รายการอ้างอิง

ภาษาไทย

มงคล ภิญโญสโมสร. 2539. การออกแบบและพัฒนาระบบการแสดงภาพสเตอริโอแบบแทรกสอด . วิทยานิพนธ์วิทยาศาสตร์มหาบัณฑิต ภาควิชาวิศวกรรมคอมพิวเตอร์ บัณฑิตวิทยาลัย จุฬาลงกรณ์มหาวิทยาลัย.

สมชาย ประสิทธิ์จูตระกูล. งานวิจัยทุนอุดหนุนโครงการสิ่งประดิษฐ์ เรื่องระบบคอมพิวเตอร์เพื่อการจินตทัศน์ข้อมูลแบบสามมิติสเตอริโอสลับเชิงเวลา . กรุงเทพมหานคร : จุฬาลงกรณ์มหาวิทยาลัย, 2540. (อัดสำเนา)

ภาษาอังกฤษ

Cameron Smith and Nancy Blachman . The Mathematica graphics guidebook . (n.p.): Addison-Wesley Publishing Company, 1995.

David B. Wagner . Power Programming with Mathematica : the Kernel . New York: McGraw-Hill , 1996.

David C. Kay and John R. Levine . Graphics File Formats . PA.: McGraw-Hill , 1992.

Kate Gregory . Special Edition Using Visual C++ 5 . Illinois: Que Corporation, 1997.

W.schroeder, K.Marlin, and B.Lorensen . The Visualization Toolkit . (n.p.): Prentice Hall PTR , 1996.

Stephen Wolfram . The Mathematica Book , 3rd Ed. London: Cambridge University Press, 1996.

StereoGraphics Coperation . Stereographics Developers' Handbook . (n.p.): StereoGraphics Coperation, 1997.

ภาคผนวก

ภาคผนวก ก.
แฟ้มข้อมูลภาพแบบดีไอบี

แฟ้มข้อมูลภาพแบบดีไอบี (Microsoft Windows Device Independence Bitmap) เป็นแฟ้มข้อมูลที่ใช้งานอย่างแพร่หลายใช้เก็บภาพแบบบิตแม็พ มี 2 แบบคือ แฟ้มข้อมูลภาพบิตแมพบนไมโครซอฟท์วินโดว์ และแฟ้มข้อมูลภาพบิตแมพบนโอเอสทู ซึ่งประกอบด้วย 3 ส่วนคือ ส่วนหัวของแฟ้มข้อมูล (File Header) ส่วนหัวของภาพบิตแมพ (Bitmap Header) และส่วนข้อมูลภาพ (Bitmap Data) ดังรายละเอียดต่อไปนี้

ก-1 ส่วนหัวของแฟ้มข้อมูล (BITMAPFILEHEADER)

ส่วนต่างๆของส่วนหัวของแฟ้มข้อมูลบิตแมพแสดงในตารางที่ ก-1

ตารางที่ ก-1 แสดงส่วนหัวของแฟ้มข้อมูล

ไบต์ออฟเซต	ขนาด	ชื่อข้อมูล	ความหมายของข้อมูล
0	2	bfType	มีค่าเป็น BM แสดงว่าเป็นแฟ้มข้อมูลบิตแมพ
2	4	bfSize	ขนาดของแฟ้มข้อมูล
6	2	bfReserved1	ค่า = 0
8	2	bfReserved2	ค่า = 0
10	4	bfOffBits	ไบต์ออฟเซตของข้อมูลภาพ

ก-2 ส่วนหัวของภาพบิตแมพ

แฟ้มข้อมูลภาพแบบบิตแมพที่ใช้บนไมโครซอฟท์วินโดว์และบนโอเอสทูจะมีลักษณะของส่วนหัวของภาพบิตแมพที่แตกต่างกันดังรายละเอียดที่แสดงต่อไป

1) ส่วนหัวของภาพบิตแมพสำหรับไมโครซอฟท์วินโดว์ (BITMAPINFOHEADER) ซึ่งจะมีโครงสร้างดังตารางที่ ก-2

ตารางที่ ก-2 โครงสร้างของส่วนหัวของแฟ้มข้อมูลภาพบิตแมพสำหรับไมโครซอฟท์วินโดว์

ไบต์ออฟเซต	ขนาด	ชื่อข้อมูล	ความหมายของข้อมูล
14	4	biSize	ขนาดของส่วนหัวของภาพบิตแมพ
18	4	biWidth	ความกว้างของภาพ หน่วยเป็นจุดภาพ
22	4	biHeight	ความสูงของภาพ หน่วยเป็นจุดภาพ
26	2	biPlanes	จำนวนระนาบของภาพ มีค่าเป็น 1
28	2	biBitCount	จำนวนบิตต่อจุดภาพมีค่าเป็น 1, 4 , 8 หรือ 24
30	4	biCompression	ชนิดของรหัสข้อมูลภาพ
34	4	biSizeImage	ขนาดเป็นไบต์ของภาพที่ถูกเข้ารหัส
38	4	biXPelsPerMeter	ความละเอียดตามแนวนอน หน่วยเป็นจุดภาพต่อเมตร
42	4	biYPelsPerMeter	ความละเอียดตามแนวตั้ง หน่วยเป็นจุดภาพต่อเมตร
46	4	biClrUsed	จำนวนสีที่ใช้
50	4	biClrImportant	จำนวนสีที่จำเป็นต้องใช้
54	4*N	bmiColors	ตารางสี

ซึ่งถ้าภาพมีความจำเป็นต้องใช้ตารางสี ตารางสีก็จะเก็บอยู่ต่อจากส่วนนี้ โครงสร้างของส่วนหัวของภาพบิตแมพที่มีส่วนของตารางสีด้วยเรียกว่า บิตแมพอินโฟ (BITMAPINFO) ซึ่งจำนวนรายการในตารางสีพิจารณาจากค่าของข้อมูล biBitCount ซึ่งถ้าเป็น 1 ก็มี 2 รายการ ถ้าเป็น 4 ก็มี 16 รายการเป็นต้น ส่วนภาพ 24 บิตจะไม่ใช้ตารางสีแต่จะใช้แต่ละจุดเก็บค่าของสีแดง เขียว และน้ำเงินโดยตรง ค่าองค์ประกอบของแต่ละรายการในตารางสีแสดงในตารางที่ ก-3

ตารางที่ ก-3 องค์ประกอบในรายการของตารางสีของภาพบิตแมพสำหรับไมโครซอฟท์วินโดว์

ไบต์ออฟเซต	ชื่อเขตข้อมูล	ความหมาย
0	rgbBlue	ค่าสีน้ำเงิน
1	rgbGreen	ค่าสีเขียว
2	rbgRed	ค่าสีแดง
3	rgbReserved	ค่าเป็น 0

2) ส่วนหัวของแฟ้มข้อมูลภาพบิตแมพสำหรับโอเอสทู(BITMAPCOREHEADER) และถ้ามีตารางสีด้วยจะเรียกว่า บิตแมพคอร์อินโฟ (BITMAPCOREINFO) ซึ่งมีรายละเอียดดังตารางที่ ก-4 และ ก-5

ตารางที่ ก-4 โครงสร้างส่วนหัวของภาพบิตแมพสำหรับโอเอสทู

ไบต์ออฟเซต	ขนาด	ชื่อข้อมูล	ความหมายของข้อมูล
14	4	bcSize	ขนาดของส่วนหัวของภาพบิตแมพ(12 ไบต์)
18	2	bcWidth	ความกว้างของภาพ หน่วยเป็นจุดภาพ
20	2	bcHeight	ความสูงของภาพ หน่วยเป็นจุดภาพ
22	2	bcPlanes	จำนวนระนาบของภาพ มีค่าเป็น 1
24	2	bcBitCount	จำนวนบิตต่อจุดภาพมีค่าเป็น 1 , 4 , 8 หรือ 24
26	3*N	bmciColors	ตารางสี

ตารางที่ ก-5 องค์ประกอบในรายการของตารางสีของภาพบิตแมพสำหรับโอเอสทู

ไบต์ออฟเซต	ชื่อเขตข้อมูล	ความหมาย
0	rgbtBlue	ค่าสีน้ำเงิน
1	rgbtGreen	ค่าสีเขียว
2	rgbtRed	ค่าสีแดง

ก-3 ส่วนข้อมูลภาพ

ข้อมูลภาพแบบใช้ตารางสีจะใช้การเก็บเป็นดัชนีที่ชี้ไปยังตารางสี ซึ่งสามารถเก็บแบบบีบอัดข้อมูลหรือไม่ก็ได้ ไบต์แรกที่ต่อจากส่วนหัวของภาพบิตแมพจะนำมาแสดงผลที่ตำแหน่งมุมล่างซ้าย ส่วนภาพ 24 บิตจะเก็บเป็นค่าของสีน้ำเงิน เขียว และแดงตามลำดับ และในแต่ละแถวจะถูกเติมด้วย 0 เพื่อให้ลงตัวในตำแหน่งจำนวนเท่าของ 4 ไบต์พอดี

ภาคผนวก ข.

รหัสต้นฉบับของโปรแกรม

ข-1 รหัสต้นฉบับของสเตอริโอแพ็คเกจ

สเตอริโอแพ็คเกจถูกพัฒนาขึ้นโดยใช้ภาษาโปรแกรมแมทแมติก้ามีรหัสต้นฉบับดังต่อไปนี้

```
(* This Mathematica' package contain commands for *)
(* generated and display Stereo image *)

BeginPackage["Stereo`"]

GenStereopair::usage = "Use to generate stereo pair images."
ShowStereo::usage = "Use to send images to StereoView to show."
GenPrevAsStereo::usage = "Use to generate stereo pair images."
RegenStereo::usage = "Use to regenerate stereo pair images in same
                      function."
ReverseStereo::usage = "Use to generate reverse stereo images"
EndStereo::usage = "End function"
NarrowView::usage = "Change image viewing"
WideView::usage = "Change image viewing"


DisplayValue = 0;
DividerViewDistance = 22;
centerview = {1.3, -2.4, 2.};
VecLen[{x_, y_, z_}] := Sqrt[N[x ^2 + y ^2 + z ^2]];
centerline = {0.0, 0.0, -1.0};
eyeline = Cross[centerline, centerview];
ueyeline = eyeline / VecLen[eyeline];
viewdist = VecLen[centerview];
eyedist = viewdist / DividerViewDistance;
leftview = centerview + eyedist * ueyeline;
rightview = centerview - eyedist * ueyeline;

stereoprogram = Install["stereoview"]

Begin["`Private`"]

GenStereopair[func_,{v_,vmin_,vmax_},{u_,umin_,umax_}] :=
      Table[nb = SelectedNotebook[];
            Plot3D[func,{v,vmin,vmax},{u,umin,umax},ViewPoint
                 -> leftview];
            Plot3D[func,{v,vmin,vmax},{u,umin,umax},ViewPoint
                 -> rightview];
            tempfunction = func;
            tempv = v;
            tempvmin = vmin;
            tempvmax = vmax;
            tempu = u;
            tempumin = umin;
            tempumax = umax;
      ]
```

```
GenStereopair[func_]:= GenStereopair[func,{x,0,4},{y,0,4}]

GenPrevAsStereo[] := Table[   nb = SelectedNotebook[];
                      Show[%,ViewPoint -> leftview];
                      Show[%,ViewPoint -> rightview];
                  ]

ShowStereo[] := Table[
                  NotebookFind[nb,"Graphics",Previous,CellStyle];
                  FrontEndTokenExecute["Copy"];
                  Global`GenerateLeftImage[];
                  NotebookFind[nb,"Graphics",Previous,CellStyle];
                  FrontEndTokenExecute["Copy"];
                  Global`GenerateRightImage[];
                  Global`ProcessImage[DisplayValue]
                  ]

RegenStereo[] := GenStereopair[tempfunction,{tempv,tempvmin,
                     tempvmax},{tempu,tempumin,tempumax}]

ReverseStereo[] := Table[
                  tempDisplayValue = DisplayValue+1;
                  DisplayValue = Mod[tempDisplayValue,2];
                  ShowStereo[];
                  ]

EndStereo[] := Uninstall[stereoprogram]

NarrowView[n_] := Table[
                  DividerViewDistance += n;
                  eyedist = viewdist / DividerViewDistance;
                  leftview = centerview + eyedist * ueyeline;
                  rightview = centerview - eyedist * ueyeline;
                  RegenStereo[];
                  ]

WideView[n_] := Table[
                  DividerViewDistance -= n;
                  eyedist = viewdist / DividerViewDistance;
                  leftview = centerview + eyedist * ueyeline;
                  rightview = centerview - eyedist * ueyeline;
                  RegenStereo[];
                  ]

End[]
EndPackage[]
```

ข-2 รหัสต้นฉบับของโปรแกรมสเตอริโอวิว

โปรแกรมสเตอริโอวิวเป็นโปรแกรมที่ถูกพัฒนาขึ้นด้วยภาษาซีโดยใช้เครื่องมือพัฒนาคือไมโครซอฟท์วิชวลซี++ ซึ่งจะมีรหัสต้นฉบับดังต่อไปนี้

```
#include<windows.h>
#include<windowsx.h>
#include<string.h>
#include<stdio.h>
#include<limits.h>

#include "StereoView.h"
#include "dibapi.h"
#include "mathlink.h"

static char szAppName[] = "Stereo View";

char Directions[100],ClipboardStatus[100];
static HWND MainWindow;
static HINSTANCE hInst;
BOOL STATUS = FALSE;
BOOL ImageFinish = FALSE;

LPSTR lpDIBHdr,lpDIBBits;
int width,height,color,bitperpixel;

HPALETTE    ghPal=NULL;       // Handle to our bitmap's palette

HPALETTE m_palDIB;
SIZE m_size;
RECT rcDIB,rcDest;
HDIB hNewDIB = NULL;          // Handle to DIB Image from clipboard
HDIB hLeftDIB = NULL;         // Handle to DIB Image from clipboard
                              // for left eye
HDIB hRightDIB = NULL;        // Handle to DIB Image from clipboard
                              // for right eye

BOOL LeftImageStatus = FALSE,RightImageStatus = FALSE;
BOOL StartImage = TRUE;       // StartImage = TRUE is mean start by
                              // Left Image
// StartImage = FALSE is mean start by Right Image (swap)

/*********************************************************************
* This part contain functioncalled from Mathematica program via
* mathlink protocal. The followed function must place in mathlink
* template file (*.tm) to specific pattern that use to call
* and complied by mprep (Mathlink preprocessor) by command
* mprep ?????.tm -o ?????.cpp
* and use that file ????.cpp to insert into project
*********************************************************************/
int GenerateImage(int x)
{
// received image from clipboard side 0=leftimage; 1=rightimage.

      if(side == 0)     // call copy image function  depend on side
            return GetLeftImage();
      else
            return GetRightImage();
}
```

```
int ProcessImage(int x)
{
      if (x == 1)
            StartImage = TRUE;
      else
            StartImage = FALSE;
      InvalidateRect(MainWindow, NULL, FALSE);
      UpdateWindow(MainWindow);
      return 0;
}

int ClearAllImage(int x)
{
      LeftImageStatus = FALSE;
      RightImageStatus = FALSE;
      return 0;
}

int WINAPI WinMain(HINSTANCE hInst,HINSTANCE hPrevInstance,LPSTR
                        lpszCmdParam,int nCmdShow)
{
      if(!hPrevInstance)
            if(!Register(hInst))
                  return FALSE;
      if(!Create(hInst, nCmdShow))
            return FALSE;

      char  buff[512];
      char FAR * buff_start = buff;
      char FAR * argv[32];
      char FAR * FAR * argv_end = argv + 32;

      MLScanString( argv, &argv_end, &lpszCmdParam, &buff_start);
      return MLMain( argv_end - argv, argv);
}

BOOL Register(HINSTANCE hInst)
{
      WNDCLASS WndClass;

      WndClass.style = CS_HREDRAW |CS_VREDRAW |CS_NOCLOSE;
      WndClass.lpfnWndProc = WndProc;
      WndClass.cbClsExtra = 0;
      WndClass.cbWndExtra = 0;
      WndClass.hInstance = hInst;
      WndClass.hIcon = LoadIcon(hInst,"StereoIcon");
      WndClass.hCursor = LoadCursor(NULL,IDC_ARROW);
      WndClass.hbrBackground  = (HBRUSH) GetStockObject
                                    (WHITE_BRUSH);
      WndClass.lpszMenuName   = NULL;
      WndClass.lpszClassName  = szAppName;

      return RegisterClass(&WndClass);
}
```

```
HWND Create(HINSTANCE hInstance,int nCmdShow)
{
      HWND hwnd;
      HDC hdc;

      hdc = GetDC(hwnd);
      int hres = GetDeviceCaps(hdc,HORZRES);

      hwnd = CreateWindow(szAppName,szAppName,WS_OVERLAPPED|
                  WS_CAPTION|WS_SYSMENU|WS_THICKFRAME,hres-1020,
                  47,1020,680,NULL,NULL,hInstance,NULL);
      if(hwnd == NULL)
            return hwnd;
      ShowWindow(hwnd, nCmdShow);
      UpdateWindow(hwnd);
      return hwnd;
}

#define EasT_DefProc DefWindowProc

long WINAPI WndProc(HWND hWindow, UINT Message, WPARAM wParam,
                        LPARAM lParam)
{
      switch(Message)
      {
            case WM_DESTROY:
                  {
                        PostQuitMessage(0);
                        break;
                  }

            case WM_PALETTECHANGED:
                  {
                        HDC hDC;     // Handle to device context
                        HPALETTE hOldPal; // Handle to previous
                                          // logical palette

// Before processing this message, make sure we are indeed using a
// palette
                  if (ghPal)
                    {
      // If this application did not change the palette, select
      // and realize this application's palette

                        if (wParam != (WPARAM)hWindow)
                        {
      // Need the window's DC for SelectPalette/RealizePalette
                              hDC = GetDC(hWindow);

      // Select and realize our palette
                              hOldPal = SelectPalette(hDC, ghPal,
                                          FALSE);
                              RealizePalette(hDC);

      // When updating the colors for an inactive window,
      // UpdateColors can be called because it is faster than
      // redrawing the client area (even though the results are
      // not as good)
```

```
                        UpdateColors(hDC);

// Clean up
                        if (hOldPal)
                              SelectPalette(hDC, hOldPal,
                                    FALSE);

                        ReleaseDC(hWindow, hDC);
                        }
                  }
                  break;
            }
      case WM_QUERYNEWPALETTE:
            {
                  HDC hDC;      // Handle to device context
                  HPALETTE hOldPal;  // Handle to previous
                                     //logical palette

// Before processing this message, make sure we are indeed
// using a palette
                  if (ghPal)
                  {
// Need the window's DC for SelectPalette/RealizePalette
                        hDC = GetDC(hWindow);

// Select and realize our palette
                        hOldPal = SelectPalette(hDC, ghPal,
                              FALSE);
                        RealizePalette(hDC);

// Redraw the entire client area
                        InvalidateRect(hWindow, NULL, TRUE);
                        UpdateWindow(hWindow);

// Clean up
                       if (hOldPal)
                          SelectPalette(hDC, hOldPal, FALSE);

                       ReleaseDC(hWindow, hDC);

// Message processed, return TRUE
                  return TRUE;
            }

// Message not processed, return FALSE
            return FALSE;
      }

      case WM_PAINT:
            {
                  PAINTSTRUCT PaintStruct;

                  HDC PaintDC = BeginPaint(hWindow,
                              &PaintStruct);
                  if(RightImageStatus && LeftImageStatus)
                  {
                        CalculateImage();
```

```
                                if (OpenClipboard(NULL))
                                {
                                  hNewDIB = (HDIB) CopyHandle
                                      (::GetClipboardData(CF_DIB));
                                  CloseClipboard();
                                  if (hNewDIB != NULL)
                                  {
                                    lpDIBHdr  = (LPSTR) ::GlobalLock
                                      ((HGLOBAL) hNewDIB);
                                    lpDIBBits = ::FindDIBBits
                                      (lpDIBHdr);
                                    width = (int) ::DIBWidth
                                      (lpDIBHdr);
                                    height = (int) ::DIBHeight
                                      (lpDIBHdr);
                                    color = (int) ::DIBNumColors
                                      (lpDIBHdr);
                                  }
                                }
                            }
                            if (hNewDIB != NULL)
                            {
                                  DoPreparePaint(hWindow);
                                  DoPaint(hWindow);
                                  PaintDIB(PaintDC, rcDest, rcDIB,
                                        m_palDIB);
                            }
                            EndPaint(hWindow,&PaintStruct);
                            break;
                      }
      default:
            return EasT_DefProc(hWindow, Message, wParam, lParam);
      }
      return 0L;
}

HGLOBAL WINAPI CopyHandle (HGLOBAL h)
{
      if (h == NULL)
            return NULL;

      DWORD dwLen = ::GlobalSize((HGLOBAL) h);
      HGLOBAL hCopy = ::GlobalAlloc(GHND, dwLen);

      if (hCopy != NULL)
      {
            void* lpCopy = ::GlobalLock((HGLOBAL) hCopy);
            void* lp     = ::GlobalLock((HGLOBAL) h);
            memcpy(lpCopy, lp, dwLen);
            ::GlobalUnlock(hCopy);
            ::GlobalUnlock(h);
      }

      return hCopy;
}
```

```
//*********************************************************************
// Function: DoPreparePaint()
// Purpose:  Called by WndProc. Does initial data for painting
// client area.
//*********************************************************************
void DoPreparePaint(HWND hWnd)
{
      // Init DIB data
      if (m_palDIB != NULL)
      {
            delete m_palDIB;
            m_palDIB = NULL;
      }
      if (hNewDIB == NULL)
      {
            return;
      }

      // Set up document size
      LPSTR lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) hNewDIB );
      if (::DIBWidth(lpDIB) > INT_MAX ||::DIBHeight(lpDIB) >
            INT_MAX)
      {
            ::GlobalUnlock((HGLOBAL) hNewDIB );
            ::GlobalFree((HGLOBAL) hNewDIB );
            hNewDIB = NULL;
            return;
      }
      m_size.cx = (int) ::DIBWidth(lpDIB);
      m_size.cy = (int) ::DIBHeight(lpDIB);
      ::GlobalUnlock((HGLOBAL) hNewDIB );
// Create copy of palette
      m_palDIB = new HPALETTE;
      if (m_palDIB == NULL)
      {
            // we must be really low on memory
            ::GlobalFree((HGLOBAL) hNewDIB ); //m_hDIB );
            hNewDIB = NULL;
            return;
      }
      if((m_palDIB = CreateDIBPalette()) == NULL)
      {
            delete m_palDIB;
            m_palDIB = NULL;
            return;
      }
}

WORD FindBitPerPixel(LPSTR lpDIB)
{
      WORD wBitCount;  // DIB bit count

      if (IS_WIN30_DIB(lpDIB))
        wBitCount = ((LPBITMAPINFOHEADER)lpDIB)->biBitCount;
      else
        wBitCount = ((LPBITMAPCOREHEADER)lpDIB)->bcBitCount;

      return wBitCount;
}
```

```
//************************************************************************
// Function: CreateDIBPalette()
// Purpose:  Called by DoPreparePaint. Does creating DIB palette.
//************************************************************************
HPALETTE CreateDIBPalette()
{
    LPLOGPALETTE lpPal;          // pointer to a logical palette
    HANDLE       hLogPal;        // handle to a logical palette
    HPALETTE     hPal = NULL;    // handle to a palette
    int          i, wNumColors;  // loop index, number of colors in
                                 // color table
    LPSTR        lpbi;           // pointer to packed-DIB
    LPBITMAPINFO lpbmi;          // pointer to BITMAPINFO structure
                                 // (Win3.0)
    LPBITMAPCOREINFO lpbmc;      // pointer to BITMAPCOREINFO
                                 // structure (OS/2)
    BOOL bWinStyleDIB;           // Win3.0 DIB?

    // if handle to DIB is invalid, return NULL
    if (!hNewDIB)
        return NULL;

    // lock DIB memory block and get a pointer to it
    lpbi = (LPSTR) GlobalLock(hNewDIB);

    // get pointer to BITMAPINFO (Win 3.0)
    lpbmi = (LPBITMAPINFO)lpbi;

    // get pointer to BITMAPCOREINFO (OS/2 1.x)
    lpbmc = (LPBITMAPCOREINFO)lpbi;

    // get the number of colors in the DIB
    wNumColors = DIBNumColors(lpbi);

    // is this a Win 3.0 DIB?
    bWinStyleDIB = IS_WIN30_DIB(lpbi);
    if (wNumColors)
    {
        // allocate memory block for logical palette
        hLogPal = GlobalAlloc(GHND, sizeof(LOGPALETTE) +
                sizeof(PALETTEENTRY) * wNumColors);

        // if not enough memory, clean up and return NULL
        if (!hLogPal)
        {
            GlobalUnlock(hNewDIB);
            return NULL;
        }

        // lock memory block and get pointer to it
        lpPal = (LPLOGPALETTE)GlobalLock(hLogPal);

        // set version and number of palette entries
        lpPal->palVersion = PALVERSION;
        lpPal->palNumEntries = wNumColors;
```

```
    // store RGB triples (if Win 3.0 DIB) or RGB quads (if OS/2
    // DIB) into palette

    for (i = 0; i < wNumColors; i++)
    {
        if (bWinStyleDIB)
        {
            lpPal->palPalEntry[i].peRed = lpbmi->bmiColors[i]
                    .rgbRed;
            lpPal->palPalEntry[i].peGreen = lpbmi->bmiColors[i]
                    .rgbGreen;
            lpPal->palPalEntry[i].peBlue = lpbmi->bmiColors[i]
                    .rgbBlue;
            lpPal->palPalEntry[i].peFlags = 0;
        }
        else
        {
            lpPal->palPalEntry[i].peRed = lpbmc->bmciColors[i].
                    rgbtRed;
            lpPal->palPalEntry[i].peGreen = lpbmc->
                    bmciColors[i].rgbtGreen;
            lpPal->palPalEntry[i].peBlue = lpbmc->bmciColors[i]
                    .rgbtBlue;
            lpPal->palPalEntry[i].peFlags = 0;
        }
    }

 // create the palette and get handle to it

    hPal = CreatePalette(lpPal);

 // if error getting handle to palette, clean up and return NULL

    if (!hPal)
    {
        GlobalUnlock(hLogPal);
        GlobalFree(hLogPal);
        return NULL;
    }
}

 // clean up

GlobalUnlock(hLogPal);
GlobalFree(hLogPal);
GlobalUnlock(hNewDIB);

// return handle to DIB's palette
return hPal;
}
```

```
BOOL PaintDIB(HDC hDC, RECT lpDCRect, RECT lpDIBRect,HPALETTE hPal)
{
    LPSTR        lpDIBHdr;           // Pointer to BITMAPINFOHEADER
    LPSTR        lpDIBBits;          // Pointer to DIB bits
    BOOL         bSuccess=FALSE;     // Success/fail flag
    HPALETTE     hOldPal=NULL;       // Previous palette

    // Check for valid DIB handle

    if (!hNewDIB)
        return FALSE;

// Lock down the DIB, and get a pointer to the beginning of the bit
// buffer

    lpDIBHdr  = (LPSTR) GlobalLock(hNewDIB);
    lpDIBBits = FindDIBBits(lpDIBHdr);

// Select and realize our palette as background

    if (hPal)
    {
        hOldPal = SelectPalette(hDC, hPal, TRUE);
        RealizePalette(hDC);
    }

// Make sure to use the stretching mode best for color pictures
      SetStretchBltMode(hDC, COLORONCOLOR);

// Determine whether to call StretchDIBits() or SetDIBitsToDevice()
      int v1 = (633-lpDCRect.bottom)/2;
      if(v1%2 != 0)
            v1 -= 1;
      if ((newRECTWIDTH(lpDCRect)  == newRECTWIDTH(lpDIBRect)) &&
            (newRECTHEIGHT(lpDCRect) == newRECTHEIGHT(lpDIBRect)))
      {
            bSuccess = SetDIBitsToDevice(hDC, (1010-lpDCRect.right)
                        /2/*lpDCRect.left*/, v1/*lpDCRect.top*/,
                        newRECTWIDTH(lpDCRect), newRECTHEIGHT
                        (lpDCRect), lpDIBRect.left,
            (int)DIBHeight(lpDIBHdr) - lpDIBRect.top - newRECTHEIGHT
                  (lpDIBRect), 0, DIBHeight(lpDIBHdr), lpDIBBits,
                  (LPBITMAPINFO)lpDIBHdr, DIB_RGB_COLORS);
      }
      else
            bSuccess = StretchDIBits(hDC, (1010-lpDCRect.right)
                  /2/*lpDCRect.left*/, v1/*lpDCRect.top*/,
                  newRECTWIDTH(lpDCRect), newRECTHEIGHT(lpDCRect),
                  lpDIBRect.left,lpDIBRect.top,
                  newRECTWIDTH(lpDIBRect), newRECTHEIGHT(lpDIBRect),
                  lpDIBBits, (LPBITMAPINFO)lpDIBHdr, DIB_RGB_COLORS,
                  SRCCOPY);

    // Unlock the memory block
    GlobalUnlock(hNewDIB);


    // Reselect old palette
    if (hOldPal)
     SelectPalette(hDC, hOldPal, FALSE);
```

```
    // Return with success/fail flag
    return bSuccess;
}

BOOL GetLeftImage()
{

      if (OpenClipboard(NULL))
      {
        hLeftDIB = (HDIB) CopyHandle(::GetClipboardData(CF_DIB));
        EmptyClipboard();
        CloseClipboard();
        if (hLeftDIB != NULL)
            LeftImageStatus = TRUE;
      }
      return LeftImageStatus;
}

BOOL GetRightImage()
{

      if (OpenClipboard(NULL))
      {
        hRightDIB = (HDIB) CopyHandle(::GetClipboardData(CF_DIB));
        EmptyClipboard();
        CloseClipboard();
        if (hRightDIB != NULL)
            RightImageStatus = TRUE;
      }
      return RightImageStatus;
}

//*********************************************************************
// Function: DoPaint()
// Purpose:  Called by WndProc. Does painting for client area.
//*********************************************************************
void DoPaint(HWND hWnd)
{
      if(hNewDIB != NULL)
      {
            LPSTR lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) hNewDIB);
            int cxDIB = (int) ::DIBWidth(lpDIB);  // Size of DIB - x
            int cyDIB = (int) ::DIBHeight(lpDIB); // Size of DIB - y
            ::GlobalUnlock((HGLOBAL) hNewDIB);
            rcDIB.top = rcDIB.left = 0;
            rcDIB.right = cxDIB;
            rcDIB.bottom = cyDIB;
            rcDest = rcDIB;
      }
}
```

```
//*********************************************************************
// Function: CalculateImage()
// Purpose:  Called by WndProc. Does calculate image details and
// create output image.
//*********************************************************************
void CalculateImage()
{

      LPSTR lpLeftDIBHdr,lpLeftDIBBits;
// Pointer to BITMAPINFOHEADER and DIB bits of left image
      LPSTR lpRightDIBHdr,lpRightDIBBits;
// Pointer to BITMAPINFOHEADER and DIB bits of right image
      LPSTR lpOutputDIBHdr,lpOutputDIBBits;
// Pointer to BITMAPINFOHEADER and DIB bits of output image
      LPSTR leftPointer,rightPointer,outputPointer;
// A 32-bit pointer to a character string.
      int i,j;                        // index for "for loop"
      int bpp_patchvalue=0;

      if(StartImage)          // if Start Image = TRUE then generate
                              // in normal style
      {
            if (hLeftDIB != NULL)
            {
                  lpLeftDIBHdr  = (LPSTR) ::GlobalLock((HGLOBAL)
                              hLeftDIB);
                  lpLeftDIBBits = ::FindDIBBits(lpLeftDIBHdr);
            }
            if (hRightDIB != NULL)
            {
                  lpRightDIBHdr  = (LPSTR) ::GlobalLock((HGLOBAL)
                              hRightDIB);
                  lpRightDIBBits = ::FindDIBBits(lpRightDIBHdr);
            }
      }
      else  // if StartImage = FALSE then generate in swap mode
      {
            if (hLeftDIB != NULL)
            {
                  lpLeftDIBHdr  = (LPSTR) ::GlobalLock((HGLOBAL)
                              hRightDIB);
                  lpLeftDIBBits = ::FindDIBBits(lpLeftDIBHdr);
            }
            if (hRightDIB != NULL)
            {
                  lpRightDIBHdr  = (LPSTR) ::GlobalLock((HGLOBAL)
                              hLeftDIB);
                  lpRightDIBBits = ::FindDIBBits(lpRightDIBHdr);
            }
      }

      // Calculate picture width
      int leftWidth = (int) ::DIBWidth(lpLeftDIBHdr);
      int rightWidth = (int) ::DIBWidth(lpRightDIBHdr);

      // Calculate picture height
      int leftHeight = (int) ::DIBHeight(lpLeftDIBHdr);
      int rightHeight = (int) ::DIBHeight(lpRightDIBHdr);

      int maxWidth = (leftWidth > rightWidth) ? leftWidth :
```

```
                rightWidth;
        int maxHeight = (leftHeight > rightHeight) ? leftHeight :
                rightHeight;

        int leftBitPerPixel = (int) FindBitPerPixel(lpLeftDIBHdr);
        int rightBitPerPixel = (int) FindBitPerPixel(lpRightDIBHdr);

        if(leftBitPerPixel == rightBitPerPixel)
              bitperpixel = leftBitPerPixel;

        // Create output DIB image
        HDIB testCreateDIB = CreateDIB(maxWidth,maxHeight,
                    bitperpixel);
        if (testCreateDIB != NULL)
        {
              lpOutputDIBHdr  = (LPSTR) ::GlobalLock((HGLOBAL)
                          testCreateDIB);
              lpOutputDIBBits = ::FindDIBBits(lpOutputDIBHdr);
        }

        // patch line

        if(bitperpixel == 16)
              bpp_patchvalue = 2;
        else if(bitperpixel == 24)
              bpp_patchvalue = 3;
        else if(bitperpixel == 32)
              bpp_patchvalue = 4;

        int left_line = bpp_patchvalue * leftWidth;
        int right_line = bpp_patchvalue * rightWidth;
        int output_line = bpp_patchvalue * maxWidth;

        if(bpp_patchvalue != 4)
        {
              left_line = ((left_line+bpp_patchvalue) / 4) *4;
              right_line = ((right_line+bpp_patchvalue) / 4) *4;
              output_line = ((output_line+bpp_patchvalue) / 4) *4;
        }

// Copy odd line from left image to desternation DIB image
        for(i=0;i<leftHeight;i+=2)
        {
              leftPointer = lpLeftDIBBits+i*left_line;
              outputPointer = lpOutputDIBBits+i*output_line;

              for (j=0;j<leftWidth*bpp_patchvalue;j++)
              {
                    unsigned char temp = *leftPointer;
                    *outputPointer = temp;
                    leftPointer++;
                    outputPointer++;
              }
        }

        // Copy even line from right image to desternation DIB image
        for(i=1;i<rightHeight;i+=2)
        {
              rightPointer = lpRightDIBBits+i*right_line;
              outputPointer = lpOutputDIBBits+i*output_line;
```

```
            for (j=0;j<rightWidth*bpp_patchvalue;j++)
            {
                  unsigned char temp = *rightPointer;
                  *outputPointer = temp;
                  rightPointer++;
                  outputPointer++;
            }
      }

      ImageFinish = TRUE;
      OpenClipboard(NULL);
      EmptyClipboard();
      SetClipboardData (CF_DIB, CopyHandle((HANDLE) testCreateDIB));
      CloseClipboard();
}
```

ประวัติผู้วิจัย

นายเรืองยุทธ อินทร์อ่อน เกิดวันที่ 18 เมษายน พ.ศ. 2517 ที่กรุงเทพมหานคร สำเร็จการศึกษาระดับปริญญาตรีวิทยาศาสตรบัณฑิต สาขาคณิตศาสตร์ คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2537 และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อ พ.ศ. 2539