بسمه تعالی

پروژه درس پردازش تصویر:

مقایسه انواع قالبهای فایلهای گرافیکی

نام و نام خانوادگی: علی بهلولی زفره

استاد درس : دکتر ترکمنی‌آذر

تابستان ۱۳۸۱

مقدمه:

ذخیره‌سازی و نگهداری تصاویر در کامپیوترها راه‌های مختلفی دارد که باعث ایجاد قالبهای مختلف برای فایلهای گرافیکی شده است.دو هدف عمده و متضاد که در ذخیره‌سازی فایلهای گرافیکی باید مورد نظر قرار گیرد عبارتند از:

۱- داشتن کیفیت خوب(از دست ندادن کیفیت)

۲- حجم فایل تا حد ممکن کوچک باشد.

برای رسیدن به هدف دوم ، با توجه به الگوریتم های مختلف فشرده‌ساری قالبهای متفاوتی بوجود آمد. اگر از الگوریتمهای فشرده سازی استفاده شود که باعث از دست ندادن دیتا شوند (نظیر RLE,LZW , Huffman) کیفیت تصویر اصلا تغییری نمی‌کند ولی درصد فشردگی در این مواقع کم است. در صورتی که هدف اول مهمتر باشد ، این روشها توصیه می‌شوند.

گاهی اوقات ممکن است از دست دادن مقداری از کیفیت برای ما مهم نباشد و مهمتر حجم کم فایل باشد ، مثلا برای انتقال تصاویر. در این مواقع با توجه به تکنیکهایی که موجود است (مثلا استفاده از تبدیلات فوریه) ابتدا مقداری از جزییات تصویر کاسته می‌شود سپس با استفاده از الگوریتمهای فشرده سازی تصویر حاصل فشرده سازی می‌شود. در این روشها حجم فایل به اندازه قابل ملاحظه‌ای کاهش می‌یابد.

در این تحقیق ابتدا به شرح پارامترهایی که در نگهداری تصویر دخالت دارند پرداخته می‌شود و در ادامه به توضیح قالبهای مختلف و در انتها برنامه‌ای که به زبان C++ برای Load کردن فایلهای با قالب BMP آورده شده است.

مشخصه‌های فایلهای گرافیکی

تمام تصویرهای گرافیکی مبتنی بر نقشه بیتی ۴ مشخصه اصلی دارند، که عبارتند از:ابعاد ، تفکیک پذیری ، عمق بیتی و مدل رنگ. که در ذیل به شرح هر یک از آنها خواهیم پرداخت.

۱ - ابعاد

تصویرهای گرافیکی مبتنی بر نقشه بیتی همواره به صورت شبکه‌های مربع شکل بزرگ می - بلشند. این شبکه‌ها مانند صفحه شطرنج میباشند. این شبکه‌های مربع شکل بزرگ تز مربع‌های کوچکی تشکیل شده‌اند. یکی از مشخصه‌هایی که همواره می‌توان در مورد شبکه‌ها بیان شد، این است که دارای ابعاد هستند.ابعاد شبکه در واقع معادل تعداد مربعهایی است که طول و عرض تصویر را تشکیل داده - اند.و ربطی به اندازه واقعی تصویر ندارند. مثلا ممکن است سطح صفحه شطرنج ۶ اینچ مربع باشد یا به اندازه زمین فوتبال ولی در هر صورت صفحه شطرنج از ۸ مربع در ۸ مربع تشکیل شده‌است.اما با افزایش اندازه تصویر اندازه فایل به صورت مربعی افزایش می‌یابد. مثلا اگر ابعاد تصویر ۳ برابر شود حجم آن ۹ برابر می‌شود.

۲ - تفکیک پذیري

تفکیک‌پذیری معادل تعداد نمونه ها در واحد اندازه‌گیری می‌باشد. اگر تصویر دارای ۷۲ نمونه در اینچ باشد و هر ۷۲ نمونه آن در کنار یکدیگر ۱ اینچ طول داشته‌باشد دیگر می‌دانید که طول هر یک از اضلاع آن ۱ اینچ خواهد بود.میتوان همین تصویر را با ۳۶ نمونه‌گیری نیز به دست آورد(یعنی بدون تاثیر بر ابعاد شبکه ، تفکیک پذیری را تغییر دهید) در این حالت ابعاد تصویر در هر بعد ۲ اینچ خواهد بود.

اگر ابعاد و تفکیک‌پذیری یک فایل گرافیکی را بدانید می‌توانید اندازه آن را مشخص کنید.مثلا اگر تصویری به اندازه ۳×۳ اینچ اسکن کرده‌باشید و در هر اینچ ۱۰۰ نمونه داشته‌باشید ، انگاه تعداد نمونه‌های موجود ۹۰۰ خواهد بود.هر چه تعداد نمونه‌ها بیشتر باشد ، حجم فایل بزرگتر خواهد بود.

۳ - عمق بیتي

هر نقطه از تصویر می‌تواند سیاه سفید ، خاکستری و یا رنگی باشد. نکته در بیتهایی است که به هر نمونه اختصاص می یابد. اگر نمونه‌ها با استفاده از یک بیت منفرد تعریف شده‌باشد، فقط می - توانند سفید یا سیاه باشند.اگر برای هر نمونه ۲ بیت در نظر گرفته شده باشد ۴ ترکیب ممکن برای رنگ آنها موجود است . تعداد این بیت‌ها عمق بیتی تصویر نامیده می‌شود.

عمق بیتی بر اندازه فایل تاثیر دارد. حجم فایلی که عمق بیتی آن ۸ بیت است ، ۸ برابر اندازه فایلی است که عمق بیتی آن ۱ است.

۴- مدل رنگ

آخری صفت مربوط به تصویر "مدل رنگ" است، که در واقع فقط به تصویرهای رنگی مربوط می‌شود. متداولترین مدل رنگ RGB است . در این مدل هر رنگ با عددی ۸ بیتی تعریف می‌شود که در مجموع ۲۴ بیت را بوجود می‌آورند.

بعضی از تصویرها با مدل رنگ CMYK (Black, Yellow , Magenta, Cyan) تعریف می‌شوند. در این روش مقادیر هر یک از رنگهای مورد استفاده در روند چاپ به صورت مجزا تعریف شده‌اند. هر چند که این روش موجب می‌شود که تعداد بیتهای مورد نیاز برای هر نمونه تا ۳۲ بیت افزایش یابد، اما رنگهای بیشتری را ایجاد نمی‌کند. روش مزبور در واقع فقط راه دیگری برای تعریف رنگ به شمار می‌آید. در بیشتر دستگاههای چاپ افست و نیز چاپگرهای رنگی این روش مورد نیاز می‌باشد. قالبهای Tiff, Eps از این روش استفاده می‌کنند.

Indexed Color

رنگهای فهرست شده در واقع روشی برای ایجاد فایلهای ۲۵۶ رنگ و ۸ بیتی می‌باشد. نقشه بیتی‌های مبتنی بر Indexed Color در جدولی متشکل از ۲۵۶ رنگ قرار می‌گیرند، که رنگهای آن از جدول رنگ ۲۴ بیتی انتخاب شده‌اند. هر کدام از رنگهای انتخاب شده در واقع به یکی از رنگهای جدول رنگ ارجاع داده می‌شود ، هر چند رنگهای فهرست شده مقدار زیادی در فضای دیسک صرفه‌جویی می‌کنند(زیرا به ازائ هر نقطه نمونه گیری شده به ۸ بیت فضا نیاز دارند، در حالی که برای تعریف کامل رنگ به ۲۴ بیت مورد نیاز می‌باشد)، اما در این حالت می‌توان ۲۵۶ رنگ را انتخاب کرد رنگها ی فهرست شده بیشتر در فایلهای GIF مورد استفاده قرار می‌گیرند. این قالب هرچند چندان مطلوب نیست اما به عنوان استاندارد WWW مورد استفاده قرار می‌گیرد.

بر خلاف تفکیک‌پذیری و عمق بیتی ، مدل رنگ الزاما بر حجم فایل اثری ندارد. اما اگر مدل رنگ را تغییر دهید، شاید کامپیوتر ناچار شود عمق بیتی تصویر را عوض کند. که این تغییر بر حجم فایل اثر می‌گذارد.مثلا اگر رنگ مبتنی بر RGB را به تصویر مبتنی بر grayscale تبدیل کنید ، کامپیوتر عمق بیتی را از ۲۴ بیت به ۸ بیت کاهش می‌دهد.

قالب بندی فایلها

اولین و مهمترین وجه افتراق قالب بندی فایلها ، تشخیص اختلاف فایلهای مبتنی بر تصویرهای گرافیکی Bitmap و فایلهای شیئ گرا می‌باشد.

تصویرهاي گرافيكي مبتني بر Bitmap

تصویرهای گرافیکی فقط نوعی شبکه بزرگ از پیکسلها می‌باشند. برای اینکه بتوان کار مفیدی با تصویرهای مبتنی بر Bitmap انجام داد ، باید قالب بندی فایل به گونه‌ای باشد ، که اطلاعات مربوط به ابعاد، تفکیک پذیری ، عمق بیتی ، مدل رنگ و متراکم سازی را شامل شود.تصویرهای Bitmap از ۳ منبع اصلی بدست می‌آیند: که عبارتند از اسکنرها، برنامه‌های ترسیم و پردازش تصویر و برنامه‌های ضبط کننده صفحه نمایش. اگر با هر یک از این ابزارهاتصویر گرافیکی ایجاد شود ، تصویر مزبور از نوع Bitmap خواهد بود.

تصویرهاي گرافيكي شيء گرا

فایلهای شیء گرا معمولا بسیار پیچیده‌تر می‌باشند. در این فایلها به جابی اینکه چهار گوشی متشکل از تعداد زیادی نقطه تعریف شود، تصویرهای گرافیکی شیءگرا ایجاد می‌شود. اطلاعات آن مثلا به این شکل می‌تواند باشد((مربعی با ابعاد تعیین شده را در محلی که مشخص شده ترسیم کن)) . این تصویر می‌تواند از چندید خط چهار گوش ، دایره ،منحنی و چند ضلعی و بلوکهای متنی تشکیل شده است.هر کدام از این موارد ممکن است صفتهای مختلفی داشته‌باشند. مثلا پهنای خط ، نوع قالب-بندی ، رنگهای پر شده و بسیاری از موارد دیگر می‌توان ذکر کرد.

تصویرهای شیءگرا معمولا از ۲ منبع اصلی بوجود می‌آیند، که عبارتند از : برنامه‌های ترسیمی (مانند freehand ,illustrator) و برنامه‌های طراحی به کمک کامپیوتر (CAD) .

قالب بندي فايلهاي Bitmap

قالبهای بسیار متعددی برای ذخیره‌سازی فایلهای گرافیکی بوجود آمده‌اندکه به عنوان مثال می‌توان قالبهای ,jpg,gif,Eps Tiff, cgm wmf,pcx, wpg , mif, ,emf hpgl را نام برد. هر کدام از این قالب بندی‌ها خصوصیاتی دارند که آنها را از یکدیگر متمایز می‌کند.هر کدام گزینه‌هایی برای فشرده-سازی دارند، اما ثابت ماندن شدت و رنگها در فایلهای مزبور با یکدیگر تفاوت دارند.در جدول ۱ ویژگی‌های بعضی از این قالبها با هم مقایسه شده‌اند.

Format	Compression	Displays in browser	Bit depths	Color	Interlacing	Labels/text[1]	Transparency
TIFF	Lossless	No	1–8, 24 32, 48	Indexed, gray, RGB, CMYK	No	Yes	Alpha channel
EPS	Lossless or lossy	No	same as TIFF	Gray, RGB, CMYK	No	Yes	Only in bilevel mode
GIF	Lossless	Yes	1–8	Indexed	Yes	Yes	One color
JPEG	Lossy	Yes	8, 24, 32	Gray, RGB, CMYK	Yes[2]	No[3]	None
PNG	Lossless	Yes[4]	1–8,16,24, 32, 48	Indexed, Gray, RGB	Yes	Yes	One color or alpha channel
FlashPix	Lossy and/ or lossless[5]	Yes[6]	8, 24	Gray, RGB, PhotoYCC[7]	No	Yes	Alpha channel
Photo CD	Lossy	Yes[6]	24	PhotoYCC[7]	No	Yes	None

۱ـقالببندي Tiff

قالب بندی TIFF (tagged image file format) متداولترین قالب بندی با استاندارد صنعتی می‌باشد، که برای فایلهای تصویر مورد استفاده قرار می‌گیرد. تصویرهای مبتنی بر Bitmap در قالب بندی TIFF می‌توانند هر ابعاد و تفکیک‌پذیری را داشته‌باشند.از نظر تئوری عمق بیتی آنها نامحدود است اما بسیاری از نرمافزارها فقط فایلهایی با عمق بیتی استاندارد را می‌خوانند و می‌نویسند.نمونه‌هایی از عمق استاندارد به ازای هر نمونه گیری عبارتند از: ۱–۸ بیت ، ۲۴ بیت ، ۳۲ بیت یا ۴۸ بیت.

TIFF می‌تواند مدلهای رنگ مختلف مانند gray scale ، RGB ، رنگهای فهرست شدهو یا CMYK را کدگذاری کند. می‌توان آن را با قالب‌بندی متراکم یا غیر متراکم ذخیره کرد. تقریبا تمام برنامه هایی که با تصویرهی Bitmap کار می‌کنند می‌توانند با فایلهای tiff نیز کار کنند.

۱-۱ مدلهاي رنگ فايلهاي tiff

مشخصه های فایلهای Tiff برای مدلهای رنگ RGB , CMYK مناسب است. یعنی اینکه میتوان ابتدا ۴ رنگ تصویر را از یکدیگر تفکیک کرد، وسپس آن را ذخیره نمود.

فایلهای tiff همچنین می‌توانند تصویرهای مبتنی بر Bitmap با رنگهای فهرست شده را نیز ذخیره کند. اما غالبا هیچکس این قالب‌بندی را برای رنگهای فهرست شده مورد استفاده قرار نمی‌دهد.همه برای ذخیره‌سازی فایلهای مبتنی بر رنگها فهرست شده فایلهای gif را ترجیح می‌دهند.

۱-۲ متراکم سازي فايلهاي tiff

برای ذخیره‌سازی فایلهای tiff می‌توان از روش فشرده‌سازی LZW استفاده کرد اما در این مورد الزامی وجود ندارد. روش فشرده‌سازی LZW از نوع فشرده‌سازی بدون کاهش داده‌ها است . یعنی هنگام ذخیره سازی فایل با این روش ، داده‌های آن کاسته نخواهد شد و بدون از دست دادن داده‌ای می‌توان آن را به حالی اول بازگرداند.

۲ - قالب‌بندي gif

ایـن قالـب‌بندی در سال ۱۹۸۷ ایجاد شد، کیفیت تصویر آن پایین است.و تصویر بدست آمده تقریبا در تمام موارد به صورت Posterized است.

۲-۱ مدل رنگ فایلها يgif

فایلهـای gif حداکثر می‌توانند ۲۵۶ رنگ مختلف داشته باشند. رنگهای مذبور در جدول رنگی قـرار مـی‌گـیرند، کـه خود بخشی از فایل gif به شمار می‌آیند.این روش ((رنگهای فهرست شده یا indexed color)) نامـیده مـی‌شـود، زیـرا رنگهـای جـدول مزبور به صورت مستقیم مقادیر RGB را نشان نمـی‌دهـند بلکـه داده‌هـا بـه صـورت یـک بایتـی فشـرده شـده‌اند، و در واقع با یکی از داده‌های ورودی موجـود در جدول رنگ اصلی در ارتباط است. مقدار عددی هر بایت منفرد در واقع به یکی از رنگهای کـامل ۲۴ بیتـی در جـدول رنـگ RGB اشـاره مـی‌کند. با این کار به میزان زیادی در حجم فایل صرفه جویی می‌شود، اما همچنان رنگهای حقیقی توسط فایل نشان داده می‌شوند.

داده‌های تصویر های مبتنی بر gif و جدول رنگهای فهرست شده آنها را می‌توان تقریبا در تمام بـرنامه‌هـای کار بر روی تصویر ، ویرایش کرد. لزومی نارد که از همه ۲۵۶ رنگ استفاده شود.داده‌های ورودی جـدول رنگ gif می‌توانند با اختصاص دادن عددهای ۸ بیتی به مقادیر فهرست شده، هر یک از ۲۵۶ رنـگ را ظاهـر کنـند.هـر چـه رنگهـای موجود در جدول رنگ کمتر باشد، بیشتر فشرده می‌شود. همچنین در برنامه‌های مرورگر وب ، و نیز کامپیوترهای مختلف نیز بهتر ظاهر می‌شود.

۲-۲ متراکم ساز ي فایلهاي gif

در قالـب‌بـندی gif از روش مـتراکم سـازی Huffman که بدون کاهش داده‌ها است استفاده می - شـود و نمی‌توان حالت متراکم سازی آن را غیر فعال کرد. این کار به میزان قابل توجهی حجم فایل را کـاهش مـی‌دهـد.مـیزان مـتراکم سـازی کاملا به محتویات صفحه بستگی دارد . تصویرهایی که بیشتر بخشـهای آنهـا سـطح تـک رنـگ و یکدسـت باشد، حتی ممکن است تا یک دهم یا یک صدم کاهش یابد در حالی که تصویرهای طبیعی در اغلب موارد چندان متراکم نمی‌شوند.

۳- قالب JPEG (Joint Photographic Expert Group)

jpeg هـنگام ضـبط از کیفیـت تصـویر مـی‌کـاهد. زیرا طرح فشرده‌سازی آن با کاهش داده‌ها مـی‌باشـد.بـه همیـن علـت معمـولا در حیـن روند تولید تصویر ، نمونه‌های متعدد آنرا با قالب‌بندی jpeg ذخـیره نمـی‌کـنند. بلکـه فقـط آخریـن نمونـه‌هایی که آماده استفاده است ، را با این فالب‌بندی ذخیره می‌کنند.

قالـب‌بـندی jpeg برای کوچک کردن فایل ، به طوری که بتوان آن را روی دیسکت قرار داد و یـا از طـریق مـودم انتقال داد ، روش مفیدی به شمار می‌آید.به خصوص اینکه تصویرهای طبیعی ، و

تصاویر اسکن شده را تا حد زیادی فشرده می کند. به همین علت است که این روش اینچنین زیاد در اینترنت مورد استفاده قرار می گیرد

۳-۱ مدلهای رنگ jpeg

روش jpeg ، امکان ذخیره سازی تصاویر مبتنی بر Bitmap (با عمق ۱ – ۸ بیت ، ۲۴ بیت ، ۳۲ بیت را با میزان فشرده‌سازی متفاوت (از میزان تراکم بسیار بالا و کیفیت پایین ، تا میزان تراکم پایین و کیفیت بالا) فراهم کرده‌است.طیف رنگ فایل اولیه فقط امکان ذخیره‌سازی تصویرهای grayScale و تصاویر رنگی RGB را فراهم کرده‌است.

۴ – قالب PCX

این قالب از قدیمی‌ترین قالبهاست. اندازه تصویر حداکثر می‌تواند 64k*64k پیکسل باشد.برای فشرده‌ساری از روش RLE استفاده می‌شود. قالب PCX از ۳ قسمت تشکیل شده‌است :

۱ – Header
۲ – داده های Bitmap
۳ – پالت ۲۵۶ رنگی

Header از ۱۲۸ بایت تشکیل شده‌است که حاوی اطلاعات زیر است:

۱ – Version ۲ – دقت تفکیک تصویر ۳ – ابعاد برحسب پیکسل ۴ – تعداد بایت در هر خط اسکن ۵ – تعداد بیتهای هر پیکسل ۶ – تعداد صفحات رنگی

این قالب به راحتی نوشته می‌شود ولی خواندن آن مشکلاتی دارد.

۴-۱ مدلهاي رنگ PCX

از حالت رنگی ۲۴ بیتی پشتیبانی می کند که یا به صورت یک فهرست ۲۵۶ رنگی و یا RGB ۲۴ بیتی پیاده‌سازی می‌شود.

۵ - فالب BMP

این فایلها می‌توانند شامل تصاویری با عمق بیتی ۱، ۴، ۸ یا ۲۴ بیت باشند. تصاویر ۱، ۴ و ۸بیتی دارای فهرست رنگ می باشند در حالی که تصاویر ۲۴ بیتی رنگ مستقیم دارند. فشرده‌سازی این نوع فایلها به روش RLE می‌باشد.

این قالب استانداردی است که شرکت میکروسافت برای ویندوز گذاشته است.فایلی که دارای قالب BMP است از سه قسمت تشکیل شده است که عبارتند از:

1) bitmap-file header
2) a bitmap-information header
3) a color table, and an array of bytes that defines the bitmap bits.

قسمت header bitmap-file حاوی اطلاعاتی راجع به نوع ، اندازه و Layout فایل است. استراکچر BITMAPFILEHEADER به این بخش اشاره می کند

قسمت bitmap-information header اندازه تصویر وروش فشرده‌سازی و فرمت رنگ را مشخص می کند.

قسمت color table, and an array of bytes آرایه‌ای از RGBQUAD است که محتوی رنگهای استفاده شده در تصویر است.این جدول برای تصویرهایی که عمق بیتی ۲۴ دارند وجود ندارد چون در این حالت ۲۴ بیت برای رنگهای قرمز و سبز و آبی در قسمت داده‌ها نگهداری می‌شود.

مقدار متغیر **biBitCount** در استراکچر BITMAPINFOHEADER نشان ذهنده عمق بیتی و مکزیمم رنگهایی است که برای یک پیکسل نگهداری می‌شود که با توجه به اینکه این مقدار چه باشد معانی متفاوتی دارد.

Value	Meaning
1	**Bitmap is monochrome and the color table contains two entries. Each bit in the bitmap array represents a pixel. If the bit is clear, the pixel is displayed with the color of the first entry in the color table. If the bit is set, the pixel has the color of the second entry in the table.**
4	**Bitmap has a maximum of 16 colors. Each pixel in the bitmap is represented by a 4-bit index into the color table. For example, if the first byte in the bitmap is 0x1F, the byte represents two pixels. The first pixel contains the color in the second table entry, and the second pixel contains the color in the sixteenth table entry.**
8	**Bitmap has a maximum of 256 colors. Each pixel in the bitmap is represented by a 1-byte index into the color table. For example, if the first byte in the bitmap is 0x1F, the first pixel has the color of the thirty-second table entry.**
24	**Bitmap has a maximum of 2^24 colors. The bmiColors (or bmciColors) member is NULL, and each 3-byte sequence in the bitmap array represents the relative intensities of red, green, and blue, respectively, for a pixel.**

ذر ضمیمه ۱ برنامه ای به زبان C++ آورده شده است که می‌تواند یک فایل با قالب BMP را خوانده و با توجه به اطلاعاتی که در Header فایل آمده است مشخصات تصویر را چاپ کرده و سپس فایل را نمایش دهد.

ضمیمه ۱

```
////////////////////////////////////////////////////////////////////////
/////////
//  BMP loading routines
//
//
////////////////////////////////////////////////////////////////////////
//////

#include <stdio.h>
#include <assert.h>
#include <conio.h>
#include <stdlib.h>
#include <alloc.h>
#include <string.h>
#include <mem.h>

#define _DEBUG_

#define BI_RGB          0    // No compression
#define BI_RLE8         1    // RLE8 compression (256 colors)
#define BI_RLE4         2    // RLE4 compression (16 colors)

typedef struct _BITMAPFILEHEADER {  // Offset   Size
    short   bfType;                 //      0      2
    long    bfSize;                 //      2      4
    short   bfReserved1;            //      6      2
    short   bfReserved2;            //      8      2
    long    bfOffBits;              //     10      4
} BITMAPFILEHEADER;                 // Total size: 14

typedef struct _BITMAPINFOHEADER {  // Offset   Size
    long    biSize;                 //      0      4
    long    biWidth;                //      4      4
    long    biHeight;               //      8      4
    short   biPlanes;               //     12      2
    short   biBitCount;             //     14      2
    long    biCompression;          //     16      4
    long    biSizeImage;            //     20      4
    long    biXPelsPerMeter;        //     24      4
    long    biYPelsPerMeter;        //     28      4
    long    biClrUsed;              //     32      4
    long    biClrImportant;         //     36      4
} BITMAPINFOHEADER;                 // Total size: 40

typedef struct _RGBQUAD {
    char    rgbBlue;
    char    rgbGreen;
    char    rgbRed;
    char    rgbReserved;
} RGBQUAD;

BITMAPFILEHEADER        bmfh;
BITMAPINFOHEADER        bmih;
RGBQUAD                *bmcoltab;
long                    bmnumcolors;
char                    pal[768];

void SetColor(char num, char r, char g, char b)
{
```

```
  outp(0x3c8, num);
  outp(0x3c9, r >> 2);
  outp(0x3c9, g >> 2);
  outp(0x3c9, b >> 2);
  pal[num*3+0] = r >> 2;
  pal[num*3+1] = g >> 2;
  pal[num*3+2] = b >> 2;
}

void BMPBufferFlip(long biWidth, long biHeight, char *buffer)
{
  char  byte;
  long  x, y;

  for (y = 0; y < biHeight/2; y++)
    for (x = 0; x < biWidth; x++) {
      byte = buffer[y*biWidth+x];
      buffer[y*biWidth+x] = buffer[(biHeight-y-1)*biWidth+x];
      buffer[(biHeight-y-1)*biWidth+x] = byte;
    }
}

char *LoadBMP(char *filename, long *biWidth, long *biHeight)
{
  FILE                    *f;
  char                    *buffer, byte;
  char                     byte1, byte2;
  char                     hinib, lonib;
  char                     deltax, deltay;
  long                     x, y;
  long                     size, k, i;

  f = fopen(filename,"rb");
  assert(f != NULL);

  fread(&bmfh, sizeof(BITMAPFILEHEADER), 1, f);
  fread(&bmih, sizeof(BITMAPINFOHEADER), 1, f);
/*
  #ifdef _DEBUG_
  printf("\nType   : %c%c",bmfh.bfType,bmfh.bfType>>8);
  printf("\nSize   : %d",bmfh.bfSize);
  printf("\nRes1   : %d",bmfh.bfReserved1);
  printf("\nRes2   : %d",bmfh.bfReserved2);
  printf("\nOffBits: %d\n",bmfh.bfOffBits);
  printf("\nSize       : %d",bmih.biSize);
  printf("\nWidth      : %d",bmih.biWidth);
  printf("\nHeight     : %d",bmih.biHeight);
  printf("\nPlanes     : %d",bmih.biPlanes);
  printf("\nBitCount   : %d",bmih.biBitCount);
  printf("\nCompression: %d",bmih.biCompression);
  printf("\nSizeImage  : %d",bmih.biSizeImage);
  printf("\nXPels      : %d",bmih.biXPelsPerMeter);
  printf("\nYPels      : %d",bmih.biYPelsPerMeter);
  printf("\nClrUsed    : %d",bmih.biClrUsed);
  printf("\nClrImpor   : %d",bmih.biClrImportant);
  printf("\n");
  getch();
  #endif
*/

  bmnumcolors = (1 << bmih.biBitCount);
```

```
  //printf("\nNumber of colors: %d", bmnumcolors);

  // Bitmap size
  size = bmih.biWidth * bmih.biHeight;

  // Load color table if biBitCount < 24
  if (bmih.biBitCount != 24) {
    bmcoltab = (RGBQUAD *)calloc(bmnumcolors, sizeof(RGBQUAD));
    #ifdef _DEBUG_
    assert(bmcoltab != NULL);
    #endif
//  mv_Error("Cannot allocate memory for BMP color table");
    fread(bmcoltab, bmnumcolors*sizeof(RGBQUAD), 1, f);
  }
  else
    size *= 3;

  for (k = 0; k < bmnumcolors; k++)
    SetColor(k, bmcoltab[k].rgbBlue, bmcoltab[k].rgbGreen,
bmcoltab[k].rgbRed);

  // Allocate memory for picture
  buffer = (char *)calloc(size, sizeof(char));
  #ifdef _DEBUG_
  assert(buffer != NULL);
  #endif

  fseek(f, bmfh.bfOffBits, SEEK_SET);
  switch (bmih.biCompression) {

    // RGB
    case BI_RGB:
      switch (bmih.biBitCount) {
      // 2 colors
      case 1:
        k = 0;
        for (y = 0; y < bmih.biHeight; y++) {
          for (x = 0; x < (bmih.biWidth >> 3); x++) {
            fread(&byte, sizeof(char), 1, f);
            for (i = 7; i >= 0; i--)
            buffer[k++] = ((byte & (1 << i)) >> i);
          }
          // biWidth % 8 pixels still to draw
          if ((bmih.biWidth % 8) > 0) {
            fread(&byte, sizeof(char), 1, f);
            size = 8 - (bmih.biWidth % 8);
            for (i = 7; i >= size; i--)
            buffer[k++] = ((byte & (1 << i)) >> i);
            // We read one more byte
            size = 1;
          }
          else
            size = 0;
          // Scan line must be zero-padded to end on a 32-bit
boundary
          size += (bmih.biWidth >> 3);
          while (size % 4 != 0) {
            fread(&byte, sizeof(char), 1, f);
            size++;
          }
        }
```

```
      break;
      // 16 colors
      case 4:
        k = 0;
        for (y = 0; y < bmih.biHeight; y++) {
          for (x = 0; x < (bmih.biWidth >> 1); x++) {
            fread(&byte, sizeof(char), 1, f);
           // buffer[k++] = ((byte >> 4) & 0x0F);
            buffer[k++] = (byte & 0x0F);
          }
          // Last pixel of row if number of pixels in row is odd
          if ((bmih.biWidth % 2) == 1) {
            fread(&byte, sizeof(char), 1, f);
            buffer[k++] = ((byte >> 4) & 0x0F);
            // We read one more byte
            size = 1;
          }
          else
            size = 0;
          // Scan line must be zero-padded to end on a 32-bit
boundary
          size += (bmih.biWidth >> 1);
          while (size % 4 != 0) {
            fread(&byte, sizeof(char), 1, f);
            size++;
          }
        }
      break;
      // 256 colors
      case 8:
        k = 0;
        for (y = 0; y < bmih.biHeight; y++) {
          for (x = 0; x < bmih.biWidth; x++)
            fread(&buffer[k++], sizeof(char), 1, f);

          // Scan line must be zero-padded to end on a 32-bit
boundary
          size += bmih.biWidth;
          while (size % 4 != 0) {
            fread(&byte, sizeof(char), 1, f);
            size++;
          }
        }
      break;
      // 16.7M colors
      case 24:
        assert(0);
        break;
      default:
        #ifdef _DEBUG_
        assert(0);
        #endif
      }
      break;

    // RLE8
    case BI_RLE8:
      x = 0;
      y = 0;

      while (y*bmih.biWidth+x < size) {
```

```
  fread(&byte1, sizeof(char), 1, f);
  fread(&byte2, sizeof(char), 1, f);

  // Absolute Mode
  if (byte1 == 0 && byte2 >= 0x03 && byte2 <= 0xFF) {
    for (k = 0; k < byte2; k++) {
      fread(&byte, sizeof(char), 1, f);
      buffer[y*bmih.biWidth+x++] = byte;
    }
    // Each run must be aligned on a word boundary
    if ((byte2 % 2) != 0)
      fread(&byte, sizeof(char), 1, f);
  }
  // Encoded Mode
  else {
    // Indicate an escape
    if (byte1 == 0) {
      switch (byte2) {
        // End of line
        case 0:
        x = 0;
        y++;
        break;
        // End of bitmap
        case 1:
        break;
        // Delta
        case 2:
        fread(&deltax, sizeof(char), 1, f);
        fread(&deltay, sizeof(char), 1, f);
        x += deltax;
        y += deltay;
        break;

        default:
        assert(0);
      }
    }
    else {
      for (k = 0; k < byte1; k++)
        buffer[y*bmih.biWidth+x++] = byte2;
    }
  }
  }
  break;

// RLE4
case BI_RLE4:
  x = 0;
  y = 0;

  while (y*bmih.biWidth+x < size) {

  fread(&byte1, sizeof(char), 1, f);
  fread(&byte2, sizeof(char), 1, f);

  // Absolute Mode
  if (byte1 == 0 && byte2 >= 0x03 && byte2 <= 0xFF) {
    byte2 >>= 1;
    for (k = 0; k < byte2; k++) {
      fread(&byte, sizeof(char), 1, f);
```

```
        hinib = ((byte >> 4) & 0x0F);
        lonib = (byte & 0x0F);
        buffer[y*bmih.biWidth+x++] = hinib;
        buffer[y*bmih.biWidth+x++] = lonib;
      }
      // Each run must be aligned on a word boundary
      if ((byte2 % 2) != 0)
        fread(&byte, sizeof(char), 1, f);
    }
    // Encoded Mode
    else {
      // Indicate an escape
      if (byte1 == 0) {
        switch (byte2) {
          // End of line
          case 0:
          x = 0;
          y++;
          break;
          // End of bitmap
          case 1:
          break;
          // Delta
          case 2:
          fread(&deltax, sizeof(char), 1, f);
          fread(&deltay, sizeof(char), 1, f);
          x += deltax;
          y += deltay;
          break;

          default:
          assert(0);
        }
      }
      else {
        hinib = ((byte2 >> 4) & 0x0F);
        lonib = (byte2 & 0x0F);

        byte2 = byte1 % 2;
        byte1 >>= 1;
        for (k = 0; k < byte1; k++) {
          buffer[y*bmih.biWidth+x++] = hinib;
          buffer[y*bmih.biWidth+x++] = lonib;
        }
        if (byte2 != 0)
          buffer[y*bmih.biWidth+x++] = hinib;
      }
    }
    }
    break;


  default:            // Should not happen
    #ifdef _DEBUG_
    assert(0);
    #endif
}
fclose(f);

BMPBufferFlip(bmih.biWidth, bmih.biHeight, buffer);
*biWidth = bmih.biWidth;
```

```
  *biHeight = bmih.biHeight;

  return buffer;
}



//  Save .BMP picture  //
//  Paramaters:
//    biFileName    = Name of file bitmap is saved into
//    biStartX      = Bitmap starting X poition
//    biStartY      = Bitmap starting Y poition
//    biWidth       = Width of bitmap
//    biHeight      = Height of bitmap
//    biBuffer      = Buffer with bitmap
//    coltab        = Bitmap color table
//    biBitCount    = Number of bits for one pixel (1, 4, 8(۲۴ ,
    //biCompression = Use RGB or RLE compression (BI_RGB, BI_RLE8,
BI_RLE4(
char SaveBMP(char *biFileName, long biStartX, long biStartY,
           long biWidth, long biHeight, char *biBuffer, char *coltab,
           char  biBitCount, char biCompression)
{
  FILE  *f;
  char   byte;
  long   size, k, i;
  long   x, y;

  bmnumcolors = (1 << biBitCount);
  // True color BMP file does not have any color map data
  if (biBitCount == 24) bmnumcolors = 0;

  // BITMAPFILEHEADER
  bmfh.bfType = 0x4D42;         // 'BM'
  bmfh.bfReserved1 = 0;         // Reserved, always 0
  bmfh.bfReserved2 = 0;         // Reserved, always 0
  // bfOffBits = Byte offset from BITMAPFILEHEADER to the actual
bitmap data
  bmfh.bfOffBits = sizeof(BITMAPFILEHEADER) +
sizeof(BITMAPINFOHEADER) +
               bmnumcolors*sizeof(RGBQUAD);
  bmfh.bfSize = bmfh.bfOffBits; // File size, will be written below

  // BITMAPINFOHEADER
  bmih.biSize = sizeof(BITMAPINFOHEADER);       // Size of
BITMAPINFOHEADER
  bmih.biWidth = biWidth;
  bmih.biHeight = biHeight;
  bmih.biPlanes = 1;            // Number of planes for the target
device
  bmih.biBitCount = biBitCount; // Number of bits per pixel (1, 4, 8,
24)

  bmih.biSizeImage = biWidth * biHeight;  // Size of image in bytes
  switch (biBitCount) {
    // Bitmap has 2 colors
    case 1:
      bmih.biCompression = BI_RGB;        // No compression allowed
      bmih.biSizeImage >>= 3;             // 8 pixels can be
represented with 1 byte
      break;
```

```
    // Bitmap has 16 colors
    case 4:
      if (biCompression == BI_RLE4)
      bmih.biCompression = BI_RLE4;     // Can have BI_RLE4
compression
      else
      bmih.biCompression = BI_RGB;      // Default compression
      bmih.biSizeImage >>= 1;             // 2 pixels can be
represented with 1 byte
      break;

    // Bitmap has 256 colors
    case 8:
      if (biCompression == BI_RLE8)
      bmih.biCompression = BI_RLE8;     // Can have BI_RLE8
compression
      else
      bmih.biCompression = BI_RGB;      // Default compression
      break;

    // True color bitmap (16.7M colors)
    case 24:
      bmih.biCompression = BI_RGB;
      bmih.biSizeImage *= 3;              // 3 bytes needed for each
pixel
      break;

    default:
      #ifdef _DEBUG_
      assert(0);
      #endif
  }

  bmih.biXPelsPerMeter = 0;
  bmih.biYPelsPerMeter = 0;
  bmih.biClrUsed = bmnumcolors;           // Number of color indexes
in color
                                // table actually used by the bitmap
  bmih.biClrImportant = bmnumcolors;      // All colors are important

  f = fopen(biFileName, "w+b");
  #ifdef _DEBUG_
  assert(f != NULL);
  #endif

  // Write BITMAPFILEHEADER and BITMAPINFOHEADER to file
  fwrite(&bmfh, sizeof(BITMAPFILEHEADER), 1, f);
  fwrite(&bmih, sizeof(BITMAPINFOHEADER), 1, f);

  // Color table
  if (bmnumcolors > 0) {
    bmcoltab = (RGBQUAD *)calloc(bmnumcolors, sizeof(RGBQUAD));
    #ifdef _DEBUG_
    assert(bmcoltab != NULL);
    #endif
    // Build color table from RGB
    for (k = 0; k < bmnumcolors; k++) {
      bmcoltab[k].rgbRed = coltab[k*3+0] << 2;
      bmcoltab[k].rgbGreen = coltab[k*3+1] << 2;
      bmcoltab[k].rgbBlue = coltab[k*3+2] << 2;
```

```
    bmcoltab[k].rgbReserved = 0;
  }
  fwrite(bmcoltab, bmnumcolors*sizeof(RGBQUAD), 1, f);
  free(bmcoltab);
}

BMPBufferFlip(biWidth, biHeight, biBuffer);
switch (biCompression) {

  // BI_RGB
  case BI_RGB:

    switch (biBitCount) {
    // 16 colors
    //case 4:
    //  size >>= 1;
    //  for (k = 0; k < size; k++) {
    //   fread(&byte, sizeof(char), 1, f);
    //    buffer[(k << 1)] = ((byte>>4) & 0x0F);
    //    buffer[(k << 1)+1] = (byte & 0x0F);
    //  }

    // 256 colors
    case 8:
      k = 0;
      byte = 0;
      for (y = 0; y < biHeight; y++) {
        for (x = 0; x < biWidth; x++) {
          fwrite(&biBuffer[y*320+x], sizeof(char), 1, f);
          bmfh.bfSize++;
        }

        size = biWidth;
        while ((size % 4) != 0) {
          fwrite(&byte, sizeof(char), 1, f);
          bmfh.bfSize++;
          size++;
        }
      }
      break;

    default:
      #ifdef _DEBUG_
      assert(0);
      #endif
    }
    break;

  // BI_RLE8
  //case BI_RLE8:
  //  break;

  default:
    #ifdef _DEBUG_
    assert(0);
    #endif
}

fseek(f, 0, SEEK_SET);

// Write BITMAPFILEHEADER and BITMAPINFOHEADER to file
```

```
  fwrite(&bmfh, sizeof(BITMAPFILEHEADER), 1, f);
  fwrite(&bmih, sizeof(BITMAPINFOHEADER), 1, f);

  fclose(f);

  return 0;
}



main(int argc, char *argv[])
{
  long   x, y, width, height;
  char  *picture;
  char *video = (char *)0xa000;

  printf("\nBMP file viewer, Copyright (c) 1381 by Hojjat Allah
Bohlooli");
  if (argc != 2) {
     printf("\nUSAGE : bmp.exe <file.bmp>");
     exit(1);
  }

  _asm {
    mov   ax,13h
    int   10h
  }

  picture = LoadBMP(argv[1], &width, &height);
  memset(video, 0, 64000);
  if (width > 320) width = 320;
  if (height > 200) height = 200;
  for (y = 0; y < height; y++) {
    for (x = 0; x < width; x++)
      video[y*320+x] = picture[y*bmih.biWidth+x];
  }
  getch();

  _asm {
    mov   ax,03h
    int   10h
  }
  return 1;
}
```