Uniwersytet w Białymstoku

Wydział Matematyczno-Fizyczny

Instytut Matematyki

Beata Szadurska

Moduł dynamicznego tworzenia i obróbki grafiki w Meta-HTML oparty o bibliotekę GD

Praca dyplomowa napisana
pod kierunkiem
Jana Kowalskiego

Białystok 2005

Spis treści

Wstęp

Obecne czasy to prawdziwy wyścig technologiczny. Niemal codziennie zaskakują nas nowinki elektroniczne. Chętnie sięgamy po nowe rozwiązania, zaawansowane technologicznie, usprawniające i przyśpieszające komunikację, bez względu na czas i dzielącą odległość. Właściwie w każdej dziedzinie życia można mówić o większym czy mniejszym postępie, wywołanym przez wynalazki i stosowanie coraz bardziej wymyślnych technologii. Z pewnością można powiedzieć, że środkiem do prawie natychmiastowej, wzajemnej, praktycznie w każdym miejscu i czasie komunikacji oraz zapewniającym dostęp do niezbędnych i zawsze aktualnych informacji jest doceniany przez nas Internet.

Liczba osób korzystających z Internetu na całym świecie rośnie z roku na rok. W samej Polsce zgodnie z wynikami cyklicznych badań NetTrack przeprowadzanych przez instytut badawczy SMG/KRC w grudniu 2004 roku korzystanie z Internetu deklarowało 24,5% Polaków, co odpowiada liczbie 7,3 miliona osób. Na przestrzeni ostatniego roku liczba osób korzystających w Polsce z Internetu wzrosła, o 2,8 punktu procentowego (z poziomu 21,7% w grudniu 2003 roku), co w przeliczeniu na liczbę osób oznacza wzrost o prawie 850 tys. użytkowników.

Kiedy powstały pierwsze sieci między komputerami w Stanach Zjednoczonych, nikt nie przypuszczał, że rośnie ogromne medium, które stanie się wizytówką ludzkości, wchodzącej powoli w XXI wiek. Jeszcze kilka lat temu Internet był jedynie modą. Posiadanie modemu i własnej skrzynki pocztowej było dużym wydarzeniem. Niektórzy nie wiedzieli nawet o jego istnieniu. Sytuacja zmieniła się diametralnie.

Kiedyś Internet był narzędziem do przesyłania wiadomości i kopiowania plików między komputerami, teraz jest ogromną bazą informacji. Powstało tysiące witryn, które zawierają ogrom wiadomości. Do budowy stron wykorzystujemy wachlarz nowoczesnych technologii, co pozwala nam tworzyć strony o najwyższej złożoności oraz funkcjonalności, a także rozbudowanej szacie graficznej i efektami wizualnymi. Czasy szarych dokumentów HTML z nielicznymi obrazkami dawno minęły. Obecnie często chcemy aby strony WWW zawierały najbardziej jak to tylko możliwe aktualne dane pobierane z baz danych czy innych aplikacji. Mało tego, chcemy aby były one przedstawione w czytelnej i przejrzystej postaci. Nie chcemy statystyk w postaci długich i nudnych tabel, lecz w postaci kolorowych wykresów. Administratorzy systemów chcie-

liby graficznego przedstawienia topologii sieci, którą zarządzają, natomiast programiści woleliby odczytywać zależności między klasami w programowaniu obiektowym z graficznych schematów niż z treści kodu programu. Aby to osiągnąć sięgamy po narzędzia do dynamicznego generowania stron łącznie z automatycznym tworzeniem grafiki. Zastosowanie mają tutaj Meta-HTML i biblioteka GD. Połączenie możliwości tych dwóch narzędzi jest tematem niniejszej pracy.

Meta-HTML umożliwia nam przygotowywanie stron interaktywnych, nawiązujących niemal dialog z internautą, dostosowuje stronę internetową do potrzeb osoby ją przeglądającej. Wszystko zależy oczywiście od programisty, ale to właśnie bogaty język Meta-HTML dostarcza tak szerokich możliwości. Niestety w Meta-HTML narzędzia do manipulowania grafiką są przestarzałe. Z pomocą przychodzi nam biblioteka GD, którą można łaczyc z programami napisanymi w języku C, takimi jak Meta-HTML, do tworzenia grafiki. Biblioteka GD umożliwia tworzenie obrazków w formacie PNG, JPEG lub GIF, których zawartość jest zmienna. Jest to profesjonalna biblioteka do manipulowania bitmapowa grafika. Intencją tej biblioteki nie jest udostępnienie pełnego, bogatego zbioru funkcji umożliwiających wykonywanie skomplikowanych operacji na grafice. Powstała ona raczej z myślą o dostarczeniu podstawowego, prostego zestawu narzedzi. Bardziej szczegółowe rozwiązania, przydatne w niektórych projektach, mogą być za ich pomoca skonstruowane później przez samego programistę. W ten sposób otrzymujemy bibliotekę, która nie ukierunkowuje się i nie faworyzuje pewnych szczególnych zagadnień związanych z grafiką. Jest to niewątpliwie jej dużą zaletą, która powoduje także, iż poznawanie i korzystanie z libgd staje się bardziej przejrzyste i proste.

Zmodyfikowany przeze mnie Meta-HTML umożliwia zamieszczanie na stronach między innymi: generowanych automatycznie liczników odwiedzin, sond czy wykresów spółek akcyjnych. Moje starania docenią na pewno osoby tworzące serwisy internetowe w Meta-HTML, ponieważ otrzymają możliwość manipulowania obrazkami, a pośrednio także osoby korzystające z Internetu, w sumie kilka miliardów na całym świecie, bo strony staną się bardziej bogate i atrakcyjne.

Rozdział 1

Biblioteki systemowe

Wraz z postępującym rozwojem informatyki mamy do czynienia z powstawaniem coraz bardziej złożonych projektów informatycznych i programistczynych. Można śmiało powiedzieć, że powstające oprogramowanie staje się coraz większe i bardziej rozbudowane. Do pracy zatrudnia się zespoły programistów i korzysta się z coraz bardziej wyspecjalizowanych środowisk, które w znaczny sposób ułatwiają i skracają czas wytwarzania oprogramowania. Jedną z najważniejszych rzeczy było niewątpliwie powstanie języków wyższych poziomów, które ułatwiły projektowanie dużych systemów. Wprowadzając wyższy poziom abstrakcji (za pośrednictwem bardziej ogólnego zestawu funkcji, operatorów itd.) pozwoliły one na szybszą i bardziej zgrabną realizację dość złożonych problemów. W znaczny sposób sam mechanizm języka odciążał programistę od wielu zbędnych drobiazgów umożliwiając skupienie się tym samym na sprawach najistotniejszych.

W obecnej chwili, kiedy dysponujemy już potężnymi środowiskami oraz wygodnymi i efektywnymi językami programowania, powstaje pytanie, czy jeszcze w jakiś sposób możemy ułatwić i przyspieszyć naszą pracę. Oczywiście tak. Jedną z kluczowych metod jest stosowanie bibliotek. Pod terminem biblioteki rozumiemy tutaj zasób gotowych podprogramów, funkcji, które służą do realizacji pewnego spójnego zagadnienia. Możemy zatem traktować ją jako tzw. 'czarną skrzynkę' odpowiedzialną za pewne operacje. Wiemy w jaki sposób z niej korzystać (podobnie jak z kalkulatora), nie wiemy jednak w jaki sposób realizuje ona dane zagadnienia. Mówiąc o bibliotece mamy na uwadze dwie rzeczy, bibliotekę jako plik zapisany w pewnym formacie i posiadającym gotowe do wykorzystania funkcje oraz jego interfejs.

Zacznijmy od krótkiego omówienia formatów. Niemal każdy system operacyjny wprowadza specyficzne dla siebie formaty rozmaitych plików, w tym także plików wykonywalnych jak i bibliotecznych. Pomimo jednak różnic w sposobie upakowania i organizacji informacji wewnątrz nich możemy wyszczególnić dwie metodyki, wg których tworzy się pliki biblioteczne niezależnie od platformy komputerowej. Wiążą się one ze sposobem dołączania informacji zawartych w bibliotekach do programu, który z nich korzysta. Dołącznie to w

żargonie informatycznym określa się jako *linkowanie*.

Pierwszym sposobem jest linkowanie statyczne, czyli dołączanie informacji z biblioteki na stałe do programu wykonywalnego. Linker wbudowany w kompilator decyduje co dołączyć i w ten sposób zbędne funkcje, z których nie korzystamy są pomijane. Drugim sposobem jest linkowanie dynamiczne czyli dołączanie informacji z biblioteki w trakcie wykonywania programu. Program aktualnie wykonywany może sam załadować bibliotekę i zlokalizować potrzebną mu funkcję.

W związku z dwoma sposobami linkowania w systemach Unix i Linux mamy dwa rodzaje bibliotek: statyczne i dynamiczne.

1.1 Biblioteki dynamiczne

Biblioteki dynamiczne są specjalnymi wykonywalnymi modułami mogącymi zawierać kod i (lub) dane, wykorzystywane przez różne aplikacje w czasie ich działania . Koncepcja tego typu modułów wzięła się stąd, że pewne operacje są charakterystyczne i jednakowe dla różnych aplikacji. Dlatego aby nie dublować w pamięci kodu i danych, przyjęto rozwiązanie, w którym z tego samego kodu i danych mogą korzystać równocześnie różne aplikacje. Znacznie zwiększa to efektywność gospodarowania pamięcią. Stosowanie bibliotek dynamicznych jest jednym ze sposobów oszczędzania zasobów komputera, zarówno jeśli chodzi o pamięć masową jak i operacyjną. Jest też sposobem na podniesienie wydajności systemu.

W przypadku systemu Unix mówimy o DSO (Dynamic Shared Object), w Windows o DLL (Dynamicaly Linked Library). Mimo różnic w terminologii i wewnętrznej implementacji zasada pozostaje taka sama w obu przypadkach. Część kodu wspólnego dla różnych programów umieszczona jest poza programem, w bibliotece. Biblioteki służą do grupowania funkcji, a także do dzielenia dużego programu na powiązane logicznie części. Biblioteka nie stanowi samoistnego programu a jej użycie w programie wymaga deklaracji. Po zadeklarowaniu modułu (biblioteki) w danym programie dostępna jest każda funkcja zdefiniowana w danym module, jak również zadeklarowane w nim stałe, typy i zmienne. W przeciwieństwie do biblioteki statycznej nie jest dołączana do pliku wykonywalnego programu, zachowuje się ona jak moduł wykonywalny: gdy jakakolwiek z jej funkcji jest wywoływana przez program, biblioteka dynamiczna jest ładowana do pamięci, a następnie jej funkcje mogą być wywoływane w tym samym czasie przez inne aplikacje. Oznacza to, że pojedyncza kopia biblioteki może być wykorzystywana wspólnie (dzielona) przez kilka aplikacji. W zasadzie możliwe jest zaktualizowanie biblioteki bez rekompilowania programów aplikacyjnych. W systemie windows biblioteka dołączana dynamicznie ma rozszerzenie dll (ang. Dynamic Link Library - DLL).

Cechy pozytywne biblioteki dynamicznej:

- Dzięki współdzieleniu kodu przez wszystkie procesy korzystające z bi-

blioteki dzielonej mniejsze jest wymaganie co do przestrzeni dyskowej potrzebnej do przechowywania aplikacji.

- Biblioteki dzielone oszczędzają pamięć systemu.
- Kod biblioteki dzielonej nie jest kopiowany do pliku wykonywalnego, a zatem tylko jedna kopia znajduje się na dysku. Gdy biblioteka jest nam potrzebna, ładujemy ją, a gdy przestaje - zwalniamy ją po prostu z pamięci.
- Jeżeli w bibliotece dzielonej znajduje się błąd to można ją zastąpić poprawna wersją bez potrzeby kompilowania wszystkich programów które z niej korzystają.
- Biblioteki mogą być ładowane na żądanie programu w trakcie jego działania. Oznacza to, że z poziomu uruchomionej aplikacji możemy ładować i zwalniać dodatkowy kod, co pozwala dynamicznie modyfikować funkcjonalność aplikacji, bez potrzeby jej rekompilacji.

Każda biblioteka dzielona posiada specjalną nazwę tzw. "nazwę-so". Nazwaso składa się z przedrostka `"lib"`, nazwy biblioteki, wyrażenia `".so"`, po którym następuje kropka oraz numer wersji(np.: libncurses.so.5).

Biblioteki dynamiczne muszą zostać umieszczone w określonym miejscu w systemie plików. Zgodnie ze standardem BSD domyślnie wszelkie biblioteki są instalowane w katalogu `/usr/lib` (a wszystkich poleceń w /usr/bin). Biblioteki prywatne należy umieszczać poza systemowymi katalogami `/lib` i `/usr/lib`. Pamiętać jednak wtedy należy o odpowiednim skompilowaniu programu tak, aby odnalazł niezbędne biblioteki.

Podczas uruchamiania programu który korzysta z bibliotek dzielonych są przeszukiwane są standardowo następujące katalogi: /lib oraz /usr/lib.

Jeśli chcielibyśmy zastąpić kilka funkcji w jednej z bibliotek, ale zachować resztę tej biblioteki, można wprowadzić nazwy zastępujących bibliotek (plików /etc/ld.so.preload; biblioteki te będą miały pierwszeństwo przed zestawem bibliotek standardowych. Plik ten zwykle jest używany w nagłych sytuacjach, do doraźnego załatania jakiejś niesprawności; normalnie dystrybucje nie zawierają tego pliku.

W związku z tym, że przeszukiwanie wszystkich podanych katalogów w momencie startu programu byłoby bardzo nieefektywne, wykorzystywany jest mechanizm cache'owania. Program tworzy odpowiednie dowiązania symboliczne w katalogach (zgodnie ze stosowanymi konwencjami) i zapisuje cache do pliku, który z kolei jest wykorzystywany przez inne programy. Przyspiesza to w znacznym stopniu dostęp do bibliotek.

1.2 Biblioteki Statyczne

Biblioteki statyczne (Library Archive) dołączane są do kodu programu w czasie linkowania, dzięki czemu plik wynikowy jest większy o dołączony kod. Biblioteki te są zbiorem plików obiektowych (rozszerzenie `*.o`). Biblioteka statyczna utworzona przez program `ar` ma rozszerzenie `*.a` i jest to archiwum plików obiektowych zawierające nagłówek, w którym znajdują się informacje m.in. o nazwach i położeniu wewnątrz archiwum poszczególnych obiektów oraz tablice symboli biblioteki. Program `ar` ma bardzo podobną składnią do programu `tar`: ar [opcje] nazwa_archiwum plik$_1$ ... plik$_n$

Argument archiwum jest nazwą biblioteki, a plik$_1$... plik$_n$ są plikami obiektowymi, z których należy stworzyć bibliotekę lub którą należy wyekstrahować lub usunąć z biblioteki.

Najczęściej używane opcje to:

- -d usuwanie z archiwum wskazanego pliku,
- -q dodawanie pliku na koniec archiwum (nie sprawdza czy dany plik jest już w archiwum),
- -r zamiana (lub dodanie) a (lub dodanie) w archiwum; jeśli pliku nie ma w archiwum to następuje jego dodanie, a w przypadku gdy nie istnieje archiwum - jego utworzenie,
- -u używana razem z -r powoduje, że zamiana następuje tylko wtedy, gdy data modyfikacji pliku w archiwum jest wcześniejsza niż data modyfikacji pliku podanego jako argumentu,
- -s ponowne utworzenie tablicy symboli biblioteki; umożliwia odtworzenie tablicy symboli po jej usunięciu programem strip,
- -t wypisanie zawartości archiwum; zwykle używana razem z opcją -v,
- -x ekstrakcja wskazanego lub wszystkich plików z archiwum (nie niszczy archiwum),
- -v wyświetlanie bardziej szczegółowych informacji podczas działania.

Przykładowo, jeśli programista chce utworzyć bibliotekę `mylib.a` z następujących plików: funkcja1.o, funkcja2.o i funkcja3.o, to polecenie może mieć postać: `ar -rv mylib.a funkcja1.o funkcja1.o funkcja1.o` lub `ar -ruv mylib.a funkcja1.o funkcja1.o funkcja1.o`

Mając już archiwum możemy dołączyć je do kodu programu. Dokonujemy tego za pomocą opcji `-lnazwaArchiwum` w kompilatorach gcc oraz g++, opcja -L powoduje wyszukiwanie biblioteki najpierw w zadanym katalogu, a następnie w katalogach domyślnych. Przypuśćmy, że w pliku app.c korzystamy z zawartej w bibliotece `libbibl.a` funkcji funkcjabibl(). Wówczas należy wywołać polecenie :`$ gcc -o app app.o -L. -lbibl`

Rozdział 2

Tworzenie i obróbka grafiki

2.1 Biblioteka GD

Biblioteka GD napisana przez Thomasa Boutella jest biblioteką C, którą można łączyć z programami w języku C do tworzenia grafiki. Jeśli mamy zamiar tworzyć obrazki w formacie PNG, JPEG lub WBMP których zawartość jest zmienna tzn. może różnić się pomiędzy kolejnymi wywołaniami, powinniśmy użyć biblioteki GD. Jest to profesjonalna biblioteka do manipulowania bitmapową grafiką.

Posiada ona przede wszystkim takie funkcje jak rysowanie linii, figur geometrycznych, definiowanie wypełnień oraz wczytywanie istniejących obrazków i manipulowanie nimi. GD jest doskonałym narzędziem do tworzenia graficznych liczników, sond z cieniowanymi paskami, wykresów odwiedzalności serwisu WWW. Rezultat naszych prac możemy później zapisać w pliku graficznym.

Intencją tej biblioteki nie jest udostępnienie pełnego bogatego zbioru funkcji umożliwiających rysowanie skomplikowanych prymitywów. Powstała ona z myślą o dostarczeniu podstawowego, prostego zestawu narzędzi. Bardziej szczegółowe rozwiązania przydatne w niektórych projektach mogą być za ich pomocą skonstruowane później przez samego programistę. W ten sposób otrzymujemy bibliotekę, która nie ukierunkowuje się i nie faworyzuje pewnych szczególnych zagadnień związanych z grafiką. Jest to niewątpliwie jej dużą zaletą, która powoduje także iż poznawanie i korzystanie z `libgd` staje się bardziej przejrzyste i proste.

Kolejną istotną rzeczą związaną z biblioteką GD jest fakt, iż powstała ona na różne platformy i nie jest zadedykowana konkretnemu systemowi operacyjnemu. Oznacza to w praktyce, iż pisząc program z jej wykorzystaniem oraz korzystając z języka dostępnego na różnych platformach możemy nasze programy kompilować bez zmian w różnych środowiskach i powinny one działać identycznie. Co prawda format plików wykonywalnych oraz bibliotek będzie się różnić lecz nasz kod wciąż pozostanie taki sam. W naszych programach nie zajmujemy się bowiem specyficznymi rzeczami zależnymi od systemu opera-

cyjnego, są one dla nas niewidoczne, oczekujemy jedynie pewnych zachowań, których realizacja spoczywa już na barkach projektantów tych systemów. Niezależność od platformy uzyskano wprowadzając wystarczająco wysoki poziom abstrakcji w opisie danych i funkcji w interfejsie.

Biblioteka GD została pierwotnie napisana w języku ANSI C i udostępniona w postaci plików źródłowych oraz skompilowanych bibliotek na różne platformy. Można z niej jednak korzystać i w innych językach (Perl, Tcl, Pascal, REXX)

Do zainstalowania GD konieczna jest biblioteka `zlib` oraz `libpng`. Biblioteka `jpeg-6b` jest wymagana tylko do generowania obrazków JPEG. Ze strony domowej GD można łatwo ściągnąć odpowiednie biblioteki i zainstalować je wg znanego schematu:

1. cd - przewodnik zawierający szyfr źródła pakunku
2. ./configure -konfiguracja pakietu dla odpowiedniego systemu
3. make -kompilacja pakietu
4. make install
5. w razie potrzeby można własnoręcznie wprowadzić zmiany i poprawki w wygenerowanym pliku `Makefile` przed kompilacją i instalacją, w razie ewentualnych błędów można także wpisać :

   ./configure -help

   aby poznać dostępne opcje i ewentualnie z nich skorzystać.

Schemat ten może się oczywiście różnić nieco dla odpowiedniej biblioteki (np. zamiast kroku ./configure będzie należało skopiować odpowiedni plik Makefile).

Za pomocą funkcji graficznych można pobrać rozmiar obrazków w formatach: JPEG, GIF i PNG. Jeżeli jest zainstalowana biblioteka GD to można również tworzyć i manipulować obrazkami. Format obrazków, którymi można manipulować, zależy od wersji zainstalowanej biblioteki GD jak i innych połączonych bibliotek. Biblioteki GD w wersji starszej od 1.6 obsługują obrazki w formacie GIF, a nie obsługują formatu PNG. Natomiast nowsze wersje tej biblioteki obsługują format PNG, a nie obsługują formatu GIF.

Wszystkie linie kodu rozpoczynające się od GD są wywołaniami funkcji z bibliotek GD.

Libgd została napisana bardzo elastycznie, umożliwiając zwykłemu programiście pewne manipulacje na kodzie. Dzięki wielu dyrektywom preprocesora #if oraz #ifdef można definiować pewne stałe symboliczne na początku kodu biblioteki w celu aktywowania pewnych cech. Żeby być bardziej precyzyjnym, należałoby powiedzieć, że można w ten sposób decydować, które z części kodu mają być uwzględnione podczas kompilacji biblioteki. W ten sposób możemy

wskazać np. że nie interesuje nas obsługa formatu graficznego JPEG bądź PNG, wówczas wymagania co do libjpeg czy libpng są już nieistotne. Innymi podobnymi stałymi są np. takie, które powodują dołączenie dodatkowych informacji przydatnych podczas debug'owania. Mogą one nieco spowolnić działanie biblioteki. Odpowiednie zbiory funkcji zostały pogrupowane tematycznie i umieszczone w osobnych plikach źródłowych (*.c). Każdy z nich posiada swój interfejs w postaci pliku nagłówkowego (*.h), który udostępnia niektóre funkcje, stałe, typy i zmienne innym modułom. Najistotniejsza część biblioteki znajduje się w pliku gd.c, który dodatkowo jest największy objętościowo. Zgromadzono tam wszystkie funkcje odpowiedzialne za tworzenie i niszczenie obrazu, manipulacje kolorami, rysowanie (zarówno figur jak i tekstu), pobieranie i ustawianie danych obrazu. W programowaniu grafiki komputerowej mamy ogromną ilość algorytmów stosowanych do różnych zagadnień. Samo rysowanie linii pomiędzy dwoma zadanymi punktami jest znacznie trudniejsze niż się to na początku może wydawać, tym bardziej jeśli zależy nam na bardzo efektywnej, szybkiej metodzie. Prędkość wykonywanych operacji nie była jednak istotnym założeniem przy tworzeniu libgd.

2.2 Zastosowanie

Pomysł operowania na obrazie w pamięci i zapisywanie go do pliku graficznego stwarza ciekawe możliwości. Twórcy stron internetowych, którzy korzystają przykładowo z PHP mogą tworzyć witryny, na których podajemy jakieś specyficzne dane (np. długość boku, kolor wypełnienia oraz pozycję) wykorzystywane dalej do generowania rysunku. Można przykładowo generować jakiś efekt (np. fraktal) pytając o liczbę iteracji, kształt, kąty obrotu itd. Efekt naszych działań można łatwo zobaczyć gdyż formaty wykorzystywane przez gd (JPEG, PNG) są zwykle obsługiwane przez znane przeglądarki.

Podobnie przy tworzeniu różnych sond czy ankiet internetowych możemy wygenerować łatwo rysunek ilustrujący otrzymane dotychczas wyniki. Wykres może mieć zarówno postać słupkową, kołową, punktową bądź funkcyjną. Użyć możemy także różnych kolorów, podpisując odpowiednie części.

Wyobraźmy sobie teraz duży rysunek ilustrujący mapę miasta. Użytkownik może wykonywać zbliżenia zaznaczając pewne prostokątne obszary na jej powierzchni. Można takie zagadnienie zrealizować w oparciu o funkcje rozciągające, ale efekt może być nieciekawy (duże piksele). Dlatego można obraz podzielić na wiele obszarów i trzymać w plikach każdą taką niewidoczną na rysunku część w dużo lepszej jakości i rozdzielczości. W momencie kiedy zostanie wydane polecenie powiększenia danego obszaru, wyliczamy, które części danych składowych należy użyć i korzystamy z mniejszych rysunków zapisanych w bardzo dobrej jakości.

Fakt, iż zapisanie obrazu do formatu JPEG czy PNG jest bardzo proste, sprawia iż napisanie konwertera jest równie łatwe. Można więc tworzyć pro-

gramy wykonujące bardziej skomplikowane operacje. Przykładowo parametrami programu mogą być dwa różne rysunki oraz liczba z przedziału 0 do 127. Określa ona w jakiej części pierwszy rysunek powinien przenikać drugi. Można wprowadzać także dodatkowe dane, np. obrót pierwszego rysunku, skalowanie, przyciemnienie itd. Kolejną rzeczą jest obsługa tekstu. Wystarczy, że w naszym programie zaimplementujemy wyświetlanie rysunków w formacie PNG. Można wówczas prezentować tekst i udostępnić użytkownikowi takie opcje jak zmiana fontu, jego wielkości itd.

Niektóre typy gier, które nie wymagają szybkiego tworzenia obrazu mogą również skorzystać z libgd. Wyobraźmy sobie grę RPG i widok planszy prezentującej sylwetkę bohatera, jego ubiór oraz przedmioty. No cóż, od razu nasuwa się pomysł przechowywania rysunków wszystkich możliwych rzeczy w oddzielnych plikach. Zależnie teraz od opisu naszej postaci decydujemy co takiego ona posiada i wyświetlamy odpowiednie komponenty. Można tutaj użyć alpha blending'u, tak że niektóre ubrania mogą się w paru miejscach przenikać (tworząc złudzenie obrazu 3D). Ktoś może jednak powiedzieć, że przy wielu takich rysunkach generowanie efektu końcowego może zająć sporo czasu. To prawda, ale głowa programisty w tym, żeby takiej generacji dokonać podczas wczytywania gry, niektóre części obrazu, tworzącego końcową postać, powtarzają się przecież.

2.3 Moduł `image` a biblioteka GD

Wiele programów, w tym także Meta-HTML, ma możliwość dołączania nowych funkcjonalności lub rozwijania istniejących bez potrzeby rekompilacji całego programu. Możliwe jest to dzięki ich modularnej budowie. Modularna budowa polega na tym, że jako integralne fragmenty programu używane są biblioteki dynamiczne. Była o tym mowa wcześniej w pracy.

Sam program Meta-HTML jest modułem dla serwera Apache, czyli formalnie jest biblioteką dynamiczną, ładowaną przez program Apache, na żądanie, przy odpowiedniej jego konfiguracji. Wiele funkcji Meta-HTML, takich jak właśnie tworzenie i obróbka obrazów, komunikacja z bazą danych, czy zaawansowane funkcje matematyczne wydzielone są jako moduły (te konkretnie tutaj wymienione moduły mają nazwy odpowiednio: `modimage.so`, `modmysql.so` i `modmath.so`), czyli tak naprawdę biblioteki dynamiczne. Tworzą się w ten sposób pewne łańcuchy zależności pomiędzy różnymi bibliotekami. W przypadku modułu `image` mamy następujące dynamiczne linkowanie:

Apache		Meta-HTML		moduł `image`		bibiloteka GD
	$\longrightarrow$		$\longrightarrow$		$\longrightarrow$	
`httpd`		`modmhtml.so`		`modimage.so`		`libgd.so`

Tak więc formalnie, z punktu widzenia systemu operacyjnego, moduł i biblioteka jest tym samym. Tak jak to zostało wcześniej opisane w 1.1 biblioteka

jest zestawem uniwersalnych funkcji, gotowych do wykorzystania w różnych programach lub innych bibliotekach. W odróżnieniu od typowej biblioteki jaką jest np. biblioteka GD, moduł jest ściśle powiązany z konkretną aplikacją. Zazwyczaj musi spełniać konkretne wymagania narzucane przez tę aplikację, innymi słowy musi być napisany zgodnie z API[1] tej aplikacji, komunikować się z nią w ustalony sposób. Tak więc, moduł nie jest tak uniwersalny jak typowa biblioteka, przeciwnie realizuje specyficzne zadania aplikacji.

Narzucające się w tym miejscu pytanie brzmi: po co wyodrębniać moduły z aplikacji? Nie robi się tego na pewno aby zaoszczędzić miejsce na dysku czy przyspieszyć wykonywanie aplikacji. Moduły pozwalają użytkownikom aplikacji na rozbudowę jej możliwości bez konieczności rekompilacji aplikacji. Ważne jest to w przypadku programów, których działania nie chcemy przerywać, lub programów komercyjnych, do których nie posiadamy kodu źródłowego.

Moduł `image` stanowi interfejs, czyli pewien łącznik, pomiędzy Meta-HTML a biblioteką GD. Meta-HTML ma dobrze określony, bardzo specyficzny sposób przekazywania argumentów, związany ze składnią języka jaki realizuje. Nie jest możliwe aby bezpośrednio z poziomu skryptu napisanego w Meta-HTML można było wołać funkcje jakiejkolwiek biblioteki zewnętrznej, wcześniej odwołania te muszą zostać "przetłumaczone". Zaimplementowane w module `image` funkcje możemy zatem traktować jako pomosty pomiędzy funkcjami GD, a konstrukcjami językowymi w Meta-HTML.

Podobna sytuacja jak w Meta-HTML występuje w wielu innych tego typu programach, np. w PHP czy Perlu.

2.4 Formaty zapisu i reprezentacja grafiki

Dostępnych jest bardzo wiele różnych sposobów na przechowywanie grafiki w komputerze od prostej bitmapy bez kompresji po zaawansowane techniki oszukujące nasz zmysł wzroku tak, aby jak najmniej zniekształcić obraz i jednocześnie mieć plik o niewielkich rozmiarach. Najbardziej popularne, chyba za sprawą Internetu, są formaty GIF i JPEG, a ostatnio dołącza do nich PNG. Pojawienie się formatu PNG było odpowiedzią środowiska Open Source na wątpliwości co do legalności GIFa. Problem polegał na tym, że w formacie GIF wykorzystuje się algorytm kompresji LZW, do którego firma Unisys rości prawa autorskie (por. [4, 5]).

W pierwszych wersjach biblioteki GD dostępny był format GIF. W momencie gdy używanie tego formatu w darmowych projektach stało się kontrowersyjne, GIF zniknął z GD. Kiedy patenty na LZW wygasły, 21 czerwca 2004, format GIF przywrócono w bibliotece GD w wersji 2.0.28.

Formaty GIF i PNG używają bezstratnej kompresji obrazu. Doskonale nadają się do zapisu grafiki komputerowej takiej jak liternictwo, wykresy itp.

[1] Application Programming Interface — czyli programowy interfejs aplikacji.

Format JPEG natomiast, służy do zapisu innego rodzaju obrazów takich jak np. zdjęcia, gdzie niewielkie zniekształcenie nie spowoduje widocznej utraty jakości i wyrazistości obrazu. Stosowana kompresja w JPEG powoduje, że część informacji z obrazu jest stracona.

JPEG (Joint Photographic Expert Group) powszechnie wykorzystywany do wyświetlania zdjęć kolorowych i w odcieniach szarości oraz innych obrazków z płynnymi przejściami tonów w dokumentach HTML (Hyper-Text Markup Language) w sieci WWW i innych serwisach online. JPEG wykorzystuje algorytm kompresji, który identyfikując i pomijając dane nieistotne dla poprawnego wyświetlenia obrazka zmniejsza rozmiar pliku. Otwarcie obrazka JPEG powoduje automatyczne rozpakowanie go. Ponieważ pewne dane są pomijane, algorytm JPEG jest zwany stratnym. Oznacza to, że obrazek raz skompresowany nie będzie po rozpakowaniu identyczny z oryginałem. Użyteczną własnością formatu JPEG jest możliwość uzyskiwania różnych wielkości strat obrazu w wyniku zmiany parametrów kompresji. Większy stopień kompresji powoduje obniżenie jakości obrazka, natomiast mniejszy ma lepszą jakość. W większości przypadków obrazek po skompresowaniu w najlepszej jakości będzie nie do odróżnienia od oryginału. Zachowując plik w formacie JPEG, podaje się pożądaną jakość i stopień kompresji obrazka. Istnieje zależność między jakością obrazka a stopniem kompresji; obrazek o jakości maksymalnej będzie słabiej skompresowany (czyli zajmie więcej miejsca na dysku) niż obrazek o niskiej jakości. Ponadto można zapisać obrazek w formacie progresywnym podając pożądaną liczbę przybliżeń. Ten format umożliwia wyświetlanie obrazka sprowadzanego z sieci za pomocą przeglądarki WWW zdefiniowanych przybliżeń, z których każde następne wnosi coraz więcej szczegółów - aż do sprowadzenia całości. Jednakże format progresywny JPEG wymaga więcej pamięci RAM do oglądania i nie jest obsługiwany przez wszystkie sieciowe przeglądarki. Format JPEG może być kodowany w dwóch odmianach: CMYK lub RGB. Te obrazy, które mają być umieszczone na sieci powinny zawsze być kodowane w formacie RGB lub odcieniach szarości.

Porównując JPEG z GIFem, ten pierwszy w kompresji jest znacznie lepszy Potrafi bowiem dokonać kompresji w stosunku 20:1 do kolorowego oryginału. Ponadto w porównaniu do GIFa kompresuje w wielu przypadkach 4 razy lepiej. Kompresowane kolory mogą osiągać wartość do 24-bitów na jeden piksel. JPEG nie jest jednak w stanie zastąpić GIFa we wszystkich przypadkach. Dla niektórych typów obrazów GIF osiąga dużo lepsze rezultaty pod względem jakości oraz objętości plików. JPEG nadaje się bardziej do obrazów kolorowych i w odcieniach szarości o dużej liczbie kolorów, czyli np. do skanowanych fotografii. Wszystkie płynne przejścia w kolorach takie jak rozjaśnienia czy cienie będą lepiej reprezentowane w formacie JPEG. Format GIF natomiast jest znacznie lepszy w kompresji tekstu, obrazów w kilku kolorach, wzorów i prostych rysunków posiadających ostre różnice w kształtach. Ponadto można go z powodzeniem stosować do kompresji obrazków, które są małe i nie są zdjęciami. Kompresuje je lepiej pomimo, że jest to kompresja bezstratna. Im

bardziej obrazy są skomplikowane tym lepiej zostaną przedstawione w formacie JPEG. GIF dobrze kompresuje także obrazy w odcieniach szarości do 256 kolorów. Biało-czarne obrazy nigdy nie powinny być kompresowane w formacie JPEG.

W formatach GIF, PNG i JPEG mamy do czynienia z dwoma sposobami kodowania koloru. Potocznie mówi się, choć formalnie niepoprawnie, że są używane dwa rodzaje *palety* kolorów: 8-bitowa i 24-bitowa. Tak naprawdę albo jest paleta 8-bitowa, albo jej nie ma.

Każdy kolor, tak jak wszystko zresztą, w komputerze reprezentowany jest jako liczba. Jest kilka standardów kodowania koloru. Najbardziej powszechny to RGB. Każdy kolor jest tutaj rozkładany na trzy składowe: czerwoną R, zieloną G i niebieską B. Każdej ze składowych przyporządkowuje się jeden bajt, czyli liczbę z przedziału od 0 do 255 włącznie. Tak więc kolor w standardzie RGB to liczba z przedziału od 0 do $2^8 2^8 2^8 = 2^{24} = 16.777.216$.

Co to jest paleta i co daje jej użycie? W przypadku grafiki komputerowej jest często tak, że stosunkowo duże obszary obrazu wypełnia się jednym kolorem. Gdyby każdemu pikslowi w obrazie 100x100 piksli przyporządkować kolor RGB to zapisany plik miałby rozmiar 30000 bajtów. Gdy jednak w tym małym obrazku używanych jest tylko powiedzmy 200 różnych kolorów to można zredukować znacznie rozmiar pliku stosując paletę 8-bitową.

Obraz można utożsamiać z pewną macierzą prostokątną. W przypadku palety 8-bitowej wartościami tej macierzy są indeksy z palety. Paleta powstaje w ten sposób, że spośród dostępnych 2^{24} kolorów wybiera się maksymalnie 256 i numeruje się je od 0 do 255. W ten sposób otrzymujemy tablicę z dwoma kolumnami, w pierwszej są kolejne liczby – indeksy kolorów, w drugiej wartości RGB tych kolorów.

Wracając do naszego przykładu, aby zapisać nasz obrazek o rozmiarach 100x100 piksli z paletą 8-bitą¸ wystarczy 10000 bajtów na sam obraz plus od 3 do 256*3 bajtów na paletę. Tak, więc otrzymujemy prawie trzykrotnie mniejszy plik.

W przypadku palety 24-bitowej wartościami macierzy odpowiadającej obrazowi są wartości kolorów RGB. Nie ma zatem tutaj żadnej pośredniej tabeli z kolorami – palety. Mówiąc paleta, jak widać, mamy raczej na myśli ilość informacji w bitach przypadającą na pojedynczy piksel w obrazie.

Format GIF zawsze używa palety 8-bitowej, PNG może używać palety 8-bitowej lub 24-bitowej, natomiast JPEG zawsze używa palety 24-bitowej do zapisu obrazów kolorowych i 8-bitowej do zapisu obrazów w odcieniach szarości.

Rozdział 3

Modyfikacje i uruchamianie Meta-HTML

3.1 Test modułu `image`

Zanim przystąpiliśmy do kompilacji przetestowaliśmy dokładnie moduł `image` w Meta-HTML. W tym celu napisaliśmy odpowiedni skrypt, który sprawdza wszystkie z dostępnych funkcji w tym module. Poniżej przedstawiamy ten skrypt wraz z opisem poszczególnych makr.

```
<image::create img width=600 height=400>
```

Za pomocą funkcji `image::create` został utworzony obrazek o rozmiarach 600 (szerokość) na 400 (wysokość) piksli, jako pewna struktura zapamiętana w zmiennej `img`. Można będzie na nim rysować linie, łuki, pojedyncze punkty oraz wstawiać tekst.

```
<image::fill img x=0 y=0 color=#aa0000>
```

Funkcja wypełnia kolorem obszar obrazu `img`. Wypełnianie zaczyna się w punkcie o współrzędnych (X,Y), liczonych od górnego lewego rogu obrazu, i kontynuowane jest we wszystkich kierunkach (piksel po pikselu), aż do wystąpienia piksela o innym kolorze jak ten w punkcie (X,Y).

```
<image::get-pixel img X=50 Y=70>
```

Ta funkcja zwraca kolor piksela z obrazu `img` w punkcie o zadanych współrzędnych (X, Y).

```
<image::arc img x=200 y=200 width=200 height=300
                                start=0 end=360 color=#ff0000>
```

Rysuje częściową elipsę o środku w punkcie o współrzędnych (X, Y), o szerokości WIDTH i wysokości HEIGHT. Argumenty START i END są dane w stopniach i określają początkowy i końcowy punkt na krzywej.

```
<image::line img X1=300 Y1=220 X2=90 Y2=190 color=#aabbcc>
```

Rysuje odcinek linii prostej od punktu (X1, Y1) do (X2, Y2) w kolorze COLOR na obrazie `img`. Można opcjonalnie podać parametr BRUSH, który jest obrazem utworzonym przez `image::create`, wówczas ten obraz będzie wykorzystany jako wzór, by narysować linię. W tym przypadku, argument COLOR jest ignorowany.

```
<image::rect img X1=50 Y1=70 X2=100 Y2=30
                                    color=#aaffaa fill=#66ffaa>
```

W obrazie `img` rysowany jest prostokąt o obramowaniu w kolorze COLOR i wypełnionym kolorem podanym jako parametr FILL. Prostokąt rysowany jest od lewego górnego wierzchołka o współrzędnych (X1,Y1) do prawego, dolnego wierzchołka (X2,Y2).

```
<image::set-pixel img X=90 Y=250 color=#ddccaa>
```

Na obrazie `img` rysujemy piksel o współrzędnych (X,Y) o kolorze COLOR.

```
<image::text img obrazek X=10 Y=20 color=#77ddee
                                                size=5 align=left>
```

Wypisuje tekst "obrazek" na obrazie `img` od pozycji (X,Y) w kolorze COLOR. Parametr ALIGN może mieć jedną z następujących wartości RIGHT, CENTER, albo LEFT i jest to wyrównanie tekstu względem punktu referencyjnego (X,Y). Domyślnie, wyrównanie tekstu jest do środka, czyli CENTER. SIZE jest to rozmiar (wysokość) tekstu i przyjmuje wartość od 1 do 6. Pominięcie tego parametru powoduje, że domyślny rozmiar wypisywanego tekstu będzie 3.

```
<image::info img>
```

Ta funkcja zwraca informację o obrazie `img`. Informacja zawiera szerokość, wysokość, użyte kolory i ich łączną liczbę.

```
<image::transparent img color=<image::get-pixel img X=55 Y=75>>
```

Kolor podany w argumencie COLOR będzie przezroczysty dla obrazu `img`.

```
<image::create img2 width=600 height=400>
<image::copy img img2
         SRC-X=10 SRC-Y=15 SRC-width=400 SRC-height=300
         DST-X=100 DST-Y=50 DST-width=200 DST-height=150>
```

Aby przetestować funkcję kopiowania tworzymy nowy obraz w zmiennej `img2`. Następnie kopiujemy bajty z `img` do `img2`. Oba obrazki muszą być wcześniej zainicjowane. Parametry z przedrostkiem SRC określają fragment obrazu `img` do skopiowanie, natomiast parametry z przedrostkiem DST określają gdzie kopiowany fragment umieścić na `img2`. Jeśli szerokość lub wysokość

w obrazku docelowym jest inna od szerokości lub wysokości fragmentu oryginalnego, obraz jest przeskalowany tak, by pasował do podanych wartości.

```
<set-var wynik=<image::write img /tmp/obrazek.gif>
<set-var wynik=<image::write img2 /tmp/obrazek2.gif>
```

Powyższe polecenia zapisują obazki `img` i `img2` w plikach odpowiednio `/tmp/obrazek.gif` i verb!/tmp/obrazek2.gif! na dysku. Zmienna `wynik` przyjmuje wartość TRUE gdy, operacja zapisu powiedzie się, w przeciwnym razie zmienna jest pusta.

```
<image::delete img>
<image::delete img2>
```

Kiedy obrazki już nie są potrzebne, aby zwolnić miejsce zajmowane przez zmienne `img`, `img2` wołane jest ta funkcja.

Efektem działania naszego skryptu jest utworzenie dwóch plików graficznych: `/tmp/obrazek.gif` i `/tmp/obrazek2.gif`.

3.2 Kompilacja Meta-HTML z biblioteką GD

Moduł `image` oparty jest o bibliotekę GD, której kod źródłowy jest dołączony do kodu źródłowego Meta-HTML. Jest to stara wersja biblioteki GD – wersja 1.3. Aby skompilować moduł `image` z najnowszą wersją biblioteki GD – wersją 2.033 – należało najpierw skompilować i zainstalować tę wersję GD w systemie. Następnym krokiem było dostosowanie plików `Makefile` w Meta-HTML. Zmiany w tych plikach polegały na tym, że odwołania do biblioteki statycznej `libgd.a` kompilowanej razem z Meta-HTML:

```
modimage$(SHARED_EXT): modimage.c libgd/libgd.a
    @echo Building module $@ from $<
    @$(CC) $(GCC_FPIC) -c -o modimage.o $(CFLAGS) $(IFLAGS) \
       -I./libgd $(DEFS) $(VERSDEF) $(INCLUDE_FLAGS) modimage.c
    @$(SHARED_LD) $(LDFLAGS) modimage.o $(DASH_SHARED) \
       -o $@ -L./libgd -lgd
```

zastąpiono odwołaniami do biblioteki dynamicznej `libgd.so` w systemie:

```
modimage$(SHARED_EXT): modimage.c
    @echo Building module $@ from $<
    $(CC) $(GCC_FPIC) -c -o modimage.o $(CFLAGS) $(IFLAGS) \
      -I./libgd $(DEFS) $(VERSDEF) $(INCLUDE_FLAGS) modimage.c
    $(SHARED_LD) $(LDFLAGS) modimage.o $(DASH_SHARED) \
      -o $@ -L/opt/cfw/lib -R/opt/cfw/lib -lgd
```

Jak widać modyfikacja polega na zmianie ścieżki w opcji `-L` i dodaniu opcji `-R`, tak aby podana ścieżka była przeszukiwana w trakcie wykonywania programu (ang. runtime, stąd R).

Po wykonaniu powyższych zmian, przy próbie kompilacji Meta-HTML występowały błędy spowodowane tym, że w implementacji modułu `image` odwoływano się bezpośrednio do części implementacyjnej biblioteki wbrew zasadom korzystania ze współdzielonych bibliotek. Te odwołania występowały w funkcji `pf_image_arc()` odpowiedzialną za kreślenie łuków. W starej wersji biblioteki GD nie było dostępne wypełnianie łuków i zrealizowane to zostało w module `image`. Konieczne było jednak korzystanie z wewnętrznych struktur danych biblioteki GD.

W nowej wersji biblioteki GD wypełnianie jest realizowane przez funkcję biblioteki i nie ma potrzeby robienia tego w module image, wystarczy skorzystać z odpowiednich funkcji `gdImagedFilledArc()`. Funkcja ta pobiera dodatkowy argument – styl wypełnienia, czego nie ma w funkcji `gdImageArc` używanej dotąd w module image. W związku z tą zmianą należało dostosować interfejs funkcji `image::arc` pochodzącej z modułu `image`. Styl wypełnienia łuku jest kombinacją czterech wartości: ARG, CHORD, NOFILL, EDGED. Te zmiany w kodzie modułu image były wystarczające by go skompilować z najnowszą wersją biblioteki GD.

Po dokonaniu poprawek i udanej kompilacji, zmiany zostały dokładnie przetestowane przy użyciu wcześniej opisywanego w 3.1 skryptu. Sprawdzone zostało zachowanie modułu przy różnych kombinacjach stylu wypełnienia. W trakcie dokonywane były poprawki w zmodyfikowanej funkcji `pf_image_arc()`.

3.3 Rozbudowa modułu `image`

Nowa wersja biblioteki GD dostarcza wielu nowych funkcjonalności, dotychczas niedostępnych. Aby wykorzystać nowe możliwości i rozbudować w ten sposób Meta-HTML wykonano szereg zmian w kodzie modułu `image`.

Podstawową zasadą jaką kierowaliśmy się dokonując zmian w kodzie było zachowanie kompatybilności z poprzednią wersją modułu `image`, tak aby stare skrypty działały w ten sam sposób jak dotąd bez potrzeby ich modyfikacji. Poniżej opisujemy dokonane modyfikacje.

`image::create`

`IMAGEVAR &key [WIDTH] [HEIGHT] [SRC] [FORMAT] [TRUECOLOR]`

Funkcja ta została zmodyfikowała tak, że poza obowiązującym dotychczas formatem GIF dodatkowo daje możliwość tworzenia obrazów w formatach PNG i JPEG. Format określa się za pomocą argumentu `FORMAT`. Domyślna wartość `FORMAT` to GIF.

W przypadku formatu PNG mamy również możliwość wyboru pomiędzy 8-bitową a 24-bitową paletą kolorów. Domyślnie używana jest paleta 8-

bitowa. Aby utworzyć obraz w pełnym kolorze 24-bitowym należy podać argument `TRUECOLOR`.

`image::set-pixel`
`IMAGEVAR &key [X] [Y] [COLOR]`

Funkcja została dostosowana do tego, że obraz może mieć paletę 8-bitową lub 24-bitową. W pierwszym przypadku wyliczany jest indeks koloru w palecie obrazu, jeśli taki kolor w palecie nie istnieje dodawany jest jako nowy. Może się zdarzyć, że paleta osiągnęła już swój maksymalny rozmiar (256 pozycji) i nie ma tam miejsca na dodanie nowej wartości. Wówczas wybierany jest z palety kolor o możliwie najbliższej wartości (najbardziej podobny).

Jeśli obraz jest 24-bitowy kolor piksla o współrzędnych (X, Y) jest ustawiany na taki jaki podał użytkownik.

Te same zmiany powinny być wykonane w pozostałych funkcjach rysujących. Aby nie powtarzać tego samego bloku kodu zostały napisane dwie nowe funkcje:

- `rgb_to_truecolor()`
  Funkcja zamienia podaną wartość koloru w formacie RGB jako tekst na wartość numeryczną.
- `rgb_to_color()` Funkcja sprawdza z jaką paletą mamy do czynienia, 8-bitową czy 24-bitową, i odpowiednio zwraca indeks koloru w palecie lub wartość numeryczną. Takie funkcje jak:
  - `image::set-pixel`,
  - `image::text`,
  - `image::arc`,
  - `image::fill`,
  - `image::line`,
  - `image::rect`

  wołają `rgb_to_color()`.

image::get-pixel
`IMAGEVAR &key [X] [Y]`

Do tej pory ta funkcja pobierała indeks koloru piksla o podanych współrzędnych (`X`, `Y`). Jako wynik wyświetlana była wartość koloru pobranego z palety o znalezionym wcześniej indeksie. W przypadku obrazów z 24 bitową paletą należało zmodyfikować działanie tej funkcji tak, aby bezpośrednio pobierała wartość koloru piksla z obrazu, a nie z palety.

`image::set-tile`
`IMAGEVAR TILE`

Jest to nowa funkcja dodana do modułu `image`. Przy jej pomocy określa się "kafelek" (ang. tile, czasem tłumaczy się to w tym kontekście również jako "dachówka"), który będzie wykorzystany przy funkcji wypełnienia `image::fill`.

Kafelek jest to obraz używany do wypełnienia obszaru powtarzającym się wzorem. Jako kafelek można wybrać dowolny obraz. Jeśli nie będzie on miał tej samej mapy kolorów co obraz, na którym będzie się go stosowało to wszystkie brakujące kolory zostaną zaalokowane. Jeśli nie wystarczy miejsc dla dodatkowych barw to kolory najbliższe brakującym zostaną użyte. Oznacza to, że nie należy przydzielać kafelka danemu obrazowi jeśli nie chce się z niego korzystać. Taka kilkukrotna operacja może szybko wypełnić paletę.

`image::set-brush`
`IMAGEVAR BRUSH`

Przy pomocy tej nowej funkcji określa się "pędzel" (ang. brush) jaki będzie wykorzystany w różnych funkcjach rysujących np. `image::line`.

Pędzel, podobnie jak opisywany wyżej kafelek, to dowolny obraz. Z jego wyborem wiążą się te same konsekwencje co z wyborem kafelka.

`image::text-al`
`IMAGEVAR TEXT &key [X] [Y] [COLOR] [SIZE] [ANGLE] [FONTNAME]`

Ta nowa funkcja służy do rysowania wielu znaków w obrazie z wykorzystaniem anty-aliasingu (wygładzanie krawędzi). Tekst narysowany w ten sposób będzie się wydawać mniej kanciasty, bardziej gładki. Wykorzystywana jest w tym celu biblioteka FreeType oraz fonty TrueType (pliki `.ttf` oraz `.ttc`). Biblioteka GD nie oferuje żadnych standardowych fontów tego rodzaju, należy więc zdobyć je samodzielnie.

Napis `TEXT` rysowany jest od punktu o współrzędnych (`X`, `Y`). `FONTNAME` określa nazwę pliku z fontem TrueType. Rysowany ciąg znaków może być przeskalowany w skali `SIZE` oraz obrócony o podany kąt `ANGLE` (w stopniach). Kierunek obrotu jest przeciwny do ruchu wskazówek zegara, gdzie 0 stopni oznacza położenie w pozycji godziny 15, natomiast 90 stopni położenie w pozycji godziny 12. Argument `COLOR` określa kolor w jakim zostanie narysowany podany napis.

W trakcie modyfikacji kodu modułu `image` robiliśmy testy przy pomocy skryptu opisanego w 3.1. Sprawdziliśmy, czy udało się zachować kompatybilność. Rzeczywiście wynik działania skryptu był dokładnie taki sam jak w poprzedniej wersji modułu `image`. Zweryfikowaliśmy oczywiście wprowadzone nowe funkcje. Testy dotyczyły wszystkich trzech formatów: GIF, PNG i JPEG, a w przypadku PNG testowane były zarówno paleta 8-bitowa jak 24-bitowa.

Dodatek A

Funkcje GD

A.1 Funkcje tworzenia i niszczenia obrazu, wgrywanie i zapisywanie

`gdImageCreate(sx, sy)`

Funkcja służy do tworzenia obrazu z paletą z maksymalną liczbą 256 kolorów. W wywołaniu podajemy dwie wartości określające jego rozdzielczość. Wartością zwracaną jest wskaźnik do struktury gdImage, czyli gdImagePtr bądź NULL kiedy niemożliwe stanie się zaalokowanie nowego obrazu. Po zakończeniu pracy z tym obiektem, powinien on zostać zniszczony za pomocą gdImageDestroy.

`gdImageCreateTrueColor(sx, sy)`

Funkcja służy do tworzenia obrazu truecolor z nieograniczoną liczbą kolorów. W wywołaniu podajemy dwie wartości określające jego rozdzielczość. Wartością zwracaną jest wskaźnik do struktury gdImage, czyli gdImagePtr bądź NULL kiedy niemożliwe stanie się zaalokowanie nowego obrazu. Po zakończeniu pracy z tym obiektem, powinien on zostać zniszczony za pomocą gdImageDestroy.

`gdImageCreateFromJpeg(FILE *in)`

`gdImageCreateFromJpegCtx(FILE *in)`

Funkcja służy do wczytywania obrazu truecolor z pliku w formacie JPEG. W wywołaniu podajemy wskaźnik do struktury FILE otwartego już pliku zawierającego rysunek. Wartością zwracaną jest wskaźnik do struktury gdImage, czyli gdImagePtr bądź NULL kiedy niemożliwe jest wgranie informacji z pliku (z reguły plik jest uszkodzony bądź nie zawiera rysunku w formacie JPEG). Funkcja ta nie zamyka pliku. Po zakończeniu pracy ze zwróconym obiektem, powinien on zostać zniszczony za pomocą gdImageDestroy. Zwracany obraz jest zawsze typu truecolor. Można poznać jego rozmiary sprawdzając pozycje sx i sy.

1. gdImageCreateFromPng(FILE *in)

   gdImageCreateFromPngCtx(gdIOCtx *in)

   Funkcja służy do wczytania obrazu z pliku w formacie PNG. W wywołaniu podajemy wskaźnik do struktury FILE otwartego już pliku zawierającego rysunek. Wartością zwracaną jest wskaźnik do struktury gdImage, czyli gdImagePtr bądź NULL kiedy niemożliwe jest wgranie informacji z pliku (z reguły plik jest uszkodzony bądź nie zawiera rysunku w formacie PNG). Funkcja ta nie zamyka pliku. Po zakończeniu pracy ze zwróconym obiektem, powinien on zostać zniszczony za pomocą gdImageDestroy. Rozmiary obrazu można poznać sprawdzając pozycje sx i sy.

   Jeżeli wczytywany rysunek z pliku PNG jest w formacie truecolor to otrzymana struktura będzie się odnosić do obrazu truecolor. W przeciwnym wypadku, kiedy wczytywany jest rysunek z paletą, otrzymamy obraz z paletą. GD uwzględnia jedynie 8 bitów na każdą ze składowych (czerwoną, zieloną i niebieską) oraz 7 bitów na kanał alpha. Pierwsze ograniczenia oddziałuje jedynie na rzadko spotykane pliki PNG o 48 bitowym kolorze oraz na PNG z 16 bitową skalą szarości. Drugie ograniczenie dotyczy różnoprzezroczystych rysunków korzystających z dużej liczby poziomów kanału alpha. Różnice są jednak niewidoczne dla oka i 7 bitów w tym przypadku to liczba wystarczająca.

2. gdImageCreateFromPngSource(gdSourcePtr in)

   Funkcja służy do wczytania obrazu PNG z innego źródła niż plik. Użycie podobne jest do funkcji gdImageCreateFromPng z tym wyjątkiem, że programista dostarcza własnego źródła danych.

   Programista musi napisać własną funkcję, która przyjmuje jako argumenty wskaźnik kontekstu, bufor oraz liczbę bajtów, która ma zostać odczytana. Funkcja musi odczytać żądaną liczbę bajtów do póki koniec pliku nie zostanie osiągnięty – w tym szczególnym przypadku funkcja powinna zwrócić 0. Zdarzyć się może również, że podczas odczytywania nastąpi pewien błąd, wówczas definiowana funkcja powinna zwrócić –1. Po zaprojektowaniu tej funkcji, programista tworzy strukturę gdSource i ustawia wskaźnik source na funkcję wejściową, zaś wskaźnik kontekstu na jakąkolwiek wartość użyteczną.

3. gdImageCreateFromGd(FILE *in)

   gdImageCreateFromGdCtx(gdIOCtx *in)

   Funkcja służy do wczytania obrazu z pliku w formacie gd. W wywołaniu podajemy wskaźnik do struktury FILE otwartego już pliku zawierającego rysunek. Format gd jest specyficzny dla libgd i służy do szybkiego

wczytywania informacji o obrazie (nie jest on przeznaczony do kompresowania danych, w tym celu należy użyć JPEG bądź PNG). Wartością zwracaną jest wskaźnik do struktury gdImage, czyli gdImagePtr bądź NULL kiedy niemożliwe jest wgranie informacji z pliku (z reguły plik jest uszkodzony bądź nie zawiera rysunku w formacie gd). Funkcja ta nie zamyka pliku. Po zakończeniu pracy ze zwróconym obiektem, powinien on zostać zniszczony za pomocą gdImageDestroy. Rozmiary obrazu można poznać sprawdzając pozycje sx i sy.

4. gdImageCreateFromGd2(FILE *in)

   gdImageCreateFromGd2Ctx(gdIOCtx *in)

   Funkcja służy do wczytania obrazu z pliku w formacie gd2. W wywołaniu podajemy wskaźnik do struktury FILE otwartego już pliku zawierającego rysunek. Format gd2 jest specyficzny dla libgd i służy do szybkiego wczytywania kawałków (części) dużego obrazu (format ten wykorzystuje kompresje, ale nie jest ona najlepsza). Wartością zwracaną jest wskaźnik do struktury gdImage, czyli gdImagePtr bądź NULL kiedy niemożliwe jest wgranie informacji z pliku (z reguły plik jest uszkodzony bądź nie zawiera rysunku w formacie gd2). Funkcja ta nie zamyka pliku. Po zakończeniu pracy ze zwróconym obiektem, powinien on zostać zniszczony za pomocą gdImageDestroy. Rozmiary obrazu można poznać sprawdzając pozycje sx i sy.

5. gdImageCreateFromGd2Part(FILE *in, int srcX, int srcY, int w, int h)

   gdImageCreateFromGd2PartCtx(gdIOCtx *in)

   Funkcja służy do wczytywania kawałków (części) z pliku gd2. Wywołanie jest podobne do gdImageCreateFromGd2, z tym że podajemy dodatkowe parametry określające punkt początkowy (x, y) oraz szerokość i wysokość obszaru do odczytania. Wartością zwracaną jest wskaźnik do struktury gdImage, czyli gdImagePtr bądź NULL kiedy niemożliwe jest wgranie informacji z pliku. Po zakończeniu pracy ze zwróconym obiektem, powinien on zostać zniszczony za pomocą gdImageDestroy.

6. gdImageCreateFromXbm(FILE *in)

   Funkcja służy do wczytania obrazu z pliku w formacie bitmapy X. W wywołaniu podajemy wskaźnik do struktury FILE otwartego już pliku zawierającego rysunek. Wartością zwracaną jest wskaźnik do struktury gdImage, czyli gdImagePtr bądź NULL kiedy niemożliwe jest wgranie informacji z pliku (z reguły plik jest uszkodzony bądź nie zawiera rysunku w formacie bitmapy X). Funkcja ta nie zamyka pliku. Po zakończeniu pracy ze zwróconym obiektem, powinien on zostać zniszczony za pomocą gdImageDestroy. Rozmiary obrazu można poznać sprawdzając pozycje sx i sy.

7. gdImageCreateFromXpm(char *filename)

   Funkcja służy do wczytania obrazu z pliku w formacie bitmapy XPM X Window System. W przeciwieństwie do innych funkcji wczytujących i tworzących obraz musimy tutaj podać nazwę pliku a nie wskaźnik do jego struktury FILE. Wartością zwracaną jest wskaźnik do struktury gdImage, czyli gdImagePtr bądź NULL kiedy niemożliwe jest wgranie informacji z pliku (z reguły plik jest uszkodzony bądź nie zawiera rysunku w formacie bitmapy XPM X Window System). Po zakończeniu pracy ze zwróconym obiektem, powinien on zostać zniszczony za pomocą gdImageDestroy. Rozmiary obrazu można poznać sprawdzając pozycje sx i sy.

8. gdImageDestroy(gdImagePtr im)

   Funkcja służy do zwalniania pamięci przydzielonej strukturze obrazu. Rzeczą ważną jest aby przed wyjściem z programu zniszczyć wszystkie stworzone wcześniej struktury gdImage.

9. void gdImageJpeg(gdImagePtr im, FILE *out, int quality)

   void gdImageJpegCtx(gdImagePtr im, gdIOCtx *out, int quality)

   Funkcja służy do zapisywania podanego rysunku do wskazanego pliku. Informacje zapisane zostaną w formacie JPEG. Plik musi być otwarty z możliwością zapisywania do niego. W systemach MSDOS i Windows jest rzeczą ważną aby plik został otworzony z flagą "wb", co nie tylko oznacza iż mamy możliwość zapisywania ale także iż pracujemy w trybie binarnym i żadne konwersje danych nie powinny być dokonywane przez system operacyjny. Funkcja nie zamyka pliku, programista powinien zrobić to sam.

   Jeżeli quality (jakość) jest wartością negatywną to używana jest kompresja domyślna, która w rzeczywistości powinna być dość dobra. Jeżeli chcemy sami zmieniać jakość rysunku to powinniśmy użyć wartości z przedziału 0 do 95. Większe wartości oznaczają lepszą jakość.

   Jeżeli programista ustawił interlacing (przeplatanie) za pomocą gdImageInterlace, to omawiana funkcja zapisze progressive (postępujący) JPEG. Niektóre programy (w szczególności przeglądarki internetowe) mają możliwość wyświetlania tego typu rysunków etapami. Może się to okazać przydatne gdy ktoś ze słabym połączeniem sieciowym wczytuje stronę internetową z rysunkiem. Zobaczy on wówczas kolejno obraz w słabszej, a dopiero później w coraz to lepszej jakości . JPEG'i zapisane w ten sposób mogą się także okazać nieco mniejsze w rozmiarach od sekwencyjnych JPEG'ów (non-progressive).

10. void* gdImageJpegPtr(gdImagePtr im, int *size)

Funkcja identyczna jak gdImageJpeg, z tym że zwraca ona wskaźnik do pamięci gdzie zapisane zostały dane JPEG'a. Obszar pamięci musi zostać zwolniony później przez programistę. W tym calu należy użyć funkcji gdFree. Parametr size określa wielkość całego bloku pamięci.

11. void gdImagePng(gdImagePtr im, FILE *out)

    Funkcja służy do zapisywania podanego rysunku do wskazanego pliku. Informacje zapisane zostaną w formacie PNG. Plik musi być otwarty z możliwością zapisywania do niego. W systemach MSDOS i Windows jest rzeczą ważną aby plik został otworzony z flagą "wb", co nie tylko oznacza iż mamy możliwość zapisywania ale także iż pracujemy w trybie binarnym i żadne konwersje danych nie powinny być dokonywane przez system operacyjny. Funkcja nie zamyka pliku, programista powinien zrobić to sam.

12. void* gdImagePngPtr(gdImagePtr im, int *size)

    Funkcja identyczna jak gdImagePng, z tym że zwraca ona wskaźnik do pamięci gdzie zapisane zostały dane PNG'a. Obszar pamięci musi zostać zwolniony później przez programistę. W tym calu należy użyć funkcji gdFree. Parametr size określa wielkość całego bloku pamięci.

13. gdImagePngToSink(gdImagePtr im, gdSinkPtr out)

    Funkcja umożliwia zapisywanie obrazu w inne miejsce niż plik. Użycie jest podobne do gdImagePng, z tym że programista dostarcza własny cel.

    Programista musi napisać własną funkcję, która przyjmuje jako argumenty wskaźnik kontekstu, bufor oraz liczbę bajtów, która ma zostać zapisana. Funkcja musi zapisać żądaną liczbę bajtów i zwrócić tą wartość. Zdarzyć się może, że podczas odczytywania nastąpi pewien błąd, wówczas definiowana funkcja powinna zwrócić –1. Po zaprojektowaniu tej funkcji, programista tworzy strukturę gdSink i ustawia wskaźnik sink na funkcję wyjściową, zaś wskaźnik kontekstu na jakąkolwiek wartość użyteczną.

14. void* gdImageWBMPPtr(gdImagePtr im, int *size)

    Funkcja identyczna jak gdImageWBMP, z tym że zwraca ona wskaźnik do pamięci gdzie zapisane zostały dane WBMP'a. Obszar pamięci musi zostać zwolniony później przez programistę. W tym calu należy użyć funkcji gdFree. Parametr size określa wielkość całego bloku pamięci.

15. void gdImageGd(gdImagePtr im, FILE *out)

    Funkcja służy do zapisywania podanego rysunku do wskazanego pliku. Informacje zapisane zostaną w formacie gd. Plik musi być otwarty z

możliwością zapisywania do niego. W systemach MSDOS i Windows jest rzeczą ważną aby plik został otworzony z flagą "wb", co nie tylko oznacza iż mamy możliwość zapisywania ale także iż pracujemy w trybie binarnym i żadne konwersje danych nie powinny być dokonywane przez system operacyjny. Funkcja nie zamyka pliku, programista powinien zrobić to sam.

Format gd jest przeznaczony do szybkiego wczytywania i zapisywania obrazów. Może się to szczególnie przydać gdy program często korzysta z innych rysunków budując nową bitmapę. Format ten nie używa kompresji i nie powinien być używany gdy jest to zbędne.

16. void* gdImageGdPtr(gdImagePtr im, int *size)

    Funkcja identyczna jak gdImageGd, z tym że zwraca ona wskaźnik do pamięci gdzie zapisane zostały dane gd'a. Obszar pamięci musi zostać zwolniony później przez programistę. W tym calu należy użyć funkcji gdFree. Parametr size określa wielkość całego bloku pamięci.

17. void gdImageGd2(gdImagePtr im, FILE *out, int chunkSize, int fmt)

    Funkcja służy do zapisywania podanego rysunku do wskazanego pliku. Informacje zapisane zostaną w formacie gd2. Plik musi być otwarty z możliwością zapisywania do niego. W systemach MSDOS i Windows jest rzeczą ważną aby plik został otworzony z flagą "wb", co nie tylko oznacza iż mamy możliwość zapisywania ale także iż pracujemy w trybie binarnym i żadne konwersje danych nie powinny być dokonywane przez system operacyjny. Funkcja nie zamyka pliku, programista powinien zrobić to sam.

    Format gd2 jest przeznaczony do szybkiego odczytywania i zapisywania kawałków (części) obrazu. Wykorzystuje on kompresje i jest dobry do pobierania niewielkich sekcji z większych rysunków.

    Plik jest zapisywany w postaci mniejszych skompresowanych podobrazków, Chunk Size określa ich rozmiar. 0 oznacza wartość domyślną wybraną przez gd. Możliwe jest również przetrzymywanie nieskompresowanych plików gd2.

18. void* gdImageGd2Ptr(gdImagePtr im, int chunkSize, int fmt, int *size)

    Funkcja identyczna jak gdImageGd, z tym że zwraca ona wskaźnik do pamięci gdzie zapisane zostały dane gd2'a. Obszar pamięci musi zostać zwolniony później przez programistę. W tym calu należy użyć funkcji gdFree. Parametr size określa wielkość całego bloku pamięci.

19. void gdImageToPalette(gdImagePtr im, int ditherFlag, int colorsWanted)

Funkcja służy do konwersji obrazu truecolor do obrazu z paletą. Używając dwóch iteracji kwantyzacji zachowywane są informacje zarówno o kolorze (RGB) jak i kanale alpha. Jeżeli argument ditherFlag jest ustawiony to dodatkowo funkcja będzie aproksymować kolory aby lepiej one odwzorowywały pierwotny rysunek. (kosztem efektu, który nazwać by można 'plamkowaniem'). Argument colorsWanted może być wartością do 256, określa on ilość kolorów którą chcemy uzyskać w obrazie wyjściowym. Jeżeli konwertujemy fotografię to powinniśmy tu użyć 256 barw.

Jednak lepiej nie używać tej funkcji i zapisywać obrazy jako truecolor PNG bądź JPEG. Przedstawiona konwersja w rzeczywistości nie oszczędza wielkiej ilości miejsca na dysku (w szczególności dla małych obrazów), natomiast utrata jakości jest ohydna.

A.2 Funkcje rysujące

1. void gdImageSetPixel(gdImagePtr im, int x, int y, int color)

   Funkcja ustawia wskazany piksel na podany numer indeksu (kolor). Zawsze powinieneś używać tej lub innych funkcji rysujących aby manipulować pikselami. Niewskazane jest bezpośrednie zapisywanie z użyciem struktury gdImage.

2. void gdImageLine(gdImagePtr im, int x1, int y1, int x2, int y2, int color)

   Funkcja służy do rysowania linii pomiędzy dwoma punktami (x1,y1, x2,y2). Linia jest malowana w kolorze, którego indeks jest podany jako ostatni argument. Może nim być zarówno kolor zwrócony przez funkcję gdImageColorAllocate lub gdStyled, gdBrushed czy gdStyledBrushed.

3. void gdImageDashedLine(gdImagePtr im, int x1, int y1, int x2, int y2, int color)

   Funkcja ta jest wciąż dostępna ze względów zachowania kompatybilności ze starszymi wersjami gd. Do rysowania linii przerywanych należy korzystać teraz z funkcji gdImageLine w połączeniu z nową funkcją gdImageSetStyle.

   Funkcja służy do rysowania przerywanych linii pomiędzy dwoma punktami (x1,y1, x2,y2). Linia jest malowana w kolorze, którego indeks jest podany jako ostatni argument. Części linii, które nie są rysowane pozostają przezroczyste, tak że tło jest widoczne.

4. void gdImagePolygon(gdImagePtr im, gdPointPtr points, int pointsTotal, int color)

   Funkcja służy do rysowania wieloboku na podstawie podanych wierzchołków, których musi być co najmniej 3. Figura malowana jest w podanym kolorze.

5. void gdImageRectangle(gdImagePtr im, int x1, int y1, int x2, int y2, int color)

   Funkcja służy do rysowania prostokąta na podstawie dwóch podanych wierzchołków (lewego górnego rogu x1,y1 oraz prawego dolnego rogu x2,y2). Figura malowana jest w podanym kolorze.

6. void gdImageFilledPolygon(gdImagePtr im, gdPointPtr points, int pointsTotal, int color)

   Funkcja służy do rysowania wypełnionego wieloboku na podstawie podanych wierzchołków, których musi być co najmniej 3. Figura malowana jest w podanym kolorze.

7. void gdImageFilledRectangle(gdImagePtr im, int x1, int y1, int x2, int y2, int color)

   Funkcja służy do rysowania wypełnionego prostokąta na podstawie dwóch podanych wierzchołków (lewego górnego rogu x1,y1 oraz prawego dolnego rogu x2,y2). Figura malowana jest w podanym kolorze

8. void gdImageArc(gdImagePtr im, int cx, int cy, int w, int h, int s, int e, int color)

   Funkcja służy do rysowania części elipsy, o podanym środku (cx, cy) oraz o podanej w pikselach szerokości i wysokości (w, h). Łuk zaczyna się w pozycji s podanej w stopniach i kończy w pozycji e także podanej w stopniach, przy czym e musi być większe od s. Wartości większe od 360 są dzielone modulo przez 360. Krzywa rysowana jest w podanym kolorze. Za pomocą tej funkcji można narysować okrąg zaczynają od 0 stopni i kończąc na 360. Podana szerokość i wysokość muszą być sobie w tym przypadku równe.

9. void gdImageFilledArc(gdImagePtr im, int cx, int cy, int w, int h, int s, int e, int color, int style)

   Funkcja służy do rysowania części elipsy o podanym środku (cx, cy) oraz o podanej w pikselach szerokości i wysokości (w, h). Łuk zaczyna się w pozycji s podanej w stopniach i kończy w pozycji e także podanej w stopniach, przy czym e musi być większe od s. Wartości większe od 360 są dzielone modulo przez 360. Figura zostaje wypełniona w kolorze określonym przez dwa ostatnie argumenty.

10. void gdImageFilledEllipse(gdImagePtr im, int cx, int cy, int w, int h, int s, int e, int color)

    Funkcja służy do rysowania wypełnionej elipsy o podanym środku (cx, cy) oraz o podanej w pikselach szerokości i wysokości (w, h). Elipsa zostaje wypełniona kolorem podanym jako ostatni argument. e musi

być większe od s. Wartości większe od 360 są dzielone modulo przez 360. Za pomocą tej funkcji można narysować koło zaczynają od 0 stopni i kończąc na 360. Podana szerokość i wysokość muszą być sobie w tym przypadku równe.

11. void gdImageFillToBorder(gdImagePtr im, int x, int y, int border, int color)

    Funkcja służy do wypełniania ograniczonego obszaru podanym kolorem. Proces rozpoczyna się w podanym punkcie i ograniczony jest konturem w kolorze border. W jaki sposób wypełnić obszar o kolorze określonym przez punkt startowy zobacz gdImageFill. Kolor granicy nie może być jednym ze specjalnych kolorów, takich jak gdTiled.

12. void gdImageFill(gdImagePtr im, int x, int y, int color)

    Funkcja służy do wypełniania części obrazu podanym kolorem zaczynając w podanym punkcie. Wypełniany jest sąsiadujący obszar, który ma tą samą barwę co punkt początkowy. Kolorem wypełnienia może być gdTiled co spowoduje wypełnienie obszaru korzystając z innego obrazka jako dachówki. Nie może on być jednak przezroczysty. Jeżeli jednak chcemy wypełnić pewien obszar obrazkiem, który posiada indeks koloru przezroczystości powinniśmy użyć na nim gdImageTransparent, a następnie ustawić indeks koloru transparent na –1.

13. void gdImageSetBrush(gdImagePtr im, gdImagePtr brush)

    Brush (szczotka, pędzel) to rysunek, którym można rysować po obrazie. Może być on złożony z wielu pikseli. Jakikolwiek obraz gd może być użyty jako pędzel i poprzez odpowiednie ustawienie jego indeksu koloru przezroczystości (za pomocą gdImageColorTransparent) można uzyskać dowolny kształt. Wszystkie funkcje rysujące linie (np. gdImageLine, gdImagePolygon) używają bieżącego pędzla jeśli jako kolor podamy gdBrushed bądź gdStyledBrushed.

    Funkcja służy do wyszczególnienia pędzla, który ma zostać przydzielony danemu obrazowi. Można ustawić którykolwiek obraz jako pędzel. Jeśli nie będzie on miał tej samej mapy kolorów co obraz, na którym będziemy go stosować to wszystkie brakujące kolory zostaną zaalokowane. Jeśli nie wystarczy miejsc dla dodatkowych barw to kolory najbliższe brakującym zostaną użyte. Oznacza to, że nie powinieneś przydzielać pędzla danemu obrazowi jeśli nie chcesz z niego korzystać. Kilkukrotna taka operacja może szybko wypełnić paletę.

    Nie trzeba podejmować żadnych operacji po zakończeniu używania pędzla. Postępujemy z nim jak z obrazem i w razie gdy przestaje on być potrzebny przekazujemy go funkcji gdImageDestroy. Kolejne wywołania przydzielające nowy pędzel zastępują stary wybór.

14. void gdImageSetTile(gdImagePtr im, gdImagePtr tile)

    Tile (kafelki) jest to obraz używany do wypełnienia obszaru powtarzającym się wzorem. Jakikolwiek obraz gd może być użyty jako kafelek i poprzez odpowiednie ustawienie jego indeksu koloru przezroczystości (za pomocą gdImageColorTransparent) można sprawić, że żądane części obrazu ulokowanego niżej będą świecić. Wszystkie funkcje wypełniające regiony (np. gdImageFill, gdImageFilledPolygon) używają bieżącego kafelka jeśli jako kolor podamy gdTiled podczas ich wywołania.

    Funkcja służy do wyszczególnienia kafelki, która ma zostać przydzielona danemu obrazowi. Można ustawić którykolwiek obraz jako kafelek. Jeśli nie będzie on miał tej samej mapy kolorów co obraz, na którym będzie się go stosowało to wszystkie brakujące kolory zostaną zaalokowane. Jeśli nie wystarczy miejsc dla dodatkowych barw to kolory najbliższe brakującym zostaną użyte. Oznacza to, że nie należy przydzielać dachówki danemu obrazowi jeśli nie chcesz z niej korzystać. Kilkukrotna taka operacja może szybko wypełnić paletę.

    Nie trzeba podejmować żadnych operacji po zakończeniu używania kafelka. Należy z nim postępować jak z obrazem i w razie gdy przestaje ona być potrzebna przekazujemy ją funkcji gdImageDestroy. Kolejne wywołania przydzielające nowego kafelka zastępują stary wybór.

15. void gdImageSetStyle(gdImagePtr im, int *style, int styleLength)

    Często chcemy narysować linię przerywaną, wykropkowaną bądź jakąkolwiek inną, posiadającą przerwy. Funkcja ta pozwala na ustawienie kolorów, włączając w to kolory specjalne pozostawiające nienaruszone tło, które mają być powtarzane podczas rysowania linii.

    Aby użyć omawianej funkcji, należy stworzyć tablicę liczb całkowitych (int) i nadać kolejnym jej elementom wartości kolorów, które mają być powtarzane. Można wykorzystać tutaj specjalny kolor gdTransparent, który oznacza iż w rysowanym punkcie o tym kolorze pozostawione zostanie tło, co jest szczególnie przydatne gdy rysujemy już na jakimś obrazie.

    Następnie kiedy chcemy już namalować linię ze zdefiniowanym wzorem, powinniśmy użyć gdImageLine, ze specjalną wartością koloru gdStyled.

    Możliwe jest mieszanie zdefiniowanego stylu z wykorzystywanym aktualnie pędzlem, tak że malując nim otrzymać możemy nieciągły wzór. Kiedy tworzysz styl do wykorzystania z pędzlem to użyte kolory są interpretowane następująco : 0 oznacza iż w danym miejscu nie powinny być rysowane piksele pędzla, natomiast 1 wskazuje odwrotnie iż w danym położeniu należy rysować zgodnie z definicją pędzla. Aby namalować tego typu linię, należy użyć specjalnego koloru gdStyledBrushed.

16. void gdImageAlphaBlending(gdImagePtr im, int blending)

    Funkcja ta umożliwia dwa róże tryby rysowania w obrazach typu truecolor. W trybie blending (mieszanie), który jest wybierany jak domyślny, składowa alpha koloru podawanego jako argument np. do funkcji gdImageSetPixel, określa jak bardzo piksel leżący już w danym miejscu powinien prześwitywać. Jako rezultat gd miesza z sobą dwa kolory, jeden napotkany w danym miejscu rysunku a drugi podany w wywołaniu funkcji, zaś rezultat zapisuje do obrazu. W trybie non-blending (brak mieszania), podany kolor jest kopiowany w podane miejsce wraz z jego kanałem alpha, zastępując w ten sposób napotkany tam piksel. Tryb blending (mieszanie) nie jest dostępny w obrazach z paletą.

17. void gdImageSaveAlpha(gdImagePtr im, int saveFlag)

    Domyślnie gd nie zapisuje pełnej informacji o kanale alpha do pliku, w przeciwieństwie do pojedynczego indeksu koloru przezroczystości. Jedynym formatem, który może te dane przechować jest PNG. Należy jednak wywołać podaną funkcję w celu zaznaczenia, że te informacje mają być podczas zapisywania uwzględnione. W tym celu saveFlag ustawiamy na wartość 1. Powinniśmy również wywołać gdImageAlphaBlending(im, 0) aby wyłączyć blending (mieszanie) wewnątrz biblioteki. W ten sposób informacje o kanale przezroczystości będą przechowywane w obrazie, a nie tworzone dopiero podczas wywoływania funkcji rysujących.

A.3 Funkcje sprawdzające, pytające

1. int gdImageAlpha(gdImagePtr im, int color) (MAKRO)

    Makro to zwraca informacje o kanale alpha podanego koloru (indeksu koloru). Wartości tej składowej są z zakresu 0 (gdAlphaOpaque), która w rzeczywistości oznacza brak mieszania z tłem, aż po 127 (gdAlphaTransparent), która pozwala na prześwitywanie tła w 100 procentach. Niewskazane jest odczytywanie informacji o kanale alpha korzystając z bezpośredniego dostępu do obrazu.

2. int gdImageRed(gdImagePtr im, int color) (MAKRO)

    Makro to zwraca wartość składowej czerwonej podanego koloru.

3. int gdImageGreen(gdImagePtr im, int color) (MAKRO)

    Makro to zwraca wartość składowej zielonej podanego koloru.

4. int gdImageBlue(gdImagePtr im, int color) (MAKRO)

    Makro to zwraca wartość składowej niebieskiej podanego koloru.

5. int gdImageGetPixel(gdImagePtr im, int x, int y)

   Funkcja zwraca indeks koloru podanego piksela.

6. int gdImageBoundsSafe(gdImagePtr im, int x, int y)

   Funkcja zwraca true (1) jeśli podany piksel leży w obrębie danego obrazu, w przeciwnym wypadku zwraca ona false (0). Ta funkcja jest zaprojektowana z myślą o programistach chcących dodać własne funkcje do gd. Funkcje rysujące libgd dokonują clippingu (obcinania) jeżeli podane współrzędne są zbyt wielkie lub zbyt małe.

7. int gdImageSX(gdImagePtr im) (MAKRO)

   Makro to zwraca szerokość obrazu w pikselach.

8. int gdImageSY(gdImagePtr im) (MAKRO)

   Makro to zwraca wysokość obrazu w pikselach.

A.4 Funkcje obsługi fontu i tekstu

1. void gdImageChar(gdImagePtr im, gdFontPtr font, int x, int y, int c, int color)

   Funkcja ta służy do stawiania pojedynczego znaku w obrazie. Drugim argumentem jest wskaźnik do struktury opisującej font. Pięć różnych fontów jest standardowo dostępnych w libgd : gdFontTiny, gdFontSmall, gdFontMediumBold, gdFontLarge, oraz gdFontGiant. Jeśli trzeba skorzystać z jednego z nich to należy pamiętać o dołączeniu odpowiedniego pliku nagłówkowego, są to odpowiednio : "gdfontt.h", "gdfonts.h", "gdfontmb.h", "gdfontl.h" oraz "gdfontg.h". Znak podany jako piąty argument rysowany jest od lewej strony do prawej używając podanego koloru

2. void gdImageCharUp(gdImagePtr im, gdFontPtr font, int x, int y, int c, int color)

   Funkcja ta służy do stawiania pojedynczego znaku w obrazie obróconego o 90 stopni. Drugim argumentem jest wskaźnik do struktury opisującej font. Pięć różnych fontów jest standardowo dostępnych w libgd : gdFontTiny, gdFontSmall, gdFontMediumBold, gdFontLarge, oraz gdFontGiant. Jeśli trzeba skorzystać z jednego z nich to należy pamiętać o dołączeniu odpowiedniego pliku nagłówkowego, są to odpowiednio : "gdfontt.h", "gdfonts.h", "gdfontmb.h", "gdfontl.h" oraz "gdfontg.h". Znak podany jako piąty argument rysowany jest od dołu do góry (obrócony o 90 stopni) używając podanego koloru.

3. void gdImageString(gdImagePtr im, gdFontPtr font, int x, int y, unsigned char *s, int color)

   Funkcja ta służy do rysowania wielu znaków w obrazie. Drugim argumentem jest wskaźnik do struktury opisującej font. Pięć różnych fontów jest standardowo dostępnych w libgd : gdFontTiny, gdFontSmall, gdFontMediumBold, gdFontLarge, oraz gdFontGiant. Jeśli trzeba skorzystać z jednego z nich to należy pamiętać o dołączeniu odpowiedniego pliku nagłówkowego, są to odpowiednio : "gdfontt.h", "gdfonts.h", "gdfontmb.h", "gdfontl.h" oraz "gdfontg.h". Ciąg znaków podany jako piąty argument rysowany jest od lewej strony do prawej używając podanego koloru. Podany łańcuch powinien kończyć się znakiem zerowym (null-terminated string).

4. void gdImageString16(gdImagePtr im, gdFontPtr font, int x, int y, unsigned short *s, int color)

   Funkcja ta służy do rysowania wielu 16 bitowych znaków w obrazie. Drugim argumentem jest wskaźnik do struktury opisującej font. Pięć różnych fontów jest standardowo dostępnych w libgd : gdFontTiny, gdFontSmall, gdFontMediumBold, gdFontLarge, oraz gdFontGiant. Jeśli trzeba skorzystać z jednego z nich to należy pamiętać o dołączeniu odpowiedniego pliku nagłówkowego, są to odpowiednio : "gdfontt.h", "gdfonts.h", "gdfontmb.h", "gdfontl.h" oraz "gdfontg.h". Ciąg 16 bitowych znaków podany jako piąty argument rysowany jest od lewej strony do prawej używając podanego koloru. Podany łańcuch powinien kończyć się znakiem zerowym (null-terminated string).

   Funkcja ta została dodana w celu umożliwienia rysowania znaków z większego zbioru niż 256 znaków, dla tych którzy mają do niego dostęp. Częściej używaną funkcją jest jednak gdImageString.

5. void gdImageStringUp(gdImagePtr im, gdFontPtr font, int x, int y, unsigned char *s, int color)

   Funkcja ta służy do rysowania wielu znaków w obrazie obróconych o 90 stopni. Drugim argumentem jest wskaźnik do struktury opisującej font. Pięć różnych fontów jest standardowo dostępnych w libgd : gdFontTiny, gdFontSmall, gdFontMediumBold, gdFontLarge, oraz gdFontGiant. Jeśli trzeba skorzystać z jednego z nich to należy pamiętać o dołączeniu odpowiedniego pliku nagłówkowego, są to odpowiednio : "gdfontt.h", "gdfonts.h", "gdfontmb.h", "gdfontl.h" oraz "gdfontg.h". Ciąg znaków podany jako piąty argument rysowany jest od dołu do góry (obrócony o 90 stopni) używając podanego koloru .Podany łańcuch powinien kończyć się znakiem zerowym (null-terminated string).

6. void gdImageStringUp16(gdImagePtr im, gdFontPtr font, int x, int y, unsigned short *s, int color)

Funkcja ta służy do rysowania wielu 16 bitowych znaków w obrazie obróconych o 90 stopni. Drugim argumentem jest wskaźnik do struktury opisującej font. Pięć różnych fontów jest standardowo dostępnych w libgd : gdFontTiny, gdFontSmall, gdFontMediumBold, gdFontLarge, oraz gdFontGiant. Jeśli trzeba skorzystać z jednego z nich to należy pamiętać o dołączeniu odpowiedniego pliku nagłówkowego, są to odpowiednio : "gdfontt.h", "gdfonts.h", "gdfontmb.h", "gdfontl.h" oraz "gdfontg.h". Ciąg 16 bitowych znaków podany jako piąty argument rysowany jest od dołu do góry (obrócony o 90 stopni) używając podanego koloru . Podany łańcuch powinien kończyć się znakiem zerowym (null-terminated string).

Funkcja ta została dodana w celu umożliwienia rysowania znaków z większego zbioru niż 256 znaków, dla tych którzy mają do niego dostęp. Częściej używaną funkcją jest jednak gdImageStringUp.

7. char *gdImageStringFT(gdImagePtr im, int *brect, int fg, char *fontname, double ptsize, double angle, int x, int y, char *string)

   Funkcja służy do rysowania wielu znaków w obrazie z wykorzystaniem anty aliasingu (wygładzanie krawędzi). Tekst narysowany w ten sposób będzie się wydawać mniej kanciasty, bardziej gładki. Wykorzystywana jest w tym celu biblioteka FreeType oraz fonty TrueType (pliki .ttf oraz .ttc). Libgd nie oferuje żadnych standardowych fontów tego rodzaju, należy więc zdobyć je samodzielnie. Fontname jest wskaźnikiem do nazwy pliku z fontem TrueType. Rysowany ciąg znaków może być przeskalowany (ptsize) oraz obrócony (angle, kąt w stopniach). Kierunek obrotu jest przeciwny do ruchu wskazówek zegara, gdzie 0 stopni oznacza położenie w pozycji godziny 15, natomiast 90 stopni położenie w pozycji godziny 12.

   Funkcja wypełnia również podaną przez programistę tablicę int brect[8] czterema współrzędnymi rogów prostokąta otaczającego narysowany napis w następującej kolejności : lewy dolny róg, prawy dolny róg, prawy górny róg oraz lewy górny róg. Współrzędne są w porządku najpierw współrzędna X a po niej współrzędna Y. Punkty te są relatywne do tekstu, nie uwzględniają one obrotu. Wynika z tego, że określenie 'lewy górny' wskazuje punkt położony najbardziej z lewej strony i najwyżej gdy tekst ustawiony jest poziomo.

   Jeżeli interesuje nas jedynie uzyskanie współrzędnych obszaru ograniczającego, należy wówczas jako wskaźnik do struktury gdImage podać NULL.

   Tekst jest rysowany w kolorze wskazanym przez indeks gf. Jeśli użyjemy wartości ujemnej, wówczas anty aliasing zostanie wyłączony.

Funkcja zwróci wskaźnik zerowy (NULL) gdy operacja przebiegnie pomyślnie. W przeciwnym razie zwróci ona wskaźnik do tekstu opisującego zaistniały błąd. Można go użyć do zidentyfikowania niepowodzenia.

8. char *gdImageStringFTEx(gdImagePtr im, int *brect, int fg, char *fontname, double ptsize, double angle, int x, int y, gdFTStringExtraPtr strex)

   Funkcja ta rozszerza możliwości gdImageStringFT umożliwiając podanie większej liczby parametrów.

   Aby narysować tekst o danym odstępie między wierszami, należy ustawić zmienną flags na gdFTEX_LINESPACE, zaś linespacing na wybraną wielkość, wyrażoną jako wielokrotność wysokości liter. Wobec powyższego wartość 1.0 jest minimalną, która gwarantuje, że kolejne wiersze tekstu nie będą z sobą kolidować.

   Jeśli nie sprecyzujemy sami jaki ten odstęp ma być, zostanie przyjęta wartość domyślna która wynosi 1.05.

A.5 Funkcje obsługi koloru

1. int gdImageColorAllocate(gdImagePtr im, int r, int g, int b)

   Funkcja ta znajduje pierwszy wolny indeks tablicy kolorów w podanym obrazie, ustawia żądane wartości RGB (każda składowa może wynosić maksymalnie 255) i zwraca numer nowego koloru w palecie bądź wartość ARGB gdy rozpatrywany obraz jest w formacie truecolor. W obu przypadkach zwrócona wartość może zostać wykorzystana jako argument jednej z funkcji rysujących określając kolor. Podczas tworzenia nowego obrazu z paletą, pierwszy zaalokowany kolor staje się automatycznie kolorem tła.

   W przypadku gdy nie ma już dostępnych miejsc w palecie (wszystkie 256 kolorów zostało już zarezerwowanych), funkcja zwróci wartość – 1.Funkcja ta nie sprawdza czy podany kolor istnieje już w tablicy kolorów. Za każdym razem stara się ona zarezerwować nową pozycję. W przypadku gdy nie ma już wolnych miejsc, można znaleźć kolor odpowiadający temu, który został sprecyzowany, w tym przypadku trzeba skorzystać z funkcji : gdImageColorExact, gdImageColorClosest, gdImageColorClosestHWB oraz gdImageColorResolve.

2. int gdImageColorAllocateAlpha(gdImagePtr im, int r, int g, int b, int a)

   Funkcja ta znajduje pierwszy wolny indeks tablicy kolorów w podanym obrazie, ustawia żądane wartości RGBA (każda składowa może wynosić maksymalnie 255, zaś składowa alpha maksymalnie 127) i zwraca numer nowego koloru w palecie bądź wartość ARGB gdy rozpatrywany

obraz jest w formacie truecolor. W obu przypadkach zwrócona wartość może zostać wykorzystana jako argument jednej z funkcji rysujących określając kolor. Podczas tworzenia nowego obrazu z paletą, pierwszy zaalokowany kolor staje się automatycznie kolorem tła.

W przypadku gdy nie ma już dostępnych miejsc w palecie (wszystkie 256 kolorów zostało już zarezerwowanych), funkcja zwróci wartość –1. Funkcja ta nie sprawdza czy podany kolor istnieje już w tablicy kolorów. Za każdym razem stara się ona zarezerwować nową pozycję. W przypadku gdy nie ma już wolnych miejsc, możesz znaleźć kolor odpowiadający temu.W tym przypadku trzeba skorzystać z funkcji : gdImageColorExactAlpha, gdImageColorClosestAlpha, oraz gdImageColorResolveAlpha.

3. int gdImageColorClosest(gdImagePtr im, int r, int g, int b)

   Funkcja ta przeszukuje paletę z zaalokowanymi już kolorami w poszukiwaniu barwy najbardziej odpowiadającej tej sprecyzowanej w wywołaniu (wykorzystywany jest euklidesowski pomiar odległości w trójwymiarowej przestrzeni). Zwrócony zostaje indeks koloru RGB najbardziej odpowiadający sprecyzowanej barwie.

   Jeśli w tablicy kolorów nie zostały jeszcze zarezerwowane żadne kolory to funkcja ta zwraca wartość –1.

   Jeśli rozpatrywany obraz jest w formacie truecolor to funkcja zawsze zwraca wartość żądanego koloru.

   Funkcja ta jest szczególnie przydatna kiedy chcemy rysować na obrazie, a mamy już zaalokowaną maksymalną liczbę kolorów (gdMaxColors, 256).

4. int gdImageColorClosestAlpha(gdImagePtr im, int r, int g, int b, int a)

   Funkcja ta przeszukuje paletę z zaalokowanymi już kolorami w poszukiwaniu barwy najbardziej odpowiadającej tej sprecyzowanej w wywołaniu (wykorzystywany jest euklidesowski pomiar odległości w czterowymiarowej przestrzeni). Zwrócony zostaje indeks koloru RGBA najbardziej odpowiadający sprecyzowanej barwie.

   Jeśli w tablicy kolorów nie zostały jeszcze zarezerwowane żadne kolory to funkcja ta zwraca wartość –1.

   Jeśli rozpatrywany obraz jest w formacie truecolor to funkcja zawsze zwraca wartość żądanego koloru.

   Funkcja ta jest szczególnie przydatna kiedy chcemy rysować na obrazie, a mamy już zaalokowaną maksymalną liczbę kolorów (gdMaxColors, 256).

5. int gdImageColorClosestHWB(gdImagePtr im, int r, int g, int b)

   Funkcja ta przeszukuje paletę z zaalokowanymi już kolorami w poszukiwaniu tego, który najbardziej odpowiada swym zabarwieniem czarnym i

białym sprecyzowanemu w wywołaniu kolorze. Zwrócony zostaje indeks koloru RGB.

Jeśli w tablicy kolorów nie zostały jeszcze zarezerwowane żadne kolory to funkcja ta zwraca wartość –1.

Jeśli rozpatrywany obraz jest w formacie truecolor to funkcja zawsze zwraca wartość żądanego koloru.

Funkcja ta jest szczególnie przydatna kiedy chcemy rysować na obrazie, a mamy już zaalokowaną maksymalną liczbę kolorów (gdMaxColors, 256). Jeżeli interesuje nas odnalezienie koloru o identycznych składowych to powinniśmy użyć funkcji gdImageColorExact.

6. int gdImageColorExact(gdImagePtr im, int r, int g, int b)

   Funkcja ta przeszukuje paletę z zaalokowanymi już kolorami w poszukiwaniu barwy takiej samej jak została sprecyzowana w wywołaniu. Jeżeli żadna z pozycji nie pasuje to funkcja zwraca wartość –1. W przeciwny wypadku zwrócony zostaje indeks koloru, którego składowe RGB są identyczne z podanymi argumentami. W celu wyszukania koloru najbardziej odpowiadającego podanej barwie zobacz opis funkcji gdImageColorClosest.

   Jeśli rozpatrywany obraz jest w formacie truecolor to funkcja zawsze zwraca wartość żądanego koloru.

7. int gdImageColorResolveAlpha(gdImagePtr im, int r, int g, int b, int a)

   Funkcja ta przeszukuje paletę z zaalokowanymi już kolorami w poszukiwaniu barwy takiej samej jak została sprecyzowana w wywołaniu i zwraca numer indeksu koloru RGBA o składowych równych argumentom. Jeżeli żadna z pozycji nie okaże się identyczna to funkcja ta stara się zaalokować dany kolor w tablicy. Może się jednak okazać, że nie ma tam już wolnych miejsc, wówczas funkcja znajdzie kolor, który najbardziej odpowiada zadanemu (podobnie jak gdImageColorClosestAlpha). Wynika z tego iż funkcja ta zawsze zwraca indeks koloru.

   Jeśli rozpatrywany obraz jest w formacie truecolor to funkcja zawsze zwraca wartość żądanego koloru.

8. int gdImageColorsTotal(gdImagePtr im) (MAKRO)

   Makro to zwraca aktualnie zaalokowaną liczbę kolorów w obrazie z paletą. Dla obrazów w formacie truecolor, wartość zwracana jest nieokreślona.

9. int gdImageColorRed(gdImagePtr im, int c) (MAKRO)

   Makro to zwraca część czerwieni danego koloru w obrazie. Zakres działania obejmuje zarówno obrazy z paletą jak i obrazy w formacie truecolor.

10. int gdImageColorGreen(gdImagePtr im, int c) (MAKRO)

    Makro to zwraca część zieleni danego koloru w obrazie. Zakres działania obejmuje zarówno obrazy z paletą jak i obrazy w formacie truecolor.

11. int gdImageColorBlue(gdImagePtr im, int c) (MAKRO)

    Makro to zwraca część niebieską danego koloru w obrazie. Zakres działania obejmuje zarówno obrazy z paletą jak i obrazy w formacie truecolor.

12. int gdImageGetInterlaced(gdImagePtr im) (MAKRO)

    Makro to zwraca wartość true (1) jeśli dany obraz podlega interlacing'owi (przeplataniu), w przeciwnym wypadku zwraca ona false (0). Nie wskazane jest odczytywanie tej informacji korzystając z bezpośredniego dostępu do obrazu.

13. int gdImageGetTransparent(gdImagePtr im) (MAKRO)

    Makro to zwraca indeks koloru przezroczystości podanego obrazu. Jeśli nie ma takiego, funkcja zwróci wartość –1. Nie wskazane jest odczytywanie tej informacji korzystając z bezpośredniego dostępu do obrazu.

14. void gdImageColorDeallocate(gdImagePtr im, int color)

    Funkcja ta zaznacza podany kolor jako pozycję, która może zostać wykorzystana ponownie. gdImageColorDeallocate nie sprawdza czy pewien piksel w obrazie używa podanej barwy. Podczas kolejnego wywołania gdImageColorAllocate, to oznaczone miejsce zostanie wykorzystane na wpisanie składowych nowego koloru. Wynika z tego iż niektóre piksele rysunku mogą zmienić swoją barwę.

15. void gdImageColorTransparent(gdImagePtr im, int color)

    Funkcja ta ustawia indeks koloru przezroczystości na podaną wartość. W celu zaznaczenia, że nie chcemy korzystać z takiej barwy, należy jako argument przekazać wartość –1. Rysunki JPEG nie obsługują przezroczystości także manipulowanie tą wartością dla tego formatu jest bezcelowe.

    Indeksem tego koloru powinna być wartość zwrócona przez funkcję gdImageColorAllocate, która została wywołana bezpośrednio przez kod bądź pośrednio przy ładowaniu obrazu. Należy używać takiego koloru przezroczystości, który nawet bez zastosowania tego efektu, będzie wyglądał dobrze z całym rysunkiem. Warto mieć na uwadze, że nie wszystkie przeglądarki obsługują kolor przezroczysty.

16. void gdImageTrueColor(int red, int green, int blue) (MAKRO)

    Makro to zwraca wartość koloru w postaci RGBA, która może zostać wykorzystana jedynie do rysowania w obrazach typu truecolor. Składowe czerwona, zielona oraz niebieska mogą być z zakresu od 0 do 255.

Jeśli piszemy kod, który ma być kompatybilny zarówno z obrazami z paletą jak i obrazami w formacie truecolor to powinieneś użyć funkcji gdImageColorResolve.

17. void gdImageTrueColorAlpha(int red, int green, int blue, int alpha) (MAKRO)

    Makro to zwraca wartość koloru w postaci RGBA, która może zostać wykorzystana jedynie do rysowania w obrazach typu truecolor z wykorzystaniem kanałów przezroczystości. Składowe czerwona, zielona oraz niebieska mogą być z zakresu od 0 do 255, natomiast wartość alpha z zakresu 0 do 127 (pełna przezroczystość).

A.6 Funkcje kopiowania, zmiany rozmiaru i obrotu

1. void gdImageCopy(gdImagePtr dst, gdImagePtr src, int dstX, int dstY, int srcX, int srcY, int w, int h)

   Funkcja ta służy do kopiowania prostokątnej części jednego obrazu do drugiego.

   Argument src wskazuje obraz, z którego będziemy pobierać wycinek, natomiast dst określa obraz, do którego zostanie on wstawiony. srcX i srcY to współrzędne w obrazie źródłowym lewego górnego rogu wycinanego prostokąta. Jego szerokość i wysokość określone są przez argumenty w i h. Zdefiniowany w ten sposób region zostanie wstawiony w punkcie dstX, dstY obrazu docelowego.

   Jeżeli kopiowanie i wstawianie odbywa się w obrębie tego samego obrazu to wszystko przebiegnie pomyślnie jeśli miejsce docelowe nie jest położone w ten sposób iż oba obszary nachodzą na siebie.

2. void gdImageCopyResized(gdImagePtr dst, gdImagePtr src, int dstX, int dstY, int srcX, int srcY, int destW, int destH, int srcW, int srcH)

   Funkcja ta służy do kopiowania prostokątnej części jednego obrazu do drugiego. Rozmiary pobieranej i wstawianej części mogą być różne, co w wyniku będzie wymagało rozciągania bądź skurczenia.

   Argument src wskazuje obraz, z którego będziemy pobierać wycinek, natomiast dst określa obraz, do którego zostanie on wstawiony. srcX i srcY to współrzędne w obrazie źródłowym lewego górnego rogu wycinanego prostokąta. Jego szerokość i wysokość określone są przez argumenty srcW i srcH. Zdefiniowany w ten sposób region zostanie wstawiony w punkcie dstX, dstY obrazu docelowego, a jego rozmiary określamy za pomocą dstW oraz dstH.

Jeżeli kopiowanie i wstawianie odbywa się w obrębie tego samego obrazu to wszystko przebiegnie pomyślnie jeśli miejsce docelowe nie jest położone w ten sposób iż oba obszary nachodzą na siebie.

3. void gdImageCopyResampled(gdImagePtr dst, gdImagePtr src, int dstX, int dstY, int srcX, int srcY, int destW, int destH, int srcW, int srcH)

   Funkcja ta służy do kopiowania prostokątnej części jednego obrazu do drugiego z użyciem interpolacji pikseli, co w praktyce oznacza iż redukcja wielkości rozpatrywanego regionu nie wpłynie tak znacząco na jego wygląd, jakość. Rozmiary pobieranej i wstawianej części mogą być różne, co w wyniku będzie wymagało rozciągania bądź skurczenia.

   Wartości pikseli są interpolowane jedynie gdy podany obraz jest w formacie truecolor. Jeżeli tak nie jest to wywoływana jest automatycznie funkcje gdImageCopyResized.

   Argument src wskazuje obraz, z którego będziemy pobierać wycinek, natomiast dst określa obraz, do którego zostanie on wstawiony. srcX i srcY to współrzędne w obrazie źródłowym lewego górnego rogu wycinanego prostokąta. Jego szerokość i wysokość określone są przez argumenty srcW i srcH. Zdefiniowany w ten sposób region zostanie wstawiony w punkcie dstX, dstY obrazu docelowego, a jego rozmiary określamy za pomocą dstW oraz dstH.

   Jeżeli kopiowanie i wstawianie odbywa się w obrębie tego samego obrazu to wszystko przebiegnie pomyślnie jeśli miejsce docelowe nie jest położone w ten sposób iż oba obszary nachodzą na siebie

4. void gdImageCopyRotated(gdImagePtr dst, gdImagePtr src, double dstX, double dstY, int srcX, int srcY, int srcW, int srcH, int angle)

   Funkcja ta służy do kopiowania prostokątnej części jednego obrazu do drugiego, obróconej o podany kąt. Rozmiary pobieranej i wstawianej części mogą być różne, co w wyniku będzie wymagało rozciągania bądź skurczenia.

   Argument src wskazuje obraz, z którego będziemy pobierać wycinek, natomiast dst określa obraz, do którego zostanie on wstawiony. srcX i srcY to współrzędne w obrazie źródłowym lewego górnego rogu wycinanego prostokąta. Jego szerokość i wysokość określone są przez argumenty srcW i srcH. Punkt o współrzędnych dstX, dstY w obrazie docelowym wskazuje środkową pozycję wstawianego wycinka, którego rozmiary określamy za pomocą dstW oraz dstH. Pozycja ta może być wyrażona jako wartość typu float, ponieważ środek regionu może wypaść w połowie piksela. Argument angle to kąt, o który chcemy obrócić wycięty prostokąt. Wyrażony jest on w stopniach (wartość od 0 do 360).

Jeżeli kopiowanie i wstawianie odbywa się w obrębie tego samego obrazu to wszystko przebiegnie pomyślnie jeśli miejsce docelowe nie jest położone w ten sposób iż oba obszary nachodzą na siebie

5. void gdImageCopyMerge(gdImagePtr dst, gdImagePtr src, int dstX, int dstY, int srcX, int srcY, int w, int h, int pct)

   Funkcja ta jest niemal identyczna w porównaniu z gdImageCopy z tą różnicą, że 'zlewa' (stapia) ona z sobą dwa wycinki w oparciu o wartość podaną jako argument pct. Jeżeli wynosi on 100 to działanie jest takie samo jak w przypadku gdImageCopy (piksele źródłowe zastępują piksele w lokacji docelowej).

   Jeżeli jednak parametr ten wynosi mniej niż 100 to oba kawałki są 'zlewane' z sobą. pct równe 0 powoduje, że nie dokonywana jest żadna akcja.

   Ta opcja jest szczególnie użyteczna jeśli chcemy aby jakiś obszar został podświetlony (dla uwidocznienia). 'Zlewamy' go wówczas z prostokątnym, wypełnionym obszarem o danym kolorze, nadając pct wartość 50.

6. void gdImageCopyMergeGray(gdImagePtr dst, gdImagePtr src, int dstX, int dstY, int srcX, int srcY, int w, int h, int pct)

   Funkcja ta jest niemal identyczna do gdImageCopyMerge z tą różnicą, że zachowuje ona odcienie obrazka źródłowego, poprzez konwersje docelowych pikseli na barwy w odcieniach szarości. Dopiero wówczas następuje proces 'zlewania'.

7. void gdImagePaletteCopy(gdImagePtr dst, gdImagePtr src)

   Funkcja ta kopiuje paletę jednego obrazka do drugiego, próbując dopasować odpowiednie barwy pochodzące z obiektu źródłowego do barw w palecie docelowej.

A.7 Funkcje rozmaite

1. int gdImageCompare(gdImagePtr im1, gdImagePtr im2)

   Funkcja ta zwraca wartość, która wskazuje czy, i w jaki sposób, dwa podane rysunki się różnią. Wszystkie możliwości zdefiniowane są w pliku nagłówkowym gd.h, lecz najczęściej używaną jest GD_CMP_IMAGE, która oznacza że oba rysunki będą się różnić kiedy zostaną wyświetlone. Inne, mniej znaczące różnice dotyczą palety kolorów. Jakiekolwiek niezbieżności w indeksie koloru przezroczystego powodują już, że obrazy są różne, nawet jeśli efekt przezroczystości nie jest wykorzystywany

2. gdImageInterlace(gdImagePtr im, int interlace)

Funkcja ta służy do wskazania czy dany obraz powinien być zapisany w liniowym porządku, gdzie linie rysunku zapisane są i pojawiają się od pierwszej do ostatniej (zwykle od góry do dołu), czy też może powinien on zostać zapisany w przeplatanym porządku (interlaced fashion), w którym obraz pojawiać się będzie w kilu iteracjach. Domyślny jest brak interlacing'u. Kiedy zapisujemy obraz do pliku w formacie JPEG, interlacing implikuje wykorzystanie zapisu progresywnego (postępującego, progressive JPEG), który jest tworzony poprzez zapis kolejnych obrazów (skanów) o coraz lepszej jakości. Obrazy bez interlacing'u są natomiast zbiorem sekwencyjnych danych o rysunku.

Niezerowy argument interlace powoduje włączenie tej opcji. Jej wykorzystanie jest jedynie widoczne i sensowne gdy wynikowy obraz zapisujemy w formacie PNG lub JPEG.

Kiedy wczytujemy obraz z pliku w formacie PNG bądź JEPG, odpowiednia informacja o wartości interlacing'u zostanie także z niego odczytana.

Należy mieć na uwadze iż nie wszystkie przeglądarki obsługują interlacing, wyświetlają one jednak tak zapisane obrazy.

3. gdFree(void *ptr)

   Funkcja to umożliwia zwolnienie obszaru pamięci przydzielonego przez takie funkcje jak np. gdImagePngPtr. Funkcja ta gwarantuje, że wersja funkcji free, która jest ostatecznie wywoływana będzie odpowiednia i zgodna z wersją funkcji malloc, która zarezerwowała dany miejsce.

A.8 Stałe

1. gdBrushed

   Używana w miejscu argumentu określającego kolor w funkcjach rysujących, takich jak np. gdImageLine bądź gdImageRectangle. Kiedy użyjemy tej stałej to pędzel, ustawiony za pomocą funkcji gdImageSetBrush, będzie wykorzystany przy rysowaniu linii zamiast standardowego pojedynczego piksela (pędzel może składać się z większej ilości pikseli). Zobacz również do opisu stałe gdStyledBrushed aby poznać w jaki sposób rysować przerywane części rysunku (pędzla).

2. gdMaxColors

   Stała o wartości 256. Wyraża ona maksymalną liczbę kolorów w obrazach PNG z paletą (zgodnie ze standardem tego formatu), w obrazach gd jest to również maksymalna liczba barw.

3. gdStyled

Używana w miejscu argumentu określającego kolor w funkcjach rysujących, takich jak np. gdImageLine bądź gdImageRectangle. Kiedy użyjemy tej stałej to kolory rysowanych pikseli są zgodne z ustawionym przez funkcje gdImageSetStyle stylem. Jeżeli kolor piksela jest równy gdTransparent to nie jest on zwiększany (pozostawiany jest w tym miejscu kolor tła). Mechanizm ten nie ma nic wspólnego z indeksem koloru przezroczystego zdefiniowanego w strukturze obrazu.

4. gdStyledBrushed

   Używana w miejscu argumentu określającego kolor w funkcjach rysujących, takich jak np. gdImageLine bądź gdImageRectangle. Kiedy użyjemy tej stałej to pędzel, ustawiony za pomocą funkcji gdImageSetBrush, jest rysowany w każdym punkcie linii, pod warunkiem, że styl, ustawiony przez gdImageSetStyle, zawiera niezerową wartość w tej (bieżącej) pozycji. Piksele są rysowane kolejno zgodnie z definicją stylu i gdy zostanie osiągnięty jego koniec, zaczyna się od początku. Zauważ, że wykorzystanie tej stałej różni się od gdStyled, gdzie kolejne piksele stylu określały barwę.

5. gdDashSize

   Określa ona długość kreski w przerywanej linii. Została ona zdefiniowana aby zachować kompatybilność ze starszymi wersjami biblioteki i wynosi ona 4. Nowe programy powinny używać funkcji gdImageSetStyle i wywoływać następnie np. gdImageLine ze specjalną wartością koloru gdStyled bądź gdStyledBrushed.

6. gdTiled

   Używana w miejscu argumentu określającego kolor w funkcjach wypełniających, takich jak np. gdImageFilledRectangle, gdImageFilledPolygon, gdImageFill oraz gdImageFillToBorder. W takim przypadku wypełniany obszar będzie składać się z tzw. dachówek, którą ustawia się za pomocą funkcji gdImageSetTile. Określa ona, które z pikseli powinny zostać narysowane i w jakim kolorze. Zobacz do opisu funkcji gdImageFill i gdImageFillToBorder aby poznać dodatkowe szczegóły związane z tym tematem.

7. gdTransparent

   Używana jako wartość koloru w definiowany stylu, który później zostanie ustawiony za pomocą funkcji gdImageSetStyle. Kolor ten wskazuje położenia, które nie powinny być wypełnione żadnym kolorem, tak aby widoczne było w tym miejscu tło, na którym odbywa się rysowanie. Stała ta nie jest równa indeksowi koloru przezroczystego danego obrazu. Aby poznać funkcjonalność wspomnianego indeksu zobacz do opisu funkcji gdImageColorTransparent.

Dodatek B

Kod modułu image

```
/* modimage.c: -*- C -*-  Module which allows the use of the GD library. */

/*  Author: Brian J. Fox (bfox@ai.mit.edu) Tue Mar 30 18:44:20 1999.

    This file is part of <Meta-HTML>(tm), a system for the rapid
    deployment of Internet and Intranet applications via the use
    of the Meta-HTML language.

    Copyright (c) 1995, 2000, Brian J. Fox (bfox@ai.mit.edu).
        Copyright (c) 2005, Beata Szadurska & Mariusz Zynel.

    Meta-HTML is free software; you can redistribute it and/or modify
    it under the terms of the General Public License as published
    by the Free Software Foundation; either version 1, or (at your
    option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    FSF GPL for more details.

    You should have received a copy of the FSF General Public License
    along with this program; if you have not, you may obtain one
    electronically at the following URL:

    http://www.fsf.org/licensing/licenses/gpl.html  */

#include "modules.h"

#if defined (__cplusplus)
extern "C"
{
#endif

#include <gd.h>
#include <gdfontt.h>
#include <gdfonts.h>
#include <gdfontmb.h>
#include <gdfontl.h>

/* #include <mtables.c> */

static void pf_image_create (PFunArgs);
static void pf_image_delete (PFunArgs);
static void pf_image_write (PFunArgs);
static void pf_image_set_pixel (PFunArgs);
```

```
static void pf_image_get_pixel (PFunArgs);
static void pf_image_text (PFunArgs);
static void pf_image_text_al (PFunArgs);
static void pf_image_line (PFunArgs);
static void pf_image_fill (PFunArgs);
static void pf_image_rect (PFunArgs);
static void pf_image_arc (PFunArgs);
static void pf_image_poly (PFunArgs);
static void pf_image_copy (PFunArgs);
static void pf_image_info (PFunArgs);
static void pf_image_transparent (PFunArgs);
static void pf_image_set_brush (PFunArgs);
static void pf_image_set_tile (PFunArgs);


static PFunDesc ftab[] =
{
  /*   tag              complex? debug_level          code     */
  { "IMAGE::CREATE",     0,       0,       pf_image_create },
  { "IMAGE::DELETE",     0,       0,       pf_image_delete },
  { "IMAGE::WRITE",      0,       0,       pf_image_write },
  { "IMAGE::SET-PIXEL", 0,       0,       pf_image_set_pixel },
  { "IMAGE::GET-PIXEL", 0,       0,       pf_image_get_pixel },
  { "IMAGE::TEXT",       0,       0,       pf_image_text },
  { "IMAGE::TEXT-AL",    0,       0,       pf_image_text_al },
  { "IMAGE::LINE",       0,       0,       pf_image_line },
  { "IMAGE::FILL",       0,       0,       pf_image_fill },
  { "IMAGE::RECT",       0,       0,       pf_image_rect },
  { "IMAGE::ARC",        0,       0,       pf_image_arc },
  { "IMAGE::POLY",       0,       0,       pf_image_poly },
  { "IMAGE::COPY",       0,       0,       pf_image_copy },
  { "IMAGE::TRANSPARENT",0,       0,       pf_image_transparent },
  { "IMAGE::INFO",       0,       0,       pf_image_info },
  { "IMAGE::SET-BRUSH", 0,       0,       pf_image_set_brush },
  { "IMAGE::SET-TILE",   0,       0,       pf_image_set_tile },
  { (char *)NULL,        0,        0,             (PFunHandler *)NULL }
};


#define IMGFORMAT_GIF           0
#define IMGFORMAT_PNG           1
#define IMGFORMAT_JPEG          2


/* 3)  Insert the following statement, with the name of your module file
   as a string.  This allows Meta-HTML to install the functions in your
   module when it is loaded. */
MODULE_INITIALIZE ("modimage", ftab)

/* 4) Write a blurb about what this module does.  The text here will make
   it directly into the documentation, as a section of its own, and the
   functions that you declare with DEFUN, DEFMACRO, etc., will be documented
   in that section. */
DEFINE_SECTION (IMAGE-MODULE, image-keywords; more keywords,
"Functions which allow the creation of GIF images.
You create an image with <var IMAGE::CREATE>, and you can then draw lines,
arcs, and points into the image.  When you are done creating the image,
you call <var IMAGE::RETRIEVE> to get the GIF image in a binary variable.

Finally, when you are totally done with the image, you call <var IMAGE::DELETE>
to make the image go away.", "")

/* An IMAGE object in Meta-HTML under this GD-based library is a variable
   name which encodes the location of a data structure within our code. */
static gdImage **images = (gdImage **)NULL;
static int image_slots = 0;
```

```
/* Create and allocate an image with width X and height Y. */
static int
allocate_image (int x, int y, int truecolor)
{
  register int i;
  register int slot = -1;

  for (i = 0; i < image_slots; i++)
    if (images[i] == (gdImage *)NULL)
      {
        slot = i;
        break;
      }

  if (slot < 0)
    {
      images = (gdImage **)xrealloc
        (images, (image_slots += 10) * sizeof (gdImage *));
      slot = i;
      for (i = slot; i < image_slots; i++)
        images[i] = (gdImage *)NULL;
    }

  if (truecolor)
          images[slot] = gdImageCreateTrueColor (x, y);
  else
          images[slot] = gdImageCreate (x, y);

  return (slot);
}

static void
deallocate_image (int which)
{
  if (which < image_slots)
    if (images[which] != (gdImage *)NULL)
      {
        gdImageDestroy (images[which]);
        images[which] = (gdImage *)NULL;
      }
}

static int
dereference_image_var (char *for_whom, char *varname)
{
  int which = -1;

  if (!empty_string_p (varname))
    {
      char *value = pagefunc_get_variable (varname);

      if (!empty_string_p (value) && (integer_p (value, 10)))
        {
          which = atoi (value);

          if (which > -1)
            {
              if (which > image_slots)
                which = -1;
              else if (images[which] == (gdImage *)NULL)
                which = -1;

              if (which == -1)
                pagefunc_set_variable (varname, "(Deallocated Image)");
            }
```

```
        }
    }
  else
    page_debug ("IMAGE::%s Missing IMAGE variable", for_whom);
  return (which);
}

static int
get_image_arg_1 (char *for_whom, Package *vars, int pos)
{
  char *varname = mhtml_evaluate_string (get_positional_arg (vars, pos));
  int which = dereference_image_var (for_whom, varname);
  xfree (varname);
  return (which);
}

static int
get_image_arg (char *for_whom, Package *vars)
{
  return (get_image_arg_1 (for_whom, vars, 0));
}

static int
float_val (char *string)
{
  double first = strtod (string, (char **)NULL);
  return (first);
}

static int
integer_val (char *string)
{
  return ((int) float_val(string));
}

DEFUNX(image::create, imagevar &key width height src format,
"Creates a new image with the specified width and height, and makes
<var imagevar> be a receptacle for that image.

If you pass <var src=/www/docs/images/foo.gif>, then <var foo.gif> will
be loaded into the image variable instead of an empty image.

<var format> specifies format of the created or loaded image. Recognized
are GIF, PNG and JPEG (or JPG). It defaults to GIF.

If PNG image is created and <var truecolor=true> is specifies, then
true color palette is used. Otherwise, 8 bit per pixel palette is used.
<var truecolor> is ignored for GIF and JPEG.")

static void
pf_image_create (PFunArgs)
{
  char *width_arg = mhtml_evaluate_string (get_value (vars, "WIDTH"));
  char *height_arg = mhtml_evaluate_string (get_value (vars, "HEIGHT"));
  char *varname = mhtml_evaluate_string (get_positional_arg (vars, 0));
  char *srcfile = mhtml_evaluate_string (get_value (vars, "SRC"));
  char *format_arg = mhtml_evaluate_string (get_value (vars, "FORMAT"));
  char *truecolor_arg = mhtml_evaluate_string (get_value (vars, "TRUECOLOR"));
  int width = 400, height = 300;
  int slot;
  int format;
  char *result = (char *)NULL;

  if (!empty_string_p (format_arg))
    {
        if (strcasecmp (format_arg, "PNG") == 0)
```

```
                format = IMGFORMAT_PNG;
        else if (strcasecmp (format_arg, "JPEG") == 0)
                format = IMGFORMAT_JPEG;
        else if (strcasecmp (format_arg, "JPG") == 0)
                format = IMGFORMAT_JPEG;
        else
                format = IMGFORMAT_GIF;
  }
else
  format = IMGFORMAT_GIF;


if (!empty_string_p (varname))
  {
    if (!empty_string_p (srcfile))
      {
        FILE *stream = fopen (srcfile, "r");
        gdImage *image = (gdImage *)NULL;

        if (stream != (FILE *)NULL)
          {
                if (format == IMGFORMAT_JPEG)
                        image = gdImageCreateFromJpeg (stream);
                else if (format == IMGFORMAT_PNG)
                        image = gdImageCreateFromPng (stream);
                else
                        image = gdImageCreateFromGif (stream);
                fclose (stream);
          }

        if (image == (gdImage *)NULL)
          {
            page_debug ("IMAGE::CREATE Couldn't load %s: %s",
                        srcfile, strerror (errno));
            return;
          }

        slot = allocate_image (1, 1, 0);
        gdImageDestroy (images[slot]);
        images[slot] = image;
      }
    else
      {
        if (!empty_string_p (width_arg))
          width = integer_val (width_arg);

        if (!empty_string_p (height_arg))
          height = integer_val (height_arg);

        slot = allocate_image (width, height,
                    (!empty_string_p (truecolor_arg) && format == IMGFORMAT_PNG)
                            || format == IMGFORMAT_JPEG);
      }

    mhtml_set_numeric_variable (varname, slot);
    result = pagefunc_get_variable (varname);
  }
else
  page_debug ("IMAGE::CREATE Missing IMAGE variable");

xfree (varname);
xfree (width_arg);
xfree (height_arg);
xfree (srcfile);
xfree (format_arg);
xfree (truecolor_arg);
```

```
#if 0
  if (result != (char *)NULL)
    {
      bprintf_insert (page, start, "%s", result);
      *newstart += strlen (result);
    }
#endif
}

DEFUNX (image::info, imagevar,
"Returns an alist representing information about the image in <var imagevar>.
The alist contains <var width>, <var height>, <var total-colors>, and
<var colors>. <var colors> is an array of the color values which appear in
the image.")

#define alist_set(name, num)                                          \
   do                                                                 \
     {                                                                \
       sprintf (numbuff, "%d", num);                                  \
       forms_set_tag_value_in_package (p, name, numbuff);             \
     } while (0)

static void
pf_image_info (PFunArgs)
{
  int slot = get_image_arg ("INFO", vars);
  char *result = (char *)NULL;

  if (slot >= 0)
    {
      register int i;
      gdImage *image = images[slot];
      Package *p = symbol_get_package ((char *)NULL);
      Symbol *colors = symbol_intern_in_package (p, "colors");
      char numbuff[20];

      alist_set ("width", image->sx);
      alist_set ("height", image->sy);
      alist_set ("total-colors", image->colorsTotal);
      if (image->transparent > -1)
        alist_set ("transparent", image->transparent);

      for (i = 0; i < gdImageColorsTotal (image); i++)
        {
          sprintf (numbuff, "#%02X%02X%02X ",
                   gdImageRed (image, i),
                   gdImageGreen (image, i),
                   gdImageBlue (image, i));
          symbol_add_value (colors, numbuff);
        }

      result = package_to_alist (p, 0);
    }

  if (result != (char *)NULL)
    {
      bprintf_insert (page, start, "%s", result);
      *newstart += strlen (result);
      free (result);
    }
}

DEFUNX (image::delete, imagevar,
"Reclaims any space that is currently being used by <var imagevar>.")
```

```
static void
pf_image_delete (PFunArgs)
{
  int slot = get_image_arg ("DELETE", vars);

  if (slot > -1)
    {
      deallocate_image (slot);
      slot = get_image_arg ("DEALLOCATION FORCE", vars);
    }
}

static int
hex_value (int c)
{
  if (islower (c)) c = toupper (c);
  if (strchr ("0123456789ABCDEF", c) != (char *)NULL)
    {
      c = c - '0';
      if (c > 9)
        c = 10 + ((c + '0') - 'A');
    }
  else
    c = -1;

  return (c);
}

static int
parse_rgb (char *rgb, int *r, int *g, int *b)
{
  register int result = -1;

  *r = -1; *g = -1; *b = -1;

  if (rgb != (char *)NULL)
    {
      while (whitespace (*rgb)) rgb++;
      while (*rgb == '#') rgb++;

      if (strchr (rgb, ',') != (char *)NULL)
        {
          /* This is RGB in decimal, values separated by commas. */
          sscanf (rgb, "%d,%d,%d", r, g, b);
        }
      else
        {
          register int i;
          /* This is RGB in hex. */

          for (i = 0; i < 6 && rgb[i]; i++)
            {
              switch (i)
                {
                case 0: *r = 16 * hex_value (rgb[i]); break;
                case 1: *r |= hex_value (rgb[i]); break;
                case 2: *g = 16 * hex_value (rgb[i]); break;
                case 3: *g |= hex_value (rgb[i]); break;
                case 4: *b = 16 * hex_value (rgb[i]); break;
                case 5: *b |= hex_value (rgb[i]); break;

                default:
                  break;
                }
            }
        }
```

```
    }

  if ((*r < 0) || (*g < 0) || (*b < 0))
    result = -1;
  else
    result = 0;

  return (result);
}

static int
rgb_to_index (gdImage *image, char *rgb)
{
  int r, g, b, pixel = -1;

  if (parse_rgb (rgb, &r, &g, &b) < 0)
    pixel = -1;
  else
    {
      pixel = gdImageColorExact (image, r, g, b);

      if (pixel == -1)
        pixel = gdImageColorAllocate (image, r, g, b);

      if (pixel == -1)
        pixel = gdImageColorClosest (image, r, g, b);
    }

  return (pixel);
}

static int
rgb_to_truecolor (char *rgb)
{
  int r, g, b;

  if (parse_rgb (rgb, &r, &g, &b) == 0)
        return gdTrueColor (r, g, b);
  else
        return -1;
}

static int
rgb_to_color (gdImage *image, char *rgb)
{

  if (strcasecmp (rgb, "BRUSHED") == 0)
    return gdBrushed;

  if (strcasecmp (rgb, "TILED") == 0)
        return gdTiled;

  if (image->trueColor)
    return rgb_to_truecolor (rgb);
  else
        return rgb_to_index (image, rgb);
}


DEFUNX (image::set-pixel, imagevar &key x y color,
"Set the pixel in <var imagevar> at location <var x>, <var y> to the color
<var color>.")
static void
pf_image_set_pixel (PFunArgs)
{
  int slot = get_image_arg ("SET-PIXEL", vars);
```

```
  if (slot > -1)
    {
      gdImage *image = images[slot];
      char *x_arg = mhtml_evaluate_string (get_value (vars, "X"));
      char *y_arg = mhtml_evaluate_string (get_value (vars, "Y"));
      char *c_arg = mhtml_evaluate_string (get_value (vars, "COLOR"));

      if ((!empty_string_p (x_arg) && number_p (x_arg)) &&
          (!empty_string_p (y_arg) && number_p (y_arg)) &&
          (!empty_string_p (c_arg)))
        {
          int x = integer_val (x_arg);
          int y = integer_val (y_arg);
          int c = rgb_to_color (image, c_arg);

          gdImageSetPixel (image, x, y, c);
        }
      else
        page_debug ("<image::set-pixel ..> requires all of X, Y, and COLOR");

      xfree (x_arg);
      xfree (y_arg);
      xfree (c_arg);
    }
}

DEFUNX (image::get-pixel, imagevar &key x y,
"Returns the color of the pixel at <var x>,<var y> in <var imagevar>")

static void
pf_image_get_pixel (PFunArgs)
{
  int slot = get_image_arg ("GET-PIXEL", vars);
  char *result = (char *)NULL;
  char buffer[20];

  if (slot > -1)
    {
      gdImage *image = images[slot];
      char *x_arg = mhtml_evaluate_string (get_value (vars, "X"));
      char *y_arg = mhtml_evaluate_string (get_value (vars, "Y"));

      if ((!empty_string_p (x_arg) && number_p (x_arg)) &&
          (!empty_string_p (y_arg) && number_p (y_arg)))
        {
          int x = integer_val (x_arg);
          int y = integer_val (y_arg);
          int c = gdImageGetPixel (image, x, y);
          if (c > -1)
            {
                  if (image->trueColor)
                  sprintf (buffer,"#%06X", c);
                  else
                      sprintf (buffer,"#%02X%02X%02X",
                       image->red[c], image->green[c], image->blue[c]);
              result = buffer;
            }
        }
      else
        page_debug ("<image::get-pixel ..> requires both X and Y");

      xfree (x_arg);
      xfree (y_arg);
    }
```

```
  if (result != (char *)NULL)
    {
      bprintf_insert (page, start, "%s", result);
      *newstart += strlen (result);
    }
}

DEFUNX (image::text, imagevar text &key x y color size align report,
"Write text on the image in <var imagevar> at position <var x>, <var y>,
in the color <var color>.
<var align> can be one of \"right\", \"center\", or \"left\", and defaults
to \"center\".
<var size> ranges frome 1 to 6 and defaults to 3.

If <var report=true> is specified, the text isn't drawn, but all of the
information about the drawing is returned as an alist.")

static void
pf_image_text (PFunArgs)
{
  int slot = get_image_arg ("TEXT", vars);

  if (slot > -1)
    {
      char *text = mhtml_evaluate_string (get_positional_arg (vars, 1));
      char *report_arg = mhtml_evaluate_string (get_value (vars, "REPORT"));
      Package *info = (Package *)NULL;

      if (!empty_string_p (report_arg))
        {
          info = symbol_get_package ((char *)NULL);
          symbol_intern_in_package (info, "left");
          symbol_intern_in_package (info, "right");
          symbol_intern_in_package (info, "height");
          symbol_intern_in_package (info, "width");
        }
      xfree (report_arg);

      if (!empty_string_p (text))
        {
          char *x_arg = mhtml_evaluate_string (get_value (vars, "X"));
          char *y_arg = mhtml_evaluate_string (get_value (vars, "Y"));
          char *c_arg = mhtml_evaluate_string (get_value (vars, "COLOR"));
          char *size_arg = mhtml_evaluate_string (get_value (vars, "SIZE"));
          char *align_arg = mhtml_evaluate_string (get_value (vars, "ALIGN"));
          gdFontPtr font;
          int x, y, c, size;
          char *temp_color;

          if (!empty_string_p (x_arg) && number_p (x_arg))
            x = integer_val (x_arg);
          else
            x = gdImageSX (images[slot]) / 2;

          if (!empty_string_p (y_arg) && number_p (y_arg))
            y = integer_val (y_arg);
          else
            y = gdImageSY (images[slot]) / 2;

          if (!empty_string_p (c_arg))
            temp_color = c_arg;
          else
            temp_color = "000000";

      c = rgb_to_color (images[slot], temp_color);
```

```
if (!empty_string_p (size_arg) && integer_p (size_arg, 10))
  size = atoi (size_arg);
else
  size = 3;

switch (size)
  {
  case 1:
    font = gdFontTiny;
    break;

  case 2:
    font = gdFontSmall;
    break;

  case 3:
    font = gdFontMediumBold;
    break;

  case 4:
    font = gdFontLarge;

  default:
    font = gdFontMediumBold;
  }

/* Finally, handle the alignment. */
{
  char *align = "center";
  int char_width = font->w;
  int char_height = font->h;
  int len = strlen (text);

  /* Adjust Y coordinate to be the center of the character. */
  y -= char_height / 2;

  if (!empty_string_p (align_arg))
    align = align_arg;

  if (strcasecmp (align, "left") == 0)
    {
      /* When we're aligned left, do nothing special. */
    }
  else if (strcasecmp (align, "right") == 0)
    {
      /* Make the last character of the string end on X. */
      x -= char_width * len;
    }
  else if (strcasecmp (align, "center") == 0)
    {
      /* Make the center character of the string land on X. */
      x -= ((char_width * len) / 2);
    }

  if (info != (Package *)NULL)
    {
      mhtml_set_numeric_variable_in_package
        (info, "width", (len * char_width));

      mhtml_set_numeric_variable_in_package
        (info, "char-width", char_width);

      mhtml_set_numeric_variable_in_package
        (info, "height", char_height);

      mhtml_set_numeric_variable_in_package
```

```
                (info, "char-height", char_height);

              mhtml_set_numeric_variable_in_package (info, "left", x);
              mhtml_set_numeric_variable_in_package
                (info, "right", x + (len * char_width));
            }
        }

        if (info == (Package *)NULL)
          {
            /* Okay, draw the damn string already! */
            gdImageString (images[slot], font, x, y, text, c);
          }
        else
          {
            char *result = package_to_alist (info, 0);
            int len = (strlen (result));
            symbol_destroy_package (info);

            bprintf_insert (page, start, "%s", result);
            *newstart += len;
          }

        xfree (x_arg);
        xfree (y_arg);
        xfree (c_arg);
        xfree (size_arg);
        xfree (align_arg);
      }

    xfree (text);
  }
}


DEFUNX (image::text_al, imagevar text &key x y color size angle fontname,
"Write anti-aliased text on the image in <var imagevar> at position <var x>,
<var y>,in the color <var color> using <var fontname> font.
<var angle> specifies an angle to ratate the text. 0 means no rotation,
rotation is counter-clockwise. <var size> is the size of text is pt, can
be a floating-point value.")

static void
pf_image_text_al (PFunArgs)
{
  int slot = get_image_arg ("TEXT-AL", vars);

  if (slot > -1)
    {
      char *text = mhtml_evaluate_string (get_positional_arg (vars, 1));

      if (!empty_string_p (text))
        {
          char *x_arg = mhtml_evaluate_string (get_value (vars, "X"));
          char *y_arg = mhtml_evaluate_string (get_value (vars, "Y"));
          char *c_arg = mhtml_evaluate_string (get_value (vars, "COLOR"));
          char *size_arg = mhtml_evaluate_string (get_value (vars, "SIZE"));
          char *angle_arg = mhtml_evaluate_string (get_value (vars, "ANGLE"));
      char *fontname_arg = mhtml_evaluate_string (get_value (vars, "FONTNAME"));

          gdFontPtr font;
          int x, y, c;
          double size, angle;
          char *temp_color;

          if (!empty_string_p (x_arg) && number_p (x_arg))
```

```
            x = integer_val (x_arg);
          else
            x = gdImageSX (images[slot]) / 2;

          if (!empty_string_p (y_arg) && number_p (y_arg))
            y = integer_val (y_arg);
          else
            y = gdImageSY (images[slot]) / 2;

          if (!empty_string_p (c_arg))
            temp_color = c_arg;
          else
            temp_color = "000000";

      c = rgb_to_color (images[slot], temp_color);

          if (!empty_string_p (size_arg) && number_p (size_arg))
            size = float_val(size_arg);
          else
            size = 12;

          if (!empty_string_p (angle_arg) && number_p (angle_arg))
            angle = float_val(angle_arg);
          else
            angle = 0;

      gdImageStringFT (images[slot], NULL, c, fontname_arg, size, angle, x, y, text);

          xfree (x_arg);
          xfree (y_arg);
          xfree (c_arg);
          xfree (size_arg);
          xfree (angle_arg);
          xfree (fontname_arg);
        }

      xfree (text);
    }
}


DEFUNX (image::line, imagevar &key x1 y1 x2 y2 color brush,
"Draw a line in <var imagevar> from (<var x1>, <var y1>) to
(<var x2>, <var y2>) in the color <var color>.
<var brush>, if supplied, is another image created with
<funref image-module image::create>, that will be used as the brush to
paint the line.  In that case, the <var color> argument is ignored.")
static void
pf_image_line (PFunArgs)
{
  int slot = get_image_arg ("LINE", vars);

  if (slot > -1)
    {
      gdImage *image = images[slot];
      char *x1_arg = mhtml_evaluate_string (get_value (vars, "X1"));
      char *y1_arg = mhtml_evaluate_string (get_value (vars, "Y1"));
      char *x2_arg = mhtml_evaluate_string (get_value (vars, "X2"));
      char *y2_arg = mhtml_evaluate_string (get_value (vars, "Y2"));
      char *c_arg = mhtml_evaluate_string (get_value (vars, "COLOR"));
      char *brush_arg = mhtml_evaluate_string (get_value (vars, "BRUSH"));

      if ((!empty_string_p (x1_arg) && number_p (x1_arg)) &&
          (!empty_string_p (x2_arg) && number_p (x2_arg)) &&
          (!empty_string_p (y1_arg) && number_p (y1_arg)) &&
          (!empty_string_p (y2_arg) && number_p (y2_arg)))
```

```
        {
          int x1 = integer_val (x1_arg);
          int y1 = integer_val (y1_arg);
          int x2 = integer_val (x2_arg);
          int y2 = integer_val (y2_arg);
          int c = 0;
          char *temp_color;

          if (!empty_string_p (c_arg))
            temp_color = c_arg;
          else
            temp_color = "000000";

          c = rgb_to_color (image, temp_color);

          if (!empty_string_p (brush_arg))
            {
              int brush_slot = dereference_image_var ("LINE", brush_arg);
              gdImage *brush = brush_slot > -1 ? images[brush_slot] : NULL;

              if (brush != (gdImage *)NULL)
                gdImageSetBrush (image, brush);
            }

          gdImageLine (image, x1, y1, x2, y2, c);

          if (!empty_string_p (brush_arg))
            gdImageSetBrush (image, (gdImage *)NULL);
        }
      else
        page_debug
          ("<image::line ..> requires all of X1, Y1, X2, Y2 and COLOR");

      xfree (x1_arg);
      xfree (x2_arg);
      xfree (y1_arg);
      xfree (y2_arg);
      xfree (c_arg);
      xfree (brush_arg);
    }
}

DEFUNX (image::fill, imagevar &key x y color border,
"Fill an area of the image in <var imagevar> with the color specified by
<var color>. The filling starts at the point specified by (<var x>, <var y>),
and continues in all directions bounded by pixels which are not the same
color as the color at (<var x>, <var y>), or, optionally, which are not the
same color as the color specified by <var border>.")

static void
pf_image_fill (PFunArgs)
{
  int slot = get_image_arg ("FILL", vars);

  if (slot > -1)
    {
      gdImage *image = images[slot];
      char *x_arg = mhtml_evaluate_string (get_value (vars, "X"));
      char *y_arg = mhtml_evaluate_string (get_value (vars, "Y"));
      char *c_arg = mhtml_evaluate_string (get_value (vars, "COLOR"));
      char *b_arg = mhtml_evaluate_string (get_value (vars, "BORDER"));

      if ((!empty_string_p (x_arg) && number_p (x_arg)) &&
          (!empty_string_p (y_arg) && number_p (y_arg)) &&
          (!empty_string_p (c_arg)))
        {
```

```
          int x = integer_val (x_arg);
          int y = integer_val (y_arg);
          int c = rgb_to_color (image, c_arg);

          if (!empty_string_p (b_arg))
            {
              int b = rgb_to_color (image, b_arg);

              gdImageFillToBorder (image, x, y, b, c);
            }
          else
            {
              gdImageFill (image, x, y, c);
            }
        }
      else
        page_debug ("<image::fill ..> requires all of X, Y, and COLOR");

      xfree (x_arg);
      xfree (y_arg);
      xfree (c_arg);
      xfree (b_arg);
    }
}

DEFUNX (image::rect, imagevar &key x1 y1 x2 y2 color fill,
"Draw a rectangle with the border lines in the color <var color> and perhaps
filled with the color <var fill>. The rectangle is drawn with the upper-left
corner specified by (<var x1> <var y1>) and the bottom-right corner specified
by (<var x2>, <var y2>).")

static void
pf_image_rect (PFunArgs)
{
  int slot = get_image_arg ("RECT", vars);

  if (slot > -1)
    {
      gdImage *image = images[slot];
      char *x1_arg = mhtml_evaluate_string (get_value (vars, "X1"));
      char *y1_arg = mhtml_evaluate_string (get_value (vars, "Y1"));
      char *x2_arg = mhtml_evaluate_string (get_value (vars, "X2"));
      char *y2_arg = mhtml_evaluate_string (get_value (vars, "Y2"));
      char *c_arg = mhtml_evaluate_string (get_value (vars, "color"));
      char *filled_arg = mhtml_evaluate_string (get_value (vars, "fill"));
      int filled = !empty_string_p (filled_arg);
      int fill_color = -1;

      if ((!empty_string_p (x1_arg) && number_p (x1_arg)) &&
          (!empty_string_p (x2_arg) && number_p (x2_arg)) &&
          (!empty_string_p (y1_arg) && number_p (y1_arg)) &&
          (!empty_string_p (y2_arg) && number_p (y2_arg)) &&
          (!empty_string_p (c_arg)))
        {
          int x1 = integer_val (x1_arg);
          int y1 = integer_val (y1_arg);
          int x2 = integer_val (x2_arg);
          int y2 = integer_val (y2_arg);
          int c = -1;

          c = rgb_to_color (image, c_arg);

          if (filled)
            {
              fill_color = rgb_to_color (image, filled_arg);
              if (fill_color == -1)
```

```
                fill_color = c;

              gdImageFilledRectangle (image, x1, y1, x2, y2, fill_color);
            }

            gdImageRectangle (image, x1, y1, x2, y2, c);
        }
      else
        page_debug
          ("<image::rect ..> requires all of X1, Y1, X2, Y2, and COLOR");

      xfree (x1_arg);
      xfree (x2_arg);
      xfree (y1_arg);
      xfree (y2_arg);
      xfree (c_arg);
      xfree (filled_arg);
    }
}

DEFUNX (image::arc, imagevar &key x y width height start end color fill,
"Draws a partial ellipse centered at the point specified by <var x> and
<var y>, with a width of <var width> and height of <var height>.

The arguments of <var start> and <var end> are given in degrees, and specify
the starting and ending points on the curve.

The following code draws a red circle with a radius of 50 pixels where
the exact center of the circle appears at 100,100:
<example>
<image::arc image x=100 y=100 width=50 height=50 start=0 end=360 color=FF0000>
</example>")

static void
pf_image_arc (PFunArgs)
{
  int slot = get_image_arg ("ARC", vars);

  if (slot > -1)
    {
      gdImage *image = images[slot];
      char *x_arg = mhtml_evaluate_string (get_value (vars, "X"));
      char *y_arg = mhtml_evaluate_string (get_value (vars, "Y"));
      char *w_arg = mhtml_evaluate_string (get_value (vars, "WIDTH"));
      char *h_arg = mhtml_evaluate_string (get_value (vars, "HEIGHT"));
      char *s_arg = mhtml_evaluate_string (get_value (vars, "START"));
      char *e_arg = mhtml_evaluate_string (get_value (vars, "END"));
      char *c_arg = mhtml_evaluate_string (get_value (vars, "COLOR"));
      char *f_arg = mhtml_evaluate_string (get_value (vars, "FILL"));
      int filled = !empty_string_p (f_arg);

      if ((!empty_string_p (x_arg) && number_p (x_arg)) &&
          (!empty_string_p (y_arg) && number_p (y_arg)) &&
          (!empty_string_p (w_arg) && number_p (w_arg)) &&
          (!empty_string_p (h_arg) && number_p (h_arg)) &&
          (!empty_string_p (s_arg) && number_p (s_arg)) &&
          (!empty_string_p (e_arg) && number_p (e_arg)) &&
          (!empty_string_p (c_arg)))
        {
          int x = integer_val (x_arg);
          int y = integer_val (y_arg);
          int w = integer_val (w_arg);
          int h = integer_val (h_arg);
          int s = 270 + integer_val (s_arg);
          int e = 270 + integer_val (e_arg);
          int c = -1;
```

```
          int f = gdArc;

          c = rgb_to_color (image, c_arg);

          if (debug_level > 5)
            page_debug ("<IMAGE::ARC x=%d y=%d w=%d h=%d s=%d e=%d c=%d>",
                        x, y, w, h, s, e, c);
          if (filled)
                {
                        char *token;
                        char *fill = f_arg;

                        while (token = strtok(fill, ",")) {
                                if (strcasecmp (token, "CHORD") == 0)
                                        f = f | gdChord;
                                if (strcasecmp (token, "NOFILL") == 0)
                                        f = f | gdNoFill;
                                if (strcasecmp (token, "EDGED") == 0)
                                        f = f | gdEdged;
                                fill = NULL;
                        }
                        gdImageFilledArc (image, x, y, w, h, s, e, c, f);
                }
          else
                        gdImageArc (image, x, y, w, h, s, e, c);

        }
      else
        page_debug
          ("<image::arc ..> requires all of X, Y, WIDTH, HEIGHT, START, END, and COLOR");

      xfree (x_arg);
      xfree (y_arg);
      xfree (w_arg);
      xfree (h_arg);
      xfree (s_arg);
      xfree (e_arg);
      xfree (c_arg);
      xfree (f_arg);
    }
}

DEFUNX (image::write, imagevar filename,
"Writes the contents of the image in <var imagevar> to the file specified
by <var filename>.  Returns \"true\" if the image was successfully written.")
static void
pf_image_write (PFunArgs)
{
  int slot = get_image_arg ("WRITE", vars);
  char *result = (char *)NULL;

  if (slot != -1)
    {
      char *output = mhtml_evaluate_string (get_positional_arg (vars, 1));

      if (!empty_string_p (output))
        {
          FILE *stream = fopen (output, "wb");

          if (stream != (FILE *)NULL)
            {
              gdImageGif (images[slot], stream);
              fclose (stream);
              result = "true";
            }
          else
```

```
            page_debug ("IMAGE::WRITE Couldn't open %s: %s",
                        output, strerror (errno));
          }
        else
          page_debug ("IMAGE::WRITE requires a filename to write the image to");

        xfree (output);
      }
    if (result != (char *)NULL)
      bprintf_insert (page, start, "%s", result);
}

DEFUNX (image::transparent, imagevar &key color,
"Makes <var color> be the transparent one for this image.")

static void
pf_image_transparent (PFunArgs)
{
  int slot = get_image_arg ("SET-TRANSPARENT", vars);

  if (slot > -1)
    {
      gdImage *image = images[slot];
      char *c_arg = mhtml_evaluate_string (get_value (vars, "COLOR"));

      if (!empty_string_p (c_arg))
        {
          int c = -1;

          /* Convert the color into an index.  If this image doesn't already
             have this color, add it now. */
          c = rgb_to_index (image, c_arg);

          if (c != -1)
            gdImageColorTransparent (image, c);
        }
    }
}

static void
pf_image_poly (PFunArgs)
{
}

DEFUNX (image::copy, src-image dst-image &key src-x src-y dst-x dst-y
  src-width src-height dst-width dst-height,
"Copies bits from <var src-image> to <var dst-image>.
Both images must exist.
If a different width or height is specified for the destination, the image is
resized to fit the specified values.")
static void
pf_image_copy (PFunArgs)
{
  int source_slot = get_image_arg_1 ("COPY", vars, 0);
  int dest_slot = get_image_arg_1 ("COPY", vars, 1);

  if ((source_slot != -1) && (dest_slot != -1))
    {
      gdImage *source = images[source_slot];
      gdImage *dest = images[dest_slot];
      char *srcx_arg = mhtml_evaluate_string
        (get_one_of (vars, "SOURCE-X", "SRC-X", (char *)NULL));
      char *srcy_arg = mhtml_evaluate_string
        (get_one_of (vars, "SOURCE-Y", "SRC-Y", (char *)NULL));
      char *dstx_arg = mhtml_evaluate_string
        (get_one_of (vars, "DEST-X", "DST-X", (char *)NULL));
```

```
      char *dsty_arg = mhtml_evaluate_string
        (get_one_of (vars, "DEST-Y", "DST-Y", (char *)NULL));
      char *srcw_arg = mhtml_evaluate_string
        (get_one_of (vars, "SOURCE-WIDTH", "SRC-W", "SRC-WIDTH", "WIDTH",
                     (char *)NULL));
      char *srch_arg = mhtml_evaluate_string
        (get_one_of (vars, "SOURCE-HEIGHT", "SRC-H", "SRC-HEIGHT", "HEIGHT",
                     (char *)NULL));
      char *dstw_arg = mhtml_evaluate_string
        (get_one_of (vars, "DEST-WIDTH", "DST-W", "DST-WIDTH", (char *)NULL));
      char *dsth_arg = mhtml_evaluate_string
        (get_one_of (vars, "DEST-HEIGHT", "DST-H", "DST-HEIGHT",
                     (char *)NULL));
      int src_x, src_y, dst_x, dst_y, src_w, dst_w, src_h, dst_h;

      if (!empty_string_p (srcx_arg) && (number_p (srcx_arg)))
        src_x = integer_val (srcx_arg);
      else
        src_x = 0;

      if (!empty_string_p (dstx_arg) && (number_p (dstx_arg)))
        dst_x = integer_val (dstx_arg);
      else
        dst_x = src_x;

      if (!empty_string_p (srcy_arg) && (number_p (srcy_arg)))
        src_y = integer_val (srcy_arg);
      else
        src_y = 0;

      if (!empty_string_p (dsty_arg) && (number_p (dsty_arg)))
        dst_y = integer_val (dsty_arg);
      else
        dst_y = src_y;

      if (!empty_string_p (srcw_arg) && (number_p (srcw_arg)))
        src_w = integer_val (srcw_arg);
      else
        src_w = gdImageSX (source);

      if (!empty_string_p (dstw_arg) && (number_p (dstw_arg)))
        dst_w = integer_val (dstw_arg);
      else
        dst_w = gdImageSX (dest);

      if (!empty_string_p (srch_arg) && (number_p (srch_arg)))
        src_h = integer_val (srch_arg);
      else
        src_h = gdImageSY (source);

      if (!empty_string_p (dsth_arg) && (number_p (dsth_arg)))
        dst_h = integer_val (dsth_arg);
      else
        dst_h = gdImageSY (dest);

      gdImageCopyResized (dest, source,
                          dst_x, dst_y, src_x, src_y,
                          dst_w, dst_h, src_w, src_h);
      xfree (srcx_arg); xfree (srcy_arg);
      xfree (srcw_arg); xfree (srch_arg);
      xfree (dstx_arg); xfree (dsty_arg);
      xfree (dstw_arg); xfree (dsth_arg);
    }
}

DEFUNX (image::set-brush, imagevar brush,
```

```
"Sets brush to be used in painting functions, e.g. IMAGE::LINE.")

static void
pf_image_set_brush (PFunArgs)
{
  int slot = get_image_arg_1 ("SET-BRUSH", vars, 0);
  int brush = get_image_arg_1 ("SET-BRUSH", vars, 1);

  if (slot > -1 && brush > -1)
    gdImageSetBrush(slot, brush);
}


DEFUNX (image::set-tile, imagevar tile,
"Sets tile to be used in IMAGE::FILL.")

static void
pf_image_set_tile (PFunArgs)
{
  int slot = get_image_arg_1 ("SET-TILE", vars, 0);
  int tile = get_image_arg_1 ("SET-TILE", vars, 1);

  if (slot > -1 && tile > -1)
    gdImageSetTile(slot, tile);
}


#if defined (__cplusplus)
}
#endif
```

Bibliografia

[1] Rafe Colburn ,*CGI*, Helion,1998.

[2] `http://www.boutell.com/gd/` — strona domowa projektu *GD Graphics Library*.

[3] `http://metahtml.sourceforge.net/` — strona domowa projektu MetaHTML.

[4] `http://cloanto.com/users/mcb/19950127giflzw.html` — The GIF Controversy: A Software Developer's Perspective.

[5] `http://www.unisys.com/about__unisys/lzw` — LZW Patent Information.