Subject: mand3.c
Posted by RJ[1] on Tue, 31 Dec 1985 15:57:47 GMT
View Forum Message <> Reply to Message

Article-I.D.: amiga.453
Posted: Tue Dec 31 10:57:47 1985
Date-Received: Wed, 1-Jan-86 04:40:48 EST
References:
Reply-To: rj@wizard.UUCP (Robert J. Mical)
Organization: Commodore-Amiga Inc., 983 University Ave #D, Los Gatos CA 95030
Lines: 494
Keywords: Amiga, Mandelbrot

```
/*
                MAND3.C - Control routines
            Mandelbrot Self-Squared Dragon Generator
                  For the Commodore Amiga
                      Version 1.00

            Copyright (C) 1985, Robert S. French
                Placed in the Public Domain

    Assorted Goodies and Intuition-stuff by =RJ Mical=  1985

This program may be distributed free of charge as long as the above
notice is retained.

*/


#include "mand.h"


extern int MathBase, MathTransBase;

extern struct Menu MainMenu[MENU_COUNT];


/*----------------------*/
/* Graphics definitions */

extern struct   GfxBase       *GfxBase;
extern struct   IntuitionBase *IntuitionBase;

struct   RastPort      *rp,*rp2;
struct   ViewPort      *vp;
```

```
struct TextAttr TextFont =
    {
    "topaz.font", /* Standard system font */
    8,    0,    0
    };

struct   Window       *w,*w2;
struct   Screen       *screen;
struct   IntuiMessage  *message;

struct NewScreen ns = {
   0, 0,                          /* start position              */
   320, 200, 6,                       /* width, height, depth          */
   0, 1,                          /* detail pen, block pen         */
   HAM,                             /* Hold and Modify ViewMode      */
   CUSTOMSCREEN,                          /* screen type                */
   &TextFont,                        /* font to use                */
   "",                             /* default title for screen      */
   NULL                             /* pointer to additional gadgets */
   };

struct NewWindow nw = {
   0, 0,                        /* start position              */
   320, 200,                        /* width, height               */
   -1, -1,                      /* detail pen, block pen         */
   MOUSEBUTTONS | MENUPICK | INTUITICKS,
                               /* IDCMP flags                 */
   ACTIVATE | BORDERLESS | BACKDROP,
                               /* window flags                */
   NULL,                        /* pointer to first user gadget  */
   NULL,                        /* pointer to user checkmark     */
   NULL,                        /* window title                */
   NULL,                        /* pointer to screen (set below) */
   NULL,                        /* pointer to superbitmap        */
   0, 0, 320, 200,                /* ignored since not sizeable    */
   CUSTOMSCREEN                       /* type of screen desired        */
};

struct NewWindow aw = {
   0, 10,
   100, 70,
   -1, -1,
   NULL,
   WINDOWDRAG|WINDOWSIZING|SMART_REFRESH,
   NULL,
   NULL,
   "Analysis",
```

```
   NULL,
   NULL,
   0, 0, 320, 200,
   CUSTOMSCREEN
};

long last_color;

BOOL SettingCenter, SettingBoxSize;

/*----------------------------------*/
/* Miscellaneous Global Definitions */

extern union kludge {
   float f;
   int i;
} start_r,end_r,start_i,end_i;  /* Block bounds for set */
extern int max_x,max_y;  /* Graphics window size */
extern int max_count,color_inc,color_offset,color_set,color_mode,color_div;
extern int color_inset,func_num;

extern int v_starty,max_mem, max_mem_y;
extern long v_offset;
extern UWORD *color_table,*v_mand_store;

extern int modified,want_read;

extern FILE *console,*v_fp,*redir_fp;

extern SHORT ZoomCenterX, ZoomCenterY, ZoomBoxSizeX, ZoomBoxSizeY;
extern SHORT ZoomBoxStartX, ZoomBoxStartY;


open_winds()
{
   int i,color;

   nw.Width = max_x;
   nw.Height = max_y+STARTY;

   if (color_mode & NOT_HOLDANDMODIFY) {
      ns.ViewModes = NULL;
      ns.Depth = 5;
   }
   else {
      ns.ViewModes = HAM;
      ns.Depth = 6;
   }
```

```
  if (color_mode & INTERLACE_MODE) {
    ns.Height = 400;
  }
  else
    ns.Height = 200;

  if (color_mode & HIRES_MODE) {
    ns.Width = 640;
    ns.ViewModes |= HIRES;
    ns.Depth = 4;
  }
  else
    ns.Width = 320;

  screen = (struct Screen *)OpenScreen(&ns);
  if (screen == NULL) {
    fputs("Can't open new screen!\n",console);
    return (1);
  }

  ShowTitle(screen,FALSE);

  nw.Screen = screen;
  w = (struct Window *)OpenWindow(&nw);
  if (w == NULL) {
    CloseScreen(screen);
    fputs("Can't open new window!\n",console);
    return (1);
  }

  SetMenuStrip(w, &MainMenu[0]);

  vp = &screen->ViewPort;
  rp = w->RPort;

  SetDrMd(rp,JAM1);
  SetBPen(rp, 0);

  /* leave registers 0 and 1 alone, set two to black, set the others
   * according to the user's design
   */
  for (i = 2; i > 8) & 0xf, (color >> 4) & 0xf, color & 0xf);
  }

  return (0);
}
```

```
void wait_close()
{
  ULONG class;
  USHORT code;
  union kludge center, distance, scale;

  if (redir_fp)
    {
    Delay(600);
    CloseDisplay();
    return;
    }

  SettingCenter = SettingBoxSize = FALSE;

  for (EVER)
    {
    Wait((1 UserPort->mp_SigBit));
    while (message = (struct IntuiMessage *)GetMsg(w->UserPort))
      {
      class = message->Class;
      code  = message->Code;
      ReplyMsg(message);

      switch (class)
        {
        case MENUPICK:
          switch MENUNUM(code)
            {
            case MENU_OPTIONS:
              switch ITEMNUM(code)
                {
                case OPTIONS_QUARTER:
                  if (color_mode & HIRES_MODE) max_x = 640 / 4;
                  else max_x = 320 / 4;
                  if (color_mode & INTERLACE_MODE) max_y = 400 / 4;
                  else max_y = 200 / 4;
                  max_mem = max_mem_y * MAXX;
                  max_mem /= max_x;
                  if (gen_mand()) return;
                  break;
                case OPTIONS_FULL:
                  if (color_mode & HIRES_MODE) max_x = 640;
                  else max_x = 320;
                  if (color_mode & INTERLACE_MODE) max_y = 400;
                  else max_y = 200;
                  max_mem = max_mem_y * MAXX;
                  max_mem /= max_x;
```

```
            /* intentionally fall into GENERATE */
          case OPTIONS_GENERATE:
            if (gen_mand()) return;
            break;
          case OPTIONS_CLOSE:
            CloseDisplay();
            fclose(v_fp);
            v_fp = NULL;
            return;
          }
        break;
      case MENU_ZOOM:
        switch ITEMNUM(code)
          {
          case ZOOM_SETCENTER:
            if (NOT SettingCenter)
              {
              ZoomCenterX = w->MouseX;
              ZoomCenterY = w->MouseY;
              DrawZoomCenter();
              SettingCenter = TRUE;
              }
            break;
          case ZOOM_SIZEBOX:
            if (NOT SettingBoxSize)
              {
              RecalcZoomBox();
              DrawZoomBox();
              SettingBoxSize = TRUE;
              }
            break;
          case ZOOM_ZOOMIN:
          case ZOOM_ZOOMIN10:
            /* first, get distance equal to the current size
             * of a single pixel width
             */
            distance.i = SPSub(start_r.i,end_r.i);
            distance.i = SPDiv(SPFlt(max_x), distance.i);
            /* center equals the number of pixels from the
             * center of the display to the zoom center
             */
            center.i = SPSub(SPFlt(max_x >> 1),
                 SPFlt(ZoomCenterX));
            /* scale equals the real displacement from the
             * center of the display to the zoom center
             */
            scale.i = SPMul(distance.i, center.i);
```

```
/* so, translate the real origin to here */
                start_r.i = SPAdd(scale.i,start_r.i);
                end_r.i = SPAdd(scale.i,end_r.i);

                /* now do it all again for the irrational axis */
                distance.i = SPSub(start_i.i,end_i.i);
                distance.i = SPDiv(SPFlt(max_y), distance.i);
/* the arguments are reversed to swap the sign */
                center.i = SPSub(SPFlt(ZoomCenterY),
                     SPFlt(max_y >> 1));
                scale.i = SPMul(distance.i, center.i);
/* so, translate the real origin to here */
                start_i.i = SPAdd(scale.i,start_i.i);
                end_i.i = SPAdd(scale.i,end_i.i);

                /* next, get the zoom-in scale
                 * if we're using the frame, then get the
                 * proportion of the box to the display;
                 * else if we're zooming in by 10, get that scale
                 */
                if (ITEMNUM(code) == ZOOM_ZOOMIN)
                  scale.i = SPDiv(SPFlt(max_x),
                       SPFlt(ZoomBoxSizeX));
                else scale.i = SPDiv(SPFlt(10), SPFlt(1));
                ZoomAlongDarling(scale.i, SPFlt(1));
                if (ITEMNUM(code) == ZOOM_ZOOMIN)
                  scale.i = SPDiv(SPFlt(max_y),
                       SPFlt(ZoomBoxSizeY));
                else scale.i = SPDiv(SPFlt(10), SPFlt(1));
                ZoomAlongDarling(SPFlt(1), scale.i);

                if (gen_mand()) return;
                break;
              case ZOOM_ZOOMOUT2:
                ZoomAlongDarling(SPFlt(2), SPFlt(2));
                if (gen_mand()) return;
                break;
              case ZOOM_ZOOMOUT10:
                ZoomAlongDarling(SPFlt(10), SPFlt(10));
                if (gen_mand()) return;
                break;
              }
            break; /* breaks MENU_ZOOM case statement */
          }
        break; /* breaks MENUPICK switch statement */
      case INTUITICKS:
        code = NULL;
      case MOUSEBUTTONS:
```

```
            if (SettingCenter)
              {
              DrawZoomCenter();
   if (code == SELECTDOWN) SettingCenter = FALSE;
              else
                {
      ZoomCenterX = w->MouseX;
      ZoomCenterY = w->MouseY;
                DrawZoomCenter();
                }
              }
            else if (SettingBoxSize)
              {
              DrawZoomBox();
   if (code == SELECTDOWN) SettingBoxSize = FALSE;
              else
                {
                RecalcZoomBox();
                DrawZoomBox();
                }
              }
            break;
          }
        }
      }
}


anal_mand()
{
   union kludge x_gap,y_gap,z_r,z_i,u_r,u_i,const0,const1,const2,const4;
   union kludge const12,temp,temp2,temp3;
   int count,x,y,width,height,last_x,last_y,select,lines;
   ULONG class;
   USHORT code;

   if (disp_mand())
      return (1);

   x_gap.i = SPDiv(SPFlt(max_x),SPSub(start_r.i,end_r.i));
   y_gap.i = SPDiv(SPFlt(max_y),SPSub(start_i.i,end_i.i));

   const0.i = SPFlt(0);
   const1.i = SPFlt(1);
   const2.i = SPFlt(2);
   const4.i = SPFlt(4);
   const12.i = SPDiv(SPFlt(2),SPFlt(1));
```

```
last_x = -1;
last_y = -1;
select = 0;
lines = 0;

aw.Screen = screen;
w2 = (struct Window *)OpenWindow(&aw);
if (w2 == NULL)
{
   CloseDisplay();
   fputs("Can't open analyzing window\n",console);
   return (1);
}

rp2 = w2->RPort;

SetDrMd(rp2,JAM1);
SetBPen(rp2, 0);

SetAPen(rp2, 1);
RectFill(rp2,LEFTW2+1,TOPW2+1,RIGHTW2-1,BOTTOMW2-1);

for (EVER) {
   if (message = (struct IntuiMessage *)GetMsg(w->UserPort)) {
      class = message->Class;
      code  = message->Code;
      ReplyMsg(message);

      if (class == CLOSEWINDOW) {
         CloseWindow(w2);
         CloseDisplay();
         return (0);
      }
      if (class == MOUSEBUTTONS)
         if (code == SELECTDOWN) {
            if (w->MouseY < STARTY)
               lines = !lines;
            else
               select = TRUE;
         }
         else if (code == SELECTUP)
            select = FALSE;
   }
   if ((last_x != w->MouseX || last_y != w->MouseY) && select &&
       w->MouseX < max_x && w->MouseY >= STARTY && w->MouseY < max_y+STARTY) {
      last_x = w-> MouseX;
      last_y = w-> MouseY;
      SetAPen(rp2,1);
```

```
RectFill(rp2,LEFTW2+1,TOPW2+1,RIGHTW2-1,BOTTOMW2-1);
width = RIGHTW2 - LEFTW2 - 6;
height = BOTTOMW2 - TOPW2 - 4;
SetAPen(rp2,3);
Move(rp2,LEFTW2+2,TOPW2+height/2+2);
Draw(rp2,RIGHTW2-2,TOPW2+height/2+2);
Move(rp2,LEFTW2+width/2+3,TOPW2+2);
Draw(rp2,LEFTW2+width/2+3,BOTTOMW2-2);
SetAPen(rp2,2);
if (func_num == 0)
   z_r.i = const0.i;
else
   z_r.i = const12.i;
z_i.i = const0.i;
u_r.i = SPAdd(start_r.i,SPMul(SPFlt(last_x),x_gap.i));
u_i.i = SPAdd(start_i.i,SPMul(SPFlt(max_y-(last_y-STARTY)-1),y_gap.i));
count = 0;
for (count=0;SPFix(SPAdd(SPMul(z_r.i,z_r.i),SPMul(z_i.i,z_i.i)))
```

Subject: Re: Mand3.c
Posted by RJ[1] on Mon, 06 Jan 1986 22:20:35 GMT
View Forum Message <> Reply to Message

Article-I.D.: amiga.483
Posted: Mon Jan 6 17:20:35 1986
Date-Received: Wed, 8-Jan-86 06:07:55 EST
References:
Reply-To: rj@wizard.UUCP (Robert J. Mical)
Organization: Commodore-Amiga Inc., 983 University Ave #D, Los Gatos CA 95030
Lines: 2

See article 1044 of net.micro.amiga.
=RJ Mical=
