UNIVERSIDAD POLITECNICA DE VALENCIA

Interacción con OpenCV: detección de movimiento para realizar un instrumento virtual con OpenCV + OpenAL

Apellidos, nombre	Agustí Melchor, Manuel[1] (magusti@disca.upv.es) Ivars Badía, Antonio (anivba@ei.upv.es)
Departamento	[1]Dpto. De Ing. De Sistemas y Computadores
Centro	Universidad Politécnica de Valencia

UNIVERSIDAD POLITECNICA DE VALENCIA

1 Resumen de las ideas clave

En este artículo vamos a presentar como el computador puede detectar las acciones que se realizan utilizando una cámara mediante el uso de OpenCV [1, 2] para la entrada de datos y OpenAL [3] para generar la información de salida, esto es, vamos a hacer un poco de ruido.

El desarrollo se puede ver en un contexto multimedia como una aplicación interactiva, que mediante técnicas de Visión por Computador (VxC) detectará dónde se produce movimiento y generará como respuesta a estos eventos sonido, aunque no profundizaremos en las características de sonido envolvente de OpenAL en este caso. Para ello tomaremos como partida OpenCV 1.1 y OpenAL 1.1. Para su desarrollo nos centraremos en la plataforma GNU/Linux, aunque todo lo expuesto es transportable a otras en que se hayan instalado estas librerías y se disponga de acceso a una cámara USB y a unos altavoces.

2 Introducción

Este artículo mostrará cómo realizar un ejercicio de interacción multimodal entre el usuario y el computador. El usuario se moverá en el espacio que monitoriza la cámara del computador para actuar sobre los objetos virtuales, los que se definen dentro del computador y que no existen físicamente en el entorno de trabajo real. El computador, por su parte recogerá datos con la cámara (información visual) y devolverá como respuesta alguna visualización en pantalla para realimentar la acción del usuario, pero básicamente sonido (información audible) en respuesta a los eventos del usuario.

Analizaremos por separado la forma de trabajo de OpenCV en cuanto a la adquisición y procesado de la información de vídeo y la de OpenAL para generar y reproducir sonidos pregrabados.. Ambas partes serán enlazadas finalmente para ofrecer el prototipo que lleve a cabo una propuesta de instrumento virtual (no existirá físicamente) que será utilizado mediante la identificación de movimiento en ciertas áreas de la imagen obtenida y que, como respuesta, darán lugar a hacer sonar determinada nota o sonido en general.

Estos ejemplos son parte de un trabajo [4] de la asignatura Integración de Medios Digitales (IMD). En la web referenciada podrá encontrar el lector el código completo de los ejemplos.

3 Objetivos

Una vez que el lector haya leído este documento y explorado el código que se proporciona, será capaz de:

- Seguir la ejecución de los ejemplos parciales.
- Explicar los diferentes elementos que componen la solución final.
- Enriquecer el contenido de audio de la solución final.
- Proponer nuevos instrumentos virtuales.

UNIVERSIDAD POLITECNICA DE VALENCIA

4 Interacción visual con OpenCV

Describe este apartado el modo de trabajo con OpenCV para la adquisición de imágenes en vivo desde una cámara. Después pasaremos a hablar de cómo limitar la exploración a determinadas áreas en la imagen. Para terminar se muestra cómo se puede detectar el movimiento que sucede en la escena observada en la imagen.

4.1 Adquisición de vídeo

(1) Indicamos que capture desde cualquier cámara con el valor

(2) Repetimos el proceso indefinidamente.

(2.1) Capturamos un cuadro desde el origen y lo guardamos en forma de imagen

(2.2) Muestra la imagen capturada desde la cámara en la ventana.

(2.3)...y lo mostramos por la ventana anteriormente creada.

(2.4) Si se pulsa la tecla ESC, entonces salimos del bucle

(3) Liberamos el dispositivo de captura y cerramos la ventana.

Listado 1: Algoritmo de adquisición de imágenes desde una cámara.

El ejercicio típico introductorio de OpenCV pasa por cargar y mostrar por pantalla lo que recoge de, por ejemplo, nuestra *webcam*. Para ello, debemos crear un sencillo programa. Por ejemplo el algoritmo mostrado en el listado 1 se muestra implementado en C en el listado 2.

Es importante destacar en la implementación que el uso de la función *cvWaitKey* es muy importante que se ejecute en el bucle principal, ya que es la única forma del módulo *HighGUI* (las funciones de alto nivel de OpenCV) para devolución y manejo de eventos. Por ello, debe ser invocado de forma periódica para un proceso de eventos normal.

4.2 Regiones de interés (ROI)

Hay un concepto muy a tener en cuenta a la hora de interactuar con un programa en OpenCV. Especialmente un programa cuya interacción no se realiza a través de medios tan concretos como el ratón y el teclado, sino más bien a través de un medio más impreciso como la *webcam*. En este contexto, ocurrirá muy a menudo que queramos detectar un objeto dentro de un área aproximada. Para ello, OpenCV ofrece una herramienta simple pero suficiente: las regiones de interés o ROI.

Una ROI, es una área rectangular delimitada por cuatro puntos. Es decir, un rectángulo (Para definir un rectángulo, disponemos de una variable OpenCV llamada *CvRect*). Para asignarle valores a este tipo de variables, llamaremos a una función a la que le indicaremos los cuatro valores que definirán el rectángulo:

```
CvRect cvRect( int pos_izquierda, int pos_superior, int ancho, int alto );
```

Luego ya, para establecer uno de estos rectángulos como ROI para una imagen dada, llamaremos a la función *cvSetImageROI*

```
void cvSetImageROI( IplImage* image, CvRect rect );
```

```
#include <stdio.h>
#include <cv.h>
#include <highgui.h>

int main() {
CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
if( !capture ) {
  fprintf( stderr, "ERROR: no se puede abrir el dispositivo.\n" ); getchar();  return -1;}
IplImage* img=cvQueryFrame(capture);
if(!img) {
   fprintf(stderr,"Error: no hay imagen de la cámara.\n"); getchar(); return -1;}
 //"CV_WINDOW_AUTOSIZE"  se asigna el tamaño a la ventana automáticamente
cvNamedWindow( "mywindow", CV_WINDOW_AUTOSIZE );
 // Repetimos el proceso indefinidamente.
 while( 1 ) {
   // Tomamos un cuadro/imagen
   img = cvQueryFrame( capture );
   if( !img ) {
      fprintf( stderr, "ERROR: no hay imagen de la cámara.\n" ); getchar(); break; }
   cvShowImage( "mywindow", img );
  if( (cvWaitKey(10) & 255) == 27 ) break;
}
// Liberamos el dispositivo de captura y eliminamos la ventana.
cvReleaseCapture( &capture );
cvDestroyWindow( "mywindow" );
return 0;
}
```

Listado 2: Ejemplo de código para la adquisición de vídeo, con OpenCV, desde la cámara.

Esta función, cambiará el valor ROI que tiene guardada *IlpImage* internamente. Las variables *IplImage*, al crearse, se les asigna por defecto una ROI igual a la superficie total de la imagen.

4.3 Interacción por movimiento: piano virtual

El movimiento ante la cámara se traduce en variaciones entre cuadros tomados consecutivamente en el tiempo. Estas variaciones son como estelas que pueden ser interpretadas como elementos a tener en cuenta a la hora de interactuar con el programa.

Siguiendo este razonamiento, podemos restar cuadros consecutivos para descubrir estas diferencias. Esta idea se desarrolla en el ejemplo de D. Millán [5].

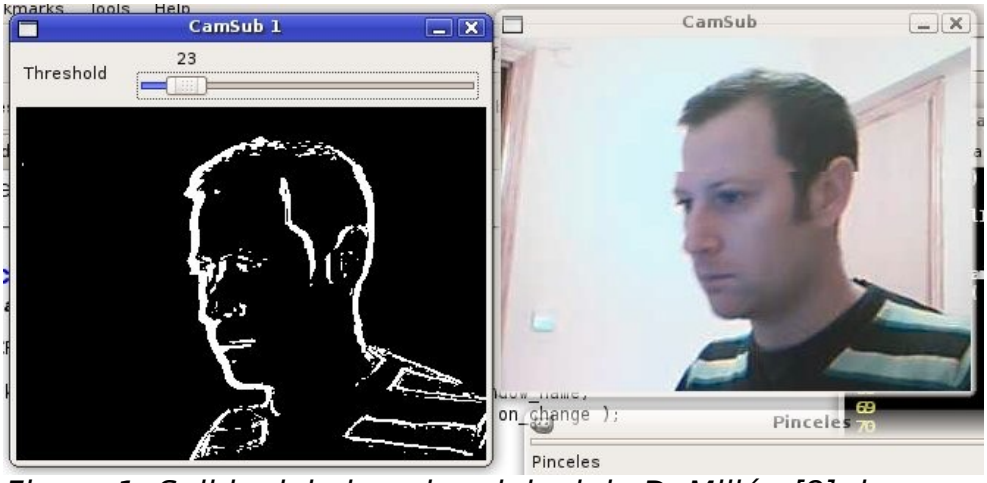

Figura 1: Salida del ejemplo original de D. Millán [8]: imagen de diferencias binarizada (izquierda) para la secuencia de vídeo (último cuadro a la derecha).

Con esta idea como base se puede desarrollar un pequeño piano virtual cuyas teclas aparecen dibujadas en la parte superior de la ventana que muestra la captura de la cámara. Las "teclas" son en realidad áreas de interés sobre las cuales se detectan los movimientos aplicando la diferencia entre cuadros consecutivos. Para ello es necesario:

- Declarar algunas variables globales.
  - El cuadro actual para poder compararlo con el anterior y así, poder detectar las diferencias.
    IplImage* image, lastImage;
  - Las diferencias entre los cuadros *image* y *lastimage*.
    IplImage* diffImage;
  - Las diferencias obtenidas en difImage en binario (blanco y negro).
    IplImage* bitImage;
  - El umbral a partir del cual un valor se considerará 0 ó 1 en una imagen de escala de grises.
    int tr;
  - Para delimitar las ROI.
    CvRect rect[8];
- Por su parte, en el programa principal, inicializamos y creamos las ROI que coincidirán con la posición de las teclas a partir de las propiedades de la imagen obtenida de la cámara.

```
for(i=0;i<8;i++)
{
                    rect[i]=cvRect((width/8)*i,0,(width/8),height/4);
```

Por su parte, el bucle principal (véase listado 3) se definen las variables de tipo imagen con las mismas características que la obtenida desde la cámara, pero con un sólo canal: hemos decidido trabajar con imágenes en escala de grises por simplicidad del código, aunque se pierda

resolución de color. Así que la secuencia de vídeo de la cámara deberá ser convertida de RGB a grises en primer lugar.

```
lint main() {
...
for(;;)
{
 IplImage* frame = 0;
 int c;
 frame = cvQueryFrame( capture );
 if( !frame ) break;
 if(!image){
   image=cvCreateImage(cvSize(frame->width,frame->height),frame->depth,1);
   bitImage=cvCreateImage(cvSize(frame->width,frame->height),frame->depth,1);
 }
 cvCvtColor(frame, image,CV_BGR2GRAY);
 if(!lastImage){ lastImage=cvCloneImage(image); }
 if(!diffImage){
    diffImage=cvCreateImage(cvSize(frame->width,frame->height),frame->depth,1);}
 for(i=0;i<8;i++) {
    cvRectangle(frame,cvPoint(width/8*i,0),cvPoint(width/8*(i+1),height/4),
                    CV_RGB(0,0,0),2,2,0);
     cvRectangle(frame,cvPoint((width/8*i)+8,0),cvPoint(width/8*(i+1),(height/4)-8),
                    CV_RGB(255,255,255),10,8,0);
 }
 cvShowImage( "CamSub", frame );
 cvAbsDiff(image,lastImage,diffImage);
  cvThreshold(diffImage,bitImage,tr,255,CV_THRESH_BINARY);
  for(i=0;i<8;i++) {
   cvSetImageROI(bitImage,rect[i]);
   media=cvMean(bitImage,NULL);
  if(media>1.0)
    cvRectangle(frame,cvPoint(width/8*i,0),cvPoint(width/8*(i+1),height/4),
                    CV_RGB(0,0,0), CV_FILLED,2,0);
 }
 lastImage=cvCloneImage(image);
 cvResetImageROI( bitImage );
 cvShowImage("CamSub 1",bitImage);
c = cvWaitKey(10);
 if( (char) c == 27 ) break;
}//Fin del bucle principal
```

Listado 3: Detalle de código para la implementación del piano virtual.

Convertidas a grises ya se pueden obtener las diferencias entre los valores de la imagen del bucle anterior y el actual. Umbralizamos la imagen de diferencias a dos valores (blanco y negro) y guardamos el resultado . Revisamos las ROI y, si la media de sus valores umbralizados supera cierto valor, pintaremos el interior de la “tecla” para demostrar el lugar donde tratar el evento de “pulsación” de tecla, si no sólo se pinta el borde de la misma. Siempre superpuesto a la imagen de la cámara para que el usuario tenga una realimentación de dónde está el instrumento virtual. El programa continua en esta dinámica mientras no se pulse la tecla ESC.

5 Audio

En este apartado utilizaremos OpenCV para extraer información de una imagen, con ese cálculo generaremos sonido. El objetivo final es crear un software que genere audio en función de la información que se obtenga a través de la cámara. Iremos poco a poco creando “instrumentos virtuales”.

La primera decisión de relevancia, pasaba por elegir una librería de audio adecuada. Nuestra elección ha sido OpenAL, por ser una librería libre, multiplataforma, gratuita y versatil. La característica que quizás da más personalidad a OpenAL, es su orientación a la generación de audio 3D. OpenAL permite ajustar la percepción del audio a la de una fuente situada en el espacio. Además, permite programar todo tipo de efectos con las funciones que incorpora.

5.1 Generador de ruido

Como primer ejemplo para generar audio, hemos pretendido generar un sencillo tono en base a la media del movimiento que la cámara perciba. ¿Como se consigue esto? Restando los cuadros consecutivos de una secuencia de vídeo. El resultado de la diferencia entre ambos identifica el movimiento que ha habido a partir de la base del ejemplo de D. Millán [5]. El algoritmo utilizado se muestra en el listado 2.

Se busca generar un tono en función de la media de movimiento que percibe la cámara. El código de partida obtiene una imagen binaria: una matriz de ceros y unos que asigna un cero a los píxeles de la entrada que no han cambiado entre el cuadro anterior y el actual; así como un uno para aquellos píxeles que hayan variado (todo esto, con un margen de relevancia o umbral). Tomando la citada imagen (*bitImage*) podemos obtener una media en coma flotante que indique el porcentaje de puntos que han sufrido una variación.

El código resultante se puede ver en el listado 5. La primera aproximación a la interacción pasa por generar un tono en función de la media de movimiento que percibe la cámara. Esta es una equivalencia con el porcentaje de puntos que han sufrido una variación.

El cálculo del tono fue una decisión subjetiva. El oído humano puede percibir frecuencias que oscilan entre los 20 y los 22000 herzios aproximadamente. Por ello, un cálculo más ajustado, teniendo en cuenta que la media de la imagen siempre estará entre los 0 y 255, sería

```
double frecuencia=20.0+(media/255)*22000
```

UNIVERSIDAD POLITECNICA DE VALENCIA

(1) Adquirir un cuadro → *lastImage*

(2) Repetimos el proceso indefinidamente.

(2.1) Capturamos un cuadro → image

(2.2) Calcular la diferencia diffImage ← image - lastImage

(2.3)..Binarizar la imagen de diferencias

(2.4) Calcular una medida del "movimiento" habido

(2.5) Asignar un tono al valor así obtenido y hacerlo sonar

(2.6) lastImage ← image

(3) Cerrar ventanas, liberar recursos y dispositivos

Listado 4: Algoritmo del generador de tonos a partir de imágenes.

```
// Otros ficheros de cabera
#include <AL/alut.h>

// Nuevas variables globales para manipular el sonido
ALuint helloBuffer, helloSource;

void generaTono(double media)
{
 double frecuencia=20.0+media*255;
 helloBuffer = alutCreateBufferWaveform(alutCreateBufferWaveform, frecuencia, 0.0, 1.0);
 alSourceStop(helloSource);
 alSourcei (helloSource, AL_BUFFER, helloBuffer);
 alSourcePlay (helloSource);
}

// En el main
...
alutInit(NULL,0);
...
double media=cvMean(bitImage,NULL);
generaTono(media);
```

Listado 5: Ejemplo de código para la generación de tonos simples a partir de imágenes.

es decir, primero convertimos la media en un valor comprendido entre 0 y 1. Después lo escalamos por la máxima frecuencia audible. No obstante, salvo que se tome un umbral de sensibilidad bajo, o que nos movamos mucho ante la cámara, los resultados que se arrojan están en

el rango de frecuencias más bajo; de ahí el que se haya optado por un cálculo más agresivo y llamativo.

Cargamos en *helloBuffer* una forma de onda que creamos mediante *alutCreateBufferWaveform*. La configuración toma la opción de forma de onda senoidal, con valor de frecuencia igual a la variable creada para el efecto, fase inicial igual a 0.0, y duración del sonido de 1.0 segundos. Detenemos la generación de sonido que pueda estar sonando, asignamos a la fuente el contenido que hemos introducido en *helloBuffer y v*olvemos a lanzar la generación de sonido.

Se puede compilar el programa en GNU/Linux con *gcc* usando la siguiente orden, asumiendo que el fichero de código fuente se llama *audifyCamSub.c*:

```
$ gcc audifyCamSub.c `pkg-config --cflags opencv --libs opencv` -lalut
-lopenal -o audifyCamSub
```

5.1.1 Mi Primer Piano

Este ejemplo, se podría decir que es el típico paso lógico. Llegados al punto de poder generar sonidos en función del área que esté sufriendo una diferencia, bien podemos delimitar varias áreas y hacerlas sonar como si de un pequeño teclado se tratara. Se utilizarán funciones de generación de sonidos simples, dejando de lado el tener un WAV por cada nota. Obviamente, esto hará que nuestro piano virtual no suene precisamente como un *Steinway*, pero podréis captar la idea, y asignarle vuestras propias notas no supondrá mayor esfuerzo que una visita al oportuno buscador o un pequeño ejercicio con un programa de edición de audio. El listado 6 muestran dos detalles del código de este ejemplo: la inicialización del audio y la determinación de qué sonido generar en el programa principal.

Se define un número de fuentes de audio (sip, también hemos hecho que la nota suene más hacia un lado u otro en función de la posición del oyente. Estas fuentes harán sonar los sonidos que crearemos con la ayuda de otros tantos *buffers* de audio. El razonamiento[1] sobre la curiosa fórmula que he utilizado para calcular los tonos de cada nota (440.0*pow(2.0,((i+3.0)/12.0))).

Para compilar en linux este código:

```
$ gcc pianoCam.c `pkg-config --cflags opencv --libs opencv` -lalut -lopenal -o pianoCam
```

6 Conclusión

Este artículo ha explorado elementos básicos hasta elaborar un ejemplo de interacción entre el usuario y el computador mediante el uso de la imagen y el sonido. Es un ejemplo de interfaz multimodal o también perceptual. Trabajando con librerías multiplataforma (OpenCV y OpenAL) y sin ninguna atadura al sistema operativo permite portar esta aplicación

[1]Francisco Roco. Como calcular la frecuencia de una nota musical y hacer música con Matlab / Gnu-Octave. (23 de enero de 2007) Última visita mayo 2011. <http://rocoblog.blogspot.com/2007/01/clculo-de-frecuencias-para-una-nota-y.html>

a cualquier sistema y entorno de desarrollo para la creación del ejecutable.

```
void inicializarAudio(){
      int i, error;
      alutInit (0,NULL);
      alGetError();
      alListenerfv(AL_POSITION,listenerPos);
      alListenerfv(AL_VELOCITY,listenerVel);
      alListenerfv(AL_ORIENTATION,listenerOri);
      error = alGetError();
      if (error) printf("%s\n", alutGetErrorString(error)); else printf("Oyente Creado\n");
      alGenBuffers(NUM_BUFFERS, buffer);
      error=alGetError(); if(error){printf("%s\n",alutGetErrorString(error));}
      for(i=0;i<8;i++)  {
            buffer[i] = alutCreateBufferWaveform(ALUT_WAVEFORM_SINE,
                                                  440.0*pow(2.0,((i+3.0)/12.0)), 0.0, 1.0);
            error = alGetError();
            if (error) printf("%s\n", alutGetErrorString(error));
           else printf("- Buffer para tono %d creado.\n",(i+1));
      }
      alGenSources(NUM_SOURCES, source);
      error = alGetError();
      if (error) printf("%s\n", alutGetErrorString(error)); else  printf("- Fuentes creadas.\n");
      for(i=0;i<8;i++) {
         alSourcef(source[i],AL_PITCH,1.0f);  alSourcef(source[i],AL_GAIN,1.0f);
         alSourcefv(source[i],AL_POSITION,sourcePos[i]);
         alSourcefv(source[i],AL_VELOCITY,sourceVel);
         alSourcei(source[i],AL_BUFFER,buffer[i]); alSourcei(source[i],AL_LOOPING,AL_FALSE);
      }
}


// En el programa principal

 for(i=0;i<8;i++)
 {
  cvSetImageROI(bitImage,rect[i]);
  media=cvMean(bitImage,NULL);
  if(media>1.0)alSourcePlay(source[i]);
 }
```

Listado 6: Ejemplo de código para la generación de tonos simples para el piano virtual.

Ahora, apreciado lector, es tu turno. Te animo a ponerte manos a la obra sobre los ejemplos y construir uno que, por ejemplo, defina áreas de interés (de un tamaño fijo) ocupando toda la imagen, Al moverse dentro de ellas deberían hacer sonar un audio y provocar un cambio en la imagen como se muestra en la fig.2. Lo llamaremos *discoCam*.

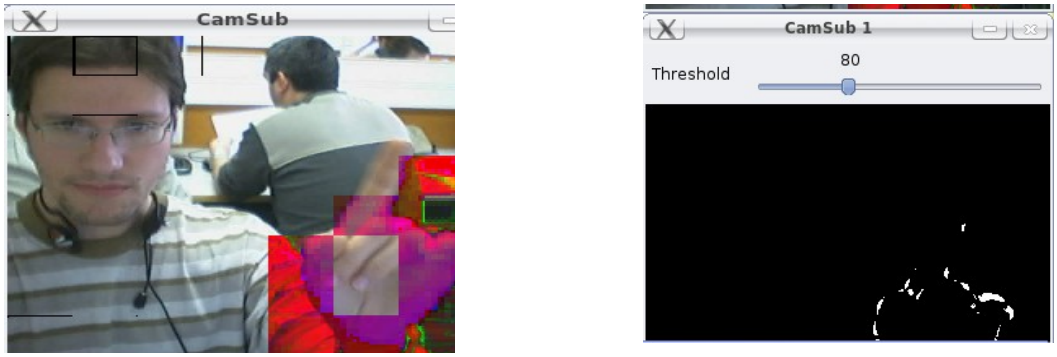

Figura 2: Salida del ejemplo de discoCam.

7 Bibliografía y referencias

[1] “Open Computer Vision Library”. Disponible en <http://sourceforge.net/projects/opencvlibrary/>. Última consulta mayo de 2011.

[2] “OpenCVWiki” , Disponible en http://opencv.w.illowgarage.com/wiki/Welcome.

[3] OpenAL, <http://connect.creativelabs.com/openal/default.aspx>. Última consulta enero de 2011.

[4] Antonio Ivars Badia . Salvapantallas activo con cámara web (Trabajando con OpenCV). Trabajo de la asignatura IMD <http://web-sisop.disca.upv.es/~imd>, curso 2008/2009.

[5] David Millán, OpenCV More with cameras, <http://blog.damiles.com/?p=67>. Última consulta mayo de 2011.