Programación del mando Xbox360

En esta práctica se aprende a programar el mando de la Xbox360. Para ello se utiliza la librería XInput, que forma parte de DirectX. XInput es una API que permite leer el estado del controlador de Xbox 360 para Windows. Además permite cambiar el estado del controlador Xbox 360, para generar vibraciones. Por otro lado, utilizando DirectSound se maneja la entrada y salida de sonido de estos mandos, lo que permite generar efectos estruendo y entrada/salida de voz.
Para aprender a programar el mando se seguirán los diferentes tutoriales que ofrece el SDK de DirectX 9 en la sección Input. Se analizarán las librerías: DirectInput, XInput y DirectSound.
Otros mandos se pueden programar de forma similar con sus propias librerías.

Objetivo

Aprender a programar el mando de XBox 360.
Aprender a utilizar la librería Xinput.
Entender la gestión de mandos sobre una aplicación interactiva.
Realizar una capa de abstracción para los dispositivos de Entrada.

Material

Microsoft DirectX SDK (2010).
Documentación online: https://msdn.microsoft.com/en-us/library/windows/desktop/ee417001(v=vs.85).aspx
Analizar los tutoriales del Sample Browser(2010), sección Input:

Tema	**Descripción**
Módulo 1. XInput SimpleController	Se enseña a comprobar si hay conectados mandos y a leer el estado de éstos.
Módulo 2. XInput RumbleController	Se enseña cómo generar realimentación sobre el usuario a través de la vibración del mando.

Es necesario el mando Xbox 360 (la facultad realiza el préstamo a través de los técnicos).
Capítulo 8 del libro: “Game Engine Architecture” de Jason Gregory, A K Peters, Ltd., Wellesley, Massachusetts, 2009.

Desarrollo

Este tutorial se explicará en clase durante varias sesiones de laboratorio. Los alumnos deberán entender en el laboratorio los tutoriales con la asistencia de los profesores de laboratorio.

Comenzando con XInput

XInput es una API que permite que las aplicaciones reciban información del Controlador Xbox 360 para Windows. Se admiten los efectos de vibración(rumble) del controlador y la entrada y salida de voz.

Este tema proporciona una breve descripción de las capacidades de XInput y cómo configurarlo en una aplicación. Incluye lo siguiente:

Introducción a XInput

La consola Xbox 360 usa un controlador de juegos que es compatible con Windows. Las aplicaciones pueden usar la API XInput para comunicarse con estos controladores cuando están conectados a una PC con Windows (se pueden conectar hasta cuatro controladores únicos a la vez).

Usando esta API, cualquier controlador Xbox 360 conectado se puede consultar para conocer su estado y se pueden establecer los efectos de vibración. Los controladores que tienen los auriculares conectados también se pueden consultar para dispositivos de entrada y salida de sonido que se pueden usar con los auriculares para el procesamiento de voz.

El controlador de Xbox 360

El controlador Xbox 360 tiene dos mandos direccionales analógicos, cada uno con un botón digital, dos disparadores analógicos, un pad direccional digital con cuatro direcciones y ocho botones digitales. Los estados de cada una de estas entradas se devuelven en la estructura **XINPUT_GAMEPAD** cuando se llama a la función **XInputGetState** .

El controlador también tiene dos motores de vibración para suministrar efectos de realimentación de fuerza al usuario. Las velocidades de estos motores se especifican en la estructura **XINPUT_VIBRATION** que se pasa a la función **XInputSetState** para establecer los efectos de vibración.

Opcionalmente, un auricular puede conectarse al controlador. El auricular tiene un micrófono para la entrada de voz y un auricular para salida de sonido. Puede llamar a los **XInputGetAudioDeviceIds** o heredadas **XInputGetDSoundAudioDeviceGuids** para obtener los identificadores de dispositivos que se corresponden con los dispositivos para el micrófono y auriculares. A continuación, puede usar **las API de audio del sistema** para recibir la entrada de voz y enviar la salida de sonido.

Usando XInput

Usar XInput es tan simple como llamar a las funciones de XInput según sea necesario. Con las funciones de XInput, puede recuperar el estado del controlador, obtener ID de audio de los auriculares y configurar los efectos de los ruidos del controlador.

Controladores Múltiples

La API XInput admite hasta cuatro controladores conectados en cualquier momento. Las funciones de *XInput* requieren un parámetro *dwUserIndex* que se transfiere para identificar el controlador que se está configurando o consultando. Esta ID estará en el rango de 0-3 y se configura automáticamente por XInput. El número corresponde al puerto en el que está conectado el controlador y no es modificable.
Cada controlador muestra qué ID está usando al iluminar un cuadrante en el "anillo de luz" en el centro del controlador. Un valor *dwUserIndex* de 0 corresponde al cuadrante superior izquierdo; la numeración procede alrededor del anillo en el sentido de las agujas del reloj.
Las aplicaciones deben ser compatibles con varios controladores.

Obtener el estado del controlador

Durante la duración de una aplicación, el estado de un controlador probablemente se hará con más frecuencia. De cuadro en cuadro en una aplicación de juego, se debe recuperar el estado y actualizar la información del juego para reflejar los cambios del controlador.
Para recuperar el estado, use la función **XInputGetState** :

```
DWORD dwResult;
for (DWORD i=0; i< XUSER_MAX_COUNT; i++ )
{
  XINPUT_STATE state;
  ZeroMemory( &state, sizeof(XINPUT_STATE) );

        // Simply get the state of the controller from XInput.
        dwResult = XInputGetState( i, &state );

        if( dwResult == ERROR_SUCCESS )
  {
      // Controller is connected
  }
        else
  {
            // Controller is not connected
  }
}
```

Tenga en cuenta que el valor de retorno de **XInputGetState** se puede usar para determinar si el controlador está conectado. Las aplicaciones deben definir una estructura para contener la información del controlador interno; esta información debe compararse con los resultados de **XInputGetState** para determinar qué cambios, como pulsaciones de botones o deltas de controladores analógicos, se realizaron en ese marco. En los ejemplos siguientes *g_Controllers* representa dicha estructura.
Una vez que se ha recuperado el estado en una estructura **XINPUT_STATE** , puede verificar los cambios y obtener información específica sobre el estado del controlador.
El miembro *dwPacketNumber* de la estructura **XINPUT_STATE** se puede usar para verificar si el estado del controlador ha cambiado desde la última llamada a **XInputGetState** . Si *dwPacketNumber* no cambia entre dos llamadas secuenciales a **XInputGetState** , entonces no ha habido cambios en el estado. Si difiere, la aplicación debe verificar el miembro de *Gamepad* de la estructura **XINPUT_STATE** para obtener información de estado más detallada.
Por motivos de rendimiento, no llame a **XInputGetState** para obtener una ranura de usuario 'vacía' en cada paso. En cambio, le recomendamos espaciar las comprobaciones de los nuevos controladores cada pocos segundos.

Zona muerta

Para que los usuarios tengan una experiencia de juego consistente, su juego debe implementar la zona muerta correctamente. La zona muerta es un valor de "movimiento" informado por el controlador incluso cuando los mandos analógicos están intactos y centrados. También hay una zona muerta para los 2 disparadores analógicos.

Nota: los juegos que usan XInput que no filtran la zona muerta experimentarán un juego deficiente. Tenga en cuenta que algunos controladores son más sensibles que otros, por lo que la zona muerta puede variar de una unidad a otra. Se recomienda que pruebe sus juegos con varios controladores Xbox 360 en diferentes sistemas.

Las aplicaciones deben usar "zonas muertas" en las entradas analógicas (disparadores, palancas) para indicar cuándo un movimiento se ha realizado suficientemente en la palanca o disparador como para considerarse válido.
Su aplicación debe verificar las zonas muertas y responder de manera adecuada, como en este ejemplo:

```
XINPUT_STATE state = g_Controllers[i].state;

float LX = state.Gamepad.sThumbLX;
float LY = state.Gamepad.sThumbLY;

//determine how far the controller is pushed
float magnitude = sqrt(LX*LX + LY*LY);

//determine the direction the controller is pushed
float normalizedLX = LX / magnitude;
float normalizedLY = LY / magnitude;

float normalizedMagnitude = 0;
```

```
//check if the controller is outside a circular dead zone
if (magnitude > INPUT_DEADZONE)
{
  //clip the magnitude at its expected maximum value
  if (magnitude > 32767) magnitude = 32767;

  //adjust magnitude relative to the end of the dead zone
  magnitude -= INPUT_DEADZONE;

  //optionally normalize the magnitude with respect to its expected
range
  //giving a magnitude value of 0.0 to 1.0
  normalizedMagnitude = magnitude / (32767 - INPUT_DEADZONE);
}
else //if the controller is in the deadzone zero out the magnitude
{
       magnitude = 0.0;
       normalizedMagnitude = 0.0;
}

//repeat for right thumb stick
```

Este ejemplo calcula el vector de dirección del controlador y qué tan lejos a lo largo del vector se ha empujado el controlador. Esto permite la aplicación de una zona muerta circular simplemente verificando si la magnitud del controlador es mayor que el valor de la zona muerta. Además, el código normaliza la magnitud del controlador, que luego se puede multiplicar por un factor específico del juego para convertir la posición del controlador en unidades relevantes para el juego.

Tenga en cuenta que puede definir sus propias zonas muertas para los palos y activadores (en cualquier lugar del 0-65534), o puede usar las zonas muertas provistas definidas como XINPUT_GAMEPAD_LEFT_THUMB_DEADZONE, XINPUT_GAMEPAD_RIGHT_THUMB_DEADZONE y XINPUT_GAMEPAD_TRIGGER_THRESHOLD en XInput.h:

```
#define XINPUT_GAMEPAD_LEFT_THUMB_DEADZONE 7849
#define XINPUT_GAMEPAD_RIGHT_THUMB_DEADZONE 8689
#define XINPUT_GAMEPAD_TRIGGER_THRESHOLD 30
```

Una vez que se aplica la zona muerta, puede que le resulte útil escalar el punto flotante del rango resultante [0.0,1.0] (como en el ejemplo anterior) y/o aplicar una transformación no lineal. Por ejemplo, con los juegos de conducción, puede ser útil dividir el resultado para proporcionar una mejor sensación al conducir los autos con un gamepad, dando más sensibilidad/precisión en los rangos inferiores, y ampliar el rango cuando se fuerza el mando al extremo.

Encuesta del Mando por Bucle o por Timer

La comunicación con el mando de Xbox se realiza mediante encuesta, por lo que es necesario dispara esta encuesta periódicamente. Esto se puede realizar mediante un Timer o modificando el bucle de mensajes:

- Utilizar un Timer es adecuado cuando se quieren consumir pocos recursos, liberando el resto del tiempo al procesador.
- En el caso de los videojuegos normalmente interesa reducir la latencia, por lo que interesa más modificar el bucle principal.

Ejercicio 1. Bucle de mensajes con encuesta al mando.

1. Instalar el Módulo 1 y entender su código.

El código del módulo 1 implementan el siguiente bucle de mensajes:

```
while (WM_QUIT != msg.message)
	{	bGotMsg = (PeekMessage(&msg, NULL, 0U, 0U, PM_REMOVE) != 0);
		if (bGotMsg)
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
		else
		{
			UpdateControllerState();
			RenderFrame();
		}
	}
```

Analizar las funciones:

- *UpdateControllerState()* que se encarga de leer el estado del mando.
- *RenderFrame()* que se encarga de representar el estado del mando.

Analizar las diferencias entre las funciones:

- *GetMessage(&msg, NULL, 0, 0)*
- *PeekMessage(&msg, NULL, 0U, 0U, PM_REMOVE).*

  Podéis comprobar que al final de la función *RenderFrame()* se duerme el proceso utilizando la función *Sleep()*. En sistemas multiproceso es aconsejable liberar a la CPU para que pueda atender otros procesos. Analizar el uso de CPU modificando el tiempo que se duerme el proceso (utilizar el analizador de rendimiento del Administrador de Tareas de Windows). Modificar el tiempo que se duerme el proceso para comprobar el consumo de CPU.

2. Modificar el bucle de mensajes para que se:
    - **Dibuje/Represente en el bucle de mensajes.**
    - **Encueste el Mando por Timer y Callback.**

Ajuste de los efectos de vibración

Además de obtener el estado del controlador, también puede enviar datos de vibración al controlador para informar al usuario del controlador. El controlador contiene dos motores de rumble que se pueden controlar de forma independiente pasando valores a la función **XInputSetState** .
La velocidad de cada motor se puede especificar utilizando un valor WORD en la estructura **XINPUT_VIBRATION** que se pasa a lafunción **XInputSetState de la** siguiente manera:

```
XINPUT_VIBRATION vibration;
ZeroMemory( &vibration, sizeof(XINPUT_VIBRATION) );
vibration.wLeftMotorSpeed = 32000; // use any value between 0-65535 here
vibration.wRightMotorSpeed = 16000; // use any value between 0-65535 here
XInputSetState( i, &vibration );
```

Tenga en cuenta que el motor derecho es el motor de alta frecuencia, el motor izquierdo es el motor de baja frecuencia. No siempre necesitan establecerse en la misma cantidad, ya que proporcionan diferentes efectos.

Emular el ratón

Si se genera el movimiento del ratón, automáticamente se generan los eventos correspondientes. El siguiente código muestra cómo manejar el ratón desde el teclado: https://msdn.microsoft.com/en-us/library/windows/desktop/ms648380(v=vs.85).aspx#_win32_Using_the_Keyboard_to_Move_the_Cursor

```
POINT pt;               // cursor location
RECT rc;                // client area coordinates
static int repeat = 1;    // repeat key counter
switch (message)
{
      GetCursorPos(&pt);
      // Convert screen coordinates to client coordinates.
      ScreenToClient(hwnd, &pt);
      switch (wParam)
      {
      // Move the cursor to reflect which
      // arrow keys are pressed.
         case VK_LEFT:            // left arrow
            pt.x -= repeat;
            break;
         case VK_RIGHT:           // right arrow
            pt.x += repeat;
            break;
         case VK_UP:              // up arrow
            pt.y -= repeat;
            break;
         case VK_DOWN:            // down arrow
            pt.y += repeat;
            break;
         default:
            return 0;
      }
      repeat++;          // Increment repeat count.
      // Keep the cursor in the client area.
       GetClientRect(hwnd, &rc);
       if (pt.x >= rc.right)
      {
         pt.x = rc.right - 1;
      }
      else
      {
         if (pt.x < rc.left)
         {
            pt.x = rc.left;
         }
      }
       if (pt.y >= rc.bottom)
         pt.y = rc.bottom - 1;
      else
         if (pt.y < rc.top)
            pt.y = rc.top;

      // Convert client coordinates to screen coordinates.
      ClientToScreen(hwnd, &pt);
      SetCursorPos(pt.x, pt.y);
      return 0;

   case WM_KEYUP:
      repeat = 1;          // Clear repeat count.
      return 0;
}
```

Emular el teclado

Se pueden generar los eventos equivalentes, esto se puede hacer mediante paso de mensajes, como se muestra en esta documentación:

https://msdn.microsoft.com/en-us/library/windows/desktop/ms646267(v=vs.85).aspx#_win32_Simulating_Input

Existen dos alternativas, *SendInput* o *keybd_event* (*mouse_event* para el mouse). A continuación, se muestra un ejemplo con *keybd_event:*

```
#include <windows.h>
  void SetNumLock( BOOL bState )
  {  BYTE keyState[256];
     GetKeyboardState((LPBYTE)&keyState);
     if( (bState && !(keyState[VK_NUMLOCK] & 1)) ||
        (!bState && (keyState[VK_NUMLOCK] & 1)) )
     {  // Simulate a key press
        keybd_event( VK_NUMLOCK,
                  0x45,
                  KEYEVENTF_EXTENDEDKEY | 0,
                  0 );
     // Simulate a key release
        keybd_event( VK_NUMLOCK,
                  0x45,
                  KEYEVENTF_EXTENDEDKEY |
                  KEYEVENTF_KEYUP,
                  0);
     }
  }
  void main()
  {     SetNumLock( TRUE );
  }
```

Módulo 2. Feedback del Mando

1. Analizar y comprender el código del Módulo 2, que permite generar un feedback de vibración sobre el jugador.

2. **Utilizando el mando de la Xbox emular el ratón y las teclas de navegación** (Enter, Esc, Cursores, Av/Re. Pág. e Inicio/Fin).

Por ejemplo:

- El joystick de la izquierda equivale al movimiento de ratón
- Botón LS/RS equivalen a botones izquierdo/derecho del ratón
- Boton X/B equivalen a Inicio/Fin.
- Boton Y/A equivalen a Av/Re PAG.
- Botones Back/Start equivalen a la tecla ESC/ENTER
- El Pad equivale a los cursores.
- En eventos PushDown/Up deben vibrar motores Izq./Der
- Los Shoulder equivalen a la rueda del ratón

3. **Extender el alcance del mando para que funcione sobre todo el escritorio**.

Para ello se debe capturar el manejador de la ventana activa:

```
HWND hWnd = GetActiveWindow()
```

Utilizando este manejador podremos **enviar los eventos de teclado a la ventana activa**. Además, será aconsejable trabajar con el ratón en coordenadas de escritorio, para no vernos limitados al área cliente.

Módulos 3. Abstracción del Mando.

- Se debe dividir la implementación en tres capas software:
  Capa *HIDHardware*: Encargada de leer/escribir el dispositivo utilizando su formato.
  - Debe implementarse para cada dispositivo.
  - Puede depender de librerías específicas para el mando.
  - Debe resolver las zonas muertas, pues esto depende fuertemente del dispositivo.
  - Debe realizar el calibrado del mando, p.e. calcular y descontar los *offsets*.
- Capa *HIDAbstracta*: Encargada de contener la información en un formato normalizado. Debe ser independiente del dispositivo y de la aplicación.
  - Asignar/configurar un mapeo de ejes.
  - Puede realizar el filtrado de los ejes analógicos (ventaja de trabajar en formato de punto flotante)
  - Debe reconocer gestos y combinaciones.
  - Debe ser independiente de librerías, S.O. y de la aplicación.
- Capa Aplicación: Encargada de utilizar la información del HID en la aplicación. Utilizará los recursos de la plataforma para la que se desarrolle.
  - Aplicar las acciones
  - Acciones dependientes del estado o contexto, como habilitar/deshabilitar controles.

La clase *HIDAbstracta* debe implementar las funcionalidades típicas que se requieren para un videojuego:

- Configure el mapeo de ejes
- Realice la detección de eventos y gestos.
- Proporcione una experiencia de usuario particular, p.e. añadir inercia al puntero.

La clase *HIDHardware* derivará de la clase *HIDAbstracta e implementará* aquellos métodos virtuales de encuesta/órdenes al mando.

La Aplicación creará un objeto de *HIDHardware* que será utilizado para consultar los eventos y gesto. La aplicación también se encargará de la temporizar de esta consulta, es decir controlar el tiempo de encuesta al mando.

- **Trabajo en Casa:** Plantear las clases, propiedades y métodos.
- **Trabajo de Laboratorio1**:Implementar las clases para el caso del mando de XBox.
- **Trabajo de Laboratorio2**:Realizar una aplicación (sin ventana) que permita manejar el escritorio de Windows con el mando de la XBox (mapeo propuesto en la práctica anterior).

Ejemplo:

```
class BaseHID
{
public:
	bool bConected;			//Mando Conectado

	//Gets & Sets
	bool gBU(WORD bit);			//Estado del Boton codificado en bit
	float gLT();				//Left Triger [0,1]
	float gRT();				//Right Triger [0,1]
	float gLJX();				//LeftJoyX [-1,1]
	float gLJY();				//LeftJoyY [-1,1]
	float gRJX();				//RightJoyX [-1,1]
	float gRJY();				//RightJoyY [-1,1]
	float gLJXf();				//LeftJoyXfiltered [-1,1]
	float gLJYf();				//LeftJoyYfiltered [-1,1]
	float gRJXf();				//RightJoyXfiltered [-1,1]
	float gRJYf();				//RigthJoyYfiltered [-1,1]
	void sLR(float cantidad,float tiempo);	//LeftRumble [0,1]: cantidad [0,1], tiempo [0,inf]
	void sRR(float cantidad,float tiempo);	//RightRumble [0,1]: cantidad [0,1], tiempo [0,inf]

	//Gestos
	bool BD(WORD Bit);			//Boton Down codificado en Bit
	bool BU(WORD Bit);			//Boton Up codificado en Bit
	bool GRLJ();				//Gesto de Rotación del LeftJoy

	BaseHID(float t)			//Constructor que recoge el periodo de muestreo
	{	T=t/1000;			//Periodo de muestreo
		a = T / (0.1 + T)		//Cte. de tiempo para filtros (depende de T)
	};
	~BaseHID()	{};
	void Actualiza();			//Actualiza Mando2HID y HID2Mando.
protected:
	//Entradas
	WORD  wButtons;				//Botones (Utilizo Codificación Xbox360)
	WORD  wLastButtons;			//Botones anteriores (Utilizo Codificación Xbox360)
	float fLeftTrigger, fRightTrigger;	//[0.0,1.0]
	float fThumbLX, fThumbLY, fThumbRX,fThumbRY;		//[-1.0,1.0]
	float fThumbLXf, fThumbLYf,  fThumbRXf, fThumbRYf;	//[-1.0,1.0] Filtrado
	float T;				//Perido de actualización
	const float a;				//Cte.Tiempo Filtro
	//Salidas
	float fLeftVibration, fRightVibration;	//[0.0,1.0]  Salida
	float tLR = 0.0;			//Tiempo que queda de vibración en LR
	float tRR = 0.0;			//Tiempo que queda de vibración en RR
	//Gestos
	WORD wButtonsDown;		//EventosDown Botones (Codificación Xbox360?)
	WORD wButtonsUp;		//EventosUp Botones (Codificación Xbox360?)
	EstadosRotacion Ro;		//Estado del gesto de rotación
	float tRo=0.0;			//Tiempo que queda para el gesto de rotación

//Funciones virtuales que se deben reimplementar para cada mando.
	virtual bool LeeMando()=0;		//Lee estado del mando
	virtual void EscribeMando()=0;		//Escribe Feeback en mando
	virtual void Mando2HID()=0;		//Vuelca el estado del mando al HID
	virtual void Calibra() = 0;		//Calibra Mando

};
```

```
void BaseHID::Actualiza()
{
	wLastButtons = wButtons;		//Copia estado de botones
	bConected = LeeMando();		//Leo Mando
	if (bConected == true)
	{
		Mando2HID();	//Vuelco de Mando a HID normalizando
		//Actualizo Gestos de entrada genéricos (entradas)
		....
		//Genero Gesto de feedback (salida)
		....
		EscribeMando();	//Escribo en Mando el feedback
	}
}
```

```
#include <Windows.h>
#include <XInput.h>
#include "HID.h"

class HIDXBox : public BaseHID
{
private:
	CONTROLER_STATE XBox;
public:
	HIDXBox(float t) :BaseHID(t){};
	bool LeeMando();
	void EscribeMando();
	void Mando2HID();
	void Calibra();
};
```

En la Aplicación crearemos un objeto para manejar el mando.

```
#define T 10   //ms para actualizar
#define TARGET_XBOX360
#ifdef TARGET_XBOX360
	HIDXBox Control(T);
#elif defined(TARGET_PS3)
	HIDPs Control(T);
#elif defined(TARGET_WII)
	HIDWii Control(T);
#endif
```

En la aplicación daremos la orden de actualizar el mando, encuestaremos sus gestos y les asignaremos una funcionalidad. Se implementar utilizando un Timer

---

```
VOID CALLBACK TimerCallBack()
{
	Control.Actualiza();  //Actualiza nuestro HID
	GeneraEfectos(&Control);//Genera los efectos en la aplicación en función de los gestos del control
}
```
