

TD 1

« Visualisation d'images sous X-Window »

Découvrir l'utilisation des programmes**1. Léna en couleur sur Central Park**

En vous servant du programme « visual_true_color.c » (voir site Web <http://www-igm.univ-mlv.fr/~riazano/> ou directement sur le disque dans le répertoire **~riazano/cours/MASTER/ITI**) et des images que vous trouverez sur le site Web ou sous le répertoire **~riazano/cours/IMAGES**, résoudre l'énigme en affichant à l'écran « **Léna en couleur sur Central Park** ». Léna est une image carrée et correspond à une femme dont la chair est de couleur rosée. L'image de New-York est-elle aussi carrée.

2. Gare au mandrill

Afficher l'image du mandrill. Ce singe est représenté en niveaux de gris dans une image « presque carrée ».

Examiner les fichiers image**3. Couleur du pixel (0,0)**

Dans l'image « Léna », quelles sont les valeurs des trois composantes du pixel (0,0) qui est le premier pixel situé en haut à gauche.

R = V = B =

Configuration du serveur X-Window**4. Mapping des couleurs**

En utilisant la commande « xdpinfo », déterminer la configuration du driver d'affichage utilisé par le serveur X :

- Quelles sont les dimensions de l'écran : lignes x colonnes
- Quelle est le visual utilisé par défaut : (classe)
- Quelle est la profondeur d'affichage : bits
- Quels sont les masques d'affichage des composantes (écrire en hexadécimal et en binaire)

Rouge	0x.....			
Vert	0x.....			
Bleu	0x.....			

- Combien de bits sont utilisés et quel est le décalage gauche pour chacune des composantes dans le frame buffer ?

Rouge	 bits	 bits
Vert	 bits	 bits
Bleu	 bits	 bits

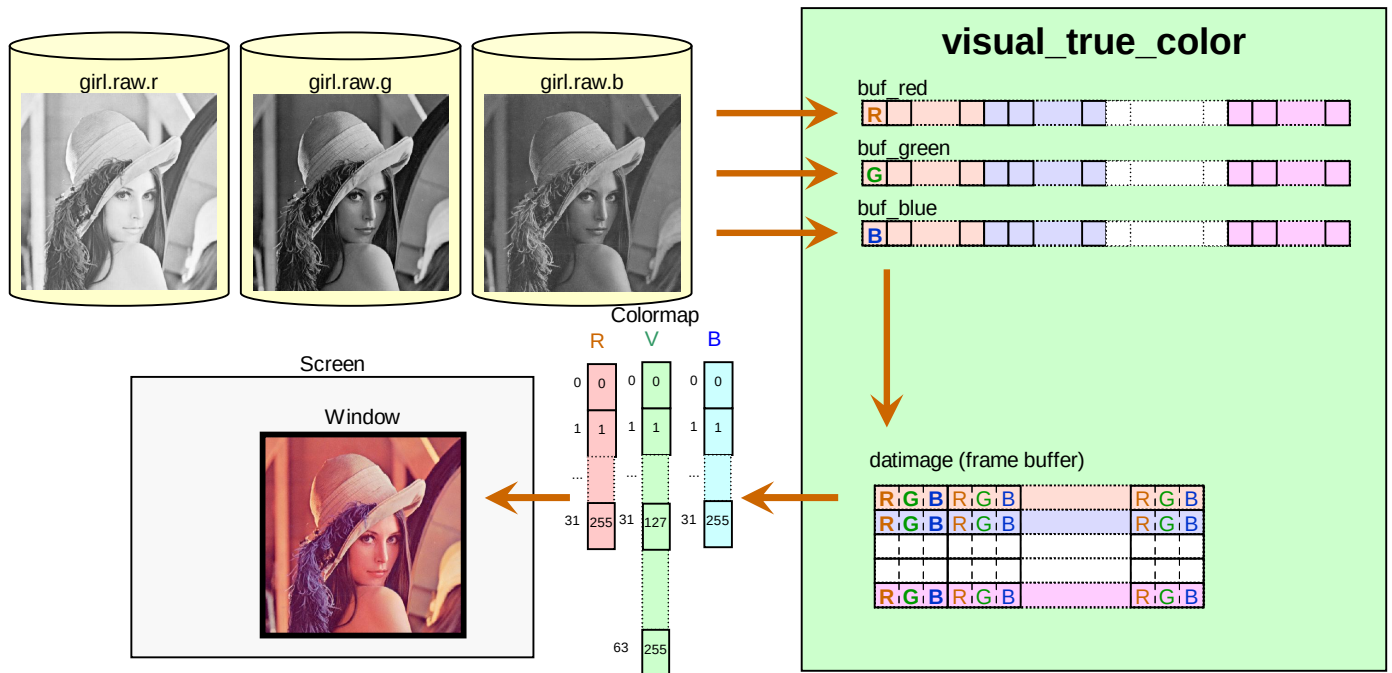
Modifier le programme visual_true_color.c**5. Couleur et représentation interne du pixel (125,17)**

Dans l'image « Léna », quelles sont les valeurs des images sur disque des trois composantes du pixel (125,17). Les coordonnées sont toujours exprimées en (ligne,colonne) et leurs valeurs commencent en 0.

R = V = B =

Quelle est la valeur dans le frame buffer du pixel (125,17) avant une éventuelle permutation LSB (little Endian) ? Ecrire en binaire puis en hexadécimal.

Pixel (125,17)	0x.....	
----------------	---------	--------



6. Comparaison des programmes « `visual_true_color` » et « `skelet` »

Le programme « `skelet` » est une amélioration du programme « `visual_true_color` ».

a. Quel est l'avantage de « `skelet` » dans la gestion des paramètres ? Illustrer ces avantages en montrant les appels possibles de « `skelet` » sur la ligne de commande.

b. Comment s'appellent les *frame buffers* dans « `visual_true_color` » et dans « `skelet` » ?

c. Pourquoi dans « `skelet` » les tâches d'initialisation de *frame buffer* ont-elles été reportées dans une fonction ?

d. Modifier le programme « `skelet` » pour que l'image traitée soit le négatif de l'image en entrée. On se servira de la constante `MAX_COLOR`.

L'image ci-contre illustre par exemple le négatif de Léna en niveaux de gris.



Le fichier « `skelet.c` » constituera dorénavant le canevas à partir duquel on réalisera dans chaque TD des essais d'algorithmes de traitement d'images dans la « **PROCESSING SECTION ».**

[illegible]

- Page 3 -

```

int          nread;          /* number of bytes actually read */

int          required_depth; /* expected depth when getting visual */
int          status;         /* status returned by X function call */

int          red_colormap_entries; /* nb.of possible values for Red */
int          red_offset; /* left offset to match the Red mask*/
int          green_colormap_entries; /* nb.of possible values for Green*/
int          green_offset; /* left offset to match the Green mask*/
int          blue_colormap_entries; /* nb.of possible values for Blue */
int          blue_offset; /* left offset to match the Blue mask*/
int          bits_per_rgb; /* bits nb.per RGB pixel in frame buf*/
int          bytes_per_rgb; /* bytes nb.per RGB pixel in frame bu*/
XSetWindowAttributes window_attributes; /* used to set window attributes */
unsigned char byte_order[4]; /* used to check byte order in int */
int          int_MSB_first; /* "Most Significant Byte first in
                           integer representation" flag */

/******
/* Connect to X server
/******
if ((display=XOpenDisplay(NULL)) == NULL)
{
    fprintf (stderr,"visual_true_color : Cannot connect to X server.\n");
    exit (1);
}
screen = DefaultScreen (display);
display_height = DisplayHeight (display,screen);
display_width = DisplayWidth (display,screen);
display_planes = DisplayPlanes (display,screen);
/******
/* Get the TrueColor visual
/******
required_depth = 24;
do
{
    if ((status=XMatchVisualInfo(display,screen,required_depth,TrueColor,
                                &visual_info)) == 0)
    {
        required_depth = required_depth - 1;
    }
} while ((status == 0) && (required_depth >= 8));
if (required_depth < 8)
{
    fprintf (stderr,
"visual_true_color : Cannot get a TrueColor visual for whatever depth.\n");
    exit (1);
}
/*=====
/* Check that the number of bits per pixels in frame buffer is multiple of 8 */
/*=====
bits_per_rgb = BitmapUnit(display);
if (required_depth == 16)
    bits_per_rgb = 16;
if (bits_per_rgb % 8 != 0)
{
    fprintf (stderr,"visual_true_color : Only number of bits per pixels in ");
    fprintf (stderr,"frame buffer multiple of 8 are supported.\n");
    exit (1);
}
bytes_per_rgb = bits_per_rgb / 8;
/******
/* Analyze the way Red, Green and Blue component are mapped in frame buffer */

```

```

/******
/* Red component
/*-----
red_colormap_entries = visual_info.red_mask;
red_offset = 0;
while ((red_colormap_entries & 0x00000001) == 0)
{
    red_offset = red_offset + 1;
    red_colormap_entries = (red_colormap_entries >> 1);
}
red_colormap_entries = red_colormap_entries + 1;
/*-----
/* Green component
/*-----
green_colormap_entries = visual_info.green_mask;
green_offset = 0;
while ((green_colormap_entries & 0x00000001) == 0)
{
    green_offset = green_offset + 1;
    green_colormap_entries = (green_colormap_entries >> 1);
}
green_colormap_entries = green_colormap_entries + 1;
/*-----
/* Blue component
/*-----
blue_colormap_entries = visual_info.blue_mask;
blue_offset = 0;
while ((blue_colormap_entries & 0x00000001) == 0)
{
    blue_offset = blue_offset + 1;
    blue_colormap_entries = (blue_colormap_entries >> 1);
}
blue_colormap_entries = blue_colormap_entries + 1;
/******
/* Get names of data files and open them
/******
/* RED file name
/*=====
if (argc >= 2)
    strcpy(file_name_red,argv[1]);
else
{
    printf("Nom du fichier image: ROUGE : ");
    scanf ("%s",file_name_red);
}
/*-----
/* Open input file
/*-----
if ((fp_red=fopen(file_name_red,"r")) == NULL)
{
    fprintf (stderr,"visual_true_color : can't open \"%s\"\n",file_name_red);
    exit (1);
}
/*=====
/* GREEN file name
/*=====
if (argc >= 3)
    strcpy(file_name_green,argv[2]);
else
{
    printf("Nom du fichier image: VERT : ");
    scanf ("%s",file_name_green);
}

```

```

/*-----*/
/* Open input file */
/*-----*/
if ((fp_green=fopen(file_name_green,"r")) == NULL)
{
    fprintf(stderr,"visual_true_color : can't open \"%s\"\n",file_name_green);
    exit (1);
}
/*=====*/
/* BLUE file name */
/*=====*/
if (argc >= 4)
    strcpy(file_name_blue,argv[3]);
else
{
    printf("Nom du fichier image: BLEU : ");
    scanf ("%s",file_name_blue);
}
/*-----*/
/* Open input file */
/*-----*/
if ((fp_blue=fopen(file_name_blue,"r")) == NULL)
{
    fprintf (stderr,"visual_true_color : can't open \"%s\"\n",file_name_blue);
    exit (1);
}
/*=====*/
/* Get size of input image */
/*=====*/
if ((argc < 5) || (sscanf(argv[4],"%d",&nliin) <= 0))
{
    printf("Nombre de lignes en entree : ");
    scanf ("%d",&nliin);
}
if ((argc < 6) || (sscanf(argv[5],"%d",&npxin) <= 0))
{
    printf("Nombre de pixels en entree : ");
    scanf ("%d",&npxin);
}
/*=====*/
/* Allocate memory for the internal frame buffer : "datimage" */
/*=====*/
if ((datimage=(unsigned char*)malloc((npxin*nliin*bytes_per_rgb)*
    sizeof(char))) == NULL)
{
    fprintf (stderr,
        "visual_true_color : Cannot allocate internal frame buffer \n");
    exit (1);
}
/*=====*/
/* Allocate memory for the buffers used to get image lines */
/*=====*/
if (((buf_red=(unsigned char*)malloc((npxin*nliin)*sizeof(char))) == NULL) ||
    ((buf_green=(unsigned char*)malloc((npxin*nliin)*sizeof(char)))==NULL) ||
    ((buf_blue=(unsigned char*)malloc((npxin*nliin)*sizeof(char))) ==NULL))
{
    fprintf (stderr,
        "visual_true_color : Cannot allocate memory for the buffers \n");
    exit (1);
}
/*=====*/
/* Initialise the frame buffer */
/*=====*/

```

```

/* Analyze order inside an integer */
/*-----*/
icolor_rgb = 0x01020304;
memcpy (byte_order,&icolor_rgb,4);
if (byte_order[0] == 0x01)
    int_MSB_first = True;
else
    int_MSB_first = False;
/*-----*/
/* Loop on lines */
/*-----*/
for (ili=0; ili<nliin; ili++)
{
    /*-----*/
    /* Read Red, Green and Blue line from files */
    /*-----*/
    if (((nread=fread(&(buf_red[ili*npxin]), sizeof(char),npxin,fp_red)) <
        npxin) ||
        ((nread=fread(&(buf_green[ili*npxin]),sizeof(char),npxin,fp_green)) <
        npxin) ||
        ((nread=fread(&(buf_blue[ili*npxin]), sizeof(char),npxin,fp_blue)) <
        npxin))
    {
        fprintf (stderr,
            "visual_true_color : error while reading record nb. %d, ",ili);
        fprintf (stderr,"(returned=%d, status=%d)\n",nread,errno);
        perror ("visual_true_color");
        exit (1);
    }
    /*-----*/
    /* Interleave Red, Green and Blue components into frame buffer */
    /*-----*/
    for (ipx=0; ipx<npxin; ipx++)
    {
        /*-----*/
        /* Set RGB values according to their colormap entries */
        /*-----*/
        icolor_red = nint((float)red_colormap_entries * buf_red[ipx] / 256);
        if (icolor_red >= red_colormap_entries)
            icolor_red = red_colormap_entries - 1;
        icolor_green = nint((float)green_colormap_entries* buf_green[ipx]/256);
        if (icolor_green >= green_colormap_entries)
            icolor_green = green_colormap_entries - 1;
        icolor_blue = nint((float)blue_colormap_entries * buf_blue[ipx] /256);
        if (icolor_blue >= blue_colormap_entries)
            icolor_blue = blue_colormap_entries - 1;
        /*-----*/
        /* Combine the three components according to their masks */
        /*-----*/
        icolor_rgb = (icolor_red << red_offset) |
            (icolor_green << green_offset) |
            (icolor_blue << blue_offset);
        /*-----*/
        /* Report value in frame buffer */
        /*-----*/
        if (int_MSB_first)
        {
            memcpy (&(datimage[(ili*npxin+ipx)*bytes_per_rgb]),
                &((unsigned char*)&icolor_rgb)[sizeof(int)-bytes_per_rgb],
                bytes_per_rgb);
        }
        else
        {

```

```

        memcpy (&(datimage[(ili*npxin+ipx)*bytes_per_rgb]),
                &icolor_rgb,bytes_per_rgb);
    }
} /* Loop on pixels */
} /* Loop on lines */
/* ***** */
/* Allocate memory for an XImage structure */
/* ***** */
visual = visual_info.visual;
format = ZPixmap;
offset = 0;
bitmap_pad = 8;
bytes_per_line = 0;
if ((ximage=XCreateImage(display,visual,display_planes,format,offset,
    (char*)datimage,npxin,nliin,bitmap_pad,bytes_per_line)) == NULL)
{
    fprintf (stderr,"visual_true_color : cannot create ximage \n");
    exit (1);
}
/* ***** */
/* Create a window */
/* ***** */
if ((win=XCreateWindow (
    display,                /* connection to X server */
    RootWindow(display,screen), /* parent window */
    x,y,                    /* coord.of UL corner in parent window*/
    npxin,nliin,            /* width/height of window */
    0,                      /* border width */
    required_depth,         /* depth of window in bits */
    InputOutput,            /* class of this window */
    visual_info.visual,     /* visual to be used for the window */
    (unsigned long)0,       /* value mask within window_attributes*/
    &window_attributes)) == 0)
{
    fprintf (stderr,"visual_true_color : Cannot create the window \n");
    exit (1);
}
sprintf (window_title,"%s / %s / %s",file_name_red,file_name_green,
    file_name_blue);
XStoreName (display,win,window_title);
/* ***** */
/* Select events that will be received by window */
/* ***** */
XSelectInput (display,win,ExposureMask | ButtonPressMask);
/* ***** */
/* Display the window */
/* ***** */
XMapWindow (display,win);
/* ***** */
/* Get graphic context */
/* ***** */
gc = DefaultGC(display,screen);
src_x = 0;
src_y = 0;
dst_x = 0;
dst_y = 0;
/* ***** */
/* Events loop */
/* ***** */
while (True)
{
    /* ***** */
    /* Get next event */
    /* ***** */

```

```

/* ***** */
XNextEvent (display,&event);
switch (event.type) {
/* ***** */
/* Expose => send image into the window */
/* ***** */
case Expose:
    XPutImage (display,win,gc,ximage,
        src_x,src_y,dst_x,dst_y,npxin,nliin);
    break;
/* ***** */
/* ButtonPress => close display and exit application */
/* ***** */
case ButtonPress:
    XCloseDisplay (display);
    exit (0);
/* ***** */
/* Any other event is not processed */
/* ***** */
default:
    break;
}
} /* event loop */
} /* Application core */

```