эта функция часто применяется для того, чтобы целиком закрасить только что созданный рисунок, например, прозрачным цветом:

```
$i = imageCreate(100, 100);
$c = imageColorAllocate($i, 1, 255, 1);
imageFilledRectangle($i, 0, 0, imageSX($i)-1, imageSY($i)-1, $c);
imageColorTransparent($i, $c);
// Дальше работаем с изначально прозрачным фоном
```

int **imageRectangle**(int $im, int $x1, int $y1, int $x2, int $y2, int $col)

Функция imageRectangle() рисует в изображении прямоугольник с границей, толщиной 1 пиксел, цветом $col. Параметры задаются так же, как и в функции imageFilledRectangle().

Вместо цвета $col можно задавать константу IMG_COLOR_STYLED, которая говорит библиотеке GD, что линию нужно рисовать не сплошную, а с использованием текущего стиля пера (см. далее).

Выбор пера

При рисовании прямоугольника толщина линии составляет всего 1 пиксел. Однако в PHP существует возможность задания любой толщины линии, для чего служит следующая функция.

bool i**mageSetThickness**(resource $im, int $thickness)

Устанавливает толщину пера, которое используется при рисовании различных фигур: прямоугольников, эллипсов, линий и т. д. По умолчанию толщина равна 1 пикселу, однако можно задать любое другое значение.

bool i**mageSetStyle**(resource $image, list $style)

Данная функция устанавливает стиль пера, который определяет, пикселы какого цвета будут составлять линию. Массив $style содержит список идентификаторов цветов, предварительно созданных в скрипте. Эти цвета будут чередоваться при выводе линий.

Если очередную точку при выводе линии необходимо пропустить, можно указать вместо идентификатора цвета специальную константу IMG_COLOR_TRANSPARENT (и получить, таким образом, пунктирную линию).

Файл pen.php

```php
<?php ## Изменение пера.
  // Создаем новое изображение.
  $im  = imageCreate(100, 100);
  $w   = imageColorAllocate($im, 255, 255, 255);
  $c1  = imageColorAllocate($im, 0, 0, 255);
  $c2  = imageColorAllocate($im, 0, 255, 0);
  // Очищаем фон.
  imageFilledRectangle($im, 0, 0, imageSX($im), imageSY($im), $w);
  // Устанавливаем стиль пера.
  $style = array($c2, $c2, $c2, $c2, $c2, $c2, $c2, $c1, $c1, $c1, $c1);
  imageSetStyle($im, $style);
  // Устанавливаем толщину пера.
  imageSetThickness($im, 2);
  // Рисуем линию.
  imageLine($im, 0, 0, 100, 100, IMG_COLOR_STYLED);
  // Выводим изображение в браузер.
  header('Content-type: image/png');
  imagePng($im);
?>
```

Линии

int **imageLine**(int $im, int $x1, int $y1, int $x2, int $y2, int $col)

Эта функция рисует сплошную тонкую линию в изображении $im, проходящую через точки ($x1, $y1) и ($x2, $y2), цветом $col. Линия получается слабо связанной (см. далее).

Задав константу IMG_COLOR_STYLED вместо идентификатора цвета, мы получим линию, нарисованную текущим установленным стилем пера. Для рисования пунктирной линии используйте совокупность функций imageSetStyle() и imageLine().

Дуга сектора

int imageArc(int $im, int $cx, int $cy, int $w, int $h, int $s, int $e, int $c)

Функция imageArc() рисует в изображении $im дугу сектора эллипса от угла $s до $e (углы указываются в градусах против часовой стрелки, отсчитываемых от горизонта- ли). Эллипс рисуется такого размера, чтобы вписываться в прямоугольник ($x, $y, $w, $h), где $w и $h задают его ширину и высоту, а $x и $y — координаты левого верхнего угла. Сама фигура не закрашивается, обводится только ее контур, для чего используется цвет $c. Если в качестве значения $c указана константа IMG_COLOR_STYLED, дуга будет нарисована в соответствии с текущим установленным стилем пера.

Закраска произвольной области

int **imageFill**(int $im, int $x, int $y, int $col)

Функция imageFill() выполняет сплошную заливку одноцветной области, содержащей точку с координатами ($x, $y), цветом $col. Нужно заметить, что современные алгорит- мы заполнения работают довольно эффективно, так что не стоит особо заботиться о скорости ее работы. Итак, будут закрашены только те точки, к которым можно про- ложить "одноцветный сильно связанный путь" из точки ($x, $y).

ПРИМЕЧАНИЕ

Две точки называются связанными сильно, если у них совпадает, по крайней мере, одна координата, а по другой координате они отличаются не более чем на 1 в любую сторону.

int **imageFillToBorder**(int $im, int $x, int $y, int $border, int $col)

Эта функция очень похожа на imageFill(), только она выполняет закраску не одно- цветных точек, а любых, но до тех пор, пока не будет достигнута граница цвета $border. Под границей здесь понимается последовательность слабо связанных точек.

Две точки называются слабо связанными, если каждая их координата отличается от другой не более чем на 1 в любом направлении. Очевидно, всякая сильная связь является также и слабой, но не наоборот. Все линии библиотека GD рисует слабо связанными: иначе бы они выглядели слишком «ступенчатыми».

Закраска текстурой

Закрашивать области можно не только одним цветом, но и некоторой фоновой картин- кой — текстурой. Это происходит, если вместо цвета всем функциям закраски указать специальную константу IMG_COLOR_TILED. При этом текстура "размножается" по верти- кали и горизонтали, что напоминает кафель на полу (от англ. tile). Это объясняет назва- ние следующей функции.

int **imageSetTile**(resource $im, resource $tile)

Устанавливает текущую текстуру закраски $tile для изображения $im. При последую- щем вызове функций закраски (таких как imageFill() или imageFilledPolygon()) с пара- метром IMG_COLOR_TILED вместо идентификатора цвета область будет заполнена данной текстурой.

Многоугольники

int **imagePolygon**(int $im, list $points, int $num_points, int $col)

Эта функция рисует в изображении $im многоугольник, заданный своими вершинами. Координаты углов передаются в массиве-списке $points, причем $points[0]=x0,

$points[1]=y0, $points[2]=x1, $points[3]=y1 и т. д. Параметр $num_points указывает об- щее число вершин — на тот случай, если в массиве их больше, чем нужно нарисовать. Многоугольник не закрашивается — только рисуется его граница цветом $col.

int **imageFilledPolygon**(int $im, list $points, int $num_points, int $col)

Функция imageFilledPolygon() делает практически то же самое, что и imagePolygon(), за исключением одного очень важного свойства: полученный многоугольник целиком заливается цветом $col. При этом правильно обрабатываются вогнутые части фигуры, если она не выпукла.

Работа с пикселами

int **imageSetPixel**(int $im, int $x, int $y, int $col)

Эта функция практически не интересна, т. к. выводит всего один пиксел, расположенный в точке ($x, $y), цвета $col в изображении $im. Вряд ли ее можно применять для закраски хоть какой-то сложной фигуры. Даже рисование обычной линии с использованием этой функции будет долгим и ресурсоёмким.

resource **imageColorAt**(int $im, int $x, int $y)

В противоположность своему антиподу — функции imageSetPixel() — функция imageColorAt() не рисует, а возвращает цвет точки с координатами ($x, $y). Возвращается идентификатор цвета, а не его RGB-представление.

Функцию удобно использовать, опять же, для определения, какой цвет в картинке должен быть прозрачным. Например, пусть для изображения птички на кислотно-зеленом фоне мы достоверно знаем, что прозрачный цвет точно приходится на точку с координатами (0, 0). Мы можем получить его идентификатор, а затем назначить прозрачным (imageColorTransparent()) .

Работа с фиксированными шрифтами

Библиотека GD имеет некоторые возможности по работе с текстом и шрифтами. Шрифты представляют собой специальные ресурсы, имеющие собственный идентификатор и чаще всего загружаемые из файла или встроенные в GD. Каждый символ шрифта может быть отображен лишь в моноцветном режиме, т. е. «рисованные» символы не поддерживаются. Встроенных шрифтов всего 5 (идентификаторы от 1 до 5), чаще всего в них входят моноширинные символы разных размеров. Остальные шрифты должны быть предварительно загружены.

ЗАМЕЧАНИЕ

Изначально библиотека GD, конечно, не поддерживает русские буквы во встроенных шрифтах. Тем не менее большинство хостинг-провайдеров добавляют эту поддержку. В любом случае лучше использовать TTF-шрифты (см. ниже).

Загрузка шрифта

int **imageLoadFont**(string $file)

Функция загружает файл шрифтов и возвращает идентификатор шрифта — это будет цифра, бóльшая 5, потому что пять первых номеров зарезервированы как встроенные. Формат файла — бинарный, а потому зависит от архитектуры машины. Это значит, что файл со шрифтами должен быть сгенерирован по крайней мере на машине с процессо- ром такой же архитектуры, как и у той, на которой используется PHP. В таблице ниже представлен формат этого файла. Левая колонка задает смещение начала данных внутри файла, а группами цифр, записанных через тире, определяется, до какого адреса продолжаются данные.

Смещение, байт	Тип	Описание
0—3	long	Число символов в шрифте (nchars)
4—7	long	Индекс первого символа шрифта (обычно 32 — пробел)
8—11	long	Ширина (в пикселах) каждого знака (width)
12—15	long	Высота (в пикселах) каждого знака (height)
От 16 и выше	array	Массив с информацией о начертании каждого символа, по одному байту на пиксел. На один символ, таким образом, приходится width×height байтов, а на все — width×height×nchars байтов. 0 означает отсутствие точки в данной позиции, все остальное — ее присутствие

В большинстве случаев гораздо удобнее брать уже готовые шрифты, чем делать новые самостоятельно. Используйте шрифты из Интернета, помня об авторском праве.

Параметры шрифта

После того как шрифт загружен, его можно использовать (встроенные шрифты, конечно же, загружать не требуется).

int **imageFontHeight**(int $font)

Возвращает высоту в пикселах каждого символа в заданном шрифте.

int **imageFontWidth**(int $font)

Возвращает ширину в пикселах каждого символа в заданном шрифте.

Вывод строки

int **imageString**(int $im, int $font, int $x, int $y, string $s, int $col)

Выводит строку $s в изображение $im, используя шрифт $font и цвет $col. Пара ($x, $y) будет координатами левого верхнего угла прямоугольника, в который вписана строка.

int **imageStringUp**(int $im, int $font, int $x, int $y, string $s, int $c)

Эта функция также выводит строку текста, но не в горизонтальном, а в вертикальном направлении. Левый верхний угол задается координатами ($x, $y).

Работа со шрифтами TrueType

Библиотека GD поддерживает работу с векторными масштабируемыми шриф- тами PostScript и TrueType. Рассмотрим только последние, т. к., во-первых, их существует множество, а во-вторых, с ними проще всего работать в PHP.

ЗАМЕЧАНИЕ

Для того чтобы заработали приведенные далее функции, PHP должен быть откомпилирован и установлен вместе с библиотекой FreeType, доступной по адресу http://www. freetype.org. В Windows-версии PHP она установлена по умолчанию. Большинство хостинг- провайдеров добавляют ее и под UNIX.

Всего существуют две функции для работы со шрифтами TrueType. Одна из них выводит строку в изображение, а вторая определяет, какое положение эта строка бы заняла при выводе.

Вывод строки

list **imageTtfText**(int $im, int $size, int $angle, int $x, int $y, int $col, string $fontfile, string $text)

Эта функция помещает строку $text в изображение $im цветом $col. Как обычно, $col должен представлять собой допустимый идентификатор цвета. Параметр $angle задает угол наклона в градусах выводимой строки, отсчитываемый от горизонтали против часовой стрелки. Координаты ($x, $y) указывают положение так называемой базовой точ- ки строки (обычно это ее левый нижний угол). Параметр $size задает размер шрифта, используемый при выводе строки. Наконец, $fontfile должен содержать имя TTF- файла, в котором, собственно, и хранится шрифт.

Параметр $fontfile должен всегда задавать абсолютный путь (от корня файловой системы) к требуемому файлу шрифтов. Если в программе задано относительное имя TTF-файла, используйте realpath() для конвертации его в абсолютное.

Функция возвращает список из 8 элементов. Первая их пара задает координаты (X, Y) левого верхнего угла прямоугольника, описанного вокруг строки текста в изображении, вторая пара — координаты правого верхнего угла, и т. д. Так как в общем случае строка может иметь любой наклон $angle, здесь требуются 4 пары координат. И, тем не менее, функция всегда возвращает координаты так, будто бы угол $angle был равен нулю.

Проблемы с русскими буквами

Если нужно выводить текст, содержащий русские буквы, то должны вначале перекодировать его в специальное представление. В этом представлении каждый знак кириллицы имеет вид &#XXXX, где XXXX — код буквы в кодировке Unicode. Знаки препинания и символы латинского алфавита в перекодировании не нуждаются.

Определение границ строки

list **imageTtfBBox**(int $size, int $angle, string $fontfile, string $text)

Эта функция ничего не выводит в изображение, а просто определяет, какой размер и положение заняла бы строка текста $text размера $size, выведенная под углом $angle в какой-нибудь рисунок. Параметр $fontfile, как и в функции imageTtfText(), задает абсолютный путь к файлу шрифта, который будет использован при выводе.

Возвращаемый список содержит всю информацию о размерах описанного прямоугольника в формате, похожем на тот, что выдает функция imageTtfText(), однако на этот раз — с учетом угла поворота. (Правда, учтен этот угол неправильно; в следующем разделе мы рассмотрим, как обойти эту ситуацию.) Для большей ясности приведем эту информацию в виде таблицы:

Индексы	Что содержится
0 и 1	(*X*, *Y*) левого нижнего угла
2 и 3	(*X*, *Y*) правого нижнего угла
4 и 5	(*X*, *Y*) правого верхнего угла
6 и 7	(*X*, *Y*) левого верхнего угла

Стороны прямоугольника не обязательно параллельны горизонтальной или вертикальной границе изображения. Они могут быть наклонными, а сам прямоугольник — повернутым. Потому-то и возвращаются 4 координаты, а не две.

Коррекция функции imageTtfBBox()

Авторы библиотеки FreeType, которая используется для вывода TTF-текста, допустили ошибки, и в результате функция imageTtfBBox() возвращает правильные данные только при нулевом угле наклона строки. В листинге ниже приведена библиотека подпрограмм, в которой этот недостаток исправляется (вводится новая функция imageTtfBBox_fixed()); кроме того, в ней содержатся ещё две полезные функции.

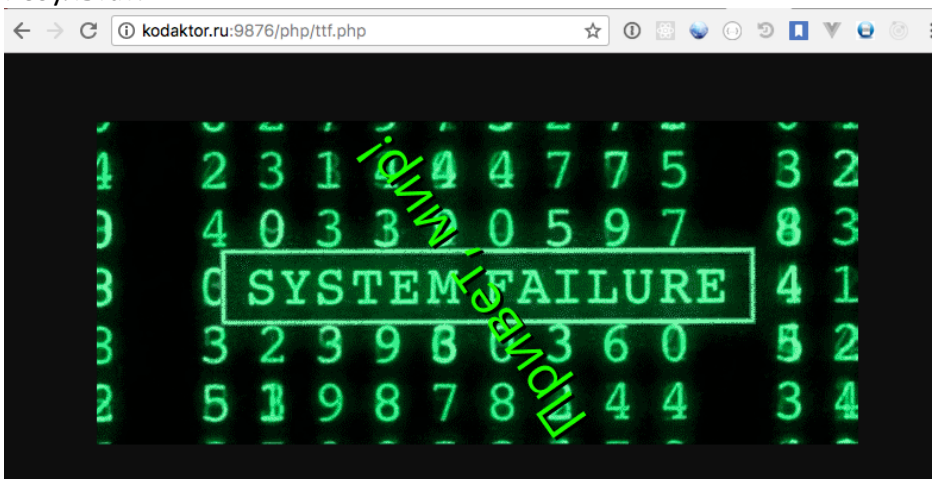
Это файл lib/imagettf.php

```php
<?php ## Библиотека функций для работы с TTF.
  // Исправленная функция imageTtfBBox(). Работает корректно
  // даже при ненулевом угле поворота $angle (исходная функция
  // при этом работает неверно).
  function imageTtfBBox_fixed($size, $angle, $fontfile, $text) {
    // Вычисляем размер при НУЛЕВОМ угле поворота.
    $horiz = imageTtfBBox($size, 0, $fontfile, $text);
    // Вычисляим синус и косинус угла поворота.
    $cos = cos(deg2rad($angle));
    $sin = sin(deg2rad($angle));
    $box = [];
    // Выполняем поворот каждой координаты.
    for ($i = 0; $i < 7; $i += 2) {
      list ($x, $y) = [$horiz[$i], $horiz[$i + 1]];
      $box[$i]   = round($x * $cos + $y * $sin);
      $box[$i+1] = round($y * $cos - $x * $sin);
    }
    return $box;
  }

  // Вычисляет размеры прямоугольника с горизонтальными и вертикальными
  // сторонами, в который вписан указанный текст. Результирующий массив
  // имеет структуру:
  // array(
  //   0  => ширина прямоугольника,
  //   1  => высота прямоугольника,
  //   2  => смещение начальной точки по X относительно левого верхнего
  //         угла прямоугольника,
  //   3  => смещение начальной точки по Y
  // )
  function imageTtfSize($size, $angle, $fontfile, $text) {
    // Вычисляем охкатывающий многоугольник.
    $box = imageTtfBBox_fixed($size, $angle, $fontfile, $text);
    $x = [$box[0], $box[2], $box[4], $box[6]];
    $y = [$box[1], $box[3], $box[5], $box[7]];
    // Вычисляем ширину, высоту и смещение начальной точки.
    $width  = max($x) - min($x);
    $height = max($y) - min($y);
    return array($width, $height, 0 - min($x), 0 - min($y));
  }

  // Функция возвращает наибольший размер шрифта, учитывая, что
  // текст $text обязательно должен поместиться в прямоугольник
  // размерами ($width, $height).
  function imageTtfGetMaxSize($angle, $fontfile, $text, $width, $height) {
    $min = 1;
    $max = $height;
    while (true) {
      // Рабочий размер - среднее между максимумом и минимумом.
      $size = round(($max + $min) / 2);
      $sz = imageTtfSize($size, $angle, $fontfile, $text);
      if ($sz[0] > $width || $sz[1] > $height) {
        // Будем уменьшать максимальную ширину до те пор, пока текст не
        // "перехлестнет" многоугольник.
        $max = $size;
      } else {
        // Наоборот, будем увеличивать минимальную, пока текст помещается.
        $min = $size;
      }
      // Минимум и максимум сошлись друг к другу.
      if (abs($max - $min) < 2) break;
    }
    return $min;
  }
```

Далее приведен пример сценария ttf.php, который использует возможности вывода TrueType-шрифтов, а также демонстрирует работу с цветом RGB. Для его работы необходим файл шрифта – в данном случае OpenSans.ttf. Он использует определённую выше библиотеку.

```php
<?php ## Пример работы с TTF-шрифтом.
  require_once 'lib/imagettf.php';
  // Выводимая строка.
  $string = 'Привет, мир!';
  // Шрифт.
  $font = getcwd().'/OpenSans.ttf'; // http://kodaktor.ru/OpenSans.ttf
  // Загружаем фоновой рисунок.
  $im = imageCreateFromPng('sample02.png'); // http://kodaktor.ru/sample02.png
  // Угол поворота зависит от текущего времени.
  $angle = (microtime(true) * 10) % 360;
  // Ксли хотите, чтобы текст шел из угла в угол, раскомментируйте строчку:
  # $angle = rad2deg(atan2(imageSY($im), imageSX($im)));
  // Подгоняем размер текста под размер изображения.
  $size = imageTtfGetMaxSize(
    $angle, $font, $string,
    imageSX($im), imageSY($im)
  );
  // Создаем в палитре новые цвета
  $shadow = imageColorAllocate($im, 0, 0, 0);
  $color  = imageColorAllocate($im, 128, 255, 0);
  // Вычисляем координаты вывода, чтобы текст оказался в центре.
  $sz = imageTtfSize($size, $angle, $font, $string);
  $x = (imageSX($im) - $sz[0]) / 2 + $sz[2];
  $y = (imageSY($im) - $sz[1]) / 2 + $sz[3];
  // Рисуем строку текста, вначале черным со сдвигом, а затем -
  // основным цветом поверх (чтобы создать эффект тени).
  imageTtfText($im, $size, $angle, $x + 3, $y + 2, $shadow, $font, $string);
  imageTtfText($im, $size, $angle, $x, $y, $color, $font, $string);
  // Сообщаем о том, что далее следует рисунок PNG.
  Header('Content-type: image/png');
  // Выводим рисунок
  imagePng($im);
```

Результат:



Сценарий генерирует изображение оттененной строки «Привет, мир!» на фоне JPEG-изображения, загруженного с диска. При этом используется TrueType-шрифт. Угол поворота строки зависит от текущего системного времени — попробуйте нажать несколько раз кнопку Обновить в браузере, и вы увидите, что строка будет все время поворачиваться против часовой стрелки. Кроме того, размер текста подбирается так, чтобы он занимал максимально возможную площадь, не выходя при этом за края картинки (см. определение функции imageTtfGetMaxSize()