Simulador de conducción de una fumigadora con banderillero satelital.

Piedrasanta, Federico
Universidad Tecnológica Nacional, Facultad Regional Córdoba

Abstract

Frente a las demandas del mercado se decidió crear un elemento capaz de capacitar a los trabajadores del sector agropecuario en el uso de los banderilleros satelitales, es así como nace la idea de diseñar un simulador en la computadora que permita visualizar y llevar a cabo el uso de un banderillero sin necesidad de conducir una fumigadora real.

A partir del reconocimiento del problema empírico se comenzó la búsqueda de materiales bibliográficos para evaluar el modo de desarrollar un simulador Además se proponía que este presentara un mapa y un resumen del proceso de fumigación virtual.

Las herramientas utilizadas fueron DirectX 9.0®, Visual C++ 6.0®, Borland C++ Builder 6.0®, MS Access®.

Con el reconocimiento del problema y con las herramientas antes mencionadas se crea un escenario, un campo en tres dimensiones, alambrado, por donde la maquina puede desplazarse siguiendo cualquier trayectoria compatible con una fumigadora. El usuario puede controlar la dirección y la velocidad. Un banderillero satelital, colocado arriba del monitor va mostrando los datos necesarios para llevar la fumigadora por el camino correcto (como ocurre en la realidad).

Los resultados obtenidos fueron que en las diversas ferias donde fue presentado generó gran atracción, de especialistas y del público en general. Además, se pudo observar que la capacitación del usuario era mucho más rápida, económica y completa que la solamente realizada "in-situ".

Palabras Claves
Simulador, banderillero satelital, gráficos 3D, sonido, dispositivos externos, base de datos

Introducción

El proyecto consistió en el desarrollo de un Simulador de conducción de una fumigadora con banderillero satelital.

El banderillero satelital es un sistema de guía para fumigadoras terrestres que utiliza señales GPS (Global Position System), este se impuso en el mercado agropecuario debido a que es una herramienta que asiste a los conductores en las tareas de desmalezamiento y control de plagas para un uso eficiente de dosis de agroquímicos y fertilizantes.

A partir de la creciente inversión en banderilleros satelitales por parte de la industria del "agro", surge la necesidad de crear un elemento capaz de capacitar a los trabajadores de este sector en el uso de los mismos, es así como nace la idea de diseñar un simulador en la computadora que permita visualizar y llevar a cabo el uso de un banderillero sin necesidad de conducir una fumigadora real para la enseñanza más rápida, económica y completa que la solamente realizada "in-situ".

La creación de un simulador significa prescindir de grandes espacios verdes, despreocuparse de la disponibilidad de una fumigadora y permitir al usuario dedicarse exclusivamente al uso del banderillero sin peligro de dañar un vehículo de gran porte a causa de la desconcentración.

Desarrollo del trabajo

A partir del reconocimiento del problema empírico se comenzó la búsqueda de materiales bibliográficos para evaluar el modo de desarrollar un simulador en donde una fumigadora debía cubrir con la dosis justa de agroquímicos y fertilizantes en un enorme

cultivo virtual, con la utilización de un banderillero real. Además se proponía que este presentara un mapa y un resumen del proceso de fumigación virtual.

A continuación se desarrollaran los elementos principales para que el simulador se haga efectivo (herramientas de 3D, de sonido, de base de datos y lenguajes de programación).

Sobre las herramientas utilizadas.

Si bien existe poca disponibilidad de recursos bibliográficos al respecto, una vez reconocidas las fuentes y posterior estudio de las mismas se decidió utilizar, para obtener los gráficos en tres dimensiones, las herramientas de desarrollo provistas por DirectX 9.0®. Si bien en el mercado informático existen diversas herramientas como puede ser OpenGL, Blender, y demás, para la creación de juegos en 3D, ninguna parecía ser tan completa para el desarrollo buscado. Además influyo que DirectX 9.0® tiene ayuda contextual suficiente como para comprender el uso de las funciones contenidas en las librerías de manera casi intuitiva.

El lenguaje que se utilizó para la programación del simulador fue Visual C++ 6.0, como ambos son desarrollos por la misma empresa fue posible su implementación.

A fin de ordenar, y presentar claramente el desarrollo del software, se lo dividirá en cuatro bloques. En el primer bloque consta de la inicialización de la placa de video, en el segundo cómo se hace un escenario en 3D, en el tercero la implementación del sonido, y por último un bosquejo de la orden de ejecución.

Primer bloque: *Inicialización de la placa de video.*

Se empleó las herramientas de Direct3D, este es un API gráfico de bajo nivel que nos permite dibujar mundos en 3D, este puede ser visto como un mediador entre la aplicación y el dispositivo gráfico.

Se hace uso de la clase LPDIRECT3D9, el cual se inicializa con la versión del SDK (software development kit) de DirectX que se esta utilizando. Luego, se setean los parámetros de visualización haciendo uso de la estructura D3DPRESENT_PARAMETERS, por último, se inicializa el objeto IDirect3DDevice9, que representa la placa de video, la cual esta inicializada con los parámetros seteados en la estructura antes mencionada.

Lo anteriormente desarrollado se expresa en el código de la siguiente manera:

```
IDirect3DDevice9 *id3dd9;
LPDIRECT3D9 lpd3d9
lpd3d9 = Direct3DCreate9( D3D_SDK_VERSION ) // Inicialización del objeto
                                            //LPDIRECT3D9, donde
                                            //D3D_SDK_VERSIÓN es una
                                            //constante con el valor de la
                                            //versión del SDK
D3DPRESENT_PARAMETERS d3dpp; // Estructura de seteo de parámetros.
d3dpp.Windowed = TRUE; // se muestra en una ventana.
d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD; // No hay efectos de transición.
d3dpp.BackBufferFormat = D3DFMT_UNKNOWN; // No se asigna ningún formato
                                            //del BackBuffer.
d3dpp.BackBufferWidth        = 1024; // Ancho de la pantalla en píxeles.
d3dpp.BackBufferHeight       = 768; // Alto de la pantalla en píxeles.
```

```
lpd3d9->CreateDevice( D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
                      D3DCREATE_SOFTWARE_VERTEXPROCESSING,
                      &d3dpp, &id3dd9) //Creación del dispositivo id3dd9 con los
                                       //parámetros seteados en la estructura d3dpp
```

Segundo Bloque: *Cómo se hace un escenario en 3D.*

Cabe destacar que DirectX genera objetos complejos a partir de triángulos, así un cuadrado consta de dos triángulos, un cubo de seis cuadrados, esto es, doce triángulos.

La declaración de los triángulos se hace a través de sus vértices, donde cada uno esta representado por su posición x, y, z; color y coordenadas de textura.

Es recomendable establecer los vértices y las dimensiones en el origen de tamaño uno y posteriormente hacer uso de la función provista para dimensionar y posicionar en el espacio 3D (rotar, mover en el eje de las x, y o z)

La orden de ejecución en seudo-código queda entonces de la siguiente forma:

Nota: En todos los casos se hace uso de los métodos de la clase IDirect3DDevice9.
Para la definición del Buffer de vértices se utiliza el objeto LPDIRECT3DVERTEXBUFFER9.
Para el uso y cálculo de vectores se empleó la clase D3DXVECTOR3, en el caso del apuntador a las texturas usadas fue la clase LPDIRECT3DTEXTURE9 y para la creación de los materiales se uso la clase D3DMATERIAL9.
Para conocer los parámetros que recibe cada método puede referirse a la Ayuda que provee DirectX.

```
//Crear el buffer de vértices
CreateVertexBuffer();

//Establecer el origen de los vértices:
SetStreamSource();

//Establecer la textura
D3DXCreateTextureFromFile();

//Establecer el formato de los vértices que se va a dibujar.
SetFVF();

//Posicionar el objeto en el mundo.
SetTransform();

//Establecer la textura para dibujar
SetTexture();

//Dibujar
BeginScene();
DrawIndexedPrimitive();
EndScene();
```

Tercer bloque: *La implementación del sonido.*

El sonido, muy importante en la sensación de realidad, acompaña el régimen de funcionamiento del motor (ralentí, transitorios de aceleración, etc.). También se escucha el ruido producido por el choque con los alambrados.

Para la creación del sonido se implemento las herramientas provistas en DirectSound.

La Clase usada fue LPDIRECTSOUND.

Las siguientes funciones en seudo código muestran la forma de implementación.

```
int Sonido::CargarDirectSound(char* nombreArchivo){// nombre del archivo ".wav"
        HMMIO wavefile; // Se abre el archivo.
        wavefile = mmioOpen(nombreArchivo, 0, MMIO_READ| MMIO_ALLOCBUF);
        DSBUFFERDESC bufdesc; // creamos un DirectSoundBuffer para contener los
                                        //datos
        memset(&bufdesc, 0, sizeof(DSBUFFERDESC)); //Se inicializa en cero.
        bufdesc.dwSize = sizeof(DSBUFFERDESC);//Se asigna el tamaño.
        dSound->CreateSoundBuffer(&bufdesc, &(buffer), NULL)//Se crea el Buffer.
        (buffer)->Lock();
                escribir();// Se escriben los datos en el buffer que se acaba de crear
        (buffer)->Unlock();
        mmioClose(wavefile, 0); // Se cierra el archivo
        return(1);
}

void Sonido::Reproducir(bool repetir, unsigned long posición){
        buffer->Stop(); //Se para el sonido que se esta reproduciendo
        buffer->SetCurrentPosition(pos);// Se setea la posición del sonido.
        if (repetir)
                buffer->Play(0, 0, DSBPLAY_LOOPING);
        else
                buffer->Play(0, 0, 0);
}
```

Cuarto bloque: *Bosquejo de la orden de ejecución*.

A continuación se presentara una estructura del uso de todas las funciones anteriormente vista para completar la creación de un escenario en 3D con sonidos.

```
LRESULT WINAPI MsgProc( HWND hWnd, UINT msg, WPARAM wParam, LPARAM
lParam )//Función definida para capturar las interrupciones o mensajes.
{
        //Si proviene del Mouse Significa un giro de volante.
        //Si proviene del teclado es aceleración o desaceleración.
        /*La posición del conductor se obtiene utilizando las ecuaciones de cinemáticas de
        movimiento calibradas empíricamente y a las que se le agregó un movimiento en el eje
        Z para simular un desplazamiento por un terreno irregular.*/

        calculoGiroVolante();
```

```
	calculoAceleracion();
	x += vel*cosf(tita/57.3f); //Próxima Posición x.
	y += vel*sinf(tita/57.3f); // Próxima  Posición y.
	lx = x + 5*cosf(tita/57.3f); // Próxima Donde se mira en x.
	ly = y + 5*sinf(tita/57.3f); // Próxima Donde se mira en y.

	D3DXMATRIXA16 matWorld; //Matriz para cálculo de movimiento y rotación.
	D3DXMatrixIdentity( &matWorld ); //Matriz identidad.
	D3DXMatrixRotationX( &matWorld, 0 ); //Se rota a la posición origen.
	D3D9->SetTransform( D3DTS_WORLD, &matWorld );//Se transforma.
	m_vEyePt    = D3DXVECTOR3(y, altura, x); //Donde Está ubicado
	m_vLookatPt = D3DXVECTOR3(ly, altura-1, lx); //Hacia Donde Mira
	m_vUpVec    = D3DXVECTOR3(0.0f, 1.0f, 0.0f); //Vector Normal.
	D3DXMATRIX V;
	D3DXMatrixLookAtLH(&V, &m_vEyePt, &m_vLookatPt, &m_vUpVec);
		//Se calcula una nueva matriz para setear los cambios realizados.
	D3D9->SetTransform(D3DTS_VIEW, &V); //Se transforma nuevamente.
}

INT WINAPI WinMain( HINSTANCE hInst, HINSTANCE, LPSTR, INT )
{
	ShowCursor(false); //Se esconde el Mouse
	WNDCLASSEX wc = {
		sizeof(WNDCLASSEX), CS_CLASSDC, MsgProc, 0L, 0L,
		GetModuleHandle(NULL), NULL, NULL, NULL, NULL,
		"Fumigadora Virtual", NULL };
	RegisterClassEx( &wc );
	HWND hWnd = CreateWindow( " Fumigadora Virtual ", "BandSat DX4 Dyson
	Simulator", WS_OVERLAPPEDWINDOW, 0, 0, 1280, 1024, GetDesktopWindow(),
	NULL, wc.hInstance, NULL ); //Creación de la Ventana
	if( SUCCEEDED( InitD3D( hWnd ) ) ) //Crear e iniciar el objeto Direct3D
	{
		if( SUCCEEDED( InitGeometry() ) ) // Cargar el buffer con los vértices.
		{
			if ((DirectSoundCreate(0, &dSound, NULL)) != DS_OK){
				MessageBox(NULL, "Error en Sonido", 0, 0);
				return(0);
			}// Crear el objeto de Sonido
		}
	}
	while(true){ //Repetir
		if( PeekMessage( &msg, NULL, 0U, 0U, PM_REMOVE ) )
		{ //Se recibe una interrupción.
			TranslateMessage( &msg );
			DispatchMessage( &msg );
		}else{
			Dibujar();
		}
	}
}
```

Sobre la base de datos.

Las bases de datos son gestionadas por un software generado en C++ Builder 6.0. La misma se hizo empleando MS Access y contiene los puntos del recorrido y los datos del usuario.

Al finalizar la fumigación, muestra además una estadística con los porcentajes de zonas solapadas (fumigadas dos veces), porcentaje de chanchos (zonas sin fumigar) y porcentaje de fumigado (aplicación correcta).

Los datos obtenidos sobre los resultados de una fumigación pueden ser impresos.

Sobre la comunicación con los dispositivos externos.

La comunicación con el banderillero es serial bajo el protocolo RS232, debido a que este es el protocolo que acepta el banderillero. El banderillero satelital es uno cualquiera de serie de la marca Dyson, inicialmente se utilizó con *Dyson DX4*[1]

El comando de dirección es externo y es un volante de automóvil (no de video juego) por la robustez requerida y para incrementar la sensación de realidad. Para poder llevar a cabo este requerimiento se diseño una caja que sostiene el volante y se detecta el giro a la derecha o la izquierda con el uso de un mouse.

El acelerador se comanda con el pie y es on/off.

Fig. 1

Fig. 2

Resultados

El tiempo de desarrollo de la creación y puesta a punto del Simulador fue de aproximadamente 6 meses. Su primera aparición al público fue el nueve de Marzo del 2006 en Feriagro en Armstrong provincia de Santa Fe.

En esta y en las siguientes ferias donde fue presentado generó gran atracción, no solo de especialistas sino del público en general, como así también, gracias al realismo logrado sorprendió aún a quienes ya poseía un banderillero desde hacía tiempo. El diario Clarín lo destacó en una nota aparecida el 13 marzo del 2006 bajo el título "Un 'juego' para pulverizar"[2].

Se pudo observar a través de la exposición del simulador que la capacitación del usuario es mucho más rápida, económica y completa que la solamente realizada "in-situ".

---

[1] Este trabajo fue realizado gracias a los aportes tecnológicos y demás recursos de la Empresa *Dyson electrónica* sin la cual hubiera sido imposible desarrollar "La fumigadora Virtual 2.0"

[2] www.clarin.com.ar en el suplemento "Negocios & Mercado". Lunes 13 de marzo. 2006

La empresa que sustento el proyecto quedo conforme sobre todo teniendo en cuenta que es una pequeña empresa nacional que pudo presentarse al mercado con una tecnología que aún no se ve en las ferias de este sector.

Profesionalmente, el autor encontró gran satisfacción y entretenimiento en todo el proceso del sistema. Principalmente porque es un gran logro personal pero también porque fue un diseño construido desde herramientas jamás exploradas por él.

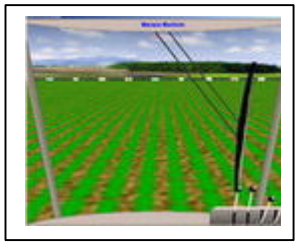

Fig. 3

Fig. 4

Conclusión

El escenario es un campo en tres dimensiones, alambrado, por donde la maquina puede desplazarse siguiendo cualquier trayectoria compatible con una fumigadora. El usuario puede controlar la dirección y la velocidad. Un banderillero satelital, colocado arriba del monitor, va mostrando, como en la realidad, los datos necesarios para llevar la fumigadora por el camino correcto.

Terminado el recorrido es posible visualizar la tarea realizada (mapa), junto con una estadística de los resultados obtenidos: total fumigado, áreas sin fumigar, solapamiento, etc. Estos resultados son guardados en una base de datos, y pueden ser impresos cuando se desee.

La presentación del simulador tuvo gran repercusión en las ferias presentadas y fue de gran utilidad para capacitar a los conductores de las fumigadoras, pues la creación del simulador se pudo utilizar en pequeños espacios mostrando el accionar del banderillero de forma tan realista que los usuarios pudieron comprender y aprender a usar la tecnología sin trasladarse a grandes campos y tener una maquina fumigadora disponible solamente para probar el uso de un banderillero.

Agradecimientos

A:

Dyson Electrónica por brindarme los recursos necesarios para la creación del proyecto.

La Cátedra de Comunicaciones de la UTN-FRC por invitarme a participar.

Mariano Luque, Mariano Ferreyra, Rafael “Bachi” Pradal por la colaboración en la construcción de los dispositivos externos.

Angélica Alvites por la ayuda incondicional en todo el proceso de desarrollo.

Bibliografía consultada.

Ayuda de los programas DirectX, Borland C++ Builder, Visual C++
Cura, Norberto Julián. “Comunicaciones de datos y redes de información” Universitas - Editorial Científica Universitaria. Primera Edición.
Deitel & Deitel. “C++ como programar con UML”. Pearson Cuarta Edición.
Gonzáles, Iván “Programación con DirectX” en http://www.idg.es/pcworld/index.asp?link=estructura/i_articulo_centroArticulo.asp&IdArticulo=61580074. 1999
http://idam.ladei.com.ar/Tutoriales/Direct3D/index.html
http://www.e-dyson.com.ar

Datos de contacto:

Nombre y Apellido: Federico Piedrasanta
Institución: Universidad Tecnológica Nacional. Facultad Regional Córdoba
Dirección postal.5010