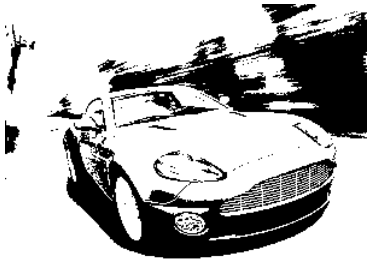
EVRARD Mickael
GLOEIKLER Florian

Projet tutoré de Visual Basic :



EVRARD Mickael
GLOEIKLER Florian

Projet tutoré de Visual Basic :
Création d'un logiciel de retouche d'image BMP 24bits

Ce projet tutoré est la mise en application de tout l'apprentissage effectué jusqu'à présent en Borland C et en Visual Basic. Il permet la manipulation de fichiers (ouvrir, lire, écrire, fermer). Il a aussi pour but de découvrir la structure d'un fichier image, et l'écriture de DLL (Dynamique Link Librairy).

Le projet tutoré consiste a modifier une image BMP 24 bits par différentes manières

Géométriques :
- *Symétrie* :
  - Horizontal
  - Verticale
- *Rotation* :
  - Horaire
  - Anti-horaire
- *Changement* de taille

Colorimétriques :
- *Luminosité* :
- *Contraste* :
- *Mise en niveaux de gris* :
- *Seuillage*
- *Contraste*
- *Estompage*

Artistique :
- *Négatif*
- *Posterisation*

Pour ce faire, 1) Etude d'un fichier BMP
2) Conception de manipulations en Borland C
3) Conception de la DLL
4) Conception du programme en VB

I) Structure d'un fichier BMP (24bits dans notre cas)

	Champ	Octet	Contenu
Entête du fichier	Type	2	Caractère B et M
	Taille du fichier	4	Taille du fichier
	Res	4	valeur 0
	Offset	4	Position de l'image après l'entête
Entête d'informations	Taille de l'entête	4	Taille de l'entête
	Largeur	4	Largeur de l'image
	Hauteur	4	Hauteur de l'image
	Plans	2	Valeur 1
	Couleurs	2	Nbre de bits par couleur (24)
	Compression	4	Description de la compression (0)
	Taille de l'image	4	Taille de l'image
	X pixels par mètre	4	Résolution Horizontale
	Y pixels par mètre	4	Résolution Verticale
	Couleurs Utilisées	4	Nbre de couleurs utilisées (0)
	Couleurs Importantes	4	Nbre de couleurs importantes (0)
Table de couleur si necessaire	Composante Bleu, Vert, Rouge	3	Intensité du bleu, du vert et du rouge (chaque intensité sur 255)
	Res	1	Valeur 0
	Composante Bleu, Vert, Rouge	3	Intensité du bleu, du vert et du rouge (chaque intensité sur 255)
	Res	1	Valeur 0
	,,,		
Table de pixels	Composante Bleu, Vert, Rouge	3	Intensité du bleu, du vert et du rouge (chaque intensité sur 255)
	Composante Bleu, Vert, Rouge	3	Intensité du bleu, du vert et du rouge (chaque intensité sur 255)
	,,,		
Bourrage	Composante Bleu, Vert, Rouge	3	Intensité du bleu, du vert et du rouge (chaque intensité sur 255)
	Composante Bleu, Vert, Rouge	3	Intensité du bleu, du vert et du rouge (chaque intensité sur 255)
	,,,		

Exemple sur une image concrète, rectangle de 3x6 en vert primaire

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	42	4D	7E	00	00	00	00	00	00	00	36	00	00	00	28	00
10	00	00	03	00	00	00	06	00	00	00	01	00	18	00	00	00
20	00	00	48	00	00	00	00	00	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	FF	00	00	FF	00	00	FF	00	00
40	00	00	00	FF	00	00	FF	00	00	FF	00	00	00	00	00	FF
50	00	00	FF	00	00	FF	00	00	00	00	00	FF	00	00	FF	00
60	00	FF	00	00	00	00	00	FF	00	00	FF	00	00	FF	00	00
70	00	00	00	FF	00	00	FF	00	00	FF	00	00	00	00		

2 octets, en ASCII, 42h 4Dh => ‘’BM’’
4 octets, fichier de taille 126octet
4 octets, => ‘’0’’
4 octets, => début du fichier en 36h
4 octets, => entête de 40 octets
4 octets, => 3 pixels de large
4 octets, => 6 pixels de haut
2 octets, => ‘’1’’
2 octets, => 24bits par pixel
4octets, => compression = 0
4 octets, => image de 72 octets
4 octets,
4 octets,
4 octets,
4 octets,
3 octets (x3), 0-255-0 => verts
3-octets, 0-0-0 => bourrage
3 octets (x3), 0-255-0 => verts
3-octets, 0-0-0 => bourrage
(…)

Pour le calcul des octets de bourrage, une ligne en octet doit être un multiple de 4, par exemple, une image de 4 pixels de largeur, soit 12 octets ne nécessite pas de bourrage mais une image de 7 pixels de largeur, soit 21, nécessite 3 octets. Dans notre cas, une ligne fait 3 pixels, donc 9 octets, il faut alors ajouter 3 octets pour arriver à 12 (octet de bourrage entouré en noir dans le tableau)

II) Manipulation

Chaque transformation partira de ce fichier original :

II-0) Traitement de l'entête

Traitement commun à toutes les fonctions

```
int entête(char *nomFicE, char *nomFicS, int dim[2])
{
  FILE *ptrE, *ptrS;                                  // Pointeurs de fichiers
  unsigned char bm[2];                                // Buffer pour les lettres BM
  short couleur;                                      // Nombre de couleurs

  ptrE=fopen(nomFicE,"rb");                           // Ouverture du fichier d'entrée en lecture

  if (ptrE==NULL)                                     // Le fichier d'entrée existe-t-il ?
    return 1;

  ptrS=fopen(nomFicS,"wb");                           // Ouverture du fichier de sortie écriture

  fread(bm,sizeof(unsigned char),2,ptrE);             // Lire et stocker les 2 premiers octets (car. B et M)

  fseek(ptrE,16,SEEK_CUR);                            // Avancer de 16 octets

  fread(dim,sizeof(int),2,ptrE);                      // Lire et recopier les dimensions

  fseek(ptrE,2,SEEK_CUR);                             // Avancer de 2 octets

  fread(&couleur,sizeof(short),1,ptrE);               // Lire et stocker le nb de couleurs

  if((bm[0]!='B') || (bm[1]!='M') || (couleur!=24))   // Vérification du format de l'image
  {
    fclose(ptrE); fclose(ptrS);                       // Si format incorrect, fermer et effacer fichier de sortie
    unlink(nomFicS);
  return 2;
  }
  fclose(ptrE); fclose(ptrS);                         // Fermeture des fichiers

  return 0;                                           // Traitement entête OK
}
```

II-1) Traitement de la fonction

Traitement commun à toutes les fonctions (sauf posterisation)

```
void main()
{
  char nomFicL[80], nomFicE[80];
  int ret;

  printf("\nNom de l'image a traiter : ");
  scanf("%s",nomFicL);
  printf("\nNom de l'image transformee : ");
  scanf("%s",nomFicE);

  ret=SymetrieHorizontale(nomFicL, nomFicE);
  switch(ret)
  {
        case 0 : printf("\nTraitement termine"); break;
        case 1 : printf("\nFichier inexistant"); break;
        case 2 : printf("\nL'image n'est pas au format BMP 24 bits");
  }
  getch();
}
```

II-2) Symétrie Horizontale

Symétrie selon un axe horizontal

Pour obtenir une symétrie horizontale, il faut inverser l'ordre des lignes de l'image. A l'aide de l'accès direct, il est inutile de stocker l'image dans un tableau.

```
int SymetrieHorizontale( char *nomFicE, char *nomFicS)
{
  FILE *ptrE, *ptrS;                                  // Pointeurs de fichiers
  int ret;                                            // Retour de la fonction
  unsigned char entête[54];                           // Buffer de recopie de l'entête
  unsigned char b, ob;                                // Nombre d'octets de bourrage, octet de bourrage
  int dim[2];                                         / Dimensions de l'image (largeur, hauteur)
  int i, j;                                           // Index courants
  unsigned char octet;                                // Pour stocker un octet
  long pixDeb;                                        // Position pixel de début de ligne
  int nbOctetParLigne;                                // Nombre d'octets, bourrage compris, d'une ligne
  ret=entête(nomFicE, nomFicS, dim);                  // Traitement de l'entête
  if(ret!=0)
    return ret;

  ptrE=fopen(nomFicE,"rb");                           // Ouverture des fichiers
  ptrS=fopen(nomFicS,"wb");                           // Lire et recopier 54 octets
  fread(entête,sizeof(unsigned char),54,ptrE);
  fwrite(entête,sizeof(unsigned char),54,ptrS);
  b=4-(3*dim[0]%4)&3;                                 // Calculer le nb d'octets à 0 de bourrage
  nbOctetParLigne=3*dim[0]+b;                         // Nombre d'octets, bourrage compris, d'une ligne
  pixDeb=54+(dim[1]-1)*nbOctetParLigne;               // Position du premier octet de la ligne (dernière ligne en premier)
  for(i=0;i<dim[1];i++)
  {
    fseek(ptrE,pixDeb,0);                             // Se positionner sur le premier pixel de la ligne
    for(j=0;j<nbOctetParLigne;j++)
    {
      fread(&octet,sizeof(octet),1,ptrE);             // Lire pixel dans E
      fwrite(&octet,sizeof(octet),1,ptrS);            // Ecrire pixel dans S
    }
    pixDeb=pixDeb-nbOctetParLigne;                    // Reculer d'une ligne
  }
  fclose(ptrE); fclose(ptrS);                         // Fermeture des fichiers
  return 0;                                           // Traitement OK
}
```

II-3) Symétrie Verticale

Symétrie selon un axe vertical

Pour obtenir une symétrie verticale, il faut inverser l'ordre des pixels de chaque ligne de l'image. A l'aide de l'accès direct, il est inutile de stocker l'image dans un tableau.

```
int SymetrieVerticale( char *nomFicE, char *nomFicS)
{
  FILE *ptrE, *ptrS;                                  // Pointeurs de fichiers
  int ret;                                            // Retour de la fonction
  unsigned char entête[54];                           // Buffer de recopie de l'entête
  unsigned char b, ob                                 // Nombre d'octets de bourrage, octet de bourrage
  int dim[2];                                         // Dimensions de l'image (largeur, hauteur)
  int i, j;                                           // Index courants
  long pixDeb, p;                                     // Position pixel début ligne, pixel courant
  struct pixel                                        // Contenu d'un pixel
  {
    unsigned char bleu;
    unsigned char vert;
    unsigned char rouge;
  } point;
  ret=entête(nomFicE, nomFicS, dim);                  // Traitement de l'entête
  if(ret!=0)
    return ret;

  ptrE=fopen(nomFicE,"rb");                           // Ouverture des fichiers
  ptrS=fopen(nomFicS,"wb");
  fread(entête,sizeof(unsigned char),54,ptrE);        // Lire et recopier 54 octets
  fwrite(entête,sizeof(unsigned char),54,ptrS);
  b=4-(3*dim[0]%4)&3;                                 // Calculer le nb d'octets à 0 de bourrage
  pixDeb=54;                                          // Position du premier pixel de la ligne
  for(i=0;i<dim[1];i++)
  {
    p=pixDeb+3*(dim[0]-1);                            // Position du dernier pixel de la ligne
    for(j=0;j<dim[0];j++)
    {
      fseek(ptrE,p,0);                                // Se positionner sur le dernier pixel de la ligne
      fread(&point,sizeof(point),1,ptrE);             // Lire pixel dans E
      fwrite(&point,sizeof(point),1,ptrS);            // Ecrire pixel dans S
      p=p-3;                                          // Reculer d'un pixel
    }
    for(j=0;j<b;j++)                                  // Ajouter les octets de bourrage
    {
        fread(&ob,sizeof(ob),1,ptrE);
        fwrite(&ob,sizeof(ob),1,ptrS);
    }
    pixDeb=pixDeb + 3*dim[0] + b;                     // Passer à la ligne suivante
  }
  fclose(ptrE); fclose(ptrS);                         // Fermeture des fichiers
  return 0;                                           // Traitement OK
}
```

II-4) Rotation Horaire

Rotation dans le sens horaire

La rotation est de 90° dans le sens horaire. A l'aide de l'accès direct, il est inutile de stocker l'image dans un tableau.

```
int RotationHoraire( char *nomFicE, char *nomFicS)
{
  FILE *ptrE, *ptrS;                          // Pointeurs de fichiers
  int ret;                                    // Retour de la fonction
  unsigned char entête[54];                   // Buffer de recopie de l'entête
  unsigned char bE, bS, ob=0;                 // Nombres d'octets de bourrage, octet de bourrage
  int dim[2];                                 // Dimensions de l'image (largeur, hauteur)
  int largeurE, hauteurE;                     // Dimensions de l'image en entrée
  int largeurS, hauteurS;                     // Dimensions de l'image en sortie
  int i, j, k;                                // Index courants
  unsigned char octet;                        // Pour stocker un octet
  long pixCE;                                 // Position pixel courant
  long pixDE;                                 // Position pixel de début de ligne
  int nbOctetParLigne;                        // Nombre d'octets, bourrage compris, d'une ligne
  struct pixel
  {
    unsigned char bleu;
    unsigned char vert;
    unsigned char rouge;
  }
  point;
  ret=entête(nomFicE, nomFicS, dim);          // Traitement de entête
  if(ret!=0)
    return ret;

  ptrE=fopen(nomFicE,"rb");                   // Ouverture des fichiers
  ptrS=fopen(nomFicS,"wb+");
  fread(entête,sizeof(unsigned char),54,ptrE);// Lire et recopier 54 octets
  fwrite(entête,sizeof(unsigned char),54,ptrS);
  largeurE=dim[0]; hauteurE=dim[1];           // Calculer le nb d'octets à 0 de bourrage en lecture
  largeurS=dim[1]; hauteurS=dim[0];           // Calculer le nb d'octets à 0 de bourrage en écriture
  bE=4-(3*largeurE%4)&3;                      // Nombre d'octets, bourrage compris, d'une ligne
  bS=4-(3*largeurS%4)&3;
  nbOctetParLigne=3*largeurE + bE;            // Position du pixel courant dans la ligne
  pixCE=(largeurE-1)*3;                       // Position du premier octet de la première ligne
  for(i=0;i<largeurE;i++)
  {
    pixDE=54;
    for(j=0;j<hauteurE;j++)                   // Se positionner sur le pixel
    {
      fseek(ptrE,pixDE+pixCE,SEEK_SET);
      fread(&point,sizeof(point),1,ptrE);     // Lire pixel dans E
      fwrite(&point,sizeof(point),1,ptrS);    // Ecrire pixel dans S
```

```
      pixDE=pixDE+nbOctetParLigne;          // Avancer d'une ligne
    }
    for(k=0;k<bS;k++)                       // Ecrire les octets de bourrage
      fwrite(&ob,sizeof(ob),1,ptrS);
    pixCE=pixCE-3;                          // Reculer d'un pixel
  }
  fseek(ptrS,18,SEEK_SET);
  fwrite(&largeurS,sizeof(largeurS),1,ptrS);  // Mettre à jour la largeur et la hauteur
  fwrite(&hauteurS,sizeof(hauteurS),1,ptrS);
  fclose(ptrE); fclose(ptrS);               // Fermeture des fichiers
  return 0;
}
```

II-5) Rotation Trigo

Rotation dans le sens trigo.

La rotation est de 90° dans le sens trigo. A l'aide de l'accès direct, il est inutile de stocker l'image dans un tableau.

```
int RotationTrigo( char *nomFicE, char *nomFicS)
{
  FILE *ptrE, *ptrS;                                  // Pointeurs de fichiers
  int ret;                                            // Retour de la fonction
  unsigned char entête[54];                           // Buffer de recopie de l'entête
  unsigned char bE, bS, ob=0;                         // Nombres d'octets de bourrage, octet de bourrage
  int dim[2];                                         // Dimensions de l'image (largeur, hauteur)
  int largeurE, hauteurE;                             // Dimensions de l'image en entrée
  int largeurS, hauteurS;                             // Dimensions de l'image en sortie
  int i, j, k;                                        // Index courants
  unsigned char octet;                                // Pour stocker un octet
  long pixCE;                                         // Position pixel courant
  long pixDE;                                         // Position pixel de début de ligne
  int nbOctetParLigne;                                // Nombre d'octets, bourrage compris, d'une ligne
  struct pixel
  {
    unsigned char bleu;
    unsigned char vert;
    unsigned char rouge;
  }
  point;
  ret=entête(nomFicE, nomFicS, dim);                  // Traitement de l'entête
  if(ret!=0)
    return ret;

  ptrE=fopen(nomFicE,"rb");                           // Ouverture des fichiers
  ptrS=fopen(nomFicS,"wb+");
  fread(entête,sizeof(unsigned char),54,ptrE);        // Lire et recopier 54 octets
  fwrite(entête,sizeof(unsigned char),54,ptrS);
  largeurE=dim[0]; hauteurE=dim[1];
  largeurS=dim[1]; hauteurS=dim[0];
  bE=4-(3*largeurE%4)&3;                              // Calculer le nb d'octets à 0 de bourrage en lecture
  bS=4-(3*largeurS%4)&3;                              // Calculer le nb d'octets à 0 de bourrage en écriture
  nbOctetParLigne=3*largeurE + bE;                    // Nombre d'octets, bourrage compris, d'une ligne
  pixCE=0;                                            // Position du pixel courant dans la ligne
  for(i=0;i<largeurE;i++)
  {
    pixDE=54+(hauteurE-1)*nbOctetParLigne;            // Position du premier octet de la dernière ligne
    for(j=0;j<hauteurE;j++)
    {
      fseek(ptrE,pixDE+pixCE,0);                      // Se positionner sur le pixel
      fread(&point,sizeof(point),1,ptrE);             // Lire pixel dans E
      fwrite(&point,sizeof(point),1,ptrS);            // Ecrire pixel dans S
```

```
      pixDE=pixDE-nbOctetParLigne;          // Reculer d'une ligne
    }
    for(k=0;k<bS;k++)                       // Ecrire les octets de bourrage
      fwrite(&ob,sizeof(ob),1,ptrS);
    pixCE=pixCE+3;
  }                                         // Avancer d'un pixel
  fseek(ptrS,18,SEEK_SET);
  fwrite(&largeurS,sizeof(largeurS),1,ptrS);  // Mettre à jour la largeur et la hauteur
  fwrite(&hauteurS,sizeof(hauteurS),1,ptrS);
  fclose(ptrE); fclose(ptrS);
  return 0;                                 // Fermeture des fichiers
}
```

II-6) Dimension

Dimension.

Le changement de dimension est obtenu en supprimant des pixels pour une réduction ou en les dupliquant pour un agrandissement.

```
int Dimension( char *nomFicE, char *nomFicS, int d)
{
  FILE *ptrE, *ptrS;                          // Pointeurs de fichiers
  int ret ;                                   // Valeur de retour
  unsigned char entête[54];                   // Buffer de recopie de l'entête
  unsigned char bE, bS, ob=0;                 // Bourrage d'entrée, de sortie, octet
  int dim[2]                                  // Dimensions de l'image (largeur, hauteur)
  int i, j, k;                                // Index courants
  float coef=d/100.;                          // coefficient d'agrandissement-réduction
  unsigned char BufL[3075];                   // Tableau pour le stockage d'une ligne
  int iBufL;                                  // Pointeur dans le tableau BufL
  int ptrLargeurE;                            // Pointeur sur la largeur de l'image d'entrée
  int ptrHauteurE;                            // Pointeur sur la hauteur de l'image d'entrée
  float ptrLargeurS;                          // Pointeur sur la largeur de l'image de sortie
  float ptrHauteurS;                          // Pointeur sur la hauteur de l'image de sortie
  int nbLigneEcrite;                          // Nombre de lignes réellement écrites
  int nbPixelEcrit;                           // Taille du fichier en octets
  int tailleEnOctets;                         // Nombre de pixels par ligne réellement écrits
  struct pixel                                // Contenu d'un pixel
  {
    unsigned char bleu;
    unsigned char vert;
    unsigned char rouge;
  }
  point;
  if(dim[0]>1024)                             // Vérification de la taille de l'image par rapport au buffer
  {
    unlink(nomFicS);                          // Si taille trop importante, fermer et effacer fichier de sortie
    ret=3;
  }
  ret=entête(nomFicE, nomFicS, dim);          // Traitement de l'entête
  if(ret!=0)
    return ret;

  ptrE=fopen(nomFicE,"rb");                   // Ouverture des fichiers
  ptrS=fopen(nomFicS,"wb");
  fread(entête,sizeof(unsigned char),54,ptrE);// Lire et recopier 54 octets
  fwrite(entête,sizeof(unsigned char),54,ptrS);
  bE=4-(3*dim[0]%4)&3;                        // Nb d'octets à 0 de bourrage en entrée
  nbLigneEcrite=0;                            // Init nombre de lignes réellement écrites
  ptrHauteurE=0;ptrHauteurS=0;                // Init pointeurs hauteur images entrée et sortie
  for(i=0;i<dim[1];i++)
  {
    nbPixelEcrit=0;                           // Init nombre de pixels réellement écrits
    ptrLargeurE=0;ptrLargeurS=0;              // Init pointeurs largeur images entrée et sortie
    iBufL=-1;                                 // Init du pointeur dans buffer de ligne
    for(j=0;j<dim[0];j++)
    {
```

```
      fread(&point,sizeof(point),1,ptrE);                  // Recopier un pixel
      fwrite(&point,sizeof(point),1,ptrS);
      nbPixelEcrit++;
      iBufL++; BufL[iBufL]=point.bleu;                     // Copier le pixel dans le buffer
      iBufL++; BufL[iBufL]=point.vert;
      iBufL++; BufL[iBufL]=point.rouge;
      ptrLargeurE=ptrLargeurE+1;                           // Incrémenter les pointeurs dans les images d'entrée et de sortie
      ptrLargeurS=ptrLargeurS+coef;
      if(ptrLargeurE<(int)ptrLargeurS)                     // Tester l'égalité des pointeurs
      {                                                    // Recopier un ou plusieurs pixels si agrandissement
        while(ptrLargeurE<(int)ptrLargeurS)
        {
          fwrite(&point,sizeof(point),1,ptrS);
          nbPixelEcrit++;
          iBufL++; BufL[iBufL]=point.bleu;
          iBufL++; BufL[iBufL]=point.vert;
          iBufL++; BufL[iBufL]=point.rouge;
          ptrLargeurE=ptrLargeurE+1;
        }
      }
      else
      if(ptrLargeurE>(int)ptrLargeurS)
      {
        while((ptrLargeurE>(int)ptrLargeurS) && j<(dim[0]-1))  // Passer un ou plusieurs pixels si réduction
        {
          fread(&point,sizeof(point),1,ptrE);
          ptrLargeurS=ptrLargeurS+coef;
          j++;
        }
      }
    }
    nbLigneEcrite++;
    bS=4-(3*nbPixelEcrit%4)&3;                             // Nb d'octets à 0 de bourrage en sortie
    for(k=0;k<bE;k++)                                      // Lire les octets de bourrage
      fread(&ob,sizeof(ob),1,ptrE);
    ob=0;                                                  // Ecrire les octets de bourrage
    for(k=0;k<bS;k++)
    {
      fwrite(&ob,sizeof(ob),1,ptrS);
      iBufL++; BufL[iBufL]=ob;
    }                                                      // Incrémenter les pointeurs dans les images d'entrée et de sortie
    ptrHauteurE=ptrHauteurE+1;
    ptrHauteurS=ptrHauteurS+coef;
    if(ptrHauteurE<(int)ptrHauteurS)                       // Tester l'égalité des pointeurs
      while(ptrHauteurE<(int)ptrHauteurS)                  // Recopier une ou plusieurs lignes si agrandissement
      {
        for(k=0;k<=iBufL;k++)
          fwrite(&BufL[k],sizeof(unsigned char),1,ptrS);
        nbLigneEcrite++;
        ptrHauteurE=ptrHauteurE+1;
      }
    }
    else
    if(ptrHauteurE>(int)ptrHauteurS)                       // Passer un ou plusieurs lignes si réduction
      {
        while((ptrHauteurE>(int)ptrHauteurS) && i<(dim[1]-1))
        {
          for(k=0;k<dim[0];k++)                            // Lire les octets de bourrage
            fread(&point,sizeof(point),1,ptrE);
          for(k=0;k<bE;k++)
            fread(&ob,sizeof(ob),1,ptrE);
          ptrHauteurS=ptrHauteurS+coef;
          i++;
        }
      }
  }

  fseek(ptrS,18,SEEK_SET);                                 // Mettre à jour la largeur et la hauteur
  fwrite(&nbPixelEcrit,sizeof(nbPixelEcrit),1,ptrS);
  fwrite(&nbLigneEcrite,sizeof(nbLigneEcrite),1,ptrS);
  fseek(ptrS,34,SEEK_SET);                                 // Mettre à jour la taille du fichier
```

tailleEnOctets=nbLigneEcrite*iBufL;
fwrite(&tailleEnOctets,sizeof(tailleEnOctets),1,ptrS);
fclose(ptrE);
fclose(ptrS);
return 0;
}
	// Fermeture des fichiers
	
	// Traitement OK

II-7) Lumière

Lumière

La modification du contraste s'obtient au augmentant ou en diminuant la valeur des composantes de chaque pixel a l'aide d'un coefficient additif ou soustractif.

```
int Lumiere( char *nomFicE, char *nomFicS, int coef)
{
  FILE *ptrE, *ptrS;                                  // Pointeurs de fichiers
  int ret;                                            // Retour de la fonction
  int dim[2];                                         // Dimensions de l'image (largeur, hauteur)
  unsigned char entête[54];                           // Buffer de recopie de l'entête
  unsigned char b, ob;                                // Nombre d'octets et octet de bourrage
  int i, j;                                           // Index courants
  struct pixel                                        // Contenu d'un pixel
  {
    unsigned char bleu;
    unsigned char vert;
    unsigned char rouge;
  }
  point;
  struct pixelInt                                     // Copie pixel dans type Entier
  {
    int bleu;
    int vert;
    int rouge;
  }
  pointInt;
  ret=entête(nomFicE, nomFicS, dim);                  // Traitement de l'entête
  if(ret!=0)
    return ret;

  ptrE=fopen(nomFicE,"rb");                           // Ouverture des fichiers
  ptrS=fopen(nomFicS,"wb");
  fread(entête,sizeof(unsigned char),54,ptrE);        // Lire et recopier 54 octets
  fwrite(entête,sizeof(unsigned char),54,ptrS);
  b=4-(3*dim[0]%4)&3;                                 // Calculer le nb d'octets à 0 de bourrage
  for(i=0;i<dim[1];i++)
  {
    for(j=0;j<dim[0];j++)
    {
      fread(&point,sizeof(point),1,ptrE);             // Lire pixel dans E
      pointInt.bleu=point.bleu + coef;                // Modifier lumière
      pointInt.vert=point.vert + coef;
      pointInt.rouge=point.rouge + coef;
      pointInt.bleu=pointInt.bleu>255?255:pointInt.bleu;   // Vérifier >255
      pointInt.vert=pointInt.vert>255?255:pointInt.vert;
      pointInt.rouge=pointInt.rouge>255?255:pointInt.rouge;
      point.bleu=pointInt.bleu<0?0:pointInt.bleu;     // Vérifier <0
      point.vert=pointInt.vert<0?0:pointInt.vert;
      point.rouge=pointInt.rouge<0?0:pointInt.rouge;
      fwrite(&point,sizeof(point),1,ptrS);            // Ecrire pixel dans S
    }
    for(j=0;j<b;j++)
```

```
    {
      fread(&ob,sizeof(ob),1,ptrE);
      fwrite(&ob,sizeof(ob),1,ptrS);
    }
  }

  fclose(ptrE);
  fclose(ptrS);
  ret=0;
  return ret;
}
```

// Lire écrire les octets de bourrage

// Fermeture des fichiers

// Traitement OK

II-9) Contraste

Contraste

La modification du contraste s'obtient au augmentant ou en diminuant la valeur des composantes de chaque pixel a l'aide d'un multiplicateur

```
int Contraste(char *nomFicE, char *nomFicS, int coef)
{
  FILE *ptrE, *ptrS;                                  // Pointeurs de fichiers
  int ret;                                            // Retour de la fonction
  unsigned char entête[54];                           // Buffer de recopie de l'entête
  unsigned char b, ob;                                // Nombre d'octets de bourrage, octet de bourrage
  int dim[2];                                         // Dimensions de l'image (largeur, hauteur)
  int i, j;                                           // Index courants
  struct pixel                                        // Contenu d'un pixel
  {
    unsigned char bleu;
    unsigned char vert;
    unsigned char rouge;
  }
  point;
  struct pixelInt                                     // Copie pixel dans type Entier
  {
    int bleu;
    int vert;
    int rouge;
  }
  pointInt;
  ret=entête(nomFicE, nomFicS, dim);                  // Traitement de l'entête
  if(ret!=0)
    return ret;

  ptrE=fopen(nomFicE,"rb");                           // Ouverture des fichiers
  ptrS=fopen(nomFicS,"wb");
  fread(entête,sizeof(unsigned char),54,ptrE);        // Lire et recopier 54 octets
  fwrite(entête,sizeof(unsigned char),54,ptrS);
  b=4-(3*dim[0]%4)&3;                                 // Calculer le nb d'octets à 0 de bourrage
  for(i=0; i<dim[1]; i++)
  {
  for(j=0; j<dim[0]; j++)
    {
      fread(&point,sizeof(point),1,ptrE);             // Lire pixel dans E
      pointInt.bleu = point.bleu  * coef/100.;        // Modifier lumiére
      pointInt.vert = point.vert  * coef/100.;
      pointInt.rouge= point.rouge * coef/100.;
      point.bleu = pointInt.bleu > 255 ? 255 : pointInt.bleu;   // Vérifier saturation
      point.vert = pointInt.vert > 255 ? 255 : pointInt.vert;
      point.rouge= pointInt.rouge> 255 ? 255 :pointInt.rouge;
      fwrite(&point,sizeof(point),1,ptrS);            // Ecrire pixel dans S
    }
    for(j=0; j<b; j++)
    {
      fread (&ob,sizeof(ob),1,ptrE);                  // Lire-écrire les octets de bourrage
      fwrite(&ob,sizeof(ob),1,ptrS);
```

```
    }
  }
  fclose(ptrE); fclose(ptrS);          // Fermeture des fichiers
  return 0;                            // Traitement OK
}
```

II-10) Niveau de gris

Niveau de gris

La mise en niveau de gris consiste à effectuer la moyenne des 3 composantes de chaque pixel, puis de remplacer ces 3 composantes par cette moyenne.

```
int NiveauDeGris( char *nomFicE, char *nomFicS)
{
  FILE *ptrE, *ptrS;                                  // Pointeurs de fichiers
  int ret;                                            // Retour de la fonction
  unsigned char entête[54];                           // Buffer de recopie de l'entête
  unsigned char b, ob;                                // Nombre d'octets de bourrage, octet de bourrage
  int dim[2];                                         // Dimensions de l'image (largeur, hauteur)
  int i, j;                                           // Index courants
  unsigned char moyenne;                              // Moyenne des 3 couleurs
  struct pixel                                        // Contenu d'un pixel
  {
    unsigned char bleu;
    unsigned char vert;
    unsigned char rouge;
  }
  point;
  ret=entête(nomFicE, nomFicS, dim);                  // Traitement de l'entête
  if(ret!=0)
    return ret;
  ptrE=fopen(nomFicE,"rb");                           // Ouverture des fichiers
  ptrS=fopen(nomFicS,"wb");
  fread(entête,sizeof(unsigned char),54,ptrE);        // Lire et recopier 54 octets
  fwrite(entête,sizeof(unsigned char),54,ptrS);
  b=4-(3*dim[0]%4)&3;                                 // Calculer le nb d'octets à 0 de bourrage
  for(i=0;i<dim[1];i++)
  {
  for(j=0;j<dim[0];j++)
    {
      fread(&point,sizeof(point),1,ptrE);             // Lire pixel dans E
      moyenne=(point.bleu+point.vert+point.rouge)/3;  // Moyenne des 3 composantes
      point.bleu=moyenne;                             // Composantes = moyenne
      point.vert=moyenne;
      point.rouge=moyenne;
      fwrite(&point,sizeof(point),1,ptrS);            // Ecrire pixel dans S
    }
    for(j=0;j<b;j++)
    {
    fread(&ob,sizeof(ob),1,ptrE);                     // Lire-écrire les octets de bourrage
    fwrite(&ob,sizeof(ob),1,ptrS);
    }
  }
  fclose(ptrE); fclose(ptrS);                         // Fermeture des fichiers
  return 0;                                           // Traitement OK
}
```

II-11) Négatif

Négatif

La transformation en négatif consiste à remplacer chaque composante de chaque pixel par son complément à 255.

```
int Negatif( char *nomFicE, char *nomFicS)
{
  FILE *ptrE, *ptrS;                          // Pointeurs de fichiers
  int ret;                                    // Retour de la fonction
  unsigned char entête[54];                   // Buffer de recopie de l'entête
  unsigned char b, ob;                        // Nombre d'octets, octet de bourrage
  int dim[2];                                 // Dimensions de l'image (largeur, hauteur)
  int i, j;                                   // Index courants
  struct pixel                                // Contenu d'un pixel
  {
  unsigned char bleu;
    unsigned char vert;
    unsigned char rouge;
  }
  point;
  ret=entête(nomFicE, nomFicS, dim);          // Traitement de l'entête
  if(ret!=0)
    return ret;

  ptrE=fopen(nomFicE,"rb");                   // Ouverture des fichiers
  ptrS=fopen(nomFicS,"wb");
  fread(entête,sizeof(unsigned char),54,ptrE); // Lire et recopier 54 octets
  fwrite(entête,sizeof(unsigned char),54,ptrS);
  b=4-(3*dim[0]%4)&3;                         // Calculer le nb d'octets à 0 de bourrage
  for(i=0;i<dim[1];i++)
  {
    for(j=0;j<dim[0];j++)
    {
      fread(&point,sizeof(point),1,ptrE);     // Lire pixel dans E
      point.bleu=255-point.bleu;              // Compléments à 255
      point.vert=255-point.vert;
      point.rouge=255-point.rouge;
      fwrite(&point,sizeof(point),1,ptrS);    // Ecrire pixel dans S
    }
    for(j=0;j<b;j++)
    {
    fread(&ob,sizeof(ob),1,ptrE);             // Ajouter les octets de bourrage
    fwrite(&ob,sizeof(ob),1,ptrS);
    }
  }
  fclose(ptrE); fclose(ptrS);                 // Fermeture des fichiers
  return 0;                                   // Traitement OK
}
```

II-12) Mosaïque

Mosaïque

La mosaique consiste à remplacer un carré de pixel par la moyenne de tous les pixels.

```
int Mosaique( char *nomFicE, char *nomFicS, int c)
{
  FILE *ptrE, *ptrS;                          // Pointeurs de fichiers
  int ret;                                    // Retour de la fonction
  unsigned char b;                            // Nombre d'octets, octet de bourrage
  unsigned char octet;                        // Octet de copie
  int dim[2];                                 // Dimensions de l'image (largeur, hauteur)
  int i, j;                                   // Index courants
  int ic, jc;                                 // Index du carré traité
  long decal;                                 // Position du pixel traité
  int nbO;                                    // Nombre d'octets par ligne
  int mb,mv,mr;                               // Moyennes bleu, vert, rouge
  struct pixel                                // Contenu d'un pixel
  {
    unsigned char bleu;
    unsigned char vert;
    unsigned char rouge;
  }
  point;
  ret=entête(nomFicE, nomFicS, dim);
  if(ret!=0)                                  // Traitement de l'entête
    return ret;

  ptrE=fopen(nomFicE,"rb");                   // Ouverture des fichiers
  ptrS=fopen(nomFicS,"wb+");
  while (!feof(ptrE))                         // Recopier l'image avant modification
  {
    fread(&octet,sizeof(octet),1,ptrE);
    if(!feof(ptrE))
      fwrite(&octet,sizeof(octet),1,ptrS);
  }
  b=4-(3*dim[0]%4)&3;                         // Nombre d'octets de bourrage
  nbO=3*dim[0] + b;                           // Nombre d'octets par ligne
  ic=0;
  while(ic<dim[1])
  {
    jc=0;
    while(jc<dim[0])
    {
      i=0;
      mb=0; mv=0; mr=0;                       // Calcul de la moyenne des pixels d'un carreau
      while(i<c && (ic+i)<dim[1])
      {
         j=0;
        while(j<c && (jc+j)<dim[0])
        {
          decal=54+ic*nbO+3*jc+i*nbO+3*j;
          fseek(ptrE,decal,SEEK_SET);
```

```
            fread(&point,sizeof(point),1,ptrE);
            mb=mb+point.bleu;
            mv=mv+point.vert;
            mr=mr+point.rouge;
            j=j+1;
          }
          i=i+1;
        }
        mb=mb/(i*j); mv=mv/(i*j); mr=mr/(i*j);      // Recopie de la moyenne dans tous les carreaux du pixel
        i=0;
        while(i<c && (ic+i)<dim[1])
        {
          j=0;
          while(j<c && (jc+j)<dim[0])
          {
            point.bleu=mb;
            point.vert=mv;
            point.rouge=mr;
            decal=54+ic*nbO+3*jc+i*nbO+3*j;
            fseek(ptrS,decal,SEEK_SET);
            fwrite(&point,sizeof(point),1,ptrS);
            j=j+1;
          }
          i=i+1;
        }
        jc=jc+c;
      }
      ic=ic+c;
    }                                               // Fermeture des fichiers
    fclose(ptrE); fclose(ptrS);                     /
    return 0;                                       // Traitement OK
}
```

II-13) Estomper

Estomper

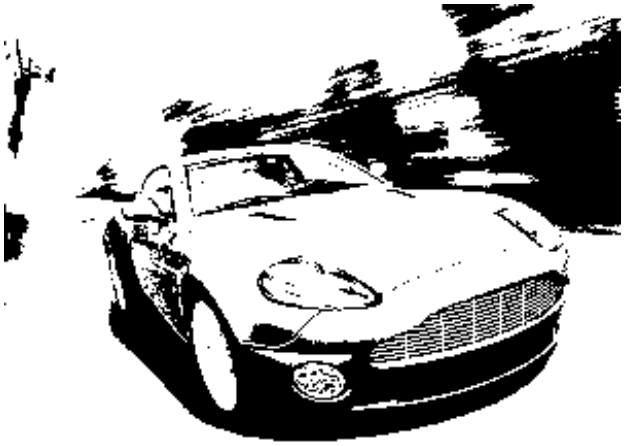
L'estompage des bords consiste à assombrir progressivement les bords de l'image sur la largeur indiquée.

```
int Estomper( char *nomFicE, char *nomFicS, int bord)
{
  typedef unsigned char oct;
  FILE *ptrE, *ptrS;                          // Pointeurs de fichiers
  int ret;                                    // Retour de la fonction
  oct bo;                                     // Nombre d'octets, octet de bourrage
  oct octet;                                  // Octet de copie
  int dim[2];                                 // Dimensions de l'image (largeur, hauteur)
  int ip, jp;                                 // Index du carré traité
  long decal;                                 // Position du pixel traité
  int nbO;                                    // Nombre d'octets par ligne
  float coef;                                 // Coefficient d'estompage
  float inc;                                  // Incrément du coef d'estompage
  struct pixel                                // Contenu d'un pixel
  {
    unsigned char bleu;
    unsigned char vert;
    unsigned char rouge;
  }
  point;
  ret=entête(nomFicE, nomFicS, dim);          // Traitement de l'entête
  if(ret!=0)
    return ret;

  if((2*bord)>dim[0])                         // Vérification de la largeur de bordure
  {
    unlink(nomFicS);                          // Si bord trop important, fermer et effacer fichier de sortie
    return 3;
  }
  ptrE=fopen(nomFicE,"rb");                   // Ouverture des fichiers
  ptrS=fopen(nomFicS,"wb+");
  while (!feof(ptrE))                         // Recopier l'image avant modification
  {
    fread(&octet,sizeof(octet),1,ptrE);
      if(!feof(ptrE))
    fwrite(&octet,sizeof(octet),1,ptrS);
  }
  bo=4-(3*dim[0]%4)&3;                        // Nombre d'octets de bourrage
  nbO=3*dim[0] + bo;                          // Nombre d'octets par ligne
  inc=100./bord;                              // Bordure inférieure
  coef=100.;
  for(ip=0;ip<bord;ip++)
  {
    for(jp=ip;jp<(dim[0]-ip);jp++)
    {
      decal=54+nbO*ip+3*jp;
      fseek(ptrE,decal,SEEK_SET);
      fread(&point,sizeof(point),1,ptrE);
      point.bleu=(unsigned char)(point.bleu * (1.-(coef/100.)));
      point.vert=(unsigned char)(point.vert * (1.-(coef/100.)));
```

```
        point.rouge=(unsigned char)(point.rouge * (1.-
(coef/100.)));
        fseek(ptrS,decal,SEEK_SET);
        fwrite(&point,sizeof(point),1,ptrS);
      }
      coef=coef-inc;
    }                                                    // Bordure supérieure
    coef=100.;
    for(ip=(dim[1]-1);ip>=(dim[1]-bord);ip--)
    {
      for(jp=(dim[1]-ip-1);jp<=(dim[0]-(dim[1]-ip));jp++)
      {
        decal=54+nbO*ip+3*jp;
        fseek(ptrE,decal,SEEK_SET);
        fread(&point,sizeof(point),1,ptrE);
        point.bleu=(unsigned char)(point.bleu * (1.-(coef/100.)));
        point.vert=(unsigned char)(point.vert * (1.-(coef/100.)));
        point.rouge=(unsigned char)(point.rouge * (1.-
(coef/100.)));
        fseek(ptrS,decal,SEEK_SET);
        fwrite(&point,sizeof(point),1,ptrS);
      }
      coef=coef-inc;                                     // Bordure gauche
    }
    coef=100.;
    for(jp=0;jp<bord;jp++)
    {
      for(ip=jp;ip<(dim[1]-jp);ip++)
      {
        decal=54+nbO*ip+3*jp;
        fseek(ptrE,decal,SEEK_SET);
        fread(&point,sizeof(point),1,ptrE);
        point.bleu=(unsigned char)(point.bleu * (1.-(coef/100.)));
        point.vert=(unsigned char)(point.vert * (1.-(coef/100.)));
        point.rouge=(unsigned char)(point.rouge * (1.-
(coef/100.)));
        fseek(ptrS,decal,SEEK_SET);
        fwrite(&point,sizeof(point),1,ptrS);
      }                                                  // Bordure droite
      coef=coef-inc;
    }
    coef=100.;
    for(jp=(dim[0]-1);jp>=(dim[0]-bord);jp--)
    {
      for(ip=(dim[0]-jp-1);ip<=(dim[1]-(dim[0]-jp));ip++)
      {
        decal=54+nbO*ip+3*jp;
        fseek(ptrE,decal,SEEK_SET);
        fread(&point,sizeof(point),1,ptrE);
        point.bleu=(unsigned char)(point.bleu * (1.-(coef/100.)));
        point.vert=(unsigned char)(point.vert * (1.-(coef/100.)));
        point.rouge=(unsigned char)(point.rouge * (1.-
(coef/100.)));
        fseek(ptrS,decal,SEEK_SET);
        fwrite(&point,sizeof(point),1,ptrS);             // Fermeture des fichiers
      }                                                  // Traitement OK
      coef=coef-inc;
    }
    fclose(ptrE); fclose(ptrS);
    return 0;
}
```

II-15) Seuillage



Seuillage

La mise en niveau de gris consiste à effectuer la moyenne des 3 composantes de chaque pixel, puis de remplacer les 3 composantes soit par 0, soit par 255 suivant que la moyenne est inférieure ou supérieure au seuil.

```
int Seuillage( char *nomFicE, char *nomFicS, unsigned char s)
{
  FILE *ptrE, *ptrS;                                  // Pointeurs de fichiers
  int ret;                                            // Retour de la fonction
  unsigned char entête[54];                           // Buffer de recopie de l'entête
  unsigned char b, ob;                                // Nombre d'octets de bourrage, octet de bourrage
  int dim[2];                                         // Dimensions de l'image (largeur, hauteur)
  int i, j;                                           // Index courants
  unsigned char moyenne;                              // Moyenne des 3 couleurs
  struct pixel                                        // Contenu d'un pixel
  {
    unsigned char bleu;
    unsigned char vert;
    unsigned char rouge;
  }
  point;
  ret=entête(nomFicE, nomFicS, dim);                  // Traitement de entête
  if(ret!=0)
    return ret;
  ptrE=fopen(nomFicE,"rb");
  ptrS=fopen(nomFicS,"wb");                           // Ouverture des fichiers
  fread(entête,sizeof(unsigned char),54,ptrE);
  fwrite(entête,sizeof(unsigned char),54,ptrS);       // Lire et recopier 54 octets
  b=4-(3*dim[0]%4)&3;
  for(i=0;i<dim[1];i++)                               // Calculer le nb d'octets à 0 de bourrage
  {
    for(j=0;j<dim[0];j++)
    {
      fread(&point,sizeof(point),1,ptrE);
      moyenne=(point.bleu+point.vert+point.rouge)/3;  // Lire pixel dans E
      if (moyenne<s)                                  // Moyenne des 3 composantes
      {                                               // Comparer la moyenne au seuil
        point.bleu=0;
        point.vert=0;
        point.rouge=0;
      }
      else
      {
        point.bleu=255;
        point.vert=255;
        point.rouge=255;
      }
      fwrite(&point,sizeof(point),1,ptrS);
    }                                                 // Ecrire pixel dans S
    for(j=0;j<b;j++)
    {                                                 // Lire-écrire les octets de bourrage
      fread(&ob,sizeof(ob),1,ptrE);
      fwrite(&ob,sizeof(ob),1,ptrS);
    }
  }
```

```
  fclose(ptrE); fclose(ptrS);
  return 0;
}
```

// Fermeture des fichiers
// Traitement OK

II-16) Posterisation

Posterisation

La moyenne des 3 composantes de chaque pixel est calculée, puis, en fonction de sa valeur, les 3 composantes sont remplacées par une valeur mémorisée dans un tableau ou une valeur aléatoire. Le nombre de couleurs peut être choisi entre 2 et 16.

```
void main()
{
  char nomFicL[80], nomFicE[80];
  int nbCouleur;
  unsigned char tabCouleur[16][3];
  int i, j;
  int ret;
  char rep;
  printf("\nNom de l'image a traiter : ");
  scanf("%s",nomFicL);
  printf("\nNom de l'image transformee : ");
  scanf("%s",nomFicE);
  do
  {
    printf("\nNombre de couleurs (2 a 16) : ");
    scanf("%d",&nbCouleur);
  }
  while(nbCouleur<2 && nbCouleur>16);
  do
  {
    printf("\nCouleurs choisies (C) ou aleatoire (A) : ");
    fflush(stdin);
    scanf("%c",&rep);
  }
  while(rep!='C' && rep!='c' && rep!='A' && rep!='a');
  if (rep=='C' || rep=='c')
  {
    for(i=0;i<nbCouleur;i++)
    {
      printf("\nCouleur %d (b,v,r) :",i+1);
      scanf("%d,%d,%d",&tabCouleur[i][0],&tabCouleur[i][1],&tabCouleur[i][2]);
    }
  }
  else
  {
    randomize();
    for(i=0;i<nbCouleur;i++)
      for(j=0;j<3;j++)
        tabCouleur[i][j]=random(256);
  }
  ret=Posterisation(nomFicL, nomFicE, nbCouleur, tabCouleur);
  switch(ret)
  {
    case 0 : printf("\nTraitement termine"); break;
    case 1 : printf("\nFichier inexistant"); break;
    case 2 : printf("\nL'image n'est pas au format BMP 24 bits");
  }
  getch();
}
```

```
int Posterisation( char *nomFicE, char *nomFicS, int nbCoul,
```

```
unsigned char tabCoul[16][3])
{
  FILE *ptrE, *ptrS;                                  // Pointeurs de fichiers
  int ret;                                            // Retour de la fonction
  unsigned char entête[54];                           // Buffer de recopie de l'entête
  unsigned char b, ob;                                // Nombre d'octets de bourrage, octet de bourrage
  int dim[2];                                         // Dimensions de l'image (largeur, hauteur)
  int i, j, k;                                        // Index courants
  unsigned char moyenne;                              // Moyenne des 3 couleurs
  int niveau;                                         // Niveau de comparaison couleur
  struct pixel                                        // Contenu d'un pixel
  {
    unsigned char bleu;
    unsigned char vert;
    unsigned char rouge;
  }
  point;
  ret=entête(nomFicE, nomFicS, dim);                  // Traitement de l'entête
  if(ret!=0)
    return ret;

  ptrE=fopen(nomFicE,"rb");                           // Ouverture des fichiers
  ptrS=fopen(nomFicS,"wb");
  fread(entête,sizeof(unsigned char),54,ptrE);        // Lire et recopier 54 octets
  fwrite(entête,sizeof(unsigned char),54,ptrS);
  b=4-(3*dim[0]%4)&3;                                 // Calculer le nb d'octets à 0 de bourrage
  for(i=0;i<dim[1];i++)
  {
  for(j=0;j<dim[0];j++)
    {
      fread(&point,sizeof(point),1,ptrE);             // Lire pixel dans E
      moyenne=(point.bleu+point.vert+point.rouge)/3;  // Moyenne des 3 composantes
      niveau=0;                                       // Niveau de comparaison
      for(k=0;k<16;k++)
      {
        niveau=niveau+256/nbCoul;
        if(moyenne<=niveau)
        {
          point.bleu=tabCoul[k][0];
          point.vert=tabCoul[k][1];
          point.rouge=tabCoul[k][2];
          break;
        }
      }
      fwrite(&point,sizeof(point),1,ptrS);            // Ecrire pixel dans S
    }
    for(k=0;k<b;k++)
    {
      fread(&ob,sizeof(ob),1,ptrE);                   // Lire-écrire les octets de bourrage
      fwrite(&ob,sizeof(ob),1,ptrS);
    }
  }
  fclose(ptrE); fclose(ptrS);                         // Fermeture des fichiers
  return 0;                                           // Traitement OK
}
```

III) Création de la DLL.

Toutes les manipulations d'images sus-citées et écrites en Borland C doivent êtres préparées pour etre utilisés dans Visual Basic. Chaques fonctions (manipulations) ont 2 paramètres communs, un nom de fichier d'entrée, un nom de fichier de sortie, et certaines fonctions utilisent des paramètres qui leurs sont propres. Ci-dessous, la liste des paramètres des fonctions :

Fonctions	Para 3	Para 4
Niveau de Gris	X	x
Négatif	X	x
Symétrie Verticale	X	x
Symétrie Horizontale	X	x
Rotation Horaire	X	x
Rotation Trigonométrique	X	x
Dimension	Facteur de Dimension	x
Contraste	Facteur de Contraste	x
Lumière	Facteur de Luminosité	x
Mosaïque	Facteur de Mosaïque	x
Estomper	Facteur d'Estompage	x
Seuiller	Facteur de Seuillage	x
Posteriser	Nombre de couleur	Couleurs choisies

NB : Les parametres 1 et 2 sont communs a tous les fichiers, il s'agit du nom de Fichier d'Entrée et du nom de Fichier de Sortie.

Les syntaxes (en Borland C) des fonctions sont les suivantes :

```
Int NiveauGris (char *nomFicE, char *nomFicS)
Int Negatif (char *nomFicE, char *nomFicS)
Int SymetrieHorizontale (char *nomFicE, char *nomFicS)
Int SymetrieVerticale (char *nomFicE, char *nomFicS)
Int RotationHoraire (char *nomFicE, char *nomFicS)
Int RotationTrigo (char *nomFicE, char *nomFicS)
Int Dimension (char *nomFicE, char *nomFicS, int d)
Int Contraste (char *nomFicE, char *nomFicS, int coef)
Int Mosaique (char *nomFicE, char *nomFicS, int c)
Int Estomper (char *nomFicE, char *nomFicS, int c)
Int Seuillage (char *nomFicE, char *nomFicS, unsigned char s)
Int Posterisation (char *nomFicE, char *nomFicS, int nbCoul, unsigned char tabCoul[16][3])
```

Chaque fonction renvoit 1 valeur parmi 3, au minimum et peuvent renvoyer d'autres valeurs selon la fonction. Ci-dessous, les valeurs renvoyées possibles :

Fonctions	Val 3
Niveau de Gris	X
Négatif	X
Symétrie Verticale	X
Symétrie Horizontale	X
Rotation Horaire	X
Rotation Trigonométrique	X
Dimension	Image trop grande
Contraste	X
Lumière	X
Mosaïque	X
Estomper	Bordure trop grande
Seuiller	X
Posteriser	X

NB : les valeurs 0, 1 et 2 sont communes a toutes les fonctions, il s'agit de :

0 : Traitement correct
1 : Fichier inexistant
2 : Image pas au bon format

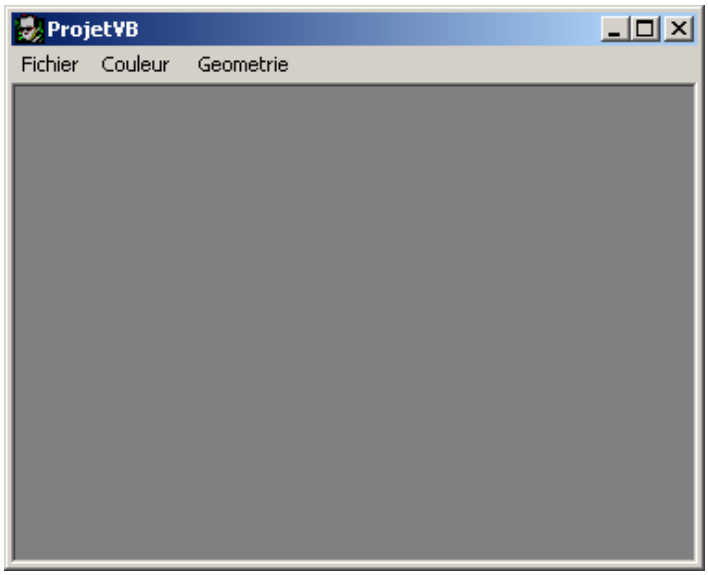
A l'aide du Borland C, on va crée une DLL qui va réunir toutes les fonctions dans le même fichier. Les syntaxes des fonctions en Borland C sont les suivantes :

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
extern 'C' int __export __stdcall NiveauGris (char *nomFicE, char *nomFicS)
extern 'C' int __export __stdcall Negatif (char *nomFicE, char *nomFicS)
extern 'C' int __export __stdcall SymetrieHorizontale (char *nomFicE, char *nomFicS)
extern 'C' int __export __stdcall SymetrieVerticale (char *nomFicE, char *nomFicS)
extern 'C' int __export __stdcall RotationHoraire (char *nomFicE, char *nomFicS)
extern 'C' int __export __stdcall RotationTrigo (char *nomFicE, char *nomFicS)
extern 'C' int __export __stdcall Dimension (char *nomFicE, char *nomFicS, int d)
extern 'C' int __export __stdcall Contraste (char *nomFicE, char *nomFicS, int coef)
extern 'C' int __export __stdcall Mosaique (char *nomFicE, char *nomFicS, int c)
extern 'C' int __export __stdcall Estomper (char *nomFicE, char *nomFicS, int c)
extern 'C' int __export __stdcall Seuillage (char *nomFicE, char *nomFicS, unsigned char s)
extern 'C' int __export __stdcall Posterisation (char *nomFicE, char *nomFicS, int nbCoul, unsigned char tabCoul[16][3])
```

Chaque syntaxe **`extern 'C'`**… etant suivi du code correspondant a la transformation
(Voir II, Manipulations), Apres compilation, on a donc notre DLL contenant toutes les manipulations d'image écrites en C et destinées a etre utilisée sous VB.

IV) Programme en VB

IV-0) Présentation

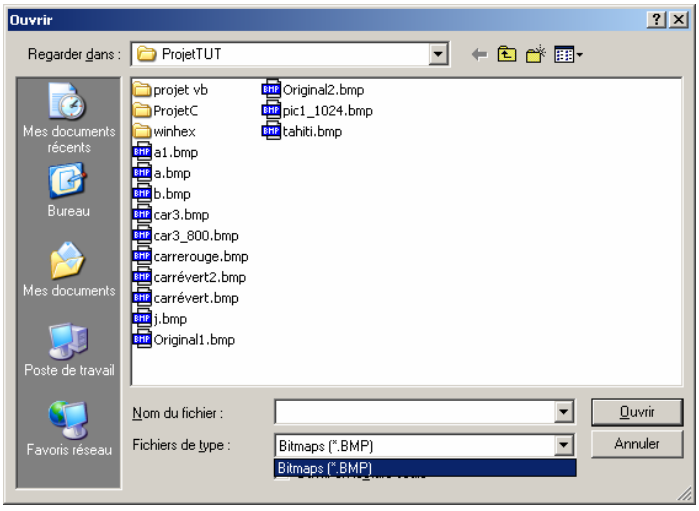


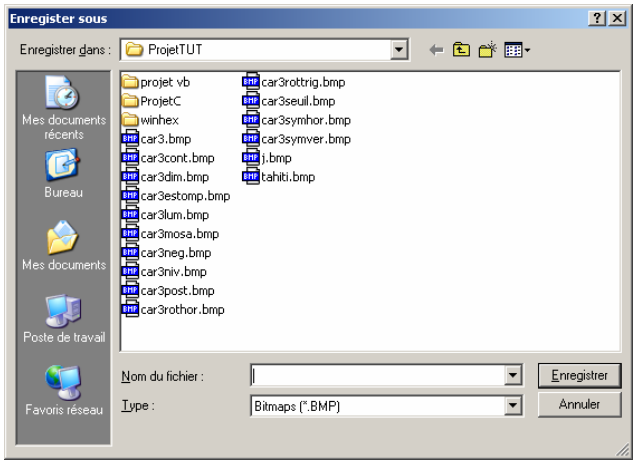
Ceci est la fenêtre principale. On retrouve les 3 menus principaux, avec les actions possibles. Les menus **Couleurs** et **Géométrie** permettent les manipulations précédentes et le menu Fichier permet d'ouvrir une image, de l'enregistrer et de quitter. Cette feuille principale est une MDI qui permet de contenir d'autres feuilles de travail. Cette autre feuille dans notre cas est l'image a manipuler. La feuille principale porte le nom de **mdiMain** et la feuille image, le nom de **frmImage**

L'ouverture des fichiers s'effectue a l'aide d'une boite de dialogue commune (**cmndialFile**) et qui permet d'ouvrir un lecteur, un répertoire et un fichier precis. Un filtre a été mis en place, ne permettant d'ouvrir que les fichiers BMP.

```
Private Sub mnuOuvrir_Click()
cmndialFile.DialogTitle = "Ouvrir"
cmndialFile.ShowOpen
nomE = cmndialFile.FileName
frmImage.picImage.Picture = LoadPicture(nomE)
frmImage.Width = frmImage.picImage.Width
frmImage.Height = frmImage.picImage.Height
End Sub
```

Lors de la sélection de **Fichier\Ouvrir**, on permet l'ouverture d'un fichier et le nom de ce fichier est stocké dans **nomE**, puis on affiche cette image dans **frmImage**

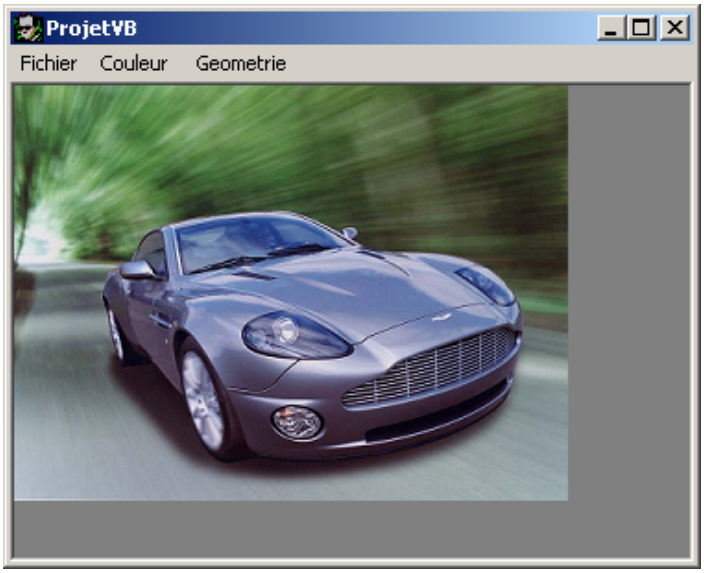




L'enregistrement des fichiers s'effectue a l'aide d'une boite de dialogue commune (**cmndialFile**) et qui permet d'ouvrir un lecteur, un repertoire.

```
Private Sub mnuEnreg_Click()
   cmndialFile.DialogTitle = "Enregister sous"
   cmndialFile.ShowSave
   nomS = cmndialFile.FileName
   SavePicture frmImage.picImage.Picture, nomS
End Sub
```

Lors de la sélection de **Fichier\Enregistrer**, on permet l'enregistrement de l'image en cours dans un fichier, donc l'extension est bloquée a BMP



On peut alors accéder aux manipulations des menus **Couleur** et **Géométrie**

Si on choisit **Niveau de gris**

```
Private Sub mnuNiveauGris_Click()
   nomS = "temp._mp"
   retour = NiveauGris(nomE, nomS)
   frmImage.picImage.Picture = LoadPicture(nomS)
   frmImage.Width = frmImage.picImage.Width
   frmImage.Height = frmImage.picImage.Height
End Sub
```

On definit le fichier de sortie comme étant le fichier temporaire de manipulation, on appelle la fonction **NiveauGris** avec les paramètres qui conviennent puis a la fin du traitement, on réaffiche dans **frmImage** l'image obtenue. L'image initiale n'est pas déformée.

On définit le fichier de sortie comme étant le fichier temporaire de manipulation, on appelle la fonction **Negatif** avec les paramètres qui conviennent puis a la fin du traitement, on réaffiche dans **frmImage** l'image obtenue. L'image initiale n'est pas déformée.

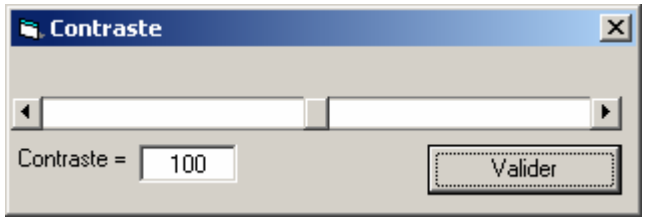
Si on choisit **Negatif**

```
Private Sub mnuNegatif_Click()
   nomS = "temp._mp"
   retour = Negatif(nomE, nomS)
   frmImage.picImage.Picture = LoadPicture(nomS)
   frmImage.Width = frmImage.picImage.Width
   frmImage.Height = frmImage.picImage.Height
End Sub
```

Si on choisit **Contraste**

On fait apparaître la barre de **Contraste**, qui renvoie la valeur désirée. Elle est limitée entre 0 et 200, si la valeur est >100, on adoucit, si <100, on durcit. On definit le fichier de sortie comme etant le fichier temporaire de manipulation, on appele la fonction **Contraste** avec les parametres qui conviennent puis a la fin du traitement, on reaffiche dans **frmImage** l'image obtenue. L'image initiale n'est pas déformée.

```
Private Sub mnuCont_Click()
   frmCont.Show 1
   nomS = "temp._mp"
   retour = Contraste(nomE, nomS, c)
   frmImage.picImage.Picture = LoadPicture(nomS)
   frmImage.Width = frmImage.picImage.Width
   frmImage.Height = frmImage.picImage.Height
End Sub
```




```
Option Explicit

Private Sub cmdValider_Click()
   frmCont.Hide
End Sub

Private Sub hsbCont_Change()
  c = hsbCont.Value
  txtCont.Text = c
End Sub
```


```
Option Explicit

Private Sub cmdValider_Click()
  frmLum.Hide
End Sub

Private Sub hsbLum_Change()
  c = hsbLum.Value
  txtLum.Text = c
End Sub
```

Si on choisit **Lumière**

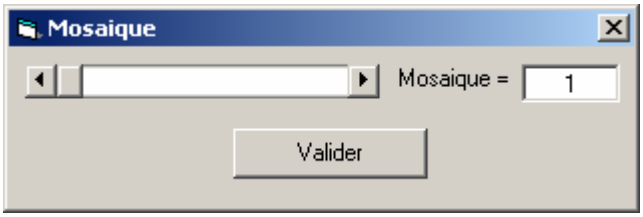
On fait apparaître la barre de **Lumière**, qui renvoie la valeur désirée. Elle est limitée entre -255 et +255, si la valeur est comprise entre -255 et -1, on assombrit, si comprise entre 1 et 255, on éclaircit. On définit le fichier de sortie comme étant le fichier temporaire de manipulation, on appelle la fonction **Lumière** avec les paramètres qui conviennent puis a la fin du traitement, on réaffiche dans **frmImage** l'image obtenue. L'image initiale n'est pas déformée.

```
Private Sub mnuLum_Click()
   frmLum.Show 1
   nomS = "temp._mp"
   retour = Lumiere(nomE, nomS, c)
   frmImage.picImage.Picture = LoadPicture(nomS)
   frmImage.Width = frmImage.picImage.Width
   frmImage.Height = frmImage.picImage.Height
End Sub
```

Si on choisit **Mosaïque**

On fait apparaître la barre de **Mosaïque**, qui renvoie la valeur désirée et elle représente le nombre de pixel que fait le carré. Elle est limitée entre 1 et 100. On définit le fichier de sortie comme étant le fichier temporaire de manipulation, on appelle la fonction **Mosaïque** avec les paramètres qui conviennent puis à la fin du traitement, on réaffiche dans **frmImage** l'image obtenue. L'image initiale n'est pas déformée.

```
Private Sub mnuMosa_Click()
   frmMosa.Show 1
   nomS = "temp._mp"
   retour = Mosaique(nomE, nomS, c)
   frmImage.picImage.Picture = LoadPicture(nomS)
   frmImage.Width = frmImage.picImage.Width
   frmImage.Height = frmImage.picImage.Height
End Sub
```




```
Option Explicit

Private Sub cmdValider_Click()
   frmMosa.Hide
End Sub

Private Sub hsbMosa_Change()
  c = hsbMosa.Value
  txtMosa.Text = c
End Sub
```


```
Option Explicit

Private Sub cmdValider_Click()
   frmEst.Hide
End Sub

Private Sub hsbEst_Change()
  l = hsbEst.Value
  txtEst.Text = l
End Sub
```

Si on choisit **Estompage**

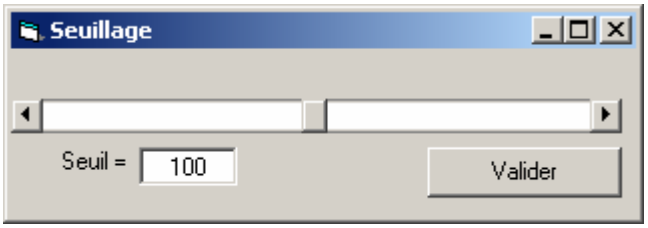
On fait apparaître la barre d'**Estompage**, qui renvoie la valeur. Elle est limitée entre 1 et 100. On définit le fichier de sortie comme étant le fichier temporaire de manipulation, on appelle la fonction **Estompage** avec les paramètres qui conviennent puis à la fin du traitement, on réaffiche dans **frmImage** l'image obtenue. L'image initiale n'est pas déformée.

```
Private Sub mnuEstomp_Click()
   frmEst.Show 1
   nomS = "temp._mp"
   retour = Estomper(nomE, nomS, l)
   frmImage.picImage.Picture = LoadPicture(nomS)
   frmImage.Width = frmImage.picImage.Width
   frmImage.Height = frmImage.picImage.Height
End Sub
```

Si on choisit Seuillage

On fait apparaître la barre de **Seuillage**, qui renvoie la valeur. Elle est limitée entre 0 et 200. On définit le fichier de sortie comme étant le fichier temporaire de manipulation, on appelle la fonction **Seuillage** avec les paramètres qui conviennent puis à la fin du traitement, on réaffiche dans **frmImage** l'image obtenue. L'image initiale n'est pas déformée.

```
Private Sub mnuSeuil_Click()
   frmSeuil.Show 1
   nomS = "temp._mp"
   retour = Seuillage(nomE, nomS, s)
   frmImage.picImage.Picture = LoadPicture(nomS)
   frmImage.Width = frmImage.picImage.Width
   frmImage.Height = frmImage.picImage.Height
End Sub
```

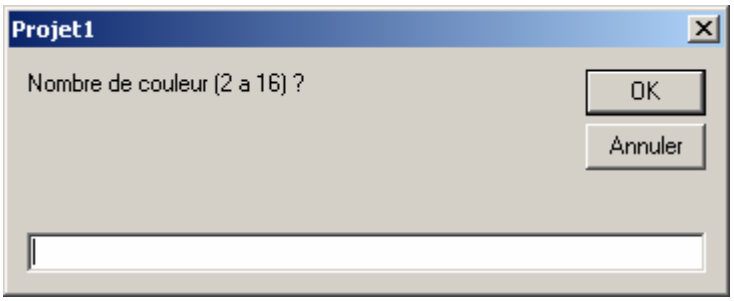



```
Option Explicit

Private Sub cmdValider_Click()
   frmSeuil.Hide
End Sub

Private Sub Form_Paint()
  txtSeuil.Text = hsbSeuil.Value
End Sub

Private Sub hsbSeuil_Change()
  s = hsbSeuil.Value
  txtSeuil.Text = s
End Sub
```



Si on choisit **Posterisation**

On fait apparaître une invite qui demande le nbre de couleur pour effectuer la comparaison. Suite a ce nbre, on fait apparaître une fenêtre contenant ce nbre sous forme de carré et chaque clic sur un carré permet de sélectionner une couleur. On appelle la fonction **Posterisation** avec les paramètres qui conviennent puis à la fin du traitement, on réaffiche dans **frmImage** l'image obtenue. L'image initiale n'est pas déformée.

```
Private Sub cmdValider_Click()
   frmPost.Hide
End Sub

Private Sub Form_Paint()
   For i = 0 To n - 1
      Picture1(i).Visible = True
   Next i
End Sub

Private Sub Picture1_Click(Index As Integer)
   cmndialColor.ShowColor
   couleur64(Index) = cmndialColor.Color
   Picture1(Index).BackColor = couleur64(Index)
End Sub
```

```
Private Sub mnuPoster_Click()
  n = Val(InputBox("Nombre de couleur (2 a 16) ?"))
  frmPost.Show 1
  nomS = "temp._mp"
  For i = 0 To 15
    c = couleur64(i)
    t(3 * i + 2) = c Mod 256
    c = c / 256
    t(3 * i + 1) = c Mod 256
    c = c / 256
    t(3 * i) = c Mod 256
  Next i
  retour = Posterisation(nomE, nomS, n, t(0))
  frmImage.picImage.Picture = LoadPicture(nomS)
  frmImage.Width = frmImage.picImage.Width
  frmImage.Height = frmImage.picImage.Height
End Sub
```

Si on choisit **Symetrie Verticale**

```
Private Sub mnuSymVer_Click()
nomS = "temp._mp"
retour = SymetrieVerticale(nomE, nomS)
frmImage.picImage.Picture = LoadPicture(nomS)
frmImage.Width = frmImage.picImage.Width
frmImage.Height = frmImage.picImage.Height
End Sub
```

On définit le fichier de sortie comme étant le fichier temporaire de manipulation, on appelle la fonction **SymetrieVerticale** avec les paramètres qui conviennent puis à la fin du traitement, on réaffiche dans **frmImage** l'image obtenue. L'image initiale n'est pas déformée.

On définit le fichier de sortie comme étant le fichier temporaire de manipulation, on appelle la fonction **SymetrieHorizontale** avec les paramètres qui conviennent puis à la fin du traitement, on réaffiche dans **frmImage** l'image obtenue. L'image initiale n'est pas déformée.

Si on choisit **Symetrie Horizontale**

```
Private Sub mnuSymHor_Click()
   nomS = "temp._mp"
   retour = SymetrieHorizontale(nomE, nomS)
   frmImage.picImage.Picture = LoadPicture(nomS)
   frmImage.Width = frmImage.picImage.Width
   frmImage.Height = frmImage.picImage.Height
End Sub
```

Si on choisit **Rotation Horaire**

```
Private Sub mnuRotHor_Click()
   nomS = "temp._mp"
   retour = RotationHoraire(nomE, nomS)
   frmImage.picImage.Picture = LoadPicture(nomS)
   frmImage.Width = frmImage.picImage.Width
   frmImage.Height = frmImage.picImage.Height
End Sub
```

On définit le fichier de sortie comme étant le fichier temporaire de manipulation, on appelle la fonction **RotationHoraire** avec les paramètres qui conviennent puis à la fin du traitement, on réaffiche dans **frmImage** l'image obtenue. L'image initiale n'est pas déformée.

On définit le fichier de sortie comme étant le fichier temporaire de manipulation, on appelle la fonction **RotationTrigo** avec les paramètres qui conviennent puis à la fin du traitement, on réaffiche dans **frmImage** l'image obtenue. L'image initiale n'est pas déformée.

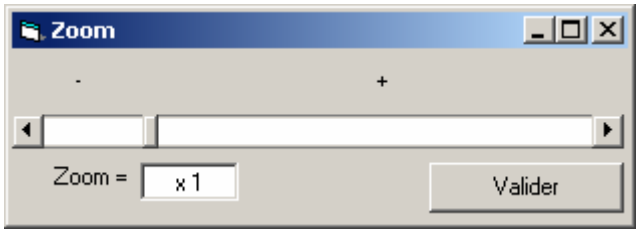
Si on choisit **Rotation Trigo**

```
Private Sub mnuRotTrig_Click()
   nomS = "temp._mp"
   retour = RotationTrigo(nomE, nomS)
   frmImage.picImage.Picture = LoadPicture(nomS)
   frmImage.Width = frmImage.picImage.Width
   frmImage.Height = frmImage.picImage.Height
End Sub
```

Si on choisit **Dimension**

On fait apparaître la barre de **Zoom**, qui renvoie la valeur. Elle est limitée entre 0.1x et 5x. On définit le fichier de sortie comme étant le fichier temporaire de manipulation, on appelle la fonction **Zoom** avec les paramètres qui conviennent puis à la fin du traitement, on réaffiche dans **frmImage** l'image obtenue. L'image initiale n'est pas déformée.

```
Private Sub mnuDimension_Click()
   frmZoom.Show 1
   nomS = "temp._mp"
   retour = Dimension(nomE, nomS, d)
   frmImage.picImage.Picture = LoadPicture(nomS)
   frmImage.Width = frmImage.picImage.Width
   frmImage.Height = frmImage.picImage.Height
End Sub
```




```
Private Sub cmdValider_Click()
frmZoom.Hide
End Sub

Private Sub hsbDim_Change()
d = hsbDim.Value
txtZoom.Text = "x" + Str(d / 100)
End Sub
```

Pour permettre l'utilisation de cette DLL, il faut déclarer chaque fonction disponible dans la DLL dans la section globale du code VB par la syntaxe :

```
Private Declare Function NiveauGris Lib "projetvb.dll" (ByVal nomE As String, ByVal nomS As String) As Integer
Private Declare Function Negatif Lib "projetvb.dll" (ByVal nomE As String, ByVal nomS As String) As Integer
Private Declare Function SymetrieVerticale Lib "projetvb.dll" (ByVal nomE As String, ByVal nomS As String) As Integer
Private Declare Function SymetrieHorizontale Lib "projetvb.dll" (ByVal nomE As String, ByVal nomS As String) As Integer
Private Declare Function RotationHoraire Lib "projetvb.dll" (ByVal nomE As String, ByVal nomS As String) As Integer
Private Declare Function RotationTrigo Lib "projetvb.dll" (ByVal nomE As String, ByVal nomS As String) As Integer
Private Declare Function Dimension Lib "projetvb.dll" (ByVal nomE As String, ByVal nomS As String, ByVal d As Long) As Integer
Private Declare Function Contraste Lib "projetvb.dll" (ByVal nomE As String, ByVal nomS As String, ByVal c As Long) As Integer
Private Declare Function Lumiere Lib "projetvb.dll" (ByVal nomE As String, ByVal nomS As String, ByVal c As Long) As Integer
Private Declare Function Mosaique Lib "projetvb.dll" (ByVal nomE As String, ByVal nomS As String, ByVal c As Long) As Integer
Private Declare Function Estomper Lib "projetvb.dll" (ByVal nomE As String, ByVal nomS As String, ByVal l As Long) As Integer
Private Declare Function Seuillage Lib "projetvb.dll" (ByVal nomE As String, ByVal nomS As String, ByVal s As Byte) As Integer
Private Declare Function Posterisation Lib "projetvb.dll" (ByVal nomE As String, ByVal nomS As String, ByVal n As Long, ByRef t
As Byte) As Integer
```

De même, on déclare toutes les variables utilisées dans un module

```
Option Explicit
   Public d As Single
   Public c As Single
   Public l As Single
   Public s As Single
   Public n As Long
   Public couleur64(0 To 15)
   Public i As Single
   Public t(0 To 47) As Byte
```

Conclusion

Le traitement des erreurs n'a pas été fait, autant les retours d'erreurs des manipulations (Image inexistante, Image au mauvais format, Image trop grande, bordure trop grande), ainsi que les erreurs de Visual Basic, ex : Si on effectue l'appel d'une fonction puis on annule, cela génère un code d'erreur.

Une coquille dans la programmation est contraignante, lorsqu'on effectue une manipulation, il y a 2 fichiers, un d'entrée, un de sortie, et si on effectue une manipulation, la prochaine s'effectuera sur le fichier original et non sur le fichier modifié, ce qui impose de sauvegarder systématiquement la progression de la transformation. De même, il aurait été possible de créer un index des manipulations, avec les fichiers temporaires s'y rattachant et de valider ou d'annuler les modifications.

Toutes les fenêtres propres aux manipulations sont modifiables en taille, ce qui est une erreur, et les barres de valeurs n'ont pas les bornes fixés correctement, ce qui peut donner des résultats disproportionnés ou aberrants, ex : une image de 100 pixels de large, et une mise en mosaïque dont le carré a pour coté 100 pixels, il en résulte une image d'une couleur unique.