Sommaire

Benoît Piranda
Équipe SISAR
Université de Marne La Vallée

Programmation de l'acquisition vidéo DV

- Le format DV / Mini DV
  - Compression de type JPEG
  - Émission de 25 images (frames) indépendantes par seconde
  - Synchronisation image et son
- Image
  - Résolution :
    - PAL 720x576, 25 images / seconde
    - NTSC 720x480, 30 images /seconde
- Son qualité CD
  - Fréquence : 44100 échantillons / seconde
  - Précision : 2 octets / échantillon
  - 1 ou 2 voies stéréos

Programmation de la vidéo DV

- Principe de l'émission DV
  - Envoi de 300 paquets de 480 octets chacun
    - Entête
    - Données son et image qui doivent être placées en mémoire à la volée
  - 144000 octets pour une trame
    - 720x576x3 = 1244160 octets pour l'image
    - 2x2x44100/25 = 7056 octets pour le son
    - Soit 1 251 216 octets compressés
      - ✓ Gain : 88,5 %

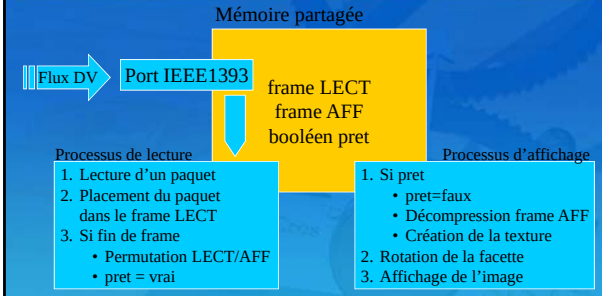
Nécessité d'un programme multithread

- Le programme
  - Lecture du frame
    - Sur 40ms
  - Décompression du frame en image et son
    - Environ 20ms (décompression logiciel)
  - Conversion de l'image en texture
  - Affichage de la texture sur une facette
- La lecture du flux DV doit être permanente
  - Risque de perte de trame
  - Besoin de deux frames en mémoire
    - Frame en cours de lecture
    - Frame en cours de décompression

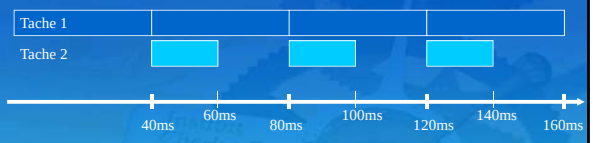
Programmation multi-processus

- Deux processus
  - Processus de lecture du port DV
  - Processus d'affichage
- Synchronisation
  - BufferDV plein : frame prêt à être décompressé
- Problème :
  - Ne pas perdre de trame pendant la décompression
  - Nécessité de deux Buffers
    - Buffer de chargement
    - Buffer de décompression

Structure du programme multithread



Analyse temporelle



Programmation de la vidéo DV

- Accès au port DV FireWire IEEE1394
  - Bibliothèque libraw1394/raw1394.h
- Décompression image et son du flux DV
  - Bibliothèques <libdv/dv_types.h> et <libdv/dv.h>
- Lecture entête avi

Décompression DV

- Initialisation du décodeur

```
dv_decoder_t *td;
 td=dv_decoder_new(TRUE,TRUE,FALSE);
td->quality=DV_QUALITY_BEST;
```

- Décompression du flux DV

```
static int pitches[3]= {720*3,0,0};
uint8_t *ptrImage;

dv_parse_header(td, bufferDV);
image = new uint8_t[720*576*3];
dv_decode_full_frame(td,bufferDV,e_dv_color_rgb,&image,pitches);
int16_t *audioBuffers[4]
for (i=0; i < 4; i++)
{ audioBuffers[i] = new int16_t[DV_AUDIO_MAX_SAMPLES];
}
dv_decode_full_audio(td, bufferDV, audioBuffers);
```

```
dv_decoder_free(td);
```

Lecture du port FireWire

- Fonctions de lecture du port IEEE1394

```
#define CHANEL_ALL     63
raw1394handle_t handle;
int g_card=1;

/* demande d'un identifiant*/
if (!(handle = raw1394_new_handle())) {exit(-1);}
```

```
/* ouverture d'un port sur la carte g_card */
if (raw1394_set_port(handle, g_card) < 0) { exit( -1); }

/* association d'une fonction de réception des paquets */
 raw1394_set_iso_handler(handle, channel, raw_iso_handler);

/* association d'une fonction de réinitialisation du
périphérique */
raw1394_set_bus_reset_handler(handle, my_reset_handler)

/* lancement de la lecture sur le port
 if (raw1394_start_iso_rcv(handle, channel) < 0) { exit( -1); }
```

Lecture du port FireWire

- Fonction appelée à chaque réception d'un paquet, encodage des informations d'une frame.

```
int raw_iso_handler(raw1394handle_t handle, int channel, size_t length, quadlet_t *data)
{ if (length>16)
  { unsigned char *p = (unsigned char*) & data[3],*buffer;
    int section_type = p[0] >> 5,
    dif_sequence = p[1] >> 4,
    dif_block = p[2];

    if (section_type == 0 && dif_sequence == 0)
    { // fin de frame
     …}

    switch (section_type)
    { case 0: memcpy(buffer+dif_sequence*150*80, p, 480); break;
      case 1: memcpy(buffer+dif_sequence*150*80+(1+dif_block)*80,p,480); break;
      case 2: memcpy(buffer+dif_sequence*150*80+(3+dif_block)*80,p,480); break;
      case 3: memcpy(buffer+dif_sequence*150*80+(6+dif_block*16)*80,p,480); break;
      case 4: memcpy(buffer+dif_sequence*150*80+(7+(dif_block/15)+dif_block)*80,p,480); break;
    }
  }
  return 0;
}
```

Lecture du port FireWire

```
/* fermeture du port,
   indispensable pour une réouverture ultérieure */
close_1394_driver(CHANEL_ALL, handle);
```

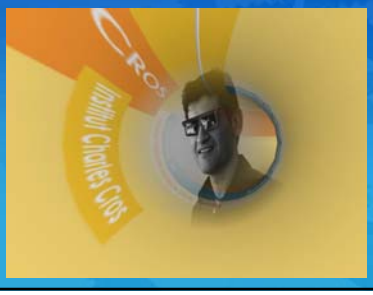
Utilisation de la bibliothèque libdv/dv.h

- Initialisation du décodeur
- Décompression du flux vidéo
- Décompression du flux son

```
int pitches[3]= { 720 * 3, 0, 0 };
dv_decoder_t *td;

td=dv_decoder_new(TRUE,TRUE,FALSE);
td->quality=DV_QUALITY_BEST;

dv_parse_header(td,buffer);
dv_decode_full_frame(td,buffer,e_dv_color_rgb,&image,pitches);

if (dv_decode_full_audio(td, buffer,(int16_t**) audioBuffers))
{ n = td->audio->samples_this_frame;
  …
}
dv_decoder_free(td);
```

Exemples d'applications

- Affichage des flux sur un maillage de facettes
  - Image : texture
  - Son : déformation de la surface



Sommaire

- ✓ Programmation temps réel
  - Exemple de programmation de la vidéo:
- ✓ Acquisition vidéo DV sous linux
- ✓ Lecture de fichier vidéo AVI-DV

Benoît Piranda
Équipe SISAR
Université de Marne La Vallée

Lecture d'une vidéo au format AVI-DV

- Syntaxe du format AVI
  - Spécification au format RIFF
    - Codage binaire little-Endian
    - Code utilisant des quadruplets d'octets
      - ✓ Identifiants : 4 caractères
      - ✓ Tailles, valeurs numériques : entier long
- Entête : 3 quadruplets puis les données
  - `'RIFF'` *`taille type données`*
    - *taille* (entier long) : taille des données dans le fichier,
    - *type* (chaîne de 4 caractères) : le format des données, dans notre cas cette chaîne contiendra 'AVI_',
    - *données* contient des structures de type :
      - ✓ LIST et des paramètres qui suivent
      - ✓ JUNK *taille*

Lecture d'une vidéo au format AVI-DV

- Structures de données
  - LIST *taille type données*
    - ✓ Type `hdrl` : paramètres de l'entête avi
    - ✓ Type `movi` : liste de blocs de données
  - JUNK *taille*
    - ✓ Permet de décaler des données dans les fichiers AVI
    - ✓ Les données ne doivent pas être interprétées

Lecture d'une vidéo au format AVI-DV

- Données de type `hdrl`

```
typedef struct{
  signed long dwMicroSecPerFrame;   // frame display rate(or 0L)
  signed long dwMaxBytesPerSec;     // max. transfer rate
  signed long dwPaddingGranularity; // pad to multiples of this
  // size; normally 2K.
  signed long dwFlags;              // the ever-present flags
  signed long dwTotalFrames;        // # frames in file
  signed long dwInitialFrames;
  signed long dwStreams;
  signed long dwSuggestedBufferSize;
  signed long dwWidth;
  signed long dwHeight;
  signed long dwReserved[4];
} MainAVIHeader;
```

Lecture d'une vidéo au format AVI-DV

- **Données de type movi**
  - **Type :**
    - 2 parties
      - ✓ Numéro de flux (2 octets)
      - ✓ Type de donnée (2octets)
    - Type 00dc *taille* : frame DV compressé pour le flux 00
    - Type 00db *taille* : frame non-compressé pour le flux 00
    - Type 01wb *taille* : données audio pour le flux 01
    - Type 00pc *taille* : changement de palette de couleur

movi	XXXX	00dc	144000	Frame DV
		00dc	144000	Frame DV
		01wb	XXXX	Flux audio
		00dc	144000	Frame DV

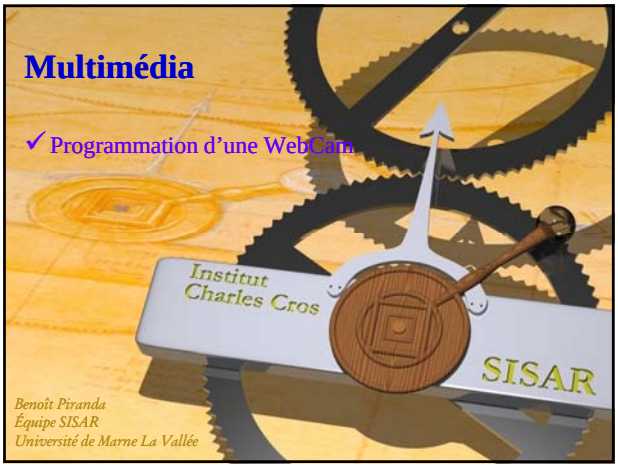
Lecture d'une vidéo au format AVI-DV

- **Format général des données AVI-DV**

```
RIFF ('AVI '
      LIST ('hdrl'
            'avih'(<Main AVI Header>)
            LIST ('strl'
                  'strh'(<Stream header>)
                  'strf'(<Stream format>)
                  [ 'strd'(<Additional header data>) ]
                  [ 'strn'(<Stream name>) ]
                  ...
                 )
            ...
           )
      LIST ('movi'
            {SubChunk | LIST ('rec '
                              SubChunk1
                              SubChunk2
                              ...
                             )
              ...
            }
            ...
           )
      ['idx1' (<AVI Index>) ]
     )
```

Exemple de programmation

```
class AVI_LIST
{ protected :  AVI_LIST *suivant;
  public :
  AVI_LIST(AVI_LIST *s)
  { suivant=s; };
  virtual ~AVI_LIST()
  { delete suivant; };
  virtual int getNbreFrames();
  virtual int getDataIndex();
};
class AVI_LIST_STRL : public AVI_LIST
{ AVIStreamHeader stream;
  public :
  AVI_LIST_STRL(ifstream &,AVI_LIST *s);
};
class AVI_LIST_HDRL : public AVI_LIST
{ MainAVIHeader main;
  AVI_LIST *als;
  public :
  AVI_LIST_HDRL(ifstream &,AVI_LIST *s);
  ~AVI_LIST_HDRL()
  { delete als; delete suivant; };
  int getNbreFrames();
};
class AVI_LIST_MOVI : public AVI_LIST
{ int index;
  public :
  AVI_LIST_MOVI(ifstream &,AVI_LIST *s);
  int getDataIndex();
};
class AVI_RIFF
{ AVI_LIST *premier;
  public :
  AVI_RIFF(ifstream &);
  ~AVI_RIFF() { delete premier; };
  int getNbreFrames();
  int getDataIndex();
};
```



WebCam : Généralités

- **Les webcam**
  - **Capacités longtemps limitées**
    - Résolution 160x128, 320x240 ou VGA 640x480
    - 15 images secondes
    - Connexion USB ou port parallèle ou spécifique
  - **Capteur**
    - CMOS : plus économique
    - CCD : meilleur qualité
  - **Avec ou sans micro intégré**

WebCam : Généralités

- **Les caméra USB 2**
  - **Meilleurs capacités**
    - Capteur à 1,3 million de pixels
    - Rafraichissement : 30 images / seconde
  - **Motorisées**
    - Logitech QuickCam Sphere
  - **Wifi**
    - Linksys WVC54GC

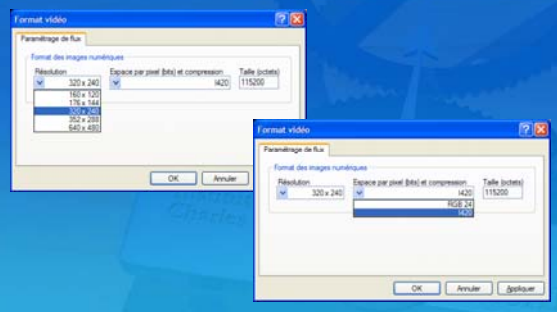
Pilotes

- Pilotes pour linux rarement fournis
  - Pilote CPiA (env. 15 caméras)
  - Pilotes OP519
    - Créative Live
    - Sony
    - Quelques autres (liste disponible)
- Drivers Windows uniformisés

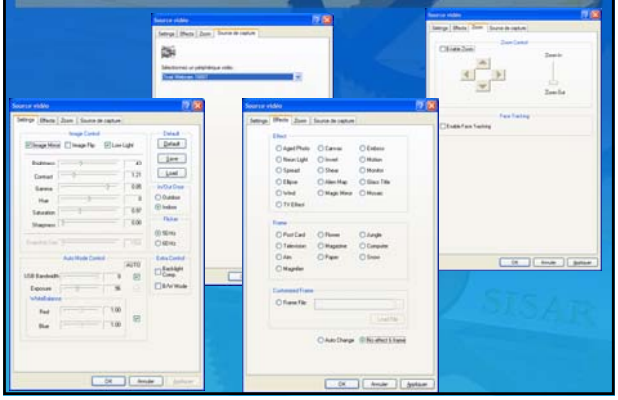
Programmation d'une WebCam

- Utilisation de l'API Windows
  - Application Programing Interface
    - Interface de développement
    - Au-dessus d'une couche logicielle
  - Ensemble de fonctions / fenêtres de dialogue
    - Traitement général des fonctions de la webcam
    - Adaptée par le constructeur pour son périphérique

Fenêtres de configuration



Fenêtres de configuration



API : AVICap

- Fonctionnalités principales
  - Visualiser la vidéo d'une caméra
  - Acquérir la vidéo et le son de la caméra
  - Sauver la vidéo sur disque
- Utilisations possibles (avec programmation)
  - Paramétrage de la caméra
    - Résolution
    - Mode de compression
  - Récupération des images en temps réel

API : AVICap

- Pilotage à base de messages
  - Macro-fonction permettant d'envoyer les messages
- Fonctions
  - Initialisation
  - Fermeture
  - Ouverture des fenêtres de configuration (dialogue)
  - Enregistrement de la vidéo dans un fichier
  - Fonction permettant d'intercepter des évènements
    - Arrivée d'un nouveau frame
    - Changement des paramètres
    - Erreur

Connexion de la caméra

- Caméra associée à une fenêtre Windows
  - Qui reçoit le flux vidéo
  - Activation
    - capDriverConnect(hWnd_WC, 0)
  - Libération
    - capDriverDisconnect(hWnd_WC);

```
hWnd = (HWND)GetHWND();
hWnd_WC = capCreateCaptureWindow("handle",WS_CHILD|WS_VISIBLE,0,0,largeur,
hauteur,hWnd,1);

// Vérifie la connection
if(!capDriverConnect(hWnd_WC, 0)) // erreur
{ MessageBox(NULL,"Erreur lors de l'initialisation de la WebCam.\nReportez-
vous à l'aide pour plus d'information.", "Erreur", MB_ICONERROR);
}
else
{ hDC_WC = GetDC(hWnd_WC); // trouve le DC
}
```

Affichage de la vidéo

- Deux modes d'affichages possibles
  - Direct (hardware) ou logiciel
  - Dépend du matériel
    - Caméra
    - Carte graphique
  - Capacité du matériel
    - BOOL capDriverGetCaps(*hwnd, psCaps, wSize*);
  - Format de la vidéo
    - DWORD capGetVideoFormat(*hwnd, psVideoFormat, wSize*);

```
// en utilisant les messages
CAPDRIVERCAPS CapDrvCaps;
SendMessage(hWndC,WM_CAP_DRIVER_GET_CAPS,sizeof(CAPDRIVERCAPS),
(LONG)(LPVOID)&CapDrvCaps);

// ou les macro-fonctions
capDriverGetCaps(hWndC, &CapDrvCaps, sizeof (CAPDRIVERCAPS));
```

Capacités vidéo

BOOL capDriverGetCaps(*hwnd, psCaps, wSize*);

```
typedef struct {
 UINT wDeviceIndex;
 BOOL fHasOverlay;
 BOOL fHasDlgVideoSource;
 BOOL fHasDlgVideoFormat;
 BOOL fHasDlgVideoDisplay;
 BOOL fCaptureInitialized;
 BOOL fDriverSuppliesPalettes;
 HANDLE hVideoIn;
 HANDLE hVideoOut;
 HANDLE hVideoExtIn;
 HANDLE hVideoExtOut;
} CAPDRIVERCAPS;
```

Capacités vidéo

DWORD capGetVideoFormat(*hwnd, psVideoFormat, wSize*);

```
typedef struct tagBITMAPINFO {
BITMAPINFOHEADER bmiHeader;
RGBQUAD bmiColors[1];
} BITMAPINFO, *PBITMAPINFO;
```

```
typedef struct tagBITMAPINFOHEADER{
DWORD biSize;
LONG biWidth;
LONG biHeight;
WORD biPlanes;
WORD biBitCount;
DWORD biCompression;
DWORD biSizeImage;
LONG biXPelsPerMeter;
LONG biYPelsPerMeter;
DWORD biClrUsed;
DWORD biClrImportant;
} BITMAPINFOHEADER, *PBITMAPINFOHEADER;
```

Affichage de la vidéo

- Mode d'affichage
  - Lien hardware entre la source et l'affichage
    - BOOL capOverlay(hwnd, TRUE);
  - Traitement software du flux vidéo
    - Actication de la fenêtre
      - ✓ capPreview(hWnd_WC, 1);
    - Échelle de la fenêtre par rapport au flux
      - ✓ capPreviewScale(hWnd_WC, TRUE);
    - Vitesse de rafraichissement en millisecondes
      - ✓ capPreviewRate(hWnd_WC, ms);

Utilisation du flux vidéo

- Fonctions d'interception des évènements
  - Arrivé d'un frame
    - BOOL capSetCallbackOnFrame(hwnd, fpProc);
    - LRESULT FrameCb (HWND,LPVIDEOHDR)
  - Changement de status
    - BOOL capSetCallbackOnStatus(hwnd, fpProc);
    - LRESULT capStatusCb(HWND, int, LPCSTR);
  - Génération d'une erreur
    - BOOL capSetCallbackOnError(hwnd, fpProc);

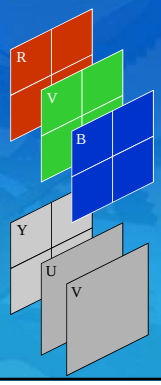
Exemple de programmation

```
LRESULT FrameCallbackProcRVB (HWND hWnd, LPVIDEOHDR lpVideoHdr)
{ CAPSTATUS capStat;

  if (capGetStatus(hWnd,&capStat, sizeof(CAPSTATUS)))
  { LPBYTE lpData = lpVideoHdr->lpData;
    int w=capStat.uiImageWidt, h=capStat.uiImageHeight,ix;
    BYTE *ptrRVB = RVB+3*w*(h-1);

    // BVR to RVB et inversion Haut/bas
    while (h--)
    { ix = w;
      while (ix--)
      { ptrRVB[2] = *lpData++;
        ptrRVB[1] = *lpData++;
        ptrRVB[0] = *lpData++;
        ptrRVB+=3;
      }
      ptrRVB-=6*w; // retourn à la ligne
    }
  }
  return 0;
}
```

Codage BVR dans le flux caméra

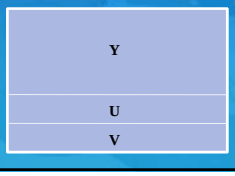
Changement d'espace de couleurs

- **Down sampling**
  - **Pour 4 pixels voisins**
    - 12 octets de codages en RVB
  - **On code en YUV**
    - la luminance Y sur 4 octets (1 par pixel)
    - la chrominance UV sur 2 octets
      - ✓ U moyen sur les 4 pixels
      - ✓ V moyen sur les 4 pixels
  - **Division de la taille de codage par 2**
  - **Perte de précision**
    - Codage YUV sur des entiers
    - Moyenne sur la chrominance



Exemple de programmation

- **Décodage du mode couleur YUV I420**
  - **I420 : format YUV par plans**
    - Première partie : image de Y
      - ✓ (w*h) octets
    - Deuxième partie : image de U (subsampling)
      - ✓ (w*h)/4 octets
    - Troisième partie : image de V (subsampling)
      - ✓ (w*h)/4 octets
  - **Compression :**
    - 12 bits par pixels
    - Perte au niveau de U et V



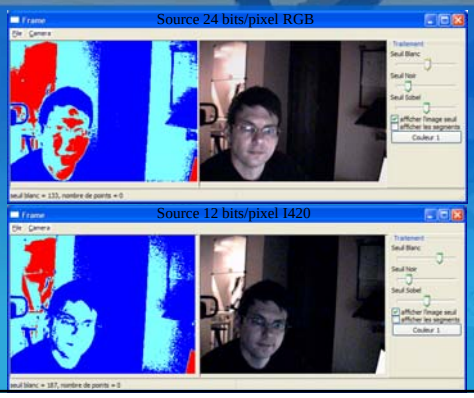
Changement d'espace de couleurs

- **Format I420 : codec Microsoft**
- **Règles de conversion**

$$\begin{pmatrix} R \\ V \\ B \end{pmatrix} = \begin{pmatrix} 1.164383 & 0 & 1.596027 \\ 1.164383 & -0.391762 & -0.812968 \\ 1.164383 & 2.017232 & 0 \end{pmatrix} \begin{pmatrix} Y-16 \\ U-128 \\ V-128 \end{pmatrix}$$

```
LRESULT FrameCallbackProcI420(HWND hWnd, LPVIDEOHDR lpVideoHdr)
{ CAPSTATUS capStat;

  if (capGetStatus(hWnd,&capStat, sizeof(CAPSTATUS)))
  { LPBYTE lpData = lpVideoHdr->lpData,ptrRVB = RVB;
    int w=capStat.uiImageWidth, h=capStat.uiImageHeight,ix;

    // décodage du I420
    double YUV[3],v;
    LPBYTE ptrRVB2 = RVB+3*w,lpData2=lpData+w, lpDataU = lpData+w*h;
    h/=2; w/=2;
    LPBYTE lpDataV = lpDataU+w*h;
    while (h--)
    { ix = w;
      while (ix--)
      { YUV[1] = *lpDataU++-128;            YUV[2] = *lpDataV++-128;
        YUV[0] = 1.164383*(*lpData++-16);
        YUV2RVB(YUV,ptrRVB);                *ptrRVB+=3;
        YUV[0] = 1.164383*(*lpData++-16);
        YUV2RVB(YUV,ptrRVB);                *ptrRVB+=3;
        YUV[0] = 1.164383*(*lpData2++-16);
        YUV2RVB(YUV,ptrRVB2);               *ptrRVB2+=3;
        YUV[0] = 1.164383*(*lpData2++-16);
        YUV2RVB(YUV,ptrRVB2);               *ptrRVB2+=3;
      }
      ptrRVB = ptrRVB2;  ptrRVB2 += 6*w;  lpData = lpData2;   lpData2+=2*w;
    }
  }
  return 0;
}
```

Exemple d'utilisation



Sauvegarde de la vidéo sur disque

- **Nommage du fichier de sauvegarde**
  - **capFileSetCaptureFile(hWndC, nom);**
- **Réservation d'une mémoire initiale**
  - **capFileAlloc(hWndC, (1024L * 1024L * 5));**
- **Lancement de l'enregistrement**
  - **capCaptureSequence(hWndC);**

Format audio

```
WAVEFORMATEX wfex;

wfex.wFormatTag = WAVE_FORMAT_PCM;
wfex.nChannels = 2; // Use stereo
wfex.nSamplesPerSec = 11025;
wfex.nAvgBytesPerSec = 22050;
wfex.nBlockAlign = 2;
wfex.wBitsPerSample = 8;
wfex.cbSize = 0;

capSetAudioFormat(hWndC, &wfex, sizeof(WAVEFORMATEX));
```

Format vidéo

```
CAPTUREPARMS CaptureParms;

float FramesPerSec = 10.0;

capCaptureGetSetup(hWndC,&CaptureParms,
sizeof(CAPTUREPARMS));
CaptureParms.dwRequestMicroSecPerFrame =
(DWORD)(1.0e6/FramesPerSec);

capCaptureSetSetup(hWndC, &CaptureParms,
sizeof (CAPTUREPARMS));
```

```
typedef struct {
DWORD dwRequestMicroSecPerFrame;
BOOL fMakeUserHitOKToCapture;
UINT wPercentDropForError;
BOOL fYield;
DWORD dwIndexSize;
UINT wChunkGranularity;
BOOL fUsingDOSMemory;
UINT wNumVideoRequested;
BOOL fCaptureAudio;
UINT wNumAudioRequested;
UINT vKeyAbort;
BOOL fAbortLeftMouse;
BOOL fAbortRightMouse;
BOOL fLimitEnabled;
UINT wTimeLimit;
BOOL fMCIControl;
BOOL fStepMCIDevice;
DWORD dwMCIStartTime;
DWORD dwMCIStopTime;
BOOL fStepCaptureAt2x;
UINT wStepCaptureAverageFrames;
DWORD dwAudioBufferSize;
BOOL fDisableWriteCache;
UINT AVStreamMaster; } CAPTUREPARMS
```