ATRテクニカルレポート表紙

[非公開]

TR-M-0042

M-motion

Andre Plante

1999.4.30

ATR 知能映像通信研究所

André Plante
ATR MIC3

Final Report

Presented to Iwadate Yuichi
Kyoto, April 1999

André Plante
MIC3
Invited Researcher (Art & Design)

Arrival Date: 15 June 1997
Departure Date: 28 April 1999

Introduction

When I arrived 23 months ago, I proposed a project called M-motion which goal was to deliver a creative environment based on previous work by art masters. The target users being non-artists or beginners.

This project was in response to a common criticism against computer art, the fact that it is somewhat shallow and that it lacks emotional impact. A comparison between digital artworks and images created with traditional medium usually demonstrates this difference quite clearly. One may argue that computer art is a relatively new art form and that it simply needs to mature, while others will point out that the apparent facility to create artworks fools anyone with paint software into believing that he/she is a seasoned artist.

In order to bridge this gap between traditional and digital art, we first envisioned a database of extracted multimedia elements (color palette, composition guides, etc...) which would act as a compendium for aiding the creation of new artworks. We soon realized that the tools that we would need to create such a database were as important as the end-result. Over the last year, our focus has been on developing some of these tools to not only be used by the database producer but eventually by the end-user in order for them to augment or create their own database.

This research area being largely uncharted an extensive review of processes and techniques used by traditional artists has been initiated. In most cases, it amounts to

reverse-engineering the work of some masters or other experts in any artistic given field.

It has been very interesting as an artist to cooperate and exchange ideas with engineers and programmers. This final report will highlight this cooperation and the progress accomplished in the past 2 years. This report makes extensive use of the English papers I have published over the last 2 years. The relevant papers are included at the end of this document.

Scope

We have concentrated our efforts on digital still imagery. The tools created for M- motion control or extract the physical features of an image, i.e. the type of pictorial features associated with size, colour, texture, composition (placement of objects within the frame), and so on. The use or selection semantic features is left to the artist. [1]

So far we have explored two of these physical features in details, namely colour and composition. We have also started to look at texture and believe that the next interesting area could be the lighting effects possible in digital art.

Users

At this point in time, there are obviously various kind of people wanting to produce computer art.

These people show varying levels of expertise in computer use and in artistic skills. We will now try to identify certain categories and analyze their specific needs.

1. Traditional Artists in the process of learning new computer skills. They typically have a fairly high level of visual literacy gained from experience with traditional media.

2. Art Schools Recent Graduates which have a mix-bag of computer and visual skills.

3. Engineers, programers and other Information Technologists which are now required to create more visual solutions.

4. Computer Hobbyists which, with the advent of the Internet for example, have found a new way to channel their creative energy. Their skill level vary widely as a group due to varying degree of formal training or expertise in the field of visual arts and/or computers.

Each of these categories of users need different kind of help in order to better perform. Looking at both ends of the spectrum, traditional artists will struggle mostly with the computer operation, while engineers and programers will have difficulty translating their ideas into graphics.

Recent graduates from art schools will have a more integrated understanding of both the computer and visual arts. But since art schools now have the dual mission of teaching these two fairly difficult fields in the same amount of time that has traditionally being dedicated to visual arts exclusively, one may argue that the recent graduates have received significantly less art instruction then their predecessors. This problem may also be more acute in shorter programs available in professional colleges.

Figure 1. shows graphically the skill mix of each categories. It also shows the ideal or desired goal, a balance of visual and computer skills. This goal can be attained through different means by each category.

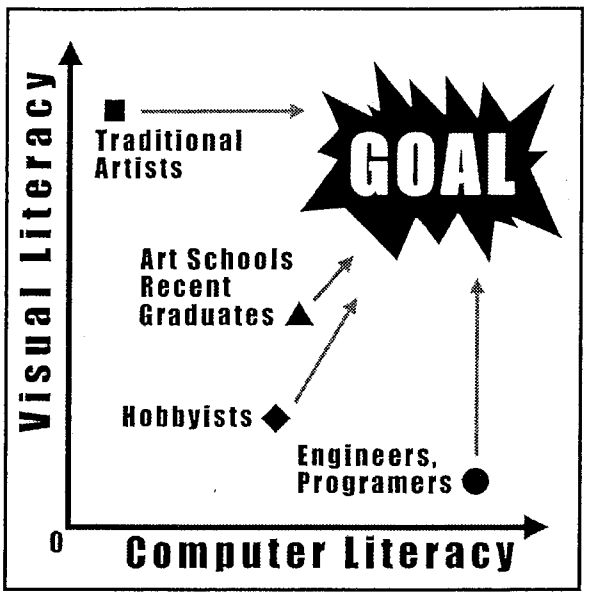


Figure 1: Computer Artist's skills

Projects

The following will be a more or less chronological list of the projects I have participated in over the last two years.

3R Toolbox

I designed a practical tool using Tanaka Shoji's HAREM segmentation method. The tool we created used HAREM in order to help better locate hotspot areas in image-based Virtual Reality. This tool was presented at ACM CHI 98 [2] in Los Angeles and later at OZCHI 98 [3] in Adelaide, Australia.

QTVR Gallery

Image based Virtual Reality has mostly been limited to photo-realistic panoramic images. I reverse-engineered the perspective grid needed to create the feeling of 3-d space and was able to paint convincing 3-D looking canvases which gave the user a new aesthetic experience, namely, walking in a painting. This work has been published in a Book

by Susan Kitchens [4] together with an tutorial document, the VR Grid and the VR movies themselves on an accompanying CD-ROM. The VR Grid has also been requested by more than 30 people through my internet site.

Etuke-shi

This is an Interactive Picture Generation System used during the Kyoto VR Museum exhibit held in 1998. It consisted in a computer environment in which people would select the shape of a vase, select a painting and some colors from which they would like to create a texture. The computer program would then execute a "recipe script" in order to generate a new and original texture for the vase. I also worked on a technique to print-out the final result on a piece of paper that could be cut and folded to make a model of the newly design vase. (the final application was developed by Inoue Seiki and a student)

EgaoKun

In collaboration with HIP laboratories, I participated in the creation of a demo for the 1997 ATR open-house. I was in charge of creating the artworks which were used as the end-result of a face recognition and emotion recognition system. This work has also been published in Japan [5] and presented at the ACM Multimedia 1998 England [6].

Re-composer

I participated in the original discussion with Tanaka and guided him toward the classical composition processes used in art. [7] Visual Balance and ratios were discussed such as the golden section and dynamic (root) rectangles. Tanaka-san has created an application and presented a demo for the 1998 open-house.

Virtual Shrine

This work is a study on photo-realistic image-based VR which presented a highly post-produced image to the viewer. It not only photographically documents the Fushimi Inari Shrine, but rather, presents the author's feelings about this space. This work was selected for the International Quicktime VR Association "Sacred World Project" and is currently exhibited on its website. (http://www.vrview.com/sacredworlds) It will also be presented in Florence at IEEE Multimedia Conference in June 1999 [8].

Noh Mask Study

In collaboration with HIP laboratories and Dr. Ruth Cambell of University College of London, we have worked on the emotions represented and/or created by Noh masks. A paper for the Royal Society of British Psychology is in the making.

Proportional Palette

The Proportional Colour Palette is a palette created by sampling the colours of a masterpiece. It automatically presents the user with an harmonized

(colours that work well side by side) and more subtle selection of colours. For more details on the procedure used in creating the palette, please read the following papers [9] [10] [11]. Two versions were developed, one for Photoshop and one for GIMP. The code of the application developed by Kita-san is available as an annex to this document.

Design Philosophy

The application and their interface design should be simple. They are a synthesis of many artistic theories but most not overwhelm the user.

In other words, the user interface design should not only be artist-friendly but it should also visually communicate a number of important elements. For example, in the proportional palette, we wanted to graphically represent the proportion of each colour and to implicitly divide them into 'main color' and 'accent color'. The palette itself relies on a sampling algorithm and presents many color theories at once. Our goal was to present a tool that seemed simple yet is very powerful.

Conclusion

There is still a lot of work to be done in this field. It is important that research groups such as MIC3 continue to design systems that make life easier for the end-user and eventually put the power of computing in the hands of a larger public.

I would like to thank everybody at ATR and MIC for their support and the great research environment they provided. I would particularly want to acknowledge the support of Nakatsu-san, Iwadate-san, my colleagues in MIC3 (Especially Tanaka-san

and Kita-san) and Inoue Seiki-san of NHK who, two years ago, presented me with the opportunity to come and do research at ATR.

References

[1] Plante, A., Tanaka, S. & Inoue S. (1998) "M-motion: A creative and learning environment facilitating the communication of emotions." Proceedings of IASTED CGIM '98, Halifax, Canada, pp. 77-80

[2] Plante, A., Tanaka, S. & Inoue S. (1998) "Evaluating the Location of Hot Spots in Interactive Scenes using the 3R Toolbox." Proceedings of ACM SIG CHI '98 Conference, Los Angeles, USA, pp. 117-123

[3] Plante, A., Tanaka, S. & Iwadate, Y. (1998) "Designing Effective Navigation for Photo-Realistic VR Environment" IEEE Australasian Conference OZCHI '98, Adelaide, Australia, pp. 4-5

[4] Kitchens, S. (1998) "The QuickTime VR Book: An Introduction to Creating and Viewing Virtual Reality at your Desktop" Book and CD-ROM published by Peachpit Press, includes a section about André Plante's digital painting techniques. ISBN: 0201696843

[5] Lyons, M., Jehan, S., Plante, A. (1998) "Egaokun: An Entertaining Application of Face Processing Technology" Proceedings of the 5th ATR Symposium on Face and Object Recognition, Kyoto, April, 1998.

[6] Lyons, M., Plante, A. Jehan, S., Inoue S. & Akamatsu S. (1998) "Avatar Creation using Automatic Face Recognition" Proceedings of the ACM Multimedia '98 Conference, Bristol, England, pp.427-434

[7] Tanaka, S., Kurumisawa, J., Plante, A., Iwadate, Y. & Inokuchi, S. (1999) "Image Re-Composer: A Post-Production Tool using Composition Information of Pictures" To be published in Proceedings of IEEE Multimedia 99, Florence, Italy.

[8] Plante, A., Tanaka, S., & Iwadate, Y. (1999) "Virtual Shinto Shrine" To be published in Proceedings of IEEE Multimedia 99, Florence, Italy

[9] Plante, A., Tanaka, S., Kita, Y. & Iwadate, Y. (1999) "M-motion Proportional Colour Palette: Abstracting Expert Practice from Art Masterpieces" To be published in Proceedings of IASTED Applied Informatics Conference AI'99, Innsbruck, Austria

[10] Plante, A., Tanaka, S. & Iwadate, Y. (1999) "M-motion: Conveying Emotions Through Digital Art." To be published in Workshop Proceedings of ITE '99, Kyoto, Japan

[11] Plante, A., Tanaka, S. & Iwadate, Y. (1999) "Wanted: Support Systems for Computer Artists" To be published in IEEE Visual Language Conference Proceedings '99, Tokyo, Japan

Plante, Tanaka,. & Kita, (1999)

"Code for M-motion Proportional Colour Palette"

平成 10 年度下期　CSK 納品物

M-motion Proportional Palette Ver2.0

Gimp バージョン

喜多 庸介
株式会社 CSK

平成１１年 ３月２０日

１．はじめに

本システムは、芸術家がよりよいコンピュータイメージを創造することを可能にするために芸術歴史とデザイン原則を活用した創造的な環境です。

芸術家が意図されている芸術的なメッセージを効果的に伝えるためには、色使い、タッチ、構図を賢明に利用しなければなりません。

システムは、初心者から中級までの芸術家がイメージ創造プロセスの上の必要な管理水準を達成することを助けるために、傑作作品から色情報を抽出して創造をサポートしています。

今回のシステムは、前回の Adobe 版の不具合点を考慮して Unix 上のフリーソフトである Gimp（GNU Image Manipulation Program）のプラグインソフトとして開発した。

2．システム構成

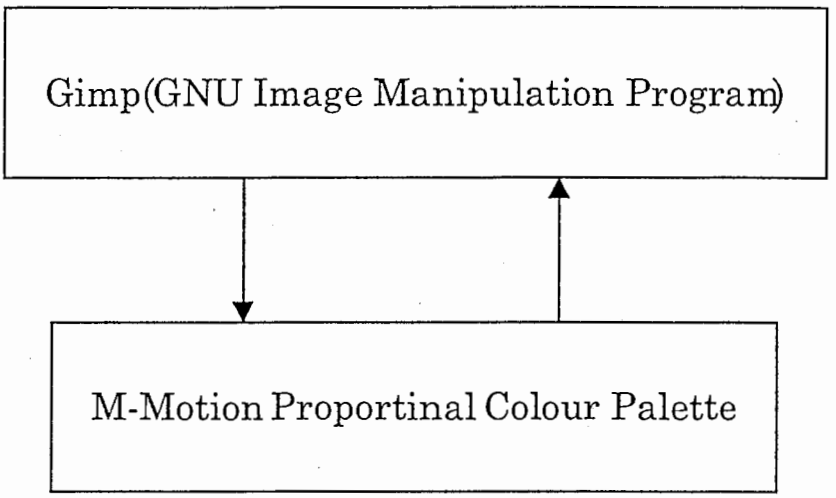


図1．システム構成図

① Gimp

フォトレタッチや画像処理といった用途に向いたフリーなソフトウェアの一つである。

② プラグイン部

指定イメージの色構成の分析及び分析結果を利用して別のイメージを評価する処理を行う。

3．動作環境

・Gimp がインストールされている。
・OS として Unix がインストールされている。
・メモリ、HD 容量に付いては、多い方が良いが特に制限はない。

4．開発環境

・GUI 開発環境として、XFORM を使用。
・プラグイン開発キットとして、GTK を使用。

５．プログラム一覧

File 名【Motion.c】

No	関数名	処理概要
1	Query	プラグインプログラムについての情報を設定する。
2	Run	Gimp のメニューより本プラグインが選択された場合、本関数がコールされる。
3	Motion_dialog	メインダイアログウィンドウの表示処理を行う。
4	Motion_check_quit	プラグインプログラムが終了する場合の処理を行う。
5	Motion_shm_open	プラグインプログラムが２重起動していないかを調べる目的で共有メモリのオープン処理を行う。
6	Motion_shm_close	共有メモリのクローズ処理を行う。

File 名【Motion_sub.c】

No	関数名	処理概要
1	PixelDataReadAndAnalyze	指定イメージデータの分析／評価を行う。分析時には、色分析を行い１２色のインデックスカラーを作成する。又、評価時には、カラーの使用率を評価する。
2	Update_box_rgb	RGB 値のヒストグラムを作成する。
3	Median_cut_rgb	ヒストグラム値の大きい順に指定数分にする。
4	Find_biggest_color_pop	一番大きな値の色を見つける処理を行う。
5	Find_biggest_volume	一番大きな値を見つける処理を行う。
6	Compute_color_rgb	RGB のインデックスカラーを計算して結果を返す処理を行う。
7	Motion_EventCallback	パレット部が押された場合の処理を行う。
8	Motion_progress_init	プログレスバーの初期化を行う。
9	Motion_progress_update	プログレスバーの更新処理を行う。
10	Motion_progress_hide	プログレスバーを非表示にする。
11	Motion_ResizeColorMap	１２色のインデックスカラーを構成比率に対応したウィンドウ領域を取得する。
12	Motion_mapping	指定パーセントより指定パレットの表示位置、大きさを割り付ける。
13	Motion_step_level	指定パレットの表示位置、大きさの優先レベルを調べる。
14	Motion_get_crank	指定パレットを割り付けた場合のクランク（段数）値を求める。

File 名【PropPalette.c】

No	関数名	処理概要
1	Create_form_PropPalette	プラグインのメインダイアログを作成する。
2	Create_form_Progress	プログレスバーのダイアログを作成する。

File 名【PropPalette_cb.c】

No	関数名	処理概要
1	NewBtnProc	指定イメージの１２色のインデックスカラーを作成する処理を行う。
2	LoadBtnProc	過去に作成していたパレット情報を読み出す処理を行う。
3	SaveBtnProc	現在のパレット情報をファイル名付きで書き出す処理を行う。
4	LevelBtnProc	現パレット情報で指定イメージの色使用率を評価する処理を行う。
5	HelpBtnProc	現在はサポートしていません。
6	BrightnessDownBtnProc	Brightness の値を－１する。
7	BrightnessUpBtnProc	Brightness の値を＋１する。
8	SatDownBtnProc	Saturation の値を－１する。
9	SatUpBtnProc	Saturation の値を＋１する。
10	HSVEditColorSet	指定 HSV 値を編集色として設定を行う。
11	HSVUpDownBtnRedraw	エディット部の UP/Down ボタンを再描画する。
12	IndexBtnProc	パレット部のボタンが押された場合の処理を行う。

File 名【PropPalette_cb.c】

No	関数名	処理概要
13	IndexColorProc	パレット部の色オブジェクトの再描画処理を行う。
14	PaletteWndResize	パレット部の各インデックスカラーのウィンドウ位置、サイズの変更処理を行う。

File 名【PropPalette_sub.c】

No	関数名	処理概要
1	Set_MapColor_Cedit	指定 RGB 値でカラー編集部及び矢印部の色設定を行う。
2	Color_RGBtoHSV	RGB カラーを HSV カラーに変換する。
3	Color_HSVtoRGB	HSV カラーを RGB カラーに変換する。
4	Color_DiffFromRGB	２つの RGB 値より NBS 色差を求める。

6．GUI 画面一覧

・プラグインプログラムを起動した最初の画面。

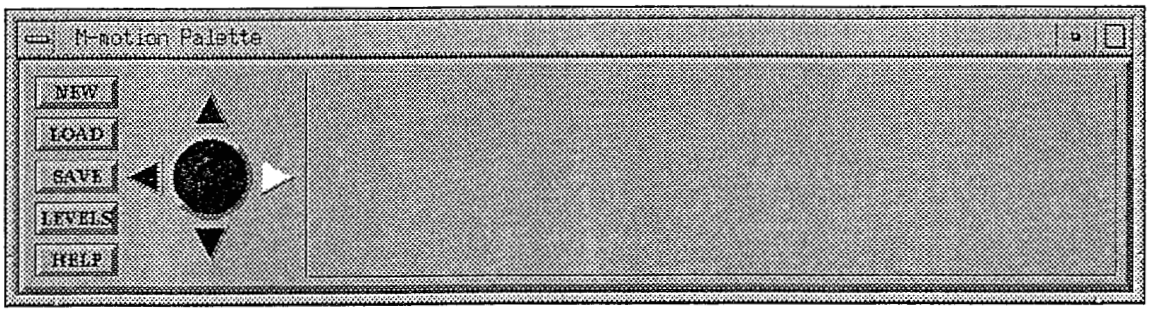


図２．起動時画面

・「NEW」ボタンによりカレントイメージを色分析した結果を表示した状態。
又は、「LOAD」ボタンにより登録してあったパレット情報を読み出した結果を表示した画面。

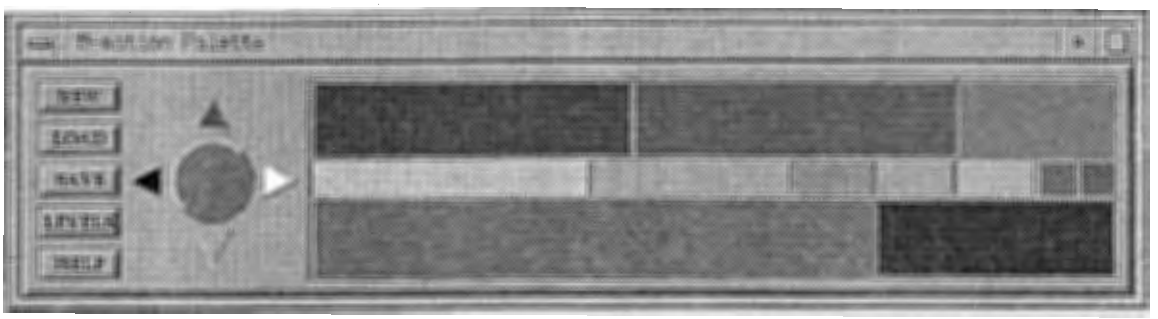


図３．パレット作成（読込み）時画面

・「LEVELS」ボタンによりカレントイメージがパレットの各色がどれだけ使用されているかを評価した結果を表示した画面

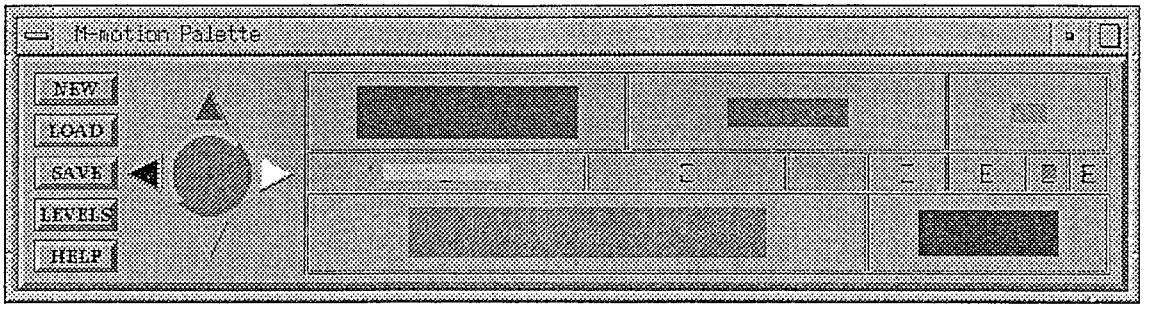


図４．パレット使用率評価時画面

・「LOAD」又は「SAVE」ボタンによりパレット情報をファイルに書込んだり読込んだりする時に表示されるダイアログ画面。

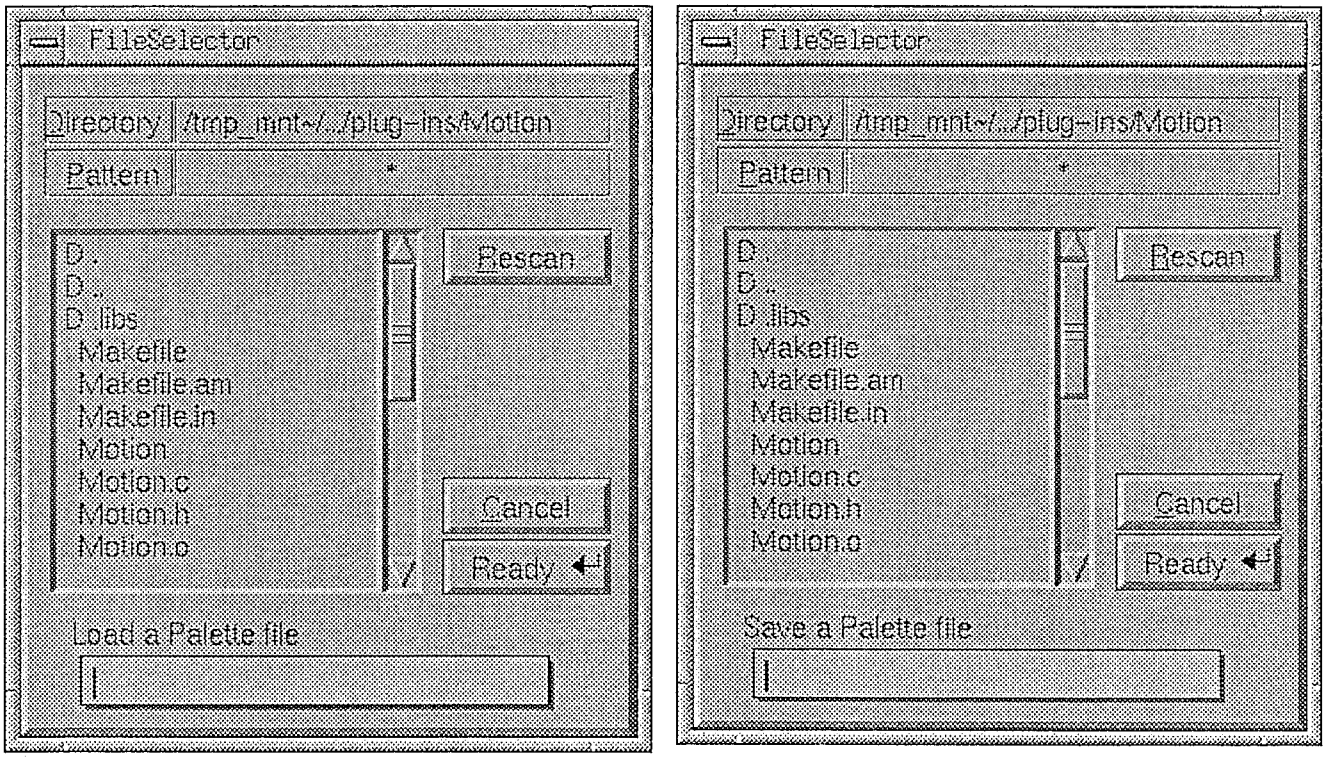


図５．ファイル読込み（書込み）時画面

・各処理の進行状況を示す画面

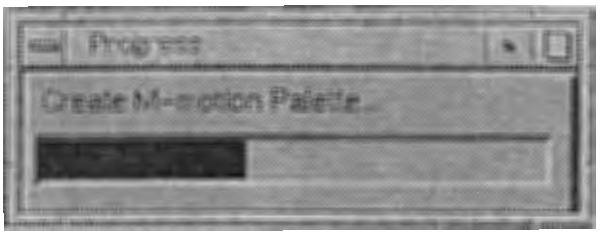
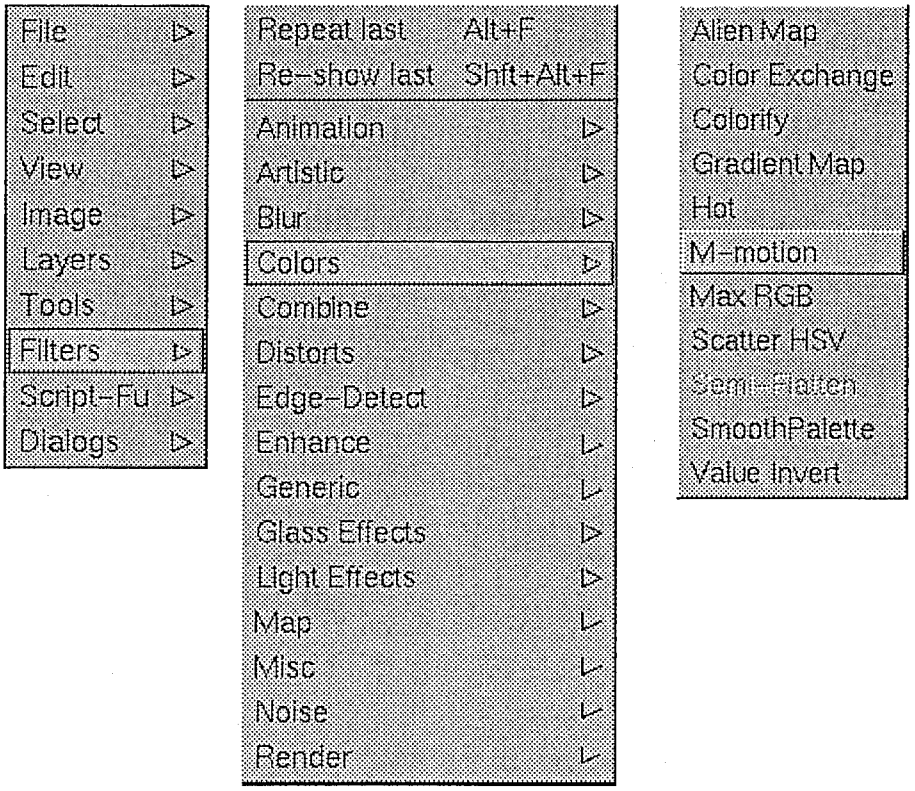


図６．プログレスバー

7．プログラム起動方法

① Gimp プログラムを起動する。
② イメージファイルを開く。
③ 右クリックメニュー「Filters」－「Colors」－「M-motion」を選択する。(下図参照)



8．パレット情報ファイルフォーマット

パレット情報ファイルは、ASCII テキストファイルです。

1行目：ヘッダー行（固定データ）

GIMP Plug-ins:M-motion Palette Data File

2行目以降：データ行

No. x x Red Green Blue Order Percent Column Row Height Width

項目	説明
No. x x	パレット番号。（x x には 1 以上の番号が入る）
Red	RGB の Red 成分。（0 ～ 2 5 5）
Green	RGB の Green 成分。（0 ～ 2 5 5）
Blue	RGB の Blue 成分。（0 ～ 2 5 5）
Order	パレットの順位。（0 ～）
Percent	構成比率。(パーセントを１００倍した値)
Column	表示位置を桁数で指定。（0 ～）
Row	表示位置を行数で指定。（0 ～）
Height	表示領域を高さで指定。（1 ～ 5）
Width	表示領域を高さで指定。（1 ～ 2 0）

※同じ番号があった場合には、後ろ側が有功となる。

9．ソースプログラム

別紙参照

```
/* プロポーショナルパレットGimpバージョン */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include "gtk/gtk.h"
#include "libgimp/gimp.h"
#include "forms.h"
#include "PropPalette.h"
#define _MOTION_C_
#include "Motion.h"
#undef  _MOTION_C_

#define PLUG_IN_NAME "M-motion"
#define PLUG_IN_VERSION "1.1"

static int      M_shmID = -1 ;

GPlugInInfo PLUG_IN_INFO =
{
        NULL,
        NULL,
        query,
        run,
};

MAIN ()

void query()
{
        static GParamDef args[] =
        {
                { PARAM_INT32, "run_mode", "Interactive, non-interactive" },
                { PARAM_IMAGE, "image", "Input image" },
                { PARAM_DRAWABLE, "drawable", "Input drawable" },
                { PARAM_COLOR, "color", "Color to apply"},
        };

        static GParamDef *return_vals  = NULL;
        static int        nargs        = sizeof(args) / sizeof(args[0]),
                          nreturn_vals = 0;

        gimp_install_procedure( "plug_in_Motion",
                                "M-motion Proportinal Palette. Ver2.0.1",
                                "Makes indexed color palette from RGB chanels.And, Co
lor constiution is analyzed.",
                                "Andre Plante & Yousuke Kita",
                                "Andre Plante & Yousuke Kita",
                                "1999",
                                "<Image>/Filters/Colors/M-motion",
                                "RGB*",
                                PROC_PLUG_IN,
                                nargs, nreturn_vals,
                                args, return_vals);
}

void run( char *name, int nparams, GParam *param, int *nreturn_vals, GParam **return_
vals )
{
    GRunModeType        run_mode;
    GStatusType         status;
    static GParam       values[1];
```



```
    status = STATUS_SUCCESS;
    run_mode = param[0].data.d_int32;

    values[0].type = PARAM_STATUS;
    values[0].data.d_status = status;

    *nreturn_vals = 1;
    *return_vals = values;

    if( Motion_shm_open() ) {
        switch (run_mode) {
            case RUN_INTERACTIVE :
                if( !Motion_dialog() ) return ;
                break ;

            case RUN_NONINTERACTIVE :
            case RUN_WITH_LAST_VALS :
                status = STATUS_CALLING_ERROR;
                break;

            default :
                break;
        }
    }
}

int Motion_dialog()
{
    gchar       **argv;
    guchar      r, g, b ;
    gint        argc;
    FL_OBJECT   *obj ;

    argc = 1;
    argv = g_new( gchar *, 1 ) ;
    argv[0] = g_strdup( "M-motion Proportional Palette" ) ;

    /* GTK 及び XFORMを初期化する */
    gtk_init( &argc, &argv ) ;
    gtk_rc_parse( gimp_gtkrc() ) ;

    fl_initialize( &argc, argv, 0, 0, 0 ) ;
    fd_Prop = create_form_PropPalette() ;
    fd_Prog = create_form_Progress() ;

    /* メインフォームを表示する */
    fl_show_form( fd_Prop->PropPalette,
                FL_PLACE_SIZE, FL_FULLBORDER, "M-motion Palette" ) ;

    /* カラーエディタ部のマップカラーを設定する */
    gimp_palette_get_foreground( &r, &g, &b ) ;
    Set_MapColor_CEdit( r, g, b ) ;

    /* イベントループ */
    M_LoopFlag = 1 ;
    while( M_LoopFlag ) {
        obj = fl_check_forms() ;
        if( M_LoopFlag != 2 ) Motion_EventCallback( obj ) ;
        Motion_check_quit() ;
    }

    gtk_main_quit() ;

    return TRUE ;
}
```

```
/* メニューより"閉じる"又は"中止"が選択されたかどうかを調べる */
void Motion_check_quit()
{
    XEvent      event ;

    XPeekEvent( fl_display, &event ) ;
    if( event.type == ClientMessage ) {
        M_LoopFlag = 0 ;
        Motion_shm_close() ;
    }
}

/* 2重起動されているか調べる為に共有メモリを使用する */
/* 共有メモリが使用しているかどうかでプラグインが起動しているか調べる */
int Motion_shm_open()
{
    struct shmid_ds     buf ;

    M_shmID = shmget( M_SHM_KEY, 1, IPC_ALLOC | 0666 ) ;
    if( M_shmID == -1 ) M_shmID = shmget( M_SHM_KEY, 1, IPC_CREAT | 0666 ) ;

    shmctl( M_shmID, IPC_STAT, &buf ) ;
    if( buf.shm_nattch == 0 ) {
        shmat( M_shmID, 0, 0 ) ;
        return 1 ;
    }
    else return 0 ;
}

void Motion_shm_close()
{
    shmctl( M_shmID, IPC_RMID, 0 ) ;
    M_shmID = -1 ;
}
```

```
/*============================================================
 * File      : Motion_sub.c
 * Date      : 1998/12/23
 * Author    : Y.Kita
 * Overview :
 *============================================================*/

#include <stdio.h>
#include "forms.h"
#include "libgimp/gimp.h"
#include "Motion.h"
#include "gtk/gtk.h"

#define RATESCALE       10000
#define HIST_RED        64
#define HIST_GREEN      64
#define HIST_BLUE       32
#define HIST_ALL        HIST_RED * HIST_GREEN * HIST_BLUE
#define HIST_RINDX      HIST_GREEN * HIST_BLUE
#define HIST_GINDX      HIST_BLUE
#define MR              HIST_RINDX
#define MG              HIST_GINDX
#define R_SCALE         30              /*  scale R distances by this much  */
#define G_SCALE         59              /*  scale G distances by this much  */
#define B_SCALE         11              /*  and B by this much  */
#define R_SHIFT         2
#define G_SHIFT         2
#define B_SHIFT         3

void Motion_progress_init( char* ) ;
void Motion_progress_update( double ) ;
void Motion_progress_hide() ;
int  Motion_mapping( int, int*, int, int*, int*, int*, int* ) ;
int  Motion_step_level( int*, int, int, int* ) ;
int  Motion_get_crank( int*, int, int, int ) ;

void PixelDataReadAndAnalyze( int mode, gint32 id )
{
    int         i, ix, iy ;
    int         indx, n_col = 0 ;
    int         plane ;
    int         r, g, b, total ;
    int         vr, vg, vb, val, min ;
    int         histogram[HIST_ALL] ;
    int         indicate_count = 0, indicate_full ;
    box         list[12] ;
    COLOR       color[15] ;
    guchar      *row ;
    GDrawable   *drawable ;
    GPixelRgn   source ;

    drawable = gimp_drawable_get( id ) ;
    plane    = drawable->bpp ;
    if( plane < 3 ) return ;
    total = drawable->height * drawable->width ;
    row   = g_malloc( drawable->width * plane * sizeof(guchar) ) ;

    gimp_pixel_rgn_init( &source, drawable,
              0, 0, drawable->width, drawable->height, FALSE, FALSE ) ;
    if( mode == 0 ) {           /* インデックスカラーの構成比率を求める */
        Motion_progress_init( "Create M-motion Palette..." ) ;
        indicate_full = drawable->height * 2 + 2 ;
        for( i = 0 ; i < HIST_ALL ; i++ ) histogram[i] = 0 ;
        for( i = 0 ; i < 12 ; i++ )
            color[i].red = color[i].green = color[i].blue = color[i].count = 0 ;
```



```
        for( iy = 0 ; iy < drawable->height ; iy++ ) {
            gimp_pixel_rgn_get_row( &source, row, 0, iy, drawable->width ) ;
            for( ix = 0 ; ix < drawable->width ; ix++ ) {
                /* 無効なピクセルをチェック */
                if( (plane == 4) && (row[ix*plane+3] == 0) ) {
                    total-- ;
                    continue ;
                }
                r = row[ix*plane+0] ;
                g = row[ix*plane+1] ;
                b = row[ix*plane+2] ;

                indx = (r>>R_SHIFT) * HIST_RINDX + (g>>G_SHIFT) * HIST_GINDX + (b>>B_
SHIFT) ;
                histogram[indx] += 1 ;
                if( n_col > 12 ) continue ;

                for( i = 0 ; i < n_col ; i++ ) {
                    if( (r == color[i].red) && (g == color[i].green) && (b == color[i
].blue) ) break ;
                }
                if( i == n_col ) {
                    color[n_col].red   = r ;
                    color[n_col].green = g ;
                    color[n_col].blue  = b ;
                    n_col++ ;
                }
            }
            Motion_progress_update( (double) ++indicate_count / indicate_full ) ;
        }

        /* 有効なカラーがあったかどうかで以下の処理を行う */
        if( n_col ) {
            /* インデックスカラーを求める */
            indx = 1 ;
            list[0].Rmin = 0 ;
            list[0].Rmax = HIST_RED - 1 ;
            list[0].Gmin = 0 ;
            list[0].Gmax = HIST_GREEN - 1 ;
            list[0].Bmin = 0 ;
            list[0].Bmax = HIST_BLUE - 1 ;
            update_box_rgb( histogram, list ) ;
            indx = median_cut_rgb( histogram, list, indx, 12 ) ;
            for( i = 0 ; i < indx ; i++ )
                compute_color_rgb( color, histogram, list+i, i ) ;
        }

        if( n_col > 12 ) n_col = 12 ;
        /* */
        for( iy = 0 ; iy < drawable->height ; iy++ ) {
            gimp_pixel_rgn_get_row( &source, row, 0, iy, drawable->width ) ;
            for( ix = 0 ; ix < drawable->width ; ix++ ) {
                /* 無効なピクセルをチェック */
                if( (plane == 4) && (row[ix*plane+3] == 0) ) continue ;
                r = row[ix*plane+0] ;
                g = row[ix*plane+1] ;
                b = row[ix*plane+2] ;

                indx = 0 ;
                min  = 0xFFFFFF ;
                for( i = 0 ; i < n_col ; i++ ) {
                    vr = color[i].red - r ;
                    vg = color[i].green - g ;
                    vb = color[i].blue - b ;
                    val = vr * vr + vg * vg + vb * vb ;
                    if( min < val ) continue ;
```

```
                    indx = i ;
                    min  = val ;
                }
                color[indx].count  += 1 ;
            }
            Motion_progress_update( (double) ++indicate_count / indicate_full ) ;
        }

        M_Palcnt = n_col ;
        for( i = 0 ; i < 12 ; i++ ) {
            M_Palette[i].color.red   = color[i].red ;
            M_Palette[i].color.green = color[i].green ;
            M_Palette[i].color.blue  = color[i].blue ;
            M_Palette[i].color.count = color[i].count ;
            M_Palette[i].ratel       = color[i].count * RATESCALE / total ;
            M_Palette[i].rate2       = 0 ;
            M_Palette[i].order       = 0 ;
            M_Palette[i].x           = 0 ;
            M_Palette[i].y           = 0 ;
            M_Palette[i].row         = 0 ;
            M_Palette[i].col         = 0 ;
        }

        Motion_progress_update( (double) ++indicate_count / indicate_full ) ;
        for( i = 0 ; i < 12 ; i++ ) {
            for( ix = i + 1 ; ix < 12 ; ix++ ) {
                if( M_Palette[i].color.count < M_Palette[ix].color.count )
                     M_Palette[i].order  += 1 ;
                else M_Palette[ix].order += 1 ;
            }
        }
        Motion_progress_hide() ;
    }
    else {                      /* インデックスカラーの使用比率を求める */
        Motion_progress_init( "Analysis M-motion Palette..." ) ;
        indicate_full = drawable->height + 1 ;

        for( i = 0 ; i < 12 ; i++ ) M_Palette[i].color.count = 0 ;
        for( iy = 0 ; iy < drawable->height ; iy++ ) {
            gimp_pixel_rgn_get_row( &source, row, 0, iy, drawable->width ) ;
            for( ix = 0 ; ix < drawable->width ; ix++ ) {
                /* 無効なピクセルをチェック */
                if( (plane == 4) && (row[ix*plane+3] == 0) ) continue ;
                r = row[ix*plane+0] ;
                g = row[ix*plane+1] ;
                b = row[ix*plane+2] ;

                indx = 0 ;
                min  = 0xFFFFFF ;
                for( i = 0 ; i < 12 ; i++ ) {
                    if( M_Palette[i].ratel == 0 ) continue ;    /* 無効パレット */

                    vr = M_Palette[i].color.red - r ;
                    vg = M_Palette[i].color.green - g ;
                    vb = M_Palette[i].color.blue - b ;
                    val = vr * vr + vg * vg + vb * vb ;
                    if( min < val ) continue ;
                    indx = i ;
                    min  = val ;
                }
                if( Color_DiffFromRGB( r, g, b,
                        M_Palette[indx].color.red,
                        M_Palette[indx].color.green,
                        M_Palette[indx].color.blue ) > M_BOUNDS_CDIFF ) continue ;
                M_Palette[indx].color.count += 1 ;
            }
```



```
            Motion_progress_update( (double) ++indicate_count / indicate_full ) ;
        }

        /* 各色の使用比率を求める */
        for( i = 0 ; i < 12 ; i++ ) {
            val = total * M_Palette[i].ratel / RATESCALE ;
            /* 上限値を超えている場合は１００％とする */
            /* 又、構成比が０％の場合も１００％とする */
            if( M_Palette[i].color.count < val )
                 M_Palette[i].rate2 = M_Palette[i].color.count * RATESCALE / val ;
            else M_Palette[i].rate2 = RATESCALE ;
        }

        Motion_progress_hide() ;
    }
    g_free( row ) ;
}

/* Shrink the min/max bounds of a box to enclose only nonzero elements, */
/* and recompute its volume and population */

void update_box_rgb(int *histogram, boxptr boxp )
{
  int *histp ;
  int R,G,B;
  int Rmin,Rmax,Gmin,Gmax,Bmin,Bmax;
  int dist0,dist1,dist2;
  long ccount;

  Rmin = boxp->Rmin;  Rmax = boxp->Rmax;
  Gmin = boxp->Gmin;  Gmax = boxp->Gmax;
  Bmin = boxp->Bmin;  Bmax = boxp->Bmax;

  if (Rmax > Rmin)
    for (R = Rmin; R <= Rmax; R++)
      for (G = Gmin; G <= Gmax; G++)
        {
          histp = histogram + R*MR + G*MG + Bmin;
          for (B = Bmin; B <= Bmax; B++)
            if (*histp++ != 0)
              {
                boxp->Rmin = Rmin = R;
                goto have_Rmin;
              }
        }
 have_Rmin:
  if (Rmax > Rmin)
    for (R = Rmax; R >= Rmin; R--)
      for (G = Gmin; G <= Gmax; G++)
        {
          histp = histogram + R*MR + G*MG + Bmin;
          for (B = Bmin; B <= Bmax; B++)
            if (*histp++ != 0)
              {
                boxp->Rmax = Rmax = R;
                goto have_Rmax;
              }
        }
 have_Rmax:
  if (Gmax > Gmin)
    for (G = Gmin; G <= Gmax; G++)
      for (R = Rmin; R <= Rmax; R++)
        {
          histp = histogram + R*MR + G*MG + Bmin;
          for (B = Bmin; B <= Bmax; B++)
            if (*histp++ != 0)
```

```
            {
              boxp->Gmin = Gmin = G;
              goto have_Gmin;
            }
        }
have_Gmin:
 if (Gmax > Gmin)
   for (G = Gmax; G >= Gmin; G--)
     for (R = Rmin; R <= Rmax; R++)
       {
         histp = histogram + R*MR + G*MG + Bmin;
         for (B = Bmin; B <= Bmax; B++)
           if (*histp++ != 0)
             {
               boxp->Gmax = Gmax = G;
               goto have_Gmax;
             }
       }
have_Gmax:
 if (Bmax > Bmin)
   for (B = Bmin; B <= Bmax; B++)
     for (R = Rmin; R <= Rmax; R++)
       {
         histp = histogram + R*MR + Gmin*MG + B;
         for (G = Gmin; G <= Gmax; G++, histp += MG)
           if (*histp != 0)
             {
               boxp->Bmin = Bmin = B;
               goto have_Bmin;
             }
       }
have_Bmin:
 if (Bmax > Bmin)
   for (B = Bmax; B >= Bmin; B--)
     for (R = Rmin; R <= Rmax; R++)
       {
         histp = histogram + R*MR + Gmin*MG + B;
         for (G = Gmin; G <= Gmax; G++, histp += MG)
           if (*histp != 0)
             {
               boxp->Bmax = Bmax = B;
               goto have_Bmax;
             }
       }
have_Bmax:

 /* Update box volume.
  * We use 2-norm rather than real volume here; this biases the method
  * against making long narrow boxes, and it has the side benefit that
  * a box is splittable iff norm > 0.
  * Since the differences are expressed in histogram-cell units,
  * we have to shift back to JSAMPLE units to get consistent distances;
  * after which, we scale according to the selected distance scale factors.
  */
 dist0 = ((Rmax - Rmin) << R_SHIFT) * R_SCALE;
 dist1 = ((Gmax - Gmin) << G_SHIFT) * G_SCALE;
 dist2 = ((Bmax - Bmin) << B_SHIFT) * B_SCALE;
 boxp->volume = dist0*dist0 + dist1*dist1 + dist2*dist2;

 /* Now scan remaining volume of box and compute population */
 ccount = 0;
 for (R = Rmin; R <= Rmax; R++)
   for (G = Gmin; G <= Gmax; G++)
     {
       histp = histogram + R*MR + G*MG + Bmin;
       for (B = Bmin; B <= Bmax; B++, histp++)
```



```
         if (*histp != 0)
           {
             ccount++;
           }
     }

 boxp->colorcount = ccount;
}

/* Repeatedly select and split the largest box until we have enough boxes */
int median_cut_rgb( int *histogram,     boxptr boxlist, int numboxes, int desired_col
ors)
{
  int n,lb;
  int R,G,B,cmax;
  boxptr b1,b2;

  while (numboxes < desired_colors) {
    /* Select box to split.
     * Current algorithm: by population for first half, then by volume.
     */
    if (numboxes*2 <= desired_colors)
      {
        b1 = find_biggest_color_pop (boxlist, numboxes);
      }
    else
      {
        b1 = find_biggest_volume (boxlist, numboxes);
      }

    if (b1 == NULL)             /* no splittable boxes left! */
      break;
    b2 = boxlist + numboxes;    /* where new box will go */
    /* Copy the color bounds to the new box. */
    b2->Rmax = b1->Rmax; b2->Gmax = b1->Gmax; b2->Bmax = b1->Bmax;
    b2->Rmin = b1->Rmin; b2->Gmin = b1->Gmin; b2->Bmin = b1->Bmin;
    /* Choose which axis to split the box on.
     * Current algorithm: longest scaled axis.
     * See notes in update_box about scaling distances.
     */
    R = ((b1->Rmax - b1->Rmin) << R_SHIFT) * R_SCALE;
    G = ((b1->Gmax - b1->Gmin) << G_SHIFT) * G_SCALE;
    B = ((b1->Bmax - b1->Bmin) << B_SHIFT) * B_SCALE;
    /* We want to break any ties in favor of green, then red, blue last.
     */
    cmax = G; n = 1;
    if (R > cmax) { cmax = R; n = 0; }
    if (B > cmax) { n = 2; }

    /* Choose split point along selected axis, and update box bounds.
     * Current algorithm: split at halfway point.
     * (Since the box has been shrunk to minimum volume,
     * any split will produce two nonempty subboxes.)
     * Note that lb value is max for lower box, so must be < old max.
     */
    switch (n)
      {
      case 0:
        lb = (b1->Rmax + b1->Rmin) / 2;
        b1->Rmax = lb;
        b2->Rmin = lb+1;
        break;
      case 1:
        lb = (b1->Gmax + b1->Gmin) / 2;
        b1->Gmax = lb;
        b2->Gmin = lb+1;
```

```
        break;
      case 2:
        lb = (b1->Bmax + b1->Bmin) / 2;
        b1->Bmax = lb;
        b2->Bmin = lb+1;
        break;
      }
    /* Update stats for boxes */
    update_box_rgb (histogram, b1);
    update_box_rgb (histogram, b2);
    numboxes++;
  }
  return numboxes;
}

/* Find the splittable box with the largest color population */
/* Returns NULL if no splittable boxes remain */
boxptr find_biggest_color_pop( boxptr boxlist, int numboxes)
{
  boxptr boxp;
  int i;
  long maxc = 0;
  boxptr which = NULL;

  for (i = 0, boxp = boxlist; i < numboxes; i++, boxp++)
    {
      if (boxp->colorcount > maxc && boxp->volume > 0)
        {
          which = boxp;
          maxc = boxp->colorcount;
        }
    }

  return which;
}


/* Find the splittable box with the largest (scaled) volume */
/* Returns NULL if no splittable boxes remain */
boxptr find_biggest_volume( boxptr boxlist, int numboxes )
{
  boxptr boxp;
  int i;
  int maxv = 0;
  boxptr which = NULL;

  for (i = 0, boxp = boxlist; i < numboxes; i++, boxp++)
    {
      if (boxp->volume > maxv)
        {
          which = boxp;
          maxv = boxp->volume;
        }
    }

  return which;
}

/* Compute representative color for a box, put it in colormap[icolor] */
void compute_color_rgb( COLOR *color, int *histogram, boxptr boxp, int icolor )
{
  /* Current algorithm: mean weighted by pixels (not colors) */
  /* Note it is important to get the rounding correct! */
  int *histp;
  int R, G, B;
  int Rmin, Rmax;
```



```
  int Gmin, Gmax;
  int Bmin, Bmax;
  long count;
  long total = 0;
  long Rtotal = 0;
  long Gtotal = 0;
  long Btotal = 0;

  Rmin = boxp->Rmin;  Rmax = boxp->Rmax;
  Gmin = boxp->Gmin;  Gmax = boxp->Gmax;
  Bmin = boxp->Bmin;  Bmax = boxp->Bmax;

  for (R = Rmin; R <= Rmax; R++)
    for (G = Gmin; G <= Gmax; G++)
      {
        histp = histogram + R*MR + G*MG + Bmin;
        for (B = Bmin; B <= Bmax; B++)
          {
            if ((count = *histp++) != 0)
              {
                total += count;
                Rtotal += ((R << R_SHIFT) + ((1<<R_SHIFT)>>1)) * count;
                Gtotal += ((G << G_SHIFT) + ((1<<G_SHIFT)>>1)) * count;
                Btotal += ((B << B_SHIFT) + ((1<<B_SHIFT)>>1)) * count;
              }
          }
      }

  if (total != 0)
    {
          color[icolor].red = (Rtotal + (total>>1)) / total;
          color[icolor].green = (Gtotal + (total>>1)) / total;
          color[icolor].blue = (Btotal + (total>>1)) / total;
    }
  else /* The only situation where total==0 is if the image was null or
        *  all-transparent.  In that case we just put a dummy value in
        *  the colourmap.
        */
    {
          color[icolor].red   =
          color[icolor].green =
          color[icolor].blue  = 0 ;
    }
}

void Motion_EventCallback( FL_OBJECT *obj )
{
    int         i ;

    if( fd_Prop->PropPalette->deactivated ) {
        fl_activate_form( fd_Prop->PropPalette ) ;
        return ;
    }

    for( i = 0 ; i < M_Palcnt ; i++ )
        fl_redraw_object( fd_Prop->IndexColor[i] ) ;

    HSVUpDownBtnRedraw() ;

    XFlush( fl_get_display() ) ;
}

void Motion_progress_init( char *title )
{
    int         x, y, w, h ;
```

```
    /* メインダイアログの表示処理を一時的に停止する */
    M_LoopFlag++ ;
    fl_deactivate_form( fd_Prop->PropPalette ) ;

    /* プログレスバーの属性を設定する */
    x = fd_Prop->PropPalette->x + 150 ;
    y = fd_Prop->PropPalette->y + 40 ;
    fl_show_form( fd_Prog->Progress, FL_PLACE_SIZE, FL_FULLBORDER, "Progress" ) ;
    fl_set_form_position( fd_Prog->Progress, x, y ) ;
    fl_set_object_label( fd_Prog->title, title ) ;
    fl_get_object_geometry( fd_Prog->waku, &x, &y, &w, &h ) ;
    fl_set_object_size( fd_Prog->bar, 0, h ) ;
    XFlush( fl_get_display() ) ;
}

void Motion_progress_update( double percent )
{
    int         x, y, w, h ;

    fl_get_object_geometry( fd_Prog->waku, &x, &y, &w, &h ) ;
    w = percent * w ;
    fl_set_object_size( fd_Prog->bar, w, h ) ;
    XFlush( fl_get_display() ) ;
}

void Motion_progress_hide()
{
    fl_hide_form( fd_Prog->Progress ) ;
    /* メインダイアログ表示の一時停止を解除する */
    M_LoopFlag-- ;
}

void Motion_ResizeColorMap( int mode )
{
    int         i, percent, total ;
    int         col_num, indx, dlt ;
    int         x, y, row, col ;
    int         ptbl[12][2], gosa ;
    int         map[5] ;

    if( mode == 0 ) {   /* 新規作成時 */
        total = 0 ;
        col_num = 0 ;
        for( i = 0 ; i < 12 ; i++ ) {
            if( M_Palette[i].color.count ) {
                percent = (M_Palette[i].ratel + 50) / 100 ;
                if( percent == 0 ) percent = 1 ;
                col_num++ ;
            }
            else percent = 0 ;

            ptbl[M_Palette[i].order][0] = percent ;
            ptbl[M_Palette[i].order][1] = i ;
            total += percent ;
        }
        if( (gosa=total-100) ) {
            while( gosa ) {
                for( i = 0 ; i < col_num ; i++ ) {
                    if( gosa < 0 ) {
                        ptbl[i][0]++ ;
                        gosa++ ;
                    }
                    else {
                        if( ptbl[i][0] == 1 ) continue ;
                        ptbl[i][0]-- ;
                        gosa-- ;
```



```
                    }
                    if( !gosa ) break ;
                }
            }
        }
        /* map initalize */
        for( i = 0 ; i < 5 ; i++ ) map[i] = 0 ;

        /* インデックス色を */
        for( i = 0 ; i < col_num ; i++ ) {
            gosa = Motion_mapping( 0, map, ptbl[i][0], &x, &y, &row, &col ) ;
            ptbl[i][0] -= gosa ;
            if( (i+2) == col_num ) ptbl[i+1][0] += gosa ;
            else {
                indx = i + 1 ;
                while( gosa ) {
                    dlt = ptbl[indx][0] ;
                    if( gosa < dlt ) dlt = gosa ;
                    ptbl[indx][0] += dlt ;
                    gosa          -= dlt ;
                    indx++ ;
                    if( indx == col_num ) indx = i + 1 ;
                }
            }
            indx = ptbl[i][1] ;
            M_Palette[indx].x   = x ;
            M_Palette[indx].y   = y ;
            M_Palette[indx].row = row ;
            M_Palette[indx].col = col ;
        }
    }
}

int Motion_mapping( int sa, int *map, int percent, int *x, int *y, int *row, int *col
 )
{
    int         i, cnt, indx=0 ;
    int         pos=0, pos1 ;
    int         level, level1 ;
    CANDI       cand[3] ;

    /* make candidacy table */
    switch( percent ) {
        case 1:                 /* for 1% table */
            cand[0].wide = 1 ;
            cand[0].hi   = 1 ;
            cnt = 1 ;
            break ;
        case 2:                 /* for 2% table */
            cand[0].wide = 2 ;
            cand[0].hi   = 1 ;
            cand[1].wide = 1 ;
            cand[1].hi   = 2 ;
            cnt = 2 ;
            break ;
        case 3:                 /* for 3% table */
            cand[0].wide = 3 ;
            cand[0].hi   = 1 ;
            cand[1].wide = 1 ;
            cand[1].hi   = 3 ;
            cnt = 2 ;
            break ;
        case 4:                 /* for 4% table */
            cand[0].wide = 2 ;
            cand[0].hi   = 2 ;
            cand[1].wide = 4 ;
```

```
        cand[1].hi   = 1 ;
        cnt = 2 ;
        break ;
    case 5:                 /* for 5% table */
        cand[0].wide = 1 ;
        cand[0].hi   = 5 ;
        cand[1].wide = 5 ;
        cand[1].hi   = 1 ;
        cnt = 2 ;
        break ;
    case 6:                 /* for 6% table */
        cand[0].wide = 2 ;
        cand[0].hi   = 3 ;
        cand[1].wide = 3 ;
        cand[1].hi   = 2 ;
        cand[2].wide = 6 ;
        cand[2].hi   = 1 ;
        cnt = 3 ;
        break ;
    case 7:
        cand[0].wide = 7 ;
        cand[0].hi   = 1 ;
        cnt = 1 ;
        break ;
    case 8:                 /* for 8% table */
        cand[0].wide = 4 ;
        cand[0].hi   = 2 ;
        cnt = 1 ;
        break ;
    case 9:                 /* for 9% table */
        cand[0].wide = 3 ;
        cand[0].hi   = 3 ;
        cnt = 1 ;
        break ;
    case 10:                /* for 10% table */
        cand[0].wide = 2 ;
        cand[0].hi   = 5 ;
        cand[1].wide = 5 ;
        cand[1].hi   = 2 ;
        cnt = 2 ;
        break ;
    case 12:                /* for 12% table */
        cand[0].wide = 4 ;
        cand[0].hi   = 3 ;
        cand[1].wide = 6 ;
        cand[1].hi   = 2 ;
        cnt = 2 ;
        break ;
    case 14:                /* for 14% table */
        cand[0].wide = 7 ;
        cand[0].hi   = 2 ;
        cnt = 1 ;
        break ;
    case 15:                /* for 15% table */
        cand[0].wide = 3 ;
        cand[0].hi   = 5 ;
        cand[1].wide = 5 ;
        cand[1].hi   = 3 ;
        cnt = 2 ;
        break ;
    case 16:                /* for 16% table */
        cand[0].wide = 8 ;
        cand[0].hi   = 2 ;
        cnt = 1 ;
        break ;
    case 18:                /* for 18% table */
```



```
        cand[0].wide = 6 ;
        cand[0].hi   = 3 ;
        cand[1].wide = 9 ;
        cand[1].hi   = 2 ;
        cnt = 2 ;
        break ;
    case 20:                /* for 20% table */
        cand[0].wide = 4 ;
        cand[0].hi   = 5 ;
        cand[1].wide = 10 ;
        cand[1].hi   = 2 ;
        cnt = 2 ;
        break ;
    case 21:                /* for 21% table */
        cand[0].wide = 7 ;
        cand[0].hi   = 3 ;
        cnt = 1 ;
        break ;
    case 22:                /* for 22% table */
        cand[0].wide = 11 ;
        cand[0].hi   = 2 ;
        cnt = 1 ;
        break ;
    case 24:                /* for 24% table */
        cand[0].wide = 8 ;
        cand[0].hi   = 3 ;
        cand[1].wide = 12 ;
        cand[1].hi   = 2 ;
        cnt = 2 ;
        break ;
    case 25:                /* for 25% table */
        cand[0].wide = 5 ;
        cand[0].hi   = 5 ;
        cnt = 1 ;
        break ;
    case 26:                /* for 26% table */
        cand[0].wide = 13 ;
        cand[0].hi   = 2 ;
        cnt = 1 ;
        break ;
    case 27:                /* for 27% table */
        cand[0].wide = 9 ;
        cand[0].hi   = 3 ;
        cnt = 1 ;
        break ;
    case 28:                /* for 28% table */
        cand[0].wide = 14 ;
        cand[0].hi   = 2 ;
        cnt = 1 ;
        break ;
    case 30:                /* for 30% table */
        cand[0].wide = 6 ;
        cand[0].hi   = 5 ;
        cand[1].wide = 10 ;
        cand[1].hi   = 3 ;
        cand[2].wide = 15 ;
        cand[2].hi   = 2 ;
        cnt = 3 ;
        break ;
    case 32:                /* for 32% table */
        cand[0].wide = 16 ;
        cand[0].hi   = 2 ;
        cnt = 1 ;
        break ;
    case 33:                /* for 33% table */
        cand[0].wide = 11 ;
```

```
            cand[0].hi   = 3 ;
            cnt = 1 ;
            break ;
        case 34:                /* for 34% table */
            cand[0].wide = 17 ;
            cand[0].hi   = 2 ;
            cnt = 1 ;
            break ;
        case 35:                /* for 35% table */
            cand[0].wide = 7 ;
            cand[0].hi   = 5 ;
            cnt = 1 ;
            break ;
        case 36:                /* for 36% table */
            cand[0].wide = 12 ;
            cand[0].hi   = 3 ;
            cand[1].wide = 18 ;
            cand[1].hi   = 2 ;
            cnt = 2 ;
            break ;
        case 38:                /* for 38% table */
            cand[0].wide = 19 ;
            cand[0].hi   = 2 ;
            cnt = 1 ;
            break ;
        case 39:                /* for 39% table */
            cand[0].wide = 13 ;
            cand[0].hi   = 3 ;
            cnt = 1 ;
            break ;
        case 40:                /* for 40% table */
            cand[0].wide = 8 ;
            cand[0].hi   = 5 ;
            cand[1].wide = 20 ;
            cand[1].hi   = 2 ;
            cnt = 2 ;
            break ;
        case 42:                /* for 42% table */
            cand[0].wide = 14 ;
            cand[0].hi   = 3 ;
            cnt = 1 ;
            break ;
        case 45:                /* for 45% table */
            cand[0].wide = 9 ;
            cand[0].hi   = 5 ;
            cand[1].wide = 15 ;
            cand[1].hi   = 3 ;
            cnt = 2 ;
            break ;
        case 48:                /* for 48% table */
            cand[0].wide = 16 ;
            cand[0].hi   = 3 ;
            cnt = 1 ;
            break ;
        case 50:                /* for 50% table */
            cand[0].wide = 10 ;
            cand[0].hi   = 5 ;
            cnt = 1 ;
            break ;
        case 51:                /* for 51% table */
            cand[0].wide = 17 ;
            cand[0].hi   = 3 ;
            cnt = 1 ;
            break ;
        case 54:                /* for 54% table */
            cand[0].wide = 18 ;
```



```
            cand[0].hi   = 3 ;
            cnt = 1 ;
            break ;
        case 55:                /* for 55% table */
            cand[0].wide = 11 ;
            cand[0].hi   = 5 ;
            cnt = 1 ;
            break ;
        case 57:                /* for 57% table */
            cand[0].wide = 19 ;
            cand[0].hi   = 3 ;
            cnt = 1 ;
            break ;
        case 60:                /* for 60% table */
            cand[0].wide = 12 ;
            cand[0].hi   = 5 ;
            cand[1].wide = 20 ;
            cand[1].hi   = 3 ;
            cnt = 2 ;
            break ;
        case 65:                /* for 65% table */
            cand[0].wide = 13 ;
            cand[0].hi   = 5 ;
            cnt = 1 ;
            break ;
        case 70:                /* for 70% table */
            cand[0].wide = 14 ;
            cand[0].hi   = 5 ;
            cnt = 1 ;
            break ;
        case 75:                /* for 75% table */
            cand[0].wide = 15 ;
            cand[0].hi   = 5 ;
            cnt = 1 ;
            break ;
        case 80:                /* for 80% table */
            cand[0].wide = 16 ;
            cand[0].hi   = 5 ;
            cnt = 1 ;
            break ;
        case 85:                /* for 85% table */
            cand[0].wide = 17 ;
            cand[0].hi   = 5 ;
            cnt = 1 ;
            break ;
        case 90:                /* for 90% table */
            cand[0].wide = 18 ;
            cand[0].hi   = 5 ;
            cnt = 1 ;
            break ;
        case 95:                /* for 95% table */
            cand[0].wide = 19 ;
            cand[0].hi   = 5 ;
            cnt = 1 ;
            break ;
        case 100:               /* for 100% table */
            cand[0].wide = 20 ;
            cand[0].hi   = 5 ;
            cnt = 1 ;
            break ;
        default:
            cnt = 0 ;
            break ;
    }

    /* mapping process */
```

```
    level = 5 ;
    for( i = 0 ; i < cnt ; i++ ) {
        /* check step level */
        level1 = Motion_step_level( map, cand[i].wide, cand[i].hi, &pos1 ) ;
        if( (level1 == -1) || (level <= level1) ) continue ;

        level = level1 ;
        indx  = i ;
        pos   = pos1 ;
        if( level == 0 ) break ;
    }

    if( level == 5 ) {
        percent-- ;
        sa++ ;
        sa = Motion_mapping( sa, map, percent, x, y, row, col ) ;
    }
    else {
        *x   = map[pos] ;
        *y   = pos ;
        *row = cand[indx].hi ;
        *col = cand[indx].wide ;
        for( i = pos ; i < (pos+cand[indx].hi) ; i++ )
                                                map[i] += cand[indx].wide ;
    }

    return sa ;
}

int Motion_step_level( int *map, int wide, int hi, int *pos )
{
    int         i ;
    int         level, crank ;

    /*  hight == 5 */
    if( hi == 5 ) {
        if( (20-map[0]) < wide ) return -1 ;
        if( (map[0] != map[1]) || (map[1] != map[2]) ||
            (map[2] != map[3]) || (map[3] != map[4]) ) return -1 ;

        *pos = 0 ;
        return 0 ;
    }

    /* hight == 3 */
    if( hi == 3 ) {
        level = -1 ;
        for( i = 0 ; i < 3 ; i++ ) {
            if( (20-map[i]) < wide ) continue ;
            if( (map[i] != map[i+1]) || (map[i+1] != map[i+2]) ) continue ;

            crank = Motion_get_crank( map, i, wide, i+hi ) ;
            if( (level == -1) || (crank < level) ) {
                level = crank ;
                *pos  = i ;
            }
        }
        return level ;
    }

    /* hight == 2 */
    if( hi == 2 ) {
        level = -1 ;
        if( wide <= (20-map[3]) ) {
            if( map[3] == map[4] ) {
                level = Motion_get_crank( map, 3, wide, 5 ) ;
```



```
                *pos  = 3 ;
            }
        }
        for( i = 0 ; i < 2 ; i++ ) {
            if( (20-map[i]) < wide ) continue ;
            if( map[i] != map[i+1] ) continue ;

            crank = Motion_get_crank( map, i, wide, i+hi ) ;
            if( (level == -1) || (crank < level) ) {
                level = crank ;
                *pos  = i ;
            }
        }
        return level ;
    }

    /* hight == 1 */
    if( hi == 1 ) {
        level = -1 ;
        for( i = 0 ; i < 5 ; i++ ) {
            if( (20-map[i]) < wide ) continue ;

            crank = Motion_get_crank( map, i, wide, i+hi ) ;
            if( (level == -1) || (crank < level) ) {
                level = crank ;
                *pos  = i ;
            }
        }
        return level ;
    }

    return -1 ;
}

int Motion_get_crank( int *map, int pos, int wide, int end )
{
    int         i, l_map[5], crank ;

    for( i = 0 ; i < 5 ; i++ ) {
        if( pos <= i && i < end ) l_map[i] = map[i] + wide ;
        else                      l_map[i] = map[i] ;
    }

    crank = 0 ;
    for( i = 0 ; i < 4 ; i++ )
        if( l_map[i] != l_map[i+1] ) crank++ ;

    return crank ;
}
```

```
/* Form definition file generated with fdesign. */

#include "forms.h"
#include <stdlib.h>
#include "PropPalette.h"
#include "libgimp/gimp.h"
#include "Motion.h"

FD_PropPalette *create_form_PropPalette(void)
{
  int           i ;
  FL_OBJECT     *obj;
  FD_PropPalette *fdui = (FD_PropPalette *) fl_calloc(1, sizeof(*fdui));

  fdui->PropPalette = fl_bgn_form(FL_NO_BOX, 666, 140);
  obj = fl_add_box(FL_UP_BOX,0,0,666,140,"");
  fdui->New = obj = fl_add_button(FL_NORMAL_BUTTON,10,10,50,20,"NEW");
    fl_set_object_lsize(obj,FL_TINY_SIZE);
    fl_set_object_lstyle(obj,13);
    fl_set_object_callback(obj,NewBtnProc,0);
  fdui->Load = obj = fl_add_button(FL_NORMAL_BUTTON,10,35,50,20,"LOAD");
    fl_set_object_lsize(obj,FL_TINY_SIZE);
    fl_set_object_lstyle(obj,13);
    fl_set_object_callback(obj,LoadBtnProc,0);
  fdui->Save = obj = fl_add_button(FL_NORMAL_BUTTON,10,60,50,20,"SAVE");
    fl_set_object_lsize(obj,FL_TINY_SIZE);
    fl_set_object_lstyle(obj,13);
    fl_set_object_callback(obj,SaveBtnProc,0);
    fl_deactivate_object( obj ) ;
  fdui->Levels = obj = fl_add_button(FL_NORMAL_BUTTON,10,85,50,20,"LEVELS");
    fl_set_object_lsize(obj,FL_TINY_SIZE);
    fl_set_object_lstyle(obj,13);
    fl_set_object_callback(obj,LevelsBtnProc,0);
    fl_deactivate_object( obj ) ;
  fdui->Help = obj = fl_add_button(FL_NORMAL_BUTTON,10,110,50,20,"HELP");
    fl_set_object_lsize(obj,FL_TINY_SIZE);
    fl_set_object_lstyle(obj,13);
    fl_set_object_callback(obj,HelpBtnProc,0);

  /* カラーエディタ部のオブジェクト生成 */
  fdui->BrightnessDown = obj = fl_add_button(FL_TOUCH_BUTTON,57,52,36,36,"@4UpArrow")
;
    fl_set_object_boxtype(obj,FL_NO_BOX);
    fl_set_object_color(obj,FL_RED,FL_COL1);
    fl_set_object_lcolor(obj,FL_RED);
    fl_set_object_callback(obj,BrightnessDownBtnProc,0);
  fdui->BrightnessUp = obj = fl_add_button(FL_NORMAL_BUTTON,138,52,36,36,"@UpArrow");
    fl_set_object_boxtype(obj,FL_NO_BOX);
    fl_set_object_callback(obj,BrightnessUpBtnProc,0);
  fdui->SatUp = obj = fl_add_button(FL_NORMAL_BUTTON,97,12,36,36,"@8UpArrow");
    fl_set_object_lcol(obj,FL_YELLOW);
    fl_set_object_boxtype(obj,FL_NO_BOX);
    fl_set_object_callback(obj,SatUpBtnProc,0);
  fdui->SatDown = obj = fl_add_button(FL_NORMAL_BUTTON,97,92,36,36,"@2UpArrow");
    fl_set_object_boxtype(obj,FL_NO_BOX);
    fl_set_object_callback(obj,SatDownBtnProc,0);
  fdui->SelectColor = obj = fl_add_box(FL_OVAL3D_UPBOX,90,45,50,50,"");
    fl_set_object_color(obj,M_SEL_COLOR,FL_COL1);
  fdui->EditColor = obj = fl_add_box(FL_RFLAT_BOX,101,56,30,30,"");
    fl_set_object_color(obj,M_EDIT_COLOR,FL_COL1);

  /* パレット部のオブジェクト生成 */
  fdui->PaletteFrame = obj = fl_add_frame(FL_ENGRAVED_FRAME,174,9,482,122,"");
  for( i = 0 ; i < 12 ; i++ ) {
    fdui->IndexBtn[i] =
    obj = fl_add_button( FL_NORMAL_BUTTON, 0, 0, 0, 0, "E" ) ;
```



```
    obj->u_ldata = i ;
    fl_set_object_boxtype( obj, FL_FRAME_BOX ) ;
    fl_set_object_callback( obj, IndexBtnProc, 0 ) ;
    fl_hide_object( obj ) ;

    fdui->IndexColor[i] =
    obj = fl_add_free( FL_NORMAL_FREE, 0, 0, 0, 0, "", IndexColorProc ) ;
    obj->u_ldata = i ;
    fl_hide_object( obj ) ;
  }

  fl_end_form();

  fdui->PropPalette->fdui = fdui;

  return fdui;
}
/*---------------------------------------*/

FD_Progress *create_form_Progress(void)
{
  FL_OBJECT *obj;
  FD_Progress *fdui = (FD_Progress *) fl_calloc(1, sizeof(*fdui));

  fdui->Progress = fl_bgn_form(FL_NO_BOX, 276, 70);
  obj = fl_add_box(FL_UP_BOX,0,0,276,105,"");
  fdui->title = obj = fl_add_text(FL_NORMAL_TEXT,5,5,250,20,"Create M-motion palette.
..");
    fl_set_object_lalign(obj,FL_ALIGN_LEFT|FL_ALIGN_INSIDE);
  fdui->waku = obj = fl_add_frame(FL_DOWN_FRAME,10,35,250,20,"");
  fdui->bar = obj = fl_add_box(FL_FLAT_BOX,10,35,96,20,"");
    fl_set_object_color(obj,FL_BLUE,FL_COL1);
  fl_end_form();

  fdui->Progress->fdui = fdui;

  return fdui;
}
/*---------------------------------------*/
```

```
#include "forms.h"
#include "PropPalette.h"
#include "gtk/gtk.h"
#include "libgimp/gimp.h"
#include "Motion.h"

/* callbacks and freeobj handles for form PropPalette */
#define COL_BORDER      1
#define COL_BLKSIZE     24
#define RATE2DIV        20000

int GetWindowPos( int ) ;
Status SearchWindowPos( char*, Window, int* ) ;

void NewBtnProc(FL_OBJECT *ob, long data)
{
    int         i, count, WinNo, TNo ;
    gint32      *imageid, ImgID ;
    gint32      DrawableID ;
    GParam      *vals ;

    /* 現在開いているイメージ数を調べ、最上位にあるイメージのIDを求める */
    imageid = gimp_query_images( &count ) ;
    if( count == 0 ) return ;
    if( count == 1 ) ImgID = imageid[0] ;
    else {
        WinNo   = -1 ;
        ImgID   = imageid[0] ;
        for( i = 0 ; i < count ; i++ ) {
            TNo = GetWindowPos( imageid[i] ) ;
            if( TNo < WinNo ) continue ;
            WinNo = TNo ;
            ImgID = imageid[i] ;
        }
    }

    /* 求めたイメージIDのDrawableIDを求める */
    vals = gimp_run_procedure( "gimp_image_active_drawable", &count,
                                        PARAM_IMAGE, ImgID, PARAM_END ) ;
    if( vals[0].data.d_status != STATUS_SUCCESS ) {
        gimp_destroy_params( vals, count ) ;
        return ;
    }
    gimp_destroy_params( vals, count ) ;
    DrawableID = vals[1].data.d_int32 ;

    /* create index color palette */
    PixelDataReadAndAnalyze( 0, DrawableID ) ;
    if( M_Palcnt == 0 ) {
        fprintf( stdout, "palette ZERO \n" ) ;
        return ;
    }

    Motion_ResizeColorMap( 0 ) ;

    /* パレットウィンドウをリサイズして再表示する */
    PaletteWndResize() ;
}

void LoadBtnProc(FL_OBJECT *ob, long data)
{
    int         no, r, g, b, ord, rate ;
    int         x, y, row, col, max_no ;
    char        buf[129], num[10] ;
    FILE        *fp ;
    const char  *fname ;
```



```
    M_LoopFlag++ ;
    fl_deactivate_form( fd_Prop->PropPalette ) ;
    if((fname = fl_show_file_selector( "Load a Palette file", 0, "*", 0 ) ) ) {
        XFlush(fl_get_display());
        if( (fp = fopen( fname, "r" )) ) {
            /* ヘッダー部読み込み */
            if( !fgets( buf, 128, fp ) ) {
                fl_show_alert( "Load", fname, "Read File Error", 0 ) ;
                goto LOAD_NEXT ;
            }
            if( strncmp( buf, M_FILE_HEADER, strlen(M_FILE_HEADER) ) ) {
                fl_show_alert( "Load", fname, "Illegal Data File", 0 ) ;
                goto LOAD_NEXT ;
            }

            /* パレットデータ部読み込み */
            max_no = 0 ;
            while( fgets( buf, 128, fp ) ) {
                sscanf( buf, "%s %d %d %d %d %d %d %d %d %d",
                        num, &r, &g, &b, &ord, &rate, &x, &y, &row, &col ) ;
                no = atoi( &(num[3]) ) - 1 ;
                M_Palette[no].color.red   = r ;
                M_Palette[no].color.green = g ;
                M_Palette[no].color.blue  = b ;
                M_Palette[no].rate1       = rate ;
                M_Palette[no].order       = ord ;
                M_Palette[no].x           = x ;
                M_Palette[no].y           = y ;
                M_Palette[no].row         = row ;
                M_Palette[no].col         = col ;
                if( max_no < no ) max_no = no ;
            }
            M_Palcnt = max_no + 1 ;
            /* 残りのパレットを無効（初期）にする */
            for( no = M_Palcnt ; no < 12 ; no++ ) {
                M_Palette[no].rate1 = 0 ;
                M_Palette[no].order = no ;
            }

            /* パレットウィンドウをリサイズして再表示する */
            PaletteWndResize() ;
LOAD_NEXT:
            fclose( fp ) ;
        }
        else fl_show_alert( "Load", fname, "Can't File open", 0 ) ;
    }
    M_LoopFlag-- ;
}

void SaveBtnProc(FL_OBJECT *ob, long data)
{
    FILE        *fp ;
    int         i ;
    char        buf[129] ;
    const char  *fname ;

    M_LoopFlag++ ;
    fl_deactivate_form( fd_Prop->PropPalette ) ;
    if((fname = fl_show_file_selector( "Save a Palette file", 0, "*", 0 ) ) ) {
        /* 既存ファイルがあるか調べる */
        if( (fp = fopen( fname, "r" )) ) {
            if( !fl_show_question( "Specified File Exist, Override OK ?", 1 ) )
                goto SAVE_NEXT ;
            fclose( fp ) ;
        }
```

```
        /* 改めてファイルをオープンする */
        if( (fp = fopen( fname, "w" )) ) {
            /* ヘッダー部書き込み */
            sprintf( buf, "%s\n", M_FILE_HEADER ) ;
            if( !fputs( buf, fp ) ) {
                fl_show_alert( "Save", fname, "Write File Error", 0 ) ;
                goto SAVE_NEXT ;
            }

            /* パレットデータ部書き込み */
            for( i = 0 ; i < 12 ; i++ ) {
                if( M_Palette[i].color.count == 0 ) continue ;
                sprintf( buf, "No.%d %d %d %d %d %d %d %d %d %d\n",
                        i+1,
                        M_Palette[i].color.red,
                        M_Palette[i].color.green,
                        M_Palette[i].color.blue,
                        M_Palette[i].order,
                        M_Palette[i].rate1,
                        M_Palette[i].x,
                        M_Palette[i].y,
                        M_Palette[i].row,
                        M_Palette[i].col ) ;
                if( !fputs( buf, fp ) ) {
                    fl_show_alert( "Save", fname, "Write File Error", 0 ) ;
                    goto SAVE_NEXT ;
                }
            }
        }
SAVE_NEXT:
        fclose( fp ) ;
    }
    M_LoopFlag-- ;
}

/* 最新イメージのインデックスカラーに対する使用比率を調べる */
void LevelsBtnProc(FL_OBJECT *ob, long data)
{
    int         indx, x, y, w, h ;
    int         dlt_w, dlt_h ;
    int         i, count, WinNo, TNo ;
    gint32      *imageid, ImgID ;
    gint32      DrawableID ;
    GParam      *vals ;

    if( M_Palcnt == 0 ) return ;

    /* 現在開いているイメージ数を調べ、最上位にあるイメージのIDを求める */
    imageid = gimp_query_images( &count ) ;
    if( count == 0 ) return ;
    if( count == 1 ) ImgID = imageid[0] ;
    else {
        WinNo   = -1 ;
        ImgID   = imageid[0] ;
        for( i = 0 ; i < count ; i++ ) {
            TNo = GetWindowPos( imageid[i] ) ;
            if( TNo < WinNo ) continue ;
            WinNo = TNo ;
            ImgID = imageid[i] ;
        }
    }

    /* 求めたイメージIDのDrawableIDを求める */
    vals = gimp_run_procedure( "gimp_image_active_drawable", &count,
                                        PARAM_IMAGE, ImgID, PARAM_END ) ;
```



```
    if( vals[0].data.d_status != STATUS_SUCCESS ) {
        gimp_destroy_params( vals, count ) ;
        return ;
    }
    gimp_destroy_params( vals, count ) ;
    DrawableID = vals[1].data.d_int32 ;

    /* インデックスカラーに対する使用比率を調べる */
    PixelDataReadAndAnalyze( 1, DrawableID ) ;

    /* パレットウィンドウをリサイズする */
    for( i = 0 ; i < 12 ; i++ ) {
        indx = M_Palette[i].order ;
        fl_get_object_geometry( fd_Prop->IndexBtn[indx], &x, &y, &w, &h ) ;
        dlt_w = (w-4) * M_Palette[i].rate2 / RATE2DIV ;
        dlt_h = (h-4) * M_Palette[i].rate2 / RATE2DIV ;
        x     = x + dlt_w + 2 ;
        y     = y + dlt_h + 2 ;
        w     = w - dlt_w * 2 - 4 ;
        h     = h - dlt_h * 2 - 4 ;
        fl_set_object_geometry( fd_Prop->IndexColor[indx], x, y, w, h ) ;
    }
}

void HelpBtnProc(FL_OBJECT *ob, long data)
{
  /* fill-in code for callback */
}

void BrightnessDownBtnProc( FL_OBJECT *ob, long data )
{
    if( M_ValMin < M_CurVal ) {
        M_CurVal-- ;
        HSVEditColorSet() ;
    }
}

void BrightnessUpBtnProc(FL_OBJECT *ob, long data)
{
    if( M_CurVal < M_ValMax ) {
        M_CurVal++ ;
        HSVEditColorSet() ;
    }
}

void SatDownBtnProc(FL_OBJECT *ob, long data)
{
    if( M_SatMin < M_CurSat ) {
        M_CurSat-- ;
        HSVEditColorSet() ;
    }
}

void SatUpBtnProc(FL_OBJECT *ob, long data)
{
    if( M_CurSat < M_SatMax ) {
        M_CurSat++ ;
        HSVEditColorSet() ;
    }
}

void HSVEditColorSet()
{
    int         red, green, blue ;

    Color_HSVtoRGB( M_CurHue, M_CurSat, M_CurVal, &red, &green, &blue ) ;
```

```
    fl_mapcolor( M_EDIT_COLOR, red, green, blue ) ;
    fl_redraw_object( fd_Prop->EditColor ) ;
    gimp_palette_set_foreground( red, green, blue ) ;
}

void HSVUpDownBtnRedraw()
{
    int         i, col ;
    FL_POINT    po[4] ;
    FL_OBJECT   *obj ;

    col = FL_BLACK ;
    for( i = 0 ; i < 4 ; i++ ) {
        switch( i ) {
            case 0:
                col = M_UP_COLOR ;
                obj = fd_Prop->SatUp ;
                po[0].x = obj->x + obj->w / 2 ;
                po[0].y = obj->y + obj->h / 4 ;
                po[1].x = obj->x + obj->w / 4 ;
                po[1].y = obj->y + obj->h * 3 / 4 ;
                po[2].x = obj->x + obj->w * 3 / 4 ;
                po[2].y = obj->y + obj->h * 3 / 4 ;
                break ;
            case 1:
                col = FL_WHITE ;
                obj = fd_Prop->BrightnessUp ;
                po[0].x = obj->x + obj->w * 3 / 4 ;
                po[0].y = obj->y + obj->h / 2 ;
                po[1].x = obj->x + obj->w / 4 ;
                po[1].y = obj->y + obj->h / 4 ;
                po[2].x = obj->x + obj->w / 4 ;
                po[2].y = obj->y + obj->h * 3 / 4 ;
                break ;
            case 2:
                col = M_DOWN_COLOR ;
                obj = fd_Prop->SatDown ;
                po[0].x = obj->x + obj->w / 2 ;
                po[0].y = obj->y + obj->h * 3 / 4 ;
                po[1].x = obj->x + obj->w * 3 / 4 ;
                po[1].y = obj->y + obj->h / 4 ;
                po[2].x = obj->x + obj->w / 4 ;
                po[2].y = obj->y + obj->h / 4 ;
                break ;
            case 3:
                col = FL_BLACK ;
                obj = fd_Prop->BrightnessDown ;
                po[0].x = obj->x + obj->w / 4 ;
                po[0].y = obj->y + obj->h / 2 ;
                po[1].x = obj->x + obj->w * 3 / 4 ;
                po[1].y = obj->y + obj->h * 3 / 4 ;
                po[2].x = obj->x + obj->w * 3 / 4 ;
                po[2].y = obj->y + obj->h / 4 ;
                break ;
        }
        fl_polyf( po, 3, col ) ;
    }

}

void IndexBtnProc( FL_OBJECT *ob, long data )
{
    int         id, i ;

    id = (int)ob->u_ldata ;
```



```
    for( i = 0 ; i < 12 ; i++ ) {
        if( M_Palette[i].order != id ) continue ;

        M_CurrentID = i ;
        Set_MapColor_CEdit( M_Palette[i].color.red,
                            M_Palette[i].color.green, M_Palette[i].color.blue ) ;
        break ;
    }
}

int IndexColorProc( FL_OBJECT *ob, int ev, FL_Coord mx, FL_Coord my, int key, void *x
ev)
{
    int         id ;

    if( ev == FL_PUSH ) {
        id = (int)ob->u_ldata ;
        IndexBtnProc( fd_Prop->IndexBtn[id], ob->u_ldata ) ;
    }
    else {
        /* free obj handler code */
        fl_rectf( ob->x, ob->y, ob->w, ob->h, ob->col1 ) ;
    }

    return 0;
}

void PaletteWndResize()
{
    int         i, indx ;
    int         x, y, w, h ;
    int         xp, yp, wide, hi ;

    /* パレットウィンドウをリサイズする */
    fl_winset( fd_Prop->PropPalette->window ) ;
    fl_freeze_form( fd_Prop->PropPalette ) ;
    fl_get_object_geometry( fd_Prop->PaletteFrame, &x, &y, &w, &h ) ;
    for( i = 0 ; i < 12 ; i++ ) {
        indx = M_Palette[i].order ;
        fl_mapcolor( FL_FREE_COL1+indx, M_Palette[i].color.red,
                M_Palette[i].color.green, M_Palette[i].color.blue ) ;
        xp   = x + M_Palette[i].x * COL_BLKSIZE + COL_BORDER ;
        yp   = y + M_Palette[i].y * COL_BLKSIZE + COL_BORDER ;
        wide = M_Palette[i].col * COL_BLKSIZE ;
        hi   = M_Palette[i].row * COL_BLKSIZE ;
        fl_set_object_geometry( fd_Prop->IndexBtn[indx], xp, yp, wide, hi ) ;
        fl_set_object_geometry( fd_Prop->IndexColor[indx], xp+2, yp+2, wide-4, hi-4 )
;
    }

    /* ボタンをアクティブにする */
    fl_activate_object( fd_Prop->Save ) ;
    fl_activate_object( fd_Prop->Levels ) ;

    /* 有効なパレットを表示する */
    for( i = 0 ; i < 12 ; i++ ) {
        if( i < M_Palcnt ) {
            fl_show_object( fd_Prop->IndexBtn[i] ) ;
            fl_show_object( fd_Prop->IndexColor[i] ) ;
        }
        else {
            fl_hide_object( fd_Prop->IndexBtn[i] ) ;
            fl_hide_object( fd_Prop->IndexColor[i] ) ;
        }
        fl_set_object_color( fd_Prop->IndexColor[i], FL_FREE_COL1+i, FL_MCOL ) ;
    }
```

```
    /* １番目のカラーをエディットカラー部に設定する */
    IndexBtnProc( fd_Prop->IndexBtn[0], 0 ) ;

    fl_unfreeze_form( fd_Prop->PropPalette ) ;
}

/* 指定イメージのIDよりファイル名を求め、ファイル名をキーにして */
/* ウィンドウの位置（Windowsで言う所のZオーダ）を調べる。 */
int GetWindowPos( int ID )
{
    int         No ;
    char        str[256], *token, *name ;
    Status      stat ;

    /* ファイル名を取得する */
    strcpy( str,gimp_image_get_filename( ID ) ) ;
    name  = str ;
    token = strtok( str, "/" ) ;
    while( token ) {
        name  = token ;
        token = strtok( NULL, "/" ) ;
    }

    /* ファイル名をキーにしてウィンドウの位置を求める */
    No   = 0 ;
    stat = SearchWindowPos( name, XDefaultRootWindow(fl_display), &No ) ;

    return stat ? No : -1 ;
}

Status SearchWindowPos( char *name, Window win, int *No )
{
    int         len, i, count ;
    char        *string ;
    Window      root, parent, *child ;
    Status      stat ;

    stat = XQueryTree( fl_display, win, &root, &parent, &child, &count ) ;
    if( stat ) {
        XFetchName( fl_display, win, &string ) ;
        if( string ) {
            len = strlen(name) ;
            if( strncmp( name, string, len ) == 0 ) {
                XFree( string ) ;
                XFree( child ) ;
                return 1 ;
            }
        }
        XFree( string ) ;
        for( i = 0 ; i < count ; i++ ) {
            *No = *No + 1 ;
            stat = SearchWindowPos( name, child[i], No ) ;
            if( stat ) return 1 ;
        }
        XFree( child ) ;
    }
    return 0 ;
}
```

```
/*============================================================
 * File     : PropPalette_sub.c
 * Date     : 1999/1/12
 * Author   : Y.Kita
 * Overview :
 *============================================================*/

#include <stdio.h>
#include <math.h>
#include "forms.h"
#include "libgimp/gimp.h"
#include "Motion.h"

/* カラーエディタ部のマップカラーを設定する */
/* 矢印ボタンの表示も同時に行う             */
void Set_MapColor_CEdit( int r, int g, int b )
{
    int         hue, sat, val ;
    int         red, green, blue ;

    fl_mapcolor( M_SEL_COLOR, r, g, b ) ;
    Color_RGBtoHSV( r, g, b, &hue, &sat, &val ) ;
    M_CurHue = hue ;
    M_CurSat = sat ;
    M_SatMin = ((sat-10)<0) ? 0 : sat-10 ;
    M_SatMax = ((sat+10)>100) ? 100 : sat+10 ;
    M_CurVal = val ;
    M_ValMin = ((val-10)<0) ? 0 : val-10 ;
    M_ValMax = ((val+10)>100) ? 100 : val+10 ;
    Color_HSVtoRGB( hue, 100, val, &red, &green, &blue ) ;
    fl_mapcolor( M_UP_COLOR, red, green, blue ) ;
    Color_HSVtoRGB( hue, 0, val, &red, &green, &blue ) ;
    fl_mapcolor( M_DOWN_COLOR, red, green, blue ) ;

    HSVEditColorSet() ;
    HSVUpDownBtnRedraw() ;
    fl_redraw_object( fd_Prop->SelectColor ) ;
    fl_redraw_object( fd_Prop->EditColor ) ;
}

/* RGBをHSVに変換する */
void Color_RGBtoHSV( int red, int blue, int green, int *hue, int *sat, int *val )
{
    int         min, max ;
    int         delta ;
    float       h, s, v ;

    if( red > green ) {
        if( red > blue ) max = red ;
        else             max = blue ;

        if( green < blue ) min = green ;
        else               min = blue ;
    }
    else {
        if( green > blue ) max = green ;
        else               max = blue ;

        if( red < blue ) min = red ;
        else             min = blue ;
    }

    v = max ;

    if( max != 0 ) s = (max - min) / (float)max ;
    else           s = 0 ;
```



```
    if( s == 0 ) h = 0 ;
    else {
        h = 0 ;
        delta = max - min ;
        if( red == max ) h = (green - blue) / (float)delta ;
        else if( green == max ) h = 2 + (blue - red) / (float)delta ;
        else if( blue == max ) h = 4 + (red - green) / (float)delta ;

        h *= 60 ;
        if( h < 0 ) h += 360 ;
        h = 360 - h ;
    }

    *hue = h ;
    *sat = s * 100 ;
    *val = v * 100 / 255 ;
}

/* HSVをRGBに変換する */
void Color_HSVtoRGB( int hue, int sat, int val, int *red, int *green, int *blue )
{
    float       h, s, v ;
    float       f, p, q, t ;

    h = hue ;
    s = sat / 100.0 ;
    v = val / 100.0 ;
    if( s == 0 ) {
        *red   = v * 255 ;
        *green = v * 255 ;
        *blue  = v * 255 ;
    }
    else {
        if( h == 360 ) h = 0 ;

        h /= 60 ;
        f = h - (int)h ;
        p = v * (1 - s) ;
        q = v * (1 - (s * f)) ;
        t = v * (1 - (s * (1 -f))) ;

        switch( (int)h ) {
            case 0:
                *red   = v * 255 ;
                *green = t * 255 ;
                *blue  = p * 255 ;
                break ;
            case 1:
                *red   = q * 255 ;
                *green = v * 255 ;
                *blue  = p * 255 ;
                break ;
            case 2:
                *red   = p * 255 ;
                *green = v * 255 ;
                *blue  = t * 255 ;
                break ;
            case 3:
                *red   = p * 255 ;
                *green = q * 255 ;
                *blue  = v * 255 ;
                break ;
            case 4:
                *red   = t * 255 ;
                *green = p * 255 ;
```

```
                *blue  = v * 255 ;
                break ;
            case 5:
                *red   = v * 255 ;
                *green = p * 255 ;
                *blue  = q * 255 ;
                break ;
        }
    }
}

/* ２つのＲＧＢよりＮＢＳ色差を求める */
float Color_DiffFromRGB( red1, green1, blue1, red2, green2, blue2 )
{
    int         hue1, sat1, val1 ;
    int         hue2, sat2, val2 ;
    float       h1, v1, c1 ;
    float       h2, v2, c2 ;
    float       dltE, dltH, dltV, dltC ;
    float       w1, w2 ;

    /* ＲＧＢをＨＳＶに変換する */
    Color_RGBtoHSV( red1, green1, blue1, &hue1, &sat1, &val1 ) ;
    Color_RGBtoHSV( red2, green2, blue2, &hue2, &sat2, &val2 ) ;

    /* Ｇｏｄｌｏｖｅ色差を求める */
    h1 = M_PI * (float)hue1 / 180.0 ;
    v1 = (float)val1 / 100.0 ;
    c1 = (float)sat1 / 100.0 ;
    h2 = M_PI * (float)hue2 / 180.0 ;
    v2 = (float)val2 / 100.0 ;
    c2 = (float)sat2 / 100.0 ;

    dltH = fabs((h1-h2)) ;
    dltV = fabs((v1-v2)) ;
    dltC = fabs((c1-c2)) ;
    w1   = dltC * dltC ;
    w2   = 16 * dltV * dltV ;
    dltE = sqrt(2 * c1 * c2 * (1 - cos(2*M_PI*dltH/100)) + w1 + w2) ;

    return (1.2 * dltE) ;
}
```

```
#ifndef _MOTION_H_
#define _MOTION_H_

#include "PropPalette.h"

/* 定数定義 */
#define M_SHM_KEY        2222
#define M_FG_COLOR       FL_FREE_COL1+20
#define M_SEL_COLOR      FL_FREE_COL1+21
#define M_EDIT_COLOR     FL_FREE_COL1+22
#define M_UP_COLOR       FL_FREE_COL1+23
#define M_DOWN_COLOR     FL_FREE_COL1+24
#define M_FILE_HEADER    "GIMP Plug-ins:M-motion Palette Data File"
#define M_BOUNDS_CDIFF   1.5

/* 構造体宣言 */
typedef struct Color {
    int          red ;
    int          green ;
    int          blue ;
    int          count ;
} COLOR, *pColor ;

typedef struct {
    int          Rmin ;
    int          Rmax ;
    int          Gmin ;
    int          Gmax ;
    int          Bmin ;
    int          Bmax ;
    int          volume ;
    long         colorcount ;
} box, *boxptr ;

typedef struct Palette {
    COLOR        color ;
    int          rate1 ;
    int          rate2 ;
    int          order ;
    int          x ;
    int          y ;
    int          row ;
    int          col ;
} PALETTE, *pPalette ;

typedef struct Candi {
    int          wide ;
    int          hi ;
} CANDI, *pCandi ;

/* プロトタイプ宣言 */
/* Motion.c */
void query() ;
void run( char*, int, GParam*, int *, GParam** ) ;
int  Motion_dialog() ;
void Motion_check_quit() ;
int  Motion_shm_open() ;
void Motion_shm_close() ;

/* Motion_sub.c */
void PixelDataReadAndAnalyze( int mode, gint32 id ) ;
void update_box_rgb( int *histogram, boxptr boxp ) ;
int median_cut_rgb( int *, boxptr, int, int ) ;
boxptr find_biggest_color_pop( boxptr, int ) ;
boxptr find_biggest_volume( boxptr, int ) ;
void compute_color_rgb( COLOR*, int*, boxptr, int ) ;
```



```
void Motion_ResizeColorMap( int ) ;

void Motion_EventCallback( FL_OBJECT* ) ;

/* PropPalette_cb.c */
void PaletteWndResize() ;
void HSVEditColorSet() ;
void HSVUpDownBtnRedraw() ;

/* PropPalette_sub.c */
void Set_MapColor_CEdit( int, int, int ) ;
void Color_RGBtoHSV( int, int, int, int*, int*, int* ) ;
void Color_HSVtoRGB( int, int, int, int*, int*, int* ) ;
float Color_DiffFromRGB( int, int, int, int, int, int ) ;

/* グローバル変数 */
#ifdef _MOTION_C_

int                     M_LoopFlag ;
FD_PropPalette          *fd_Prop ;
FD_Progress             *fd_Prog ;
int                     M_Palcnt ;
int                     M_CurrentID ;
PALETTE                 M_Palette[12] ;
int                     M_CurHue ;
int                     M_CurSat ;
int                     M_CurVal ;
int                     M_SatMin ;
int                     M_SatMax ;
int                     M_ValMin ;
int                     M_ValMax ;

#else

extern int              M_LoopFlag ;
extern FD_PropPalette   *fd_Prop ;
extern FD_Progress      *fd_Prog ;
extern int              M_Palcnt ;
extern int              M_CurrentID ;
extern PALETTE          M_Palette[12] ;
extern int              M_CurHue ;
extern int              M_CurSat ;
extern int              M_CurVal ;
extern int              M_SatMin ;
extern int              M_SatMax ;
extern int              M_ValMin ;
extern int              M_ValMax ;

#endif
#endif
```

```
/** Header file generated with fdesign on Tue Dec 15 14:59:40 1998.**/

#ifndef FD_PropPalette_h_
#define FD_PropPalette_h_

/** Callbacks, globals and object handlers **/
extern void NewBtnProc(FL_OBJECT *, long);
extern void LoadBtnProc(FL_OBJECT *, long);
extern void SaveBtnProc(FL_OBJECT *, long);
extern void LevelsBtnProc(FL_OBJECT *, long);
extern void HelpBtnProc(FL_OBJECT *, long);
extern void BrightnessDownBtnProc(FL_OBJECT *, long);
extern void BrightnessUpBtnProc(FL_OBJECT *, long);
extern void SatUpBtnProc(FL_OBJECT *, long);
extern void SatDownBtnProc(FL_OBJECT *, long);
extern void IndexBtnProc( FL_OBJECT*, long ) ;
extern int  IndexColorProc( FL_OBJECT*, int, FL_Coord, FL_Coord, int, void* ) ;
extern void CancelBtnProc(FL_OBJECT*, long);

/**** Forms and Objects ****/
typedef struct {
        FL_FORM *PropPalette;
        void *vdata;
        char *cdata;
        long  ldata;
        FL_OBJECT *SelectColor;
        FL_OBJECT *New;
        FL_OBJECT *Load;
        FL_OBJECT *Save;
        FL_OBJECT *Levels;
        FL_OBJECT *Help;
        FL_OBJECT *BrightnessDown;
        FL_OBJECT *BrightnessUp;
        FL_OBJECT *SatUp;
        FL_OBJECT *SatDown;
        FL_OBJECT *PaletteFrame;
        FL_OBJECT *EditColor;
        FL_OBJECT *IndexBtn[12];
        FL_OBJECT *IndexColor[12];
} FD_PropPalette;

extern FD_PropPalette * create_form_PropPalette(void);

typedef struct {
        FL_FORM *Progress;
        void *vdata;
        char *cdata;
        long  ldata;
        FL_OBJECT *title;
        FL_OBJECT *waku;
        FL_OBJECT *bar;
} FD_Progress;

extern FD_Progress * create_form_Progress(void);


#endif /* FD_PropPalette_h_ */
```