INDEPENDENCIA: JUEGO DE ESTRATEGIA EN 3D BASADO EN HECHOS IMPORTANTES DE LA CAMPAÑA LIBERTADORA DE COLOMBIA

NORMAN HERNANDO MUÑOZ FANDIÑO
WILLIAM ALEXANDER RIVERA LÓPEZ

ANEXOS

DIRECTOR: MSC. CARLOS ALBERTO COBOS LOZADA

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE SISTEMAS
LÍNEA DE INTERÉS EN DESARROLLO DE JUEGOS
POPAYÁN, ABRIL DE 2005

INDEPENDENCIA: JUEGO DE ESTRATEGIA EN 3D BASADO EN HECHOS IMPORTANTES DE LA CAMPAÑA LIBERTADORA DE COLOMBIA

ANEXOS

NORMAN HERNANDO MUÑOZ FANDIÑO
WILLIAM ALEXANDER RIVERA LÓPEZ

DIRECTOR: MSC. CARLOS ALBERTO COBOS LOZADA

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE SISTEMAS
LÍNEA DE INTERÉS EN DESARROLLO DE JUEGOS
POPAYÁN, ABRIL DE 2005

TABLA DE CONTENIDO

ANEXO A: DOCUMENTO DE ESTÁNDARES DE PROGRAMACIÓN

Para el desarrollo del proyecto "INDEPENDENCIA: Juego de estrategia en 3D basado en hechos importantes de la campaña libertadora de Colombia", se utilizará el lenguaje de programación *"C++"* y se seguirá el paradigma de *"Programación Orientada a Objetos".*

Las siguientes son los estándares de nomenclatura que se usarán:

1. El idioma para el nombrado de clases, métodos, objetos, variables, etc. es el español con algunas excepciones como la utilización de Get, Set, On.

2. Se hará uso del estándar de nombrado de Microsoft, así:
    a. El nombre de una clase empieza por C, seguido del nombre que se quiera dar, utilizando letras capitales. Ejemplos: CAplicacion, CModeloAnimado.
    b. El nombre de las variables empieza por minúscula y describe el tipo de variable que es, luego el nombre en letras capitales. Ejemplos:

```
HANDLE      hVentana;
BOOL        bBandera;
FLOAT fValor;
BYTE        byPosicion;
int         nContador;
TCHAR szCadena;
LONG *      plPtr;
CClase oClase;
CClase *    pClase;
```

    c. Quedan prohibidas las variables cuyo nombre no diga nada acerca de su utilidad, por ejemplo: i, j, fp.
    d. Cuando una variable sea de ámbito global se antepondrá a su nombre la **g_** así: g_hVentana, g_fValor. Si es una variable miembro de una clase se antepondrá **m_** así: oClase.m_szCadena, oClase.m_nContador.

e. Las funciones miembro de una clase deben comenzar con mayúscula. Si es una función para obtener una variable miembro debe empezar con **Get** seguido del nombre de la variable a obtener. Si es una función para asignar un valor a una variable miembro debe empezar por **Set** seguido del nombre de la variable. Ejemplos

    int GetContador( )

    void SetContador(int nContador)

    void Pintar( )

    void DibujarTriangulo( )

f. Para asignar el valor de una variable float se debe escribir el valor constante seguido por una f, por ejemplo: fValor = 1.0f. Si el valor es un LONG se debe usar la letra L, así: *plPtr = 1.5L.

g. Los eventos se escriben con On seguido del nombre en español del mismo. Ejemplos:

    OnComando( )

    OnClick( )

3. Cada clase estará contenida en dos archivos de código: el archivo de cabecera (CNombre.h) y el de desarrollo de la clase (CNombre.cpp).

Todo archivo de código debe quedar documentado a nivel de archivo, de clase, de método y de ser posible de línea.

1. A nivel de archivo se debe especificar el nombre del archivo, la descripción del archivo (que contiene y su fin general), la fecha de creación, por quien fue creado (incluyendo iniciales del nombre) y fecha de modificaciones con las iniciales de quién o quienes hicieron la modificación además de una la descripción de la misma.

```
/**
 +----------------+-------------------------------------------------------------------+
 | NOMBRE:        | Nombre del archivo                                                |
 | DESCRIPCION:   | Toda la descripcion                                               |
 |                |                                                                   |
 +----------------+-------------------------------------------------------------------+
 | CREADO POR:    | - William Alexander Rivera López.  WR                             |
 |                | - Norman Hernando Muñoz Fandiño.   NM                             |
 +----------------+-------------------------------------------------------------------+
 | FECHAS         |                                                                   |
 | CREACION:      | - DD, Mes de 2004                                                 |
 | MODIFICACIONES:| - DD, Mes de 2004 (Iniciales de quien modifica): Descripción de la |
 |                |   modificación                                                    |
 +----------------+-------------------------------------------------------------------+
 */
```

2. A nivel de clase se debe hacer una breve descripción de la función de la clase.

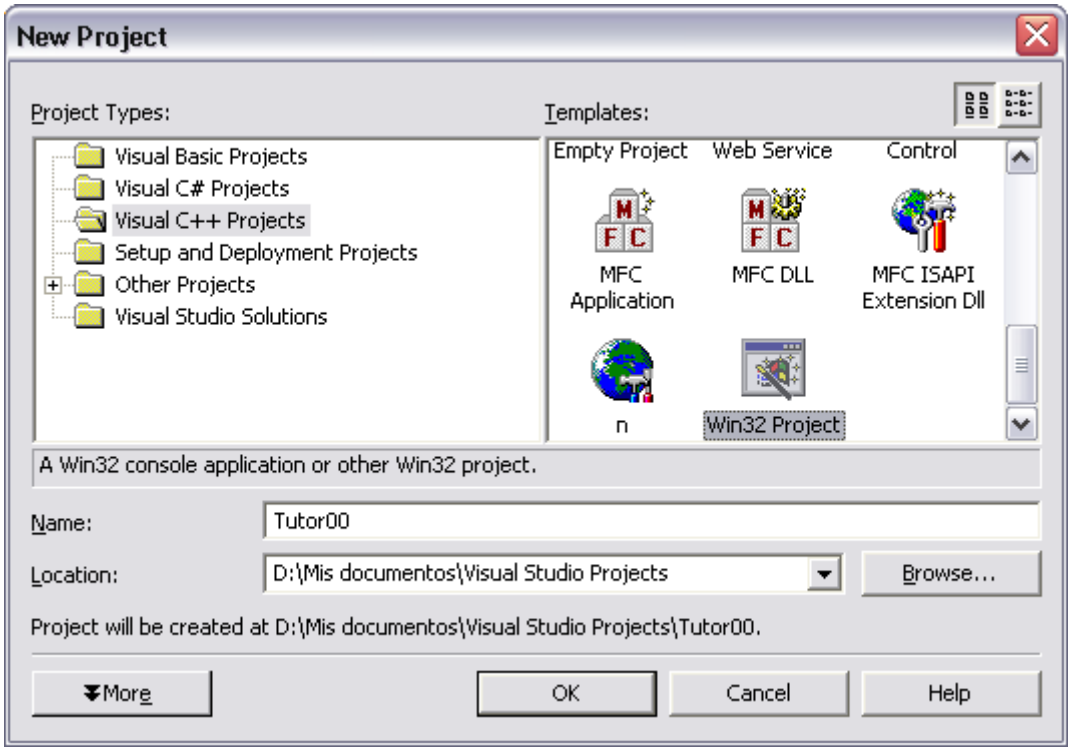
```
/**
 +-----------------+-------------------------------------------------------------------------+
 | NOMBRE:         | Nombre de la clase.                                                     |
 | DESCRIPCION:    | Toda la descripción.                                                    |
 +-----------------+-------------------------------------------------------------------------+
 */
```

3. A nivel de método se especifica el nombre, descripción de lo que hace y los parámetros de entrada y salida.

```
/**
 +-----------------+-------------------------------------------------------------------------+
 | NOMBRE:         | Nombre del método.                                                      |
 | DESCRIPCION:    | Toda la descripción.                                                    |
 +-----------------+-------------------------------------------------------------------------+
 | ENTRADAS:       | - Nombre del parámetro: Descripción.                                    |
 |                 | - Nombre del parámetro: Descripción.                                    |
 +-----------------+-------------------------------------------------------------------------+
 | SALIDA:         | - Tipo de retorno: Descripción.                                         |
 +-----------------+-------------------------------------------------------------------------+
 */
```

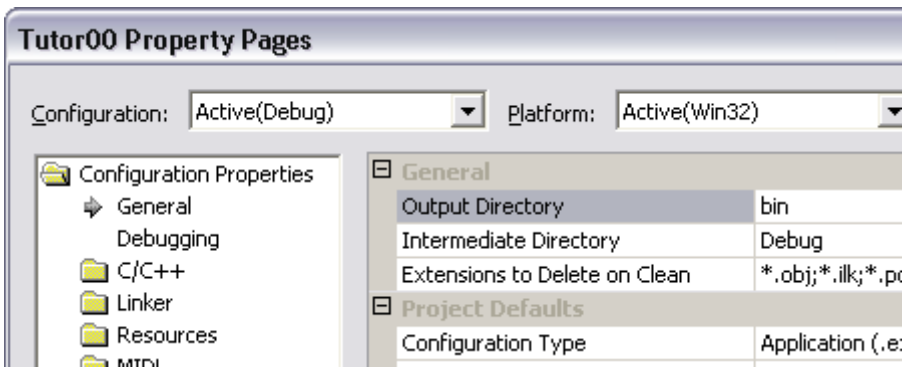
ANEXO B: EJEMPLO DE UTILIZACION DEL MOTOR 3D

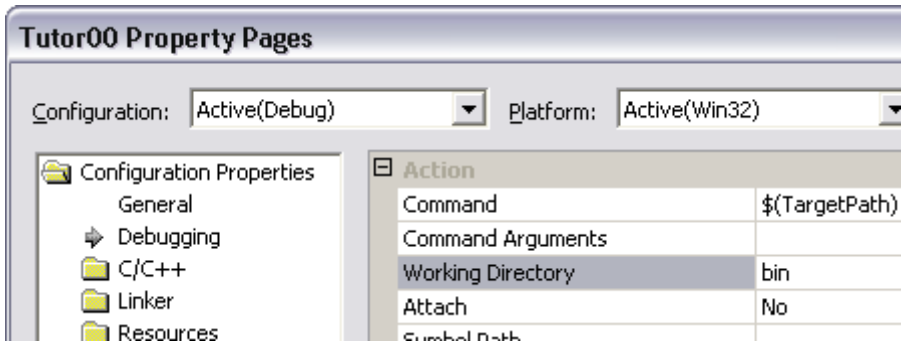
1. Se elige crear un nuevo proyecto vacío de tipo Win32 llamado Tutor00



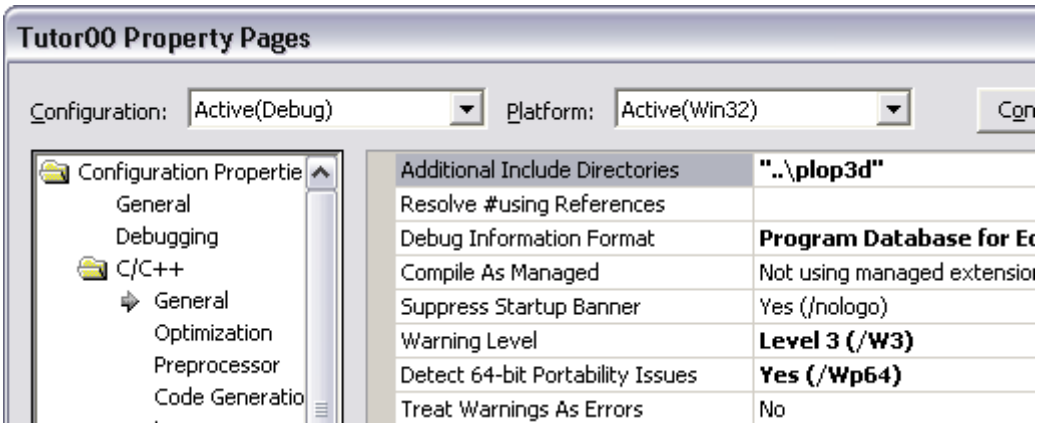
2. Se elige cambiar las propiedades para el proyecto Tutor00.

2. 1. Se cambia en las opciones generales el directorio donde va a quedar el ejecutable a bin, así como el directorio de trabajo definido para el depurado (Debugging).

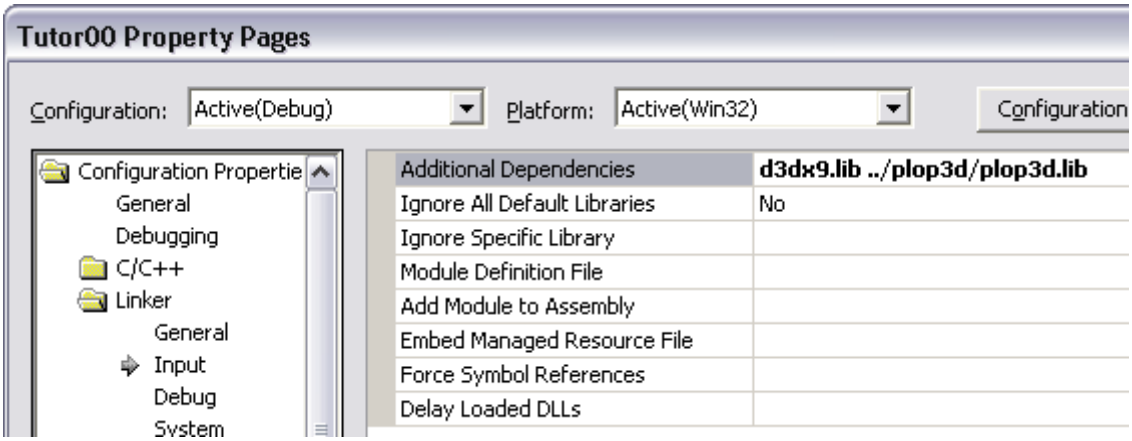




2. 2. Se cambian las propiedades de C/C++ definiendo directorios a incluir adicionales. Aquí se coloca la carpeta donde se encuentran los archivos de cabecera del motor 3d (plop3d)



2. 3. Se definen las librerías necesarias para el link, las cuales incluyen la librería de Direct3D y la del Motor 3D



3. Se crea la carpeta bin dentro de la del Tutor00 y se copia la dll del motor (plop3d.dll) y se verifica que en la carpeta que tiene los archivos de cabecera del motor exista la librería plop3d.lib.

4. Se crea una clase llamada CTutor00 que hereda de la clase CAplicacionD3D, y que va a quedar guardado en los archivos ctutor00.h y ctutor00.cpp. Es necesario incluir el archivo de cabecera del motor mediante la directiva:

```
#include "..\plop3d\plop3d.h"
```

5. El constructor de CTutor00 se debe redeclarar de la siguiente manera:

```
CTutor00( HINSTANCE hInstancia,
          const CTexto & szTexto,
          const CTexto & szNombreClase,
          const CTexto & szArchivoParametros );
```

6. Debido a que CAplicacionD3D es abstracta, es necesario declarar y posteriormente implementar el método público VOID Correr( ).
7. Se adiciona un nuevo ítem (un nuevo archivo cpp) al proyecto de nombre Main.cpp; este va a ser el archivo que contenga el punto de entrada al programa.
8. Se declara dentro de Main.cpp:

```
/* Archivo de cabecera de la clase principal */
#include "ctutor00.h"

INT WINAPI WinMain( HINSTANCE hInstancia,
                    HINSTANCE hInstPrevia,
                    LPSTR szArgumentos,
                    INT nCmdMostrar ) {

    /* Puntero a la clase CTutor00 */
    CTutor00 *    tutor00;

    try {

        /* Intenta crear un nuevo juego */
        tutor00 = new CTutor00( hInstancia,
                                "Tutor 00",
                                "Tutor00 - Motor",
                                NULL );

        /* Pone a correr al juego */
        tutor00->Correr( );

        /* Libera memoria */
        if( tutor00 )
            delete tutor00;
    }
    catch( CExcepcion e ) {

        /* Si hay error lo muestra */
        MessageBox( NULL, e.GetMensaje( ), "Error", MB_ICONERROR );

        /* Sale con error */
        return 1;
    }

    return 0;
}
```

9. De vuelta en la clase CTutor, se define el método Correr( ) así:

```
VOID CTutor00::Correr( ) {
    /* Variables locales */
    MSG               msg;

    /* Variable para determinar si ya se terminó la aplicación */
    BOOL bTermino = FALSE;

    /* Procesa los mensajes */
    while( !bTermino ) {

        /* ¿Hay un mensaje esperando? */
        if( ::PeekMessage( &msg, NULL, 0, 0, PM_REMOVE ) ) {

            /* Si el mensaje es de terminación */
            if( msg.message == WM_QUIT ) {

                /* Termina la aplicación */
                bTermino  = TRUE;
            }
            else {

                /* Traslada mensaje */
                ::TranslateMessage( &msg );

                /* Despacha mensaje */
                ::DispatchMessage( &msg );
            }
        }
        else {
        }
    }
}
```

Si en este momento se ejecuta la aplicación, se puede notar que ya es funcional y muestra una ventana con sus respectivos botones. En términos generales, lo que hace el método correr es preguntar al sistema operativo si hay algún mensaje para la aplicación y de ser así, que se los pase para que los procese. Dentro de esos mensajes que el sistema operativo le envía, puede estar los eventos del teclado, los de ratón, de terminación de la aplicación, entre otros. Toda esta funcionalidad ya se encuentra encapsulada dentro de CAplicacionD3D y lo único que hay que hacer para procesar, por ejemplo un evento de ratón, es sobrecargar el método adecuado.

10. Ahora se regresa al archivo de cabecera de la clase CTutor00 y se le adicionan las variables miembros, funciones privadas y funciones sobrecargadas para que tenga la funcionalidad que se desea.

```
/* Clase de ejemplo que hereda de CAplicacionD3D */
class CTutor00 : public CAplicacionD3D {

    private:
```

```
    /* Enumeración para el movimiento de la camara */
    enum EN_MOVIMIENTO_CAMARA { EMC_ATRAS,
                                EMC_ADELANTE,
                                EMC_IZQUIERDA,
                                EMC_DERECHA,
                                EMC_ARRIBA,
                                EMC_ABAJO };

    /* Puntero para crear el terreno */
    CTerreno *      m_pTerreno;

    /* Puntero para la cámara */
    CCamara *       m_pCamara;

    /* Variable para el manejo del teclado */
    CHAR            m_szTeclado[ 256 ];

    /* Variable para controlar la altura de la cámara */
    FLOAT           m_fAlturaCamara;

    /* Puntero para la creación del modelo */
    CModelo *       m_pModelo;

    /* Puntero del objeto que carga el objeto 3D */
    CEstatico *     m_pEstatico;

    /* Angulo para movimiento del modelo */
    FLOAT           m_fAngulo;

    /* Método privado para fijar las luces */
    VOID FijarLuces( );

    /* Método privado para fijar las matrices de transformación */
    VOID FijarMatrices( );

    /* Método para procesar las entradas de teclado */
    VOID ProcesarTeclado( );

    /* Método para actualizar la posición de la cámara */
    VOID MoverCamara( EN_MOVIMIENTO_CAMARA enMovimiento );

    /* Método para la animación de aeroplano */
    VOID Mover( );

public:

    /* Constructor */
    CTutor00( HINSTANCE hInstancia,
              const CTexto & szTexto,
              const CTexto & szNombreClase,
              const CTexto & szArchivoParametros );

    /* Destructor */
    ~CTutor00( );

    /* Método sobrecargado para correr la aplicación */
```

```
        VOID Correr( );

        /* Se presionó una tecla */
        VOID OnTeclaAbajo( const DWORD & dwIdObjeto, INT nTecla );

        /* Se libera una tecla */
        VOID OnTeclaArriba( const DWORD & dwIdObjeto, INT nTecla );
};
```

Obsérvese que del motor se están utilizando varias clases que ya proveen la funcionalidad necesaria para la creación de juegos, como son las clases para manejo de terrenos (CTerreno), cámara (CCamara) y objetos 3D sin animación (CModelo y CEstatico).

Adicionalmente se han definido métodos que permiten encapsular mucha de la lógica de la aplicación, tal y como es el caso de FijarLuces( ), FijarMatrices( ), ProcesarTeclado( ), MoverCamara ( ) y Mover ( ), siendo los cuatro primeros casi de utilización obligada para cualquier aplicación y no variando mucho entre una y otra aplicación.

Por otro lado, debe notarse que se sobrecargaron dos métodos de la clase CAplicacionD3D y que sirven para el manejo de los eventos de teclado de presionar y liberar una tecla.

11. Ahora de regreso al archivo donde se definen los métodos de la clase CTutor00, se empieza a definir cada uno de los métodos necesarios.

11. 1. En el constructor

```
CTutor00::CTutor00( HINSTANCE hInstancia,
                    const CTexto & szTexto,
                    const CTexto & szNombreClase,
                    const CTexto & szArchivoParametros ) :
    CAplicacionD3D( hInstancia,
                    szTexto,
                    szNombreClase,
                    szArchivoParametros ) {

    /* Crea el terreno */
    m_pTerreno  = new CTerreno( m_pDispositivoD3D,
                                "tutor00.ter",
                                ( CEscuchadorEventos * ) this, TRUE );

    /* Crea la cámara */
    m_pCamara   = new CCamara( CCamara::ETC_CAMARA_PLANO );

    /* Inicia la altura de la cámara */
    m_fAlturaCamara = 120.0f;

    /* Inicia ángulo de animación */
    m_fAngulo       = 0.0f;
```

```
    /* Fja la posición inicial de la cámara */
    m_pCamara->SetPosicion( D3DXVECTOR3( 0.0f, m_pTerreno->GetAltura(
0.0f, 0.0f ) + m_fAlturaCamara, 0.0f ) );
    m_pCamara->Girar( D3DXToRadian( -45.0f ) );
    m_pCamara->Cabecear( ( 58.0f * D3DX_PI ) / 180.0f );

    /* Limpia la variable para el teclado */
    ::ZeroMemory( m_szTeclado, 256 );

    /* Crea el objeto 3D */
    m_pEstatico  = new CEstatico( m_pDispositivoD3D );

    /* Carga desde el archivo .x */
    m_pEstatico->Cargar( "Modelos/aeroplano.x" );

    /* Separa memoria para el modelo */
    m_pModelo   = new CModelo( m_pDispositivoD3D, D3DXVECTOR3( -25.0f,
m_pTerreno->GetAltura( 0.0f, 0.0f ) + m_fAlturaCamara / 2.0f, 20.0f )
);

    /* Fija el objeto 3D al modelo */
    m_pModelo->SetEstatico( m_pEstatico );

    /* Rota el modelo */
    m_pModelo->SetRotacion( D3DXVECTOR3( 0.0f, 135.0f, 0.0f ) );

}
```

En el constructor en primera instancia se crea y se carga el terreno que ha sido creado previamente (puede hacer uso de la utilidad para la edición de terrenos que se incluye en el juego independencia). Una vez creado el terreno se copia el archivo generado (*.ter) en la carpeta bin, al igual que las carpetas Modelos y Texturas/Losas.

Posteriormente se crea la cámara que va a definir desde que posición se va a observar la acción. Para el caso, se fija en la posición (0, *y*, 0), donde *y* se debe hallar a partir de la altura que tenga en ese momento el terreno en la posición 0 en x y 0 en z.

Una vez creada la cámara se carga el mesh (archivo en formato de DirectX *.x) mediante la utilización de un objeto CEstatico. Se usa este tipo de objeto, porque se sabe que el objeto 3D que se cargo para el ejemplo no poseía animaciones. Una vez cargado el objeto 3D, se procede a crear el modelo (CModelo). El modelo separa la lógica de la presentación, por lo cual fácilmente se podrían crear más modelos (por ejemplo mediante un array) y fijar únicamente el mismo objeto 3D.

11. 2. De regreso al método correr

```
VOID CTutor00::Correr( ) {
    /* Variables locales */
    MSG               msg;

    /* Variable para determinar si ya se terminó la aplicación */
    BOOL bTermino = FALSE;

    /* Procesa los mensajes */
    while( !bTermino ) {

        /* ¿Hay un mensaje esperando? */
        if( ::PeekMessage( &msg, NULL, 0, 0, PM_REMOVE ) ) {

            /* Si el mensaje es de terminación */
            if( msg.message == WM_QUIT ) {

                /* Termina la aplicación */
                bTermino  = TRUE;
            }
            else {

                /* Traslada mensaje */
                ::TranslateMessage( &msg );

                /* Despacha mensaje */
                ::DispatchMessage( &msg );
            }
        }
        else {

            /* Limpia buffer de salida */
            m_pDispositivoD3D->Clear( 0, 0, D3DCLEAR_TARGET |
D3DCLEAR_ZBUFFER, 0x00000000, 1.0f, 0 );

            /* Inicializa escena */
            m_pDispositivoD3D->BeginScene( );

            /* Anima el aeroplano */
            Mover( );

            /* Actualiza entradas de teclado */
            ProcesarTeclado( );

            /* Fija las luces */
            FijarLuces( );

            /* Fija las matrices */
            FijarMatrices( );

            /* Si hay terreno */
            if( m_pTerreno )
                m_pTerreno->Pintar( );

            /* Pinta el modelo */
            if( m_pModelo )
                m_pModelo->Pintar(  );
```

```
            /* Fin de la escena */
            m_pDispositivoD3D->EndScene( );

            /* Cambia al buffer primario */
            m_pDispositivoD3D->Present( NULL, NULL, NULL, NULL );
        }
    }
}
```

En este método se adicionó toda la parte que corresponde al procesamiento de la aplicación Direct3D, es decir, la parte de procesamiento del juego.

En primera instancia, se limpia el buffer de salida, luego se inicia la escena, después se realiza el procesamiento de acciones propias del juego, para pasar a pintar la escena, posteriormente se finaliza la escena y se cambia todo lo que se dibujó en el buffer secundario al primario. Este es el esquema básico y de esa manera se controla la acción del juego.

Es de destacar que entre el inicio y fin de la escena está la parte correspondiente a la lógica de la aplicación, que para el caso incluye animar al modelo (mediante Mover()), actualizar las entradas de teclado, fijar luces y matrices de transformación y pintar el terreno y el modelo.

11. 3. Fijar las luces

```
VOID CTutor00::FijarLuces( ) {

    /* Variables auxiliares */
    D3DMATERIAL9    stMaterial;
    D3DXVECTOR3     vecDir;
    D3DLIGHT9       light;

    /* Prepara el material */
    ZeroMemory( & stMaterial, sizeof(D3DMATERIAL9) );
    stMaterial.Diffuse    = D3DXCOLOR( 1.0f, 1.0f, 1.0f, 1.0f )* 0.8f;
    stMaterial.Ambient    = D3DXCOLOR( 1.0f, 1.0f, 1.0f, 1.0f );

    /* Fija el material */
    m_pDispositivoD3D->SetMaterial( & stMaterial );

    /* Crea una luz direccional para el terreno */
    ZeroMemory( &light, sizeof(D3DLIGHT9) );
    light.Type        = D3DLIGHT_DIRECTIONAL;
    light.Diffuse     = D3DXCOLOR( 1.0f, 1.0f, 1.0f, 0.0f );
    light.Range       = 1000.0f;
    vecDir            = D3DXVECTOR3( -1.0f, -3.0f, 2.0f );
    D3DXVec3Normalize( (D3DXVECTOR3*)&light.Direction, &vecDir );

    /* Fija la luz 0 */
    m_pDispositivoD3D->SetLight( 0, &light );
```

```
    /* Habilita la utilización de luces */
    m_pDispositivoD3D->SetRenderState( D3DRS_LIGHTING, TRUE );

    /* Activa la luz de ambiente */
    m_pDispositivoD3D->SetRenderState( D3DRS_AMBIENT, 0x00323232 );

    /* Habilita solo la luz para el terreno */
    m_pDispositivoD3D->LightEnable( 0, TRUE );
}
```

11. 4. Fijar las matrices

```
VOID CTutor00::FijarMatrices( ) {

    /* Matrices para las transformaciones */
    D3DXMATRIX stMundo, stVista, stProyeccion;

    /* Crea una matriz identidad para la transformación del mundo */
    D3DXMatrixIdentity( &stMundo );
    m_pDispositivoD3D->SetTransform( D3DTS_WORLD, &stMundo );

    /* Obtiene la matriz de vista desde la cámara */
    stVista = m_pCamara->GetMatrizVista( );
    m_pDispositivoD3D->SetTransform( D3DTS_VIEW, &stVista );

    /* Construye la matriz de proyeccion */
    D3DXMatrixPerspectiveFovLH( &stProyeccion, D3DX_PI/4, 1.0f, 1.0f,
1000.0f );
    m_pDispositivoD3D->SetTransform( D3DTS_PROJECTION, &stProyeccion );
}
```

11. 5. Procesar las entradas de teclado

```
VOID CTutor00::ProcesarTeclado( ) {

    /* Posicion de la camara */
    D3DXVECTOR3 stPosicion = m_pCamara->GetPosicion( );

    /* Mover la camara hacia adelante */
    if( m_szTeclado[ VK_UP ] ) {
        MoverCamara( EMC_ADELANTE );
    }

    /* Mover la camara hacia atras */
    if( m_szTeclado[ VK_DOWN ] ) {
        MoverCamara( EMC_ATRAS );
    }

    /* Mover la camara hacia la izquierda */
    if( m_szTeclado[ VK_LEFT ] ) {
        MoverCamara( EMC_IZQUIERDA );
    }

    /* Mover la camara hacia la derecha */
    if( m_szTeclado[ VK_RIGHT ] ) {
        MoverCamara( EMC_DERECHA );
    }

    /* Elevar la camara del terreno */
    if( m_szTeclado[ 'X' ] ) {
```

```
        MoverCamara( EMC_ARRIBA );
    }

    /* Baja la camara hacia el terreno */
    if( m_szTeclado[ 'Z' ] ) {
        MoverCamara( EMC_ABAJO );
    }

    /* Escape */
    if( m_szTeclado[ VK_ESCAPE ] ) {

        /* Termina la aplicación */
        ::SendMessage( m_pVentanaPrincipal->GetManejador( ), WM_CLOSE,
0, 0 );

    }

    /* Toma de nuevo la posicion de la camara */
    stPosicion = m_pCamara->GetPosicion( );

    /* Actualiza la posicion en Y */
    stPosicion.y = m_pTerreno->GetAltura( stPosicion.x, stPosicion.z )
+ m_fAlturaCamara;

    /* Fija la posicion actualizada */
    m_pCamara->SetPosicion( stPosicion );
}
```

11. 6. Mover la cámara

```
VOID CTutor00::MoverCamara( EN_MOVIMIENTO_CAMARA enMovimiento ) {

    /* Mover la camara hacia adelante */
    if( enMovimiento == EMC_ADELANTE ) {

        /* Si la camara esta aún dentro del terreno */
        if( m_pCamara->GetPosicion( ).z <= ( m_pTerreno-
>GetProfundidad( ) / 2 ) - (m_pTerreno->GetEspacioLosas( ) * 5) ) {

            /* Mueve la camara hacia Z */
            m_pCamara->Caminar( 3.0f );
        }
    }

    /* Mover la camara hacia atras */
    if( enMovimiento == EMC_ATRAS ) {

        /* Si la camara esta aún dentro del limite inferior */
        if( m_pCamara->GetPosicion( ).z >= ( - m_pTerreno-
>GetProfundidad( ) / 2 ) - (m_pTerreno->GetEspacioLosas( ) * 5) ) {

            /* Mueve la camara en contra de Z */
            m pCamara->Caminar( -3.0f );
        }
    }

    /* Mover la camara hacia la izquierda */
    if( enMovimiento == EMC_IZQUIERDA ) {
```

```
        /* Si la camara esta aún dentro del terreno */
        if( m_pCamara->GetPosicion( ).x >= ( - m_pTerreno->GetAncho( )
/ 2 ) + 2.0 ) {

            /* Mueve la camara en contra de Z */
            m_pCamara->Lateralizar( -3.0f );
        }
    }

    /* Mover la camara hacia la derecha */
    if( enMovimiento == EMC_DERECHA ) {

        /* Si la camara esta aún dentro del terreno */
        if( m_pCamara->GetPosicion( ).x <= ( m_pTerreno->GetAncho( ) /
2 ) - 2.0 ) {

            /* Mueve la camara en contra de Z */
            m_pCamara->Lateralizar( 3.0f );
        }
    }

    /* Elevar la camara del terreno */
    if( enMovimiento == EMC_ARRIBA ) {

        /* Si la cámara no ha exedido la altura máxima */
        if( m_fAlturaCamara <= 78.0f ) {
            m_pCamara->Volar( 2.0f );
            m_pCamara->Caminar( -1.0f );
            m_pCamara->Cabecear( 0.016f );
            m_fAlturaCamara = m_pCamara->GetPosicion( ).y - m_pTerreno-
>GetAltura( m_pCamara->GetPosicion( ).x, m_pCamara->GetPosicion( ).z );
        }
    }

    /* Baja la camara hacia el terreno */
    if( enMovimiento == EMC_ABAJO ) {

        /* Si la cámara no ha exedido la altura mínima */
        if( m_fAlturaCamara >= 14.0f ) {
            m_pCamara->Volar( -2.0f );
            m_pCamara->Caminar( 1.0f );
            m_pCamara->Cabecear( -0.016f );
            m_fAlturaCamara = m_pCamara->GetPosicion( ).y - m_pTerreno-
>GetAltura( m_pCamara->GetPosicion( ).x, m_pCamara->GetPosicion( ).z );
        }
    }
}
```

11. 7. Procesar eventos de teclado

```
VOID CTutor00::OnTeclaAbajo( const DWORD & dwIdObjeto, INT nTecla ) {

    /* Se presionó una tecla */
    m_szTeclado[ nTecla ] = TRUE;
}

VOID CTutor00::OnTeclaArriba( const DWORD & dwIdObjeto, INT nTecla ) {
```

```
    /* Se liberó una tecla */
    m_szTeclado[ nTecla ] = FALSE;
}
```

11. 8. Animar el modelo

```
VOID CTutor00::Mover( ) {

    /* Variables auxiliares */
    D3DXVECTOR3     vPosicion;
    D3DXVECTOR3     vRotacion;
    FLOAT           fRadio;

    /* Si se cargó el modelo */
    if( m_pModelo ) {

        /* Radio de la circunferencia */
        fRadio      = 30.0f;

        /* Posición en z, según el radio y el ángulo */
        vPosicion.z = fRadio * cos( m_fAngulo );

        /* Posición en x, según el radio y el ángulo */
        vPosicion.x = fRadio * sin( m_fAngulo );

        /* Posición en y, según la posición en x,z */
        vPosicion.y = m_pTerreno->GetAltura( vPosicion.x, vPosicion.z )
+ m_fAlturaCamara / 2;

        /* Actualiza el ángulo */
        m_fAngulo   = ( m_fAngulo <= 360.0f )? m_fAngulo - 0.01f :
0.0f;

        /* Fija la posición para el modelo */
        m_pModelo->SetPosicion( vPosicion );

        /* Obtiene la rotación actual del modelo */
        vRotacion   = m_pModelo->GetRotacion( );

        /* Actualiza la rotación */
        vRotacion.x += m_fAngulo;
        m_pModelo->SetRotacion( vRotacion );
    }
}
```

Una vez terminada la codificación anterior, se procede a ejecutar la aplicación.

De esta manera se nota que la creación de aplicaciones se simplifica con la utilización de las clases que ofrece el Motor 3D, ya que encapsula mucha de la funcionalidad necesaria para la creación de juegos, sólo basta un poco de imaginación para poder crear tu propio juego.

ANEXO C: MANUAL DE REFERENCIA DEL MOTOR

- **CAlojarJerarquia**

Clase que permite alojar toda la jerarquía para modelos 3D animados

CAlojarJerarquia - Atributos

Atributo	Tipo	Descripción
m_pAnimado	protected : *CAnimado*	Puntero al modelo

CAlojarJerarquia - Métodos

Método	Tipo	Descripción
MemoriaNombre (*LPCTSTR, LPTSTR**)	protected: *HRESULT*	Separa memoria para un nombre
CAlojarJerarquia (*CAnimado**)	public:	Constructor
CreateFrame (*THIS_, LPD3DXFRAME**)	public:	Crea el frame
CreateMeshContainer (*THIS_, LPD3DXMESHDATA, LPD3DXMATERIAL, LPD3DXEFFECTINSTANCE, DWORD, DWORD*, LPD3DXSKININFO, LPD3DXMESHCONTAINER**)	public:	Crea el contenedor del mesh
DestroyFrame (*THIS_*)	public:	Destruye el frame
DestroyMeshContainer (*THIS_*)	public:	Destruye el contenedor

- **CAnimado**

Clase que permite la creación y manipulación de modelos 3D animados (skin meshes)

CAnimado - Atributos

Atributo	Tipo	Descripción
m_enMetodo	protected : *EN_METODO_SKIN*	Define el tipo de método de skinning
m_pFrameRaiz	protected : *LPD3DXFRAME*	Puntero al frame raiz
m_pControlAnimacion	protected : *LPD3DXANIMATIONCONTROLLER*	Puntero al control de animación
m_pMatrizHuesos	protected :	Puntero para la matriz de huesos

	D3DXMATRIXA16	
m_nNumMatrizHuesos	protected : *UINT*	Número de matrices de hueso

CAnimado - Métodos

Método	Tipo	Descripción
IniciarPunterosMatrizHuesos (*LPD3DXFRAME*)	protected: *HRESULT*	Inicializa los punteros del frame a la matriz de huesos
IniciarPunterosMatrizHuesosEnMesh (*LPD3DXMESHCONTAINER*)	protected: *HRESULT*	Inicializa los punteros del mesh a la matriz de hueso
DibujarContenedor (*LPD3DXMESHCONTAINER, LPD3DXFRAME*)	protected: *VOID*	Dibuja cada contenedor dependiendo del modo de skinning
DibujarFrame (*LPD3DXFRAME*)	protected: *VOID*	Dibuja cada frame de manera recursiva, llamando a cada contenedor
CAnimado (*LPDIRECT3DDEVICE9*)	public:	Constructor de inicialización
~CAnimado ()	public:	Destructor
GetMetodo ()	public const query: *EN_METODO_SKIN&*	Obtiene el método de skining
GetFrameRaiz ()	public: *LPD3DXFRAME*	Puntero al frame raiz
GetControlAnimacion ()	public: *LPD3DXANIMATIONCONTROLLER*	Puntero al control de animación
GetMatrizHuesos ()	public query: *D3DXMATRIXA16**	Puntero para la matriz de huesos
GetNumMatrizHuesos ()	public const query: *UINT&*	Número de matrices de hueso
SetMetodo (*EN_METODO_SKIN&*)	public: *VOID*	Define el tipo de método de skinning
SetControlAnimacion (*LPD3DXANIMATIONCONTROLLER*)	public: *VOID*	Fija un nuevo control de animación
Generar (*CContenedor**)	public: *VOID*	Genera el Modelo 3D
Pintar ()	public: *VOID*	Método para pintar el modelo 3D
Cargar (*CTexto&*)	public: *VOID*	Carga el modelo
ActualizarMatrices (*LPD3DXFRAME, LPD3DXMATRIX*)	public: *VOID*	Actualiza las matrices del frame

- **CAnimado::EN_METODO_SKIN**

Enumera los distintos métodos de skinning

CAnimado::EN_METODO_SKIN - Atributos

Atributo	Tipo	Descripción
EMS_D3DNONINDEXED	public : *int*	No indexado
EMS_D3DINDEXED	public : *int*	Indexado
EMS_SOFTWARE	public : *int*	Por software
EMS_D3DINDEXEDVS	public : *int*	
EMS_D3DINDEXEDHLSLVS	public : *int*	
EMS_NONE	public : *int*	Ninguno

- **CAplicacion**

Esta clase se encarga de registrar la clase de Windows de la aplicación, así como de su finalización.

CAplicacion - Atributos

Atributo	Tipo	Descripción
m_pVentanaPrincipal	protected static : *CVentanaPrincipal*	Puntero a la ventana principal
m_hInstancia	protected : *HINSTANCE*	Manejador de la instancia
m_szNombre	protected : *CTexto*	Nombre de la clase Windows

CAplicacion - Métodos

Método	Tipo	Descripción
ProcedimientoVentana (*HWND, UINT, WPARAM, LPARAM*)	private static: *LRESULT*	Procedimiento para el manejo de mensajes
CAplicacion (*HINSTANCE, CTexto&*)	public:	Constructor de inicialización de CAplicacion
~CAplicacion ()	public:	Este método se encarga de liberar los recursos de la clase.
GetInstancia ()	public: *HINSTANCE*	Obtiene el manejador de la instancia
GetNombre ()	public const: *CTexto*	Obtiene el nombre de la clase de windows
GetVentanaPrincipal ()	public const: *CVentanaPrincipal**	Puntero a la ventana principal
SetVentanaPrincipal (*CVentanaPrincipal**)	public: *VOID*	Método para fijar la Ventana principal
Registrar ()	public: *VOID*	Registra la clase de windows.
Correr ()	«pure» public abstract: *VOID*	Espera por mensajes de windows

- **CAplicacionD3D**

Clase para el manejo de una aplicación con soporte para Direct3D.

CAplicacionD3D - Atributos

Atributo	Tipo	Descripción
m_pDirect3D	protected : *CDirect3D*	Puntero a un objeto para el manejo del Direct3D

CAplicacionD3D - Métodos

Método	Tipo	Descripción
CAplicacionD3D (*HINSTANCE, CTexto&, CTexto&, CTexto&*)	public:	Constructor de inicialización de la clase CAplicacionD3D.
~CAplicacionD3D ()	public:	Destructor de la clase CAplicacionD3D.
OnActivada (*DWORD&*)	public abstract: *VOID*	Cuando se activa la ventana
OnDesactivada (*DWORD&*)	public abstract: *VOID*	Cuando se desactiva la ventana
OnCreada (*DWORD&*)	public abstract: *VOID*	Cuando se crea la ventana
OnCerrar (*DWORD&*)	public abstract: *VOID*	Llama al evento que maneja este mensaje
OnComando (*DWORD&, WPARAM&, LPARAM&*)	public abstract: *VOID*	Cuando se da un evento de comando
OnRatonAbajo (*DWORD&, SHORT, WPARAM, DWORD, DWORD*)	public abstract: *VOID*	Se presionó un botón del ratón
OnRatonArriba (*DWORD&, SHORT, WPARAM, DWORD, DWORD*)	public abstract: *VOID*	Se liberó un botón del ratón
OnRatonMueve (*DWORD&, WPARAM, DWORD, DWORD*)	public abstract: *VOID*	Se movió el puntero del ratón
OnClick (*DWORD&, SHORT, WPARAM, DWORD, DWORD*)	public abstract: *VOID*	Se hizo click sobre un objeto
OnRuedaRatonGira (*DWORD&, WPARAM, DWORD, DWORD*)	public abstract: *VOID*	Se giró la rueda del mouse
OnTeclaAbajo (*DWORD&, INT*)	public abstract: *VOID*	Se presionó una tecla
OnTeclaArriba (*DWORD&, INT*)	public abstract: *VOID*	Se libera una tecla
OnCambio (*DWORD&, CHAR&*)	public abstract: *VOID*	Se presiona una tecla pero con un valor ASCII
LeerParametrosPresentacion (*CTexto&*)	public: *D3DPRESENT_PARAMETERS*	Lee los parámetros de configuración de la aplicación para la creación del dispositivo Direct3D.
TiempoElapsado ()	public: *DOUBLE*	Retorna el tiempo elapsado (transcurrido)
TicksPorSegundo ()	public: *LONGLONG*	Retorna los ticks por segundo para el procesador
IniciarMaterial (*D3DXCOLOR, D3DXCOLOR, D3DXCOLOR, D3DXCOLOR, FLOAT*)	public: *D3DMATERIAL9*	Permite inicializar un material
IniciarLuz	public:	Permite inicializar una luz

(*D3DXVECTOR3*, D3DXVECTOR3*, D3DXCOLOR*, D3DLIGHTTIPO*)	*D3DLIGHT9*	

- **CBoton**

Declaración de una clase que permite la creación y manipulación de botones 2D

CBoton - Métodos

Método	Tipo	Descripción
CBoton (*LPDIRECT3DDEVICE9, RECT, CEscuchadorEventos*, CTexto&*)	public:	Constructor de inicialización
CBoton (*LPDIRECT3DDEVICE9, CTexto&*)	public:	Constructor de inicialización
~CBoton ()	public:	Destructor
Pintar ()	public: *VOID*	Pinta el objeto
operator = (*CBoton&*)	public const: *CBoton&*	Sobrecarga del operador de asignación
operator == (*CBoton&*)	public: *BOOL*	Sobrecarga del operador de comparación
operator != (*CBoton&*)	public: *BOOL*	Sobrecarga del operador de comparación

- **CBoton::EN_ESTADO_BOTON**

Enumeracion de los estados del botón

CBoton::EN_ESTADO_BOTON - Atributos

Atributo	Tipo	Descripción
EEB_LIBRE	public : *int*	Libre
EEB_ARRIBA	public : *int*	Arriba
EEB_ABAJO	public : *int*	Abajo
EEB_CLICK	public : *int*	Click

- **CBotonImagen**

Clase que permite crear botones con imágenes como estado.

CBotonImagen - Atributos

Atributo	Tipo	Descripción
m_pTexturas	private : *LPDIRECT3DTEXTURE9*	Puntero a las texturas con los 3 estados

CBotonImagen - Métodos

Método	Tipo	Descripción
CBotonImagen (*LPDIRECT3DDEVICE9, RECT, CEscuchadorEventos**)	public:	Constructor de inicialización
CBotonImagen (*LPDIRECT3DDEVICE9*)	public:	Constructor de inicialización
~CBotonImagen ()	public:	Destructor
GetTexturas ()	public query: *LPDIRECT3DTEXTURE9**	Retorna el puntero a las texturas con los 3 estados
Pintar ()	public: *VOID*	Pinta el objeto
CargarTextura (*CTexto&, DWORD&*)	public: *VOID*	Carga una imagen para un estado del boton

- **CCajaTexto**

Declaración de una clase que permite crear una caja de texto para mostrar en coordenadas 3D, pero a partir de coordenadas 2D

CCajaTexto - Atributos

Atributo	Tipo	Descripción
m_dwTiempoMostrarCursor	private : *DWORD*	Bandera para controlar cuando hay que mostrar el cursor titilante del cuadro
m_bMostrarCursor	private : *BOOL*	Bandera para controlar cuando hay que mostrar el cursor titilante del cuadro
m_nMaximoTexto	private : *UINT*	Guarda cual es el máximo de texto permitido

CCajaTexto - Métodos

Método	Tipo	Descripción
AdicionarCaracter (*INT*)	private: *VOID*	Adiciona un caracter al final de la cadena de texto
QuitarUltimoCaracter ()	private: *VOID*	Elimina el último caracter
MostrarCursor ()	private: *VOID*	Muestra el cursor titilante
OcultarCursor ()	private: *VOID*	Oculta el cursor
CCajaTexto (*LPDIRECT3DDEVICE9, RECT, CEscuchadorEventos*, CTexto&*)	public:	Constructor de inicialización
CCajaTexto (*LPDIRECT3DDEVICE9, CTexto&*)	public:	Constructor de inicialización
~CCajaTexto ()	public:	Destructor
GetMaximoTexto ()	public: *UINT*	Guarda cual es el máximo de texto permitido
GetTexto ()	public const: *CTexto&*	Devuelve el puntero al texto del objeto ocultando el cursor
SetMaximoTexto (*UINT*)	public: *VOID*	Asigna cual es el máximo de texto permitido
Pintar ()	public: *VOID*	Pinta el objeto 2D
IsTeclaAbajo (*INT*)	public: *BOOL*	Se presionó una tecla

- **CCamara**

Define una clase para el manejo de una cámara en el espacio 3D.

CCamara - Atributos

Atributo	Tipo	Descripción
m_enTipo	private : *EN_TIPO_CAMARA*	Tipo de la cámara
m_stPosicion	private : *D3DXVECTOR3*	Vector de Posición de la cámara
m_stDerecha	private : *D3DXVECTOR3*	Vector de la Derecha de la cámara
m_stArriba	private : *D3DXVECTOR3*	Vector de Arriba de la cámara
m_stFrente	private : *D3DXVECTOR3*	Vector de Vista de la cámara

CCamara - Métodos

Método	Tipo	Descripción
CCamara (*EN_TIPO_CAMARA*)	public:	Constructor por defecto y de inicialización de la clase
~CCamara ()	public:	Destructor de la clase CCamara.
GetTipo ()	public const query: *EN_TIPO_CAMARA*	Obtiene el tipo de la cámara
GetPosicion ()	public const query: *D3DXVECTOR3*	Retorna el vector de posición
GetDerecha ()	public const query: *D3DXVECTOR3*	Retorna el vector de la derecha
GetArriba ()	public const query: *D3DXVECTOR3*	Retorna el vector de arriba
GetFrente ()	public const query: *D3DXVECTOR3*	Retorna el vector de en frente
GetMatrizVista ()	public const: *D3DXMATRIX*	Crea una matriz de vista para la camara en el espacio 3D.
SetTipo (*EN_TIPO_CAMARA*)	public: *VOID*	Fija el tipo de la camara
SetPosicion (*D3DXVECTOR3*)	public: *VOID*	Fija el vector de Posicion
Caminar (*FLOAT*)	public: *VOID*	Mueve la camara fUnidades hacia adelante (+) o hacia atras.
Volar (*FLOAT*)	public: *VOID*	Mueve la camara fUnidades hacia arriba (+) o hacia abajo
Lateralizar (*FLOAT*)	public: *VOID*	Mueve la camara fUnidades hacia la derecha (+) o hacia la izquierda
Cabecear (*FLOAT*)	public: *VOID*	Rota la camara fAngulo radianes sobre el eje de la derecha.
Girar (*FLOAT*)	public: *VOID*	Rota la camara fAngulo radianes sobre el eje de arriba
Rodar (*FLOAT*)	public: *VOID*	Rota la camara fAngulo radianes sobre el eje del frente

- **CCamara::EN_TIPO_CAMARA**

Enumeracion para el tipo de Camara

CCamara::EN_TIPO_CAMARA - Atributos

Atributo	Tipo	Descripción
ETC_CAMARA_PLANO	public : *int*	Camara para el plano
ETC_CAMARA_AIRE	public : *int*	Cámara tipo aire

- **CContenedor**

Estructura derivada de D3DXMESHCONTAINER ayuda a adicionar algunas cosas especificas para la carga de un mesh

CContenedor - Atributos

Atributo	Tipo	Descripción
m_ppTexturas	public : *LPDIRECT3DTEXTURE9*	Puntero a las texturas
m_pMeshOrigen	public : *LPD3DXMESH*	Mesh de origen
m_pTablaAtributo	public : *LPD3DXATRIBUTORANGE*	Puntero a la tabla de atributos
m_dwNumGruposAtributos	public : *DWORD*	Numero de grupos de atributos
m_dwNumInfl	public : *DWORD*	Numero de infl
m_pBufferCombinacionHuesos	public : *LPD3DXBUFFER*	Puntero al buffer de huesos combinados
m_ppMatrizHuesos	public : *D3DXMATRIX**	Puntero al puntero de la matriz de huesos
m_pOffsetMatrizHuesos	public : *D3DXMATRIX*	Puntero al corrimientos de la matriz de huesos
m_NumEntradasPaleta	public : *DWORD*	Número de entradas de la paleta
m_UsaSoftwareVP	public : *BOOL*	Indica si usa procesamiento de vertices por software
m_dwAtributoSW	public : *DWORD*	usuado para denotar el cambio entre SW y HW si es necesario para skinning no indexado

- **CControl**

Clase base para la creación de objetos 2D dentro de D3D

CControl - Atributos

Atributo	Tipo	Descripción
m_pVertices	protected : *LPDIRECT3DVERTEXBUFFER9*	Puntero al buffer de vertices
m_stRectangulo	protected : *RECT*	Rectángulo que define la posición y las dimensiones del objeto
m_stColor	protected :	Color del objeto

	D3DXCOLOR	
m_bTransparente	protected : *BOOL*	El objeto tiene color de fondo o no
m_bVisible	protected : *BOOL*	Dice si el objeto esta visible o no
m_stCentro	protected : *POINT*	Posicion en (x,y) que define el centro del objeto
m_nAncho	protected : *INT*	Ancho del objeto en pixels
m_nAlto	protected : *INT*	Alto del objeto en pixels
m_fRotacionEjes	protected : *FLOAT*	Angulo de rotacion de los ejes sobre el ejex X+
m_stSeleccion	protected : *POINT*	Punto que guarda la información del píxel sobre el que se hizo clic

CControl - Métodos

Método	**Tipo**	**Descripción**
IsDentroDelObjeto (*DWORD, DWORD*)	protected: *BOOL*	Determina si un punto (x,y) está dentro de las coordenadas del objeto
CControl ()	public:	Constructor por defecto
CControl (*LPDIRECT3DDEVICE9, RECT, CEscuchadorEventos**)	public:	Constructor de inicialización
CControl (*LPDIRECT3DDEVICE9, POINT, INT, INT, FLOAT, CEscuchadorEventos**)	public:	Constructor de inicialización
~CControl ()	public:	Destructor de la clase
GetVertices ()	public: *LPDIRECT3DVERTEXBUFFER9*	Retorna el puntero al buffer de vértices
GetRectangulo ()	public: *RECT*	Devuelve el rectángulo que define la posición y las dimensiones del objeto
GetX ()	public: *DWORD*	Devuelve la posición en x del objeto
GetY ()	public: *DWORD*	Devuelve la posición en y del objeto
GetAncho ()	public: *INT*	Retorna el ancho del objeto
GetAlto ()	public: *INT*	Retorna el alto del objeto
GetColor ()	public: *D3DXCOLOR*	Retorna el color del objeto
IsVisible ()	public: *BOOL*	Dice si el objeto esta visible o no
IsTransparente ()	public: *BOOL*	Retorna un boolean que dice si el objeto es transparente
GetSeleccion ()	public: *POINT*	Punto que guarda la información del píxel sobre el que se hizo clic
SetVertices (*LPDIRECT3DVERTEXBUFFER9*)	public: *VOID*	Asigna el puntero al buffer de vértices
SetRectangulo (*RECT*)	public: *VOID*	Asigna el rectángulo que define la posición y las dimensiones del objeto
SetX (*DWORD*)	public: *VOID*	Asigna la posición en x del objeto
SetY (*DWORD*)	public: *VOID*	Asigna la posición en y del objeto
SetAncho (*DWORD*)	public: *VOID*	Asigna el ancho del objeto
SetAlto (*DWORD*)	public: *VOID*	Asigna el alto del objeto
SetPosicion (*DWORD,*	public: *VOID*	Asigna posición del objeto

DWORD)		
SetTamano (*DWORD, DWORD*)	public: *VOID*	Asigna tamaño del objeto
SetColor (*D3DXCOLOR*)	public: *VOID*	Asigna el color del objeto
SetVisible (*BOOL*)	public: *VOID*	Asigna visibilidad
SetTransparente (*BOOL*)	public: *VOID*	Asigna un booleano que dice si el objeto es transparente o no
Pintar ()	public: *VOID*	Pinta el control
CrearVertices ()	public: *VOID*	Permite crea el buffer de vértices. Unicamente se puede crear una vez, las demás veces debe llamar a GetVertices para obtener el puntero a l información.
CargarDesdeArchivo (*CTexto&*)	public: *VOID*	Carga el objeto desde un archvio tipo INI
operator = (*CControl&*)	public const: *CControl&*	Sobrecarga del operador de asignación
operator == (*CControl&*)	public: *BOOL*	Sobrecarga del operador de comparación
operator != (*CControl&*)	public: *BOOL*	Sobrecarga del operador de comparación

- **CCuadroAceptarCancelar**

Declara la clase CCuadroAceptarCancelar, la cual es un dialogo que permite mostrar un mensaje al usuario con las opciones Aceptar y Cancelar.

CCuadroAceptarCancelar - Métodos

Método	Tipo	Descripción
CCuadroAceptarCancelar (*LPDIRECT3DDEVICE9, RECT, INT, CTexto&, INT, D3DXCOLOR, CTexto&, CTexto&, CTexto&, CTexto&, CEscuchadorEventos**)	public:	Constructor de inicialización.
~CCuadroAceptarCancelar ()	public:	Destructor de la clase
GetIdBotonAceptar ()	public: *DWORD*	Obtiene el ID del boton Aceptar
GetIdBotonCancelar ()	public: *DWORD*	Obtiene el ID del boton Cancelar
SetTexto (*CTexto&*)	public: *VOID*	Fija el texto para el dialogo
SetEscuchadorEventos (*CEscuchadorEventos**)	public: *VOID*	Fija el escuchador de eventos para los controles
Pintar ()	public: *VOID*	Método para pintar el dialogo

- **CCuadroMensaje**

Declara la clase CEntradaTexto, la cual es un dialogo que permite solicitar al usuario una entrada de texto.

CCuadroMensaje - Métodos

Método	Tipo	Descripción
CCuadroMensaje (*LPDIRECT3DDEVICE9, RECT, INT, CTexto&,*	public:	Constructor de inicialización.

*INT, D3DXCOLOR, CTexto&, CTexto&, CTexto&, CTexto&, CEscuchadorEventos**)		
~CCuadroMensaje ()	public:	Destructor de la clase
GetIdBotonAceptar ()	public: *DWORD*	Obtiene el ID del boton Aceptar
SetTexto (*CTexto&*)	public: *VOID*	Fija el texto para el dialogo
SetEscuchadorEventos (*CEscuchadorEventos**)	public: *VOID*	Fija el escuchador de eventos para los controles
Pintar ()	public: *VOID*	Método para pintar el dialogo

- **CDirect3D**

Declara una clase que permite iniciar y manipular Direct3D

CDirect3D - Atributos

Atributo	Tipo	Descripción
m_pD3D	private : *LPDIRECT3D9*	Puntero al objeto de tipo Direct3D
m_stParametrosD3D	private : *D3DPRESENT_PARAMETERS*	Estructura con los parametros de configuración de D3D
m_nAdaptador	private : *UINT*	Adaptador a usar para el Direct3D

CDirect3D - Métodos

Método	Tipo	Descripción
CDirect3D ()	public:	Constructor por defecto.
CDirect3D (*D3DPRESENT_PARAMETERS*)	public:	Constructor de inicialización
~CDirect3D ()	public:	Destructor de la clase.
GetD3D9 ()	public: *LPDIRECT3D9*	Retorna la interfaz IDirect3D9
GetParametrosD3D ()	public const query: *D3DPRESENT_PARAMETERS&*	Retorna la estructura con los parametros de configuración de D3D
GetAdaptador ()	public const query: *UINT&*	Retorna el adaptador que se esta usando
SetParametrosD3D (*D3DPRESENT_PARAMETERS*)	public: *VOID*	Fija los parametros de presentacion del Direct3D
SetAdaptador (*UINT*)	public: *VOID*	Fija el adaptador de pantalla que va a usar el Direct3D
SetVentana (*HWND*)	public: *VOID*	Fija el manejador de ventana
Iniciar ()	public: *VOID*	Inicializa la interfaz IDirect3D9
TieneProcesamientoHardware ()	public: *BOOL*	Verifica si el dispositivo por defecto soporta procesamiento de vértices por hardware.
VerificarCapacidadesD3D ()	public: *D3DCAPS9*	Verifica las capacidades del dispositivo Direct3D.
GetModosPantalla (*D3DFORMAT*)	public: *CLista**	Lista los distintos modos de pantalla soportados por el adaptador en un formato de pixel dado. Los formatos soportados son:

		- D3DFMT_X8R8G8B8 32-bit RGB - 8-bit color - D3DFMT_R5G6B5 16-bit RGB - 5-6-5- bit color - D3DFMT_X1R5G5B5 16-bit pixel - 5-bit color - D3DFMT_A2R10G10B10 32-bit pixel - 10-bit color 2-bit alpha (solo para pantalla completa)
GetTasasRefresco (*D3DDISPLAYMODE*)	public: *CLista**	Lista las distintos frecuencias de refresco soportadas por el adaptador en un formato de pixel dado y en una resolución específica.
ParametrosPorDefecto ()	public: *VOID*	Fija los parametros de presentacion por defecto.
CrearDispositivo (*D3DDEVTIPO*)	public: *VOID*	Intenta crear el dispositivo 3D para el renderizado

- **CEntradaTexto**

Declara la clase CEntradaTexto, la cual es un dialogo que permite solicitar al usuario una entrada de texto.

CEntradaTexto - Métodos

Método	Tipo	Descripción
CEntradaTexto (*LPDIRECT3DDEVICE9, RECT, INT, CTexto&, CTexto&, D3DXCOLOR, CTexto&, CTexto&, CTexto&, CTexto&, CEscuchadorEventos*, INT*)	public:	Constructor de inicialización.
~CEntradaTexto ()	public:	Destructor de la clase
GetIdCajaTexto ()	public: *DWORD*	Obtiene el ID de la caja de texto
GetIdBotonAceptar ()	public: *DWORD*	Obtiene el ID del boton Aceptar
GetIdBotonCancelar ()	public: *DWORD*	Obtiene el ID del boton Cancelar
GetCajaTexto ()	public: *CCajaTexto**	Obtiene la referencia de la caja de texto
GetNombreArchivo ()	public const: *CTexto&*	Obtiene el texto con el nombre para el archivo
SetTexto (*CTexto&*)	public: *VOID*	Fija el texto para el dialogo
SetEntradaPorDefecto (*CTexto&*)	public: *VOID*	Fija el texto para la entrada
SetEscuchadorEventos (*CEscuchadorEventos**)	public: *VOID*	Fija el escuchador de eventos para los controles
SetMaximoTexto (*INT*)	public: *VOID*	Fija el escuchador de eventos para los controles
Pintar ()	public: *VOID*	Método para pintar el dialogo

- **CEscuchadorEventos**

Escucha por los eventos de mouse y teclado de un objeto

CEscuchadorEventos - Métodos

Método	Tipo	Descripción
CEscuchadorEventos ()	public:	Constructor por defecto
OnRatonAbajo (*DWORD&, SHORT, WPARAM, DWORD, DWORD*)	«pure» public abstract: *VOID*	Se presionó un botón del ratón

OnRatonArriba (*DWORD&, SHORT, WPARAM, DWORD, DWORD*)	«pure» public abstract: *VOID*	Se liberó un botón del ratón
OnRatonMueve (*DWORD&, WPARAM, DWORD, DWORD*)	«pure» public abstract: *VOID*	Se movió el puntero del ratón
OnClick (*DWORD&, SHORT, WPARAM, DWORD, DWORD*)	«pure» public abstract: *VOID*	Se hizo click sobre un objeto
OnRuedaRatonGira (*DWORD&, WPARAM, DWORD, DWORD*)	«pure» public abstract: *VOID*	Se giró la rueda del mouse
OnTeclaAbajo (*DWORD&, INT*)	«pure» public abstract: *VOID*	Se presionó una tecla
OnTeclaArriba (*DWORD&, INT*)	«pure» public abstract: *VOID*	Se libera una tecla
OnCambio (*DWORD&, CHAR&*)	«pure» public abstract: *VOID*	Se presiona una tecla pero con un valor ascii

- **CEscuchadorVentana**

Escucha por los eventos de ventana de Windows

CEscuchadorVentana - Métodos

Método	Tipo	Descripción
CEscuchadorVentana ()	public:	Constructor
OnActivada (*DWORD&*)	«pure» public abstract: *VOID*	Cuando se activa la ventana
OnDesactivada (*DWORD&*)	«pure» public abstract: *VOID*	Cuando se desactiva la ventana
OnCreada (*DWORD&*)	«pure» public abstract: *VOID*	Cuando se crea la ventana
OnCerrar (*DWORD&*)	«pure» public abstract: *VOID*	Llama al evento que maneja este mensaje
OnComando (*DWORD&, WPARAM&, LPARAM&*)	«pure» public abstract: *VOID*	Cuando se da un evento de comando

- **CEstatico**

Maneja modelos 3d estáticos (sn animaciones)

CEstatico - Atributos

Atributo	Tipo	Descripción
m_pMesh	private : *LPD3DXMESH*	Puntero al mesh
m_pMateriales	private :	Puntero a los materiales

	D3DMATERIAL9	
m_pTexturas	private : *LPDIRECT3DTEXTURE9*	Puntero a las texturas
m_dwNumMateriales	private : *DWORD*	Número de materiales del mesh

CEstatico - Métodos

Método	Tipo	Descripción
CEstatico (*LPDIRECT3DDEVICE9*)	public:	Constructor de inicialización
~CEstatico ()	public:	Destructor
GetMesh ()	public const query: *LPD3DXMESH*	Retorna el puntero al mesh
GetMateriales ()	public query: *D3DMATERIAL9**	Devuelve un puntero a los materiales
GetTexturas ()	public const query: *LPDIRECT3DTEXTURE9**	Retorna un puntero a las texturas
GetNumMateriales ()	public const query: *DWORD*	Devuelve el número de materiales del mesh
Pintar ()	public: *VOID*	Método que se encarga de pintar el mesh estático
Cargar (*CTexto&*)	public: *VOID*	Método que se encarga de cargar el mesh desde un archivo

- **CEtiqueta**

Clase para la manipulación de texto

CEtiqueta - Atributos

Atributo	Tipo	Descripción
m_szTexto	protected : *CTexto*	Texto del objeto
m_pFuente	protected : *CFuente*	Puntero a la fuente
m_stColorTexto	protected : *D3DXCOLOR*	Color de la fuente
m_dwFormato	protected : *DWORD*	Formato del texto

CEtiqueta - Métodos

Método	Tipo	Descripción
CEtiqueta (*LPDIRECT3DDEVICE9, RECT, CEscuchadorEventos*, CTexto&*)	public:	Constructor de inicialización
CEtiqueta (*LPDIRECT3DDEVICE9, CTexto&*)	public:	Constructor de inicialización
~CEtiqueta ()	public:	Destructor
GetTexto ()	public const	Devuelve el puntero al texto del objeto

	query: *CTexto&*	
GetFuente ()	public: *CFuente**	Devuelve el puntero a la fuente
GetColorTexto ()	public: *D3DXCOLOR*	Retorna el color del texto
GetFormato ()	public: *DWORD*	Retorna el formato del texto
SetTexto (*CTexto&*)	public: *VOID*	Asigna el texto del objeto
SetTexto (*CHAR**)	public: *VOID*	Asigna el texto del objeto
SetFuente (*CFuente&*)	public: *VOID*	Asigna el puntero a la fuente
SetColorTexto (*D3DXCOLOR*)	public: *VOID*	Asigna el color del texto
SetFormato (*DWORD*)	public: *VOID*	Asigna el formato del texto
Pintar ()	public: *VOID*	Pinta el objeto
CargarDesdeArchivo (*CTexto&*)	public: *VOID*	Carga el objeto desde un archvio tipo INI

- **CExcepcion**

Clase para el manejo genérico de excepciones

CExcepcion - Atributos

Atributo	Tipo	Descripción
m_pMensaje	protected : *CHAR*	Guarda el mensaje de error con el que fue lanzada
m_enTipo	protected : *EN_TIPO_EXCEPCION*	Define el tipo de excepcion de la clase

CExcepcion - Métodos

Método	Tipo	Descripción
CExcepcion ()	public:	Constructor por defecto para la Excepcion.
CExcepcion (*CHAR*, EN_TIPO_EXCEPCION*)	public:	Constructor de inicialización para la Excepcion
CExcepcion (*CExcepcion&*)	public:	Constructor de copia para la excepcion.
~CExcepcion ()	public:	Libera la memoria del mensaje
GetMensaje ()	public const query: *CHAR**	Obtiene el mensaje de error
GetTipo ()	public: *EN_TIPO_EXCEPCION*	Obtiene el tipo de excepcion
SetMensaje (*CHAR**)	public: *VOID*	Fija el mensaje de error
SetTipo (*EN_TIPO_EXCEPCION*)	public: *VOID*	Fija el tipo de excepcion

- **CExcepcion::EN_TIPO_EXCEPCION**

Enumeracion para los tipos de excepcion

CExcepcion::EN_TIPO_EXCEPCION - Atributos

Atributo	Tipo	Descripción
ETE_GENERAL	public : *int*	Excepciones de tipo general
ETE_WINDOWS	public :	Excepciones de Windows

	int	
ETE_DIRECT3D	public : *int*	Excepciones de tipo Direct3D
ETE_MEDIO	public : *int*	Excepciones de tipo Medios
ETE_OBJETOS2D	public : *int*	Excepciones de tipo objetos 2D

- **CFormulario**

Declara la clase CFormulario, la cual es un contenedor de controles y se encarga de pasar los mensajes a estos.

CFormulario - Atributos

Atributo	Tipo	Descripción
m_pControles	protected : *CLista*	Lista para los controles contenidos en ella

CFormulario - Métodos

Método	Tipo	Descripción
CFormulario ()	public:	Constructor por defecto
CFormulario (*LPDIRECT3DDEVICE9*)	public:	Constructor de inicializacion
~CFormulario ()	public:	Destructor de la clase
GetLargo ()	public: *INT*	
GetControl (*INT*)	public: *CControl**	Obtiene la referencia de uno de los controles del formulario por su posición en la lista de controles.
IsObjetoContenido (*DWORD*)	public: *BOOL*	Verifica si un objeto esta en la lista de controles del formulario
IsRatonAbajo (*SHORT, WPARAM, DWORD, DWORD*)	public: *BOOL*	Se presionó un botón del ratón.
IsRatonArriba (*SHORT, WPARAM, DWORD, DWORD*)	public: *BOOL*	Se liberó un botón del ratón.
IsRatonMueve (*WPARAM, DWORD, DWORD*)	public: *BOOL*	Se movió el puntero del ratón
IsClick (*SHORT, WPARAM, DWORD, DWORD*)	public: *BOOL*	Se hizo click sobre un objeto
IsRuedaRatonGira (*WPARAM, DWORD, DWORD*)	public: *BOOL*	Se giró la rueda del ratón.
IsTeclaAbajo (*INT*)	public: *BOOL*	Se presionó una tecla
IsTeclaArriba (*INT*)	public: *BOOL*	Se liberó una tecla
IsCambio (*CHAR*)	public: *BOOL*	Se presiono una tecla de caracter
AdicionarControl (*CControl**)	public: *VOID*	Adiciona un control al formulario.
EliminarControl (*INT*)	public: *CControl**	Elimina la referencia al control de la lista de controles del formulario.
Pintar ()	public: *VOID*	Pinta
operator[] (*INT*)	public: *CControl**	Retorna la referencia a un control contenido en el formulario dada su posición en la lista de controles.

- **CFuente**

Clase para el manejo de fuentes en Windows

CFuente - Atributos

Atributo	Tipo	Descripción
m_stOpciones	private : *LOGFONT*	Estructura para almacenar las características de la fuente
m_pFuenteD3D	private : *LPD3DXFONT*	Interfaz para crear la fuente

CFuente - Métodos

Método	Tipo	Descripción
CFuente (*LPDIRECT3DDEVICE9, LOGFONT*)	public:	Constructor de inicialización
CFuente (*LPDIRECT3DDEVICE9, CHAR*, UINT, UINT, BOOL, BOOL*)	public:	Constructor de inicialización
~CFuente ()	public:	Destructor de la clase
GetOpciones ()	public: *LOGFONT*	Devuelve la estructura con las opciones de creación de la fuente
GetAncho ()	public: *UINT*	Devuelve el ancho de la fuente
GetAlto ()	public: *UINT*	Devuelve el alto de la fuente
GetNombre ()	public: *CHAR**	Devuelve el nombre de la fuente
GetFuenteD3D ()	public: *LPD3DXFONT*	Devuelve el puntero a la interfaz D3D de la fuente
IsNegrilla ()	public: *BOOL*	Devuelve true si la fuente está definida como negrita
IsCursiva ()	public: *BOOL*	Devuelve true si la fuente está definida como cursiva
IsSubrayada ()	public: *BOOL*	Devuelve true si la fuente está subrayada
SetOpciones (*LOGFONT*)	public: *VOID*	Asigna las opciones de creación de la fuente
SetAncho (*UINT*)	public: *VOID*	Asigna el ancho lógico a la fuente
SetAlto (*UINT*)	public: *VOID*	Asigna el alto lógico de la fuente
SetNombre (*CHAR**)	public: *VOID*	Asigna el nombre de la fuente
SetNegrilla (*BOOL*)	public: *VOID*	Asigna true si la fuente está definida como negrita
SetCursiva (*BOOL*)	public: *VOID*	Asigna true si la fuente está definida como cursiva
SetSubrayada (*BOOL*)	public: *VOID*	Asigna true si la fuente está subrayada
Crear ()	public: *VOID*	Crea la fuente D3D a aprtir de las ocpiones especificadas
operator = (*CFuente&*)	public const: *CFuente&*	Sobrecarga del operador de asignación

- **CImagen**

Clase para manipular imagenes y mostrarlas dentro de D3D

CImagen - Atributos

Atributo	Tipo	Descripción
m_pTextura	private : *LPDIRECT3DTEXTURE9*	Buffer para la textura

CImagen - Métodos

Método	Tipo	Descripción
CImagen (*LPDIRECT3DDEVICE9, RECT, CEscuchadorEventos*, CHAR**)	public:	Constructor de inicialización
CImagen (*LPDIRECT3DDEVICE9, CHAR**)	public:	Constructor de inicialización
CImagen (*LPDIRECT3DDEVICE9, POINT, INT, INT, FLOAT, CEscuchadorEventos**)	public:	Constructor de inicializacion
~CImagen ()	public:	Destructor
GetTextura ()	public: *LPDIRECT3DTEXTURE9*	Retorna el puntero a la textura actual
SetTextura (*LPDIRECT3DTEXTURE9*)	public: *VOID*	Fija la textura de la imagen.
Pintar ()	public: *VOID*	Pinta el objeto 2D
CargarDesdeArchivo (*CTexto&*)	public: *VOID*	Carga el objeto desde un archivo tipo INI
CargarTextura (*CHAR**)	public: *VOID*	Carga una textura desde archivo

- **CLista**

Clase para la creación de listas enlazadas

CLista - Atributos

Atributo	Tipo	Descripción
m_pPrimero	private : *CNodo*	Puntero al primer nodo
m_nLargo	private : *unsigned int*	Dice cual es el largo de la lista
m_bLiberarMemoria	private : *bool*	Dice si hay que liberar memoria o no

CLista - Métodos

Método	Tipo	Descripción
CLista (*bool&*)	public:	Constructor por defecto
CLista (*CLista&*)	public:	Constructor de copia
~CLista ()	public:	Destructor
GetPrimero ()	public: *CNodo**	Devuelve el puntero al primer nodo
GetLargo ()	public: *unsigned int*	Retorna el largo de la lista
SetLiberarMemoria (*bool&*)	public: *void*	Dice si hay que liberar memoria o no
Insertar (*void*, unsigned int*)	public: *bool*	Inserta un nuevo nodo en la lista
Buscar (*unsigned int*)	public: *void**	Busca un nodo en la lista y retorna su contenido.
Eliminar (*unsigned int*)	public: *bool*	Elimina un nodo de la lista
GetNodo (*unsigned int*)	public: *CNodo**	Devuelve el nodo especificado por nPosicion
Adicionar (*void**)	public: *bool*	Adiciona un nodo al final de la lista

Vaciar ()	public: *void*	Elimina todo los nodos de la lista
operator[] (*unsigned int*)	public: *void**	Busca un nodo en la lista y retorna su contenido
operator= (*CLista&*)	public const: *CLista&*	Operador de asignación

- **CLosa**

Clase que guarda informacion de un vértice de una losa del terreno.

CLosa - Atributos

Atributo	Tipo	Descripción
m_stVertice	protected : *CVertice*	Vertice de la esquina superior izquierda de la losa
m_pTextura	protected : *IDirect3DTexture9*	Puntero a la textura para la losa
m_pModelo	protected : *CModelo*	Puntero al modelo de ambientación de la losa
m_bSeleccionada	protected : *BOOL*	Indica que la losa está seleccionada o nó
m_bVisible	protected : *BOOL*	Dice si la losa es visible actualmente
m_fFactorNiebla	protected : *FLOAT*	Factor de niebla para el dibujado de la losa

CLosa - Métodos

Método	Tipo	Descripción
CLosa (*IDirect3DDevice9*, IDirect3DTexture9*, CVertice, BOOL, FLOAT*)	public:	Constructor de inicializacion de la clase.
~CLosa ()	public:	Destructor de la clase.
GetVertice ()	public: *CVertice**	Obtiene una referencia al vértice de la losa
GetTextura ()	public: *IDirect3DTexture9**	Obtiene una referencia a la textura de la losa
GetModelo ()	public: *CModelo**	Obtiene una referencia al modelo de la losa
IsSeleccionada ()	public: *BOOL*	Dice si la losa esta seleccionada
IsVisible ()	public: *BOOL*	Dice si la losa es visible
GetFactorNiebla ()	public: *FLOAT*	Obtiene el factor de niebla de la losa
SetTextura (*IDirect3DTexture9**)	public: *VOID*	Fija la textura para la losa
SetModelo (*CModelo**)	public: *VOID*	Fija el modelo para la losa
SetSeleccionada (*BOOL*)	public: *VOID*	Fija el estado de Seleccionada a la losa
SetVisible (*BOOL*)	public: *VOID*	Fija el estado de visibilidad a la losa
SetFactorNiebla (*FLOAT*)	public: *VOID*	Fija el valor del factor de niebla para la losa.
Pintar (*IDirect3DVertexBuffer9 *, CLosa*, CLosa*, CLosa**)	public: *VOID*	Pinta la losa, tomando como referencia su vértice y el de las tres losas adyacentes.
PintarModelo (*D3DXVECTOR3, DOUBLE*)	public: *VOID*	Pinta el modelo de ambientacion de la losa.

- **CMarco**

Estructura derivada de D3DXFRAME que nos ayuda a adicionar algunas cosas especificas para la carga de un frame

CMarco - Atributos

Atributo	Tipo	Descripción
m_mTransformacionCombinada	public : *D3DXMATRIXA16*	Transformación de la matriz combinada

- **CMedio**

Esta clase se encarga de la manipulación de contenido multimedia, tal como videos y musica en formatos avi, mpeg, wav, mp3, etc, ha- ciendo uso de la API DirectShow.

CMedio - Atributos

Atributo	Tipo	Descripción
m_hVentana	private : *HWND*	Manejador de la ventana de la aplicacion
m_hInstancia	private : *HINSTANCE*	Manejador de la instancia de la aplicacion
m_pConstructorFiltro	private : *IGraphBuilder*	Constructor del Filtro
m_pControlMedio	private : *IMediaControl*	Control para el flujo de medio
m_pEventoMedio	private : *IMediaEvent*	Interfaz para el control de eventos en el medio
m_pUbicadorMedio	private : *IMediaSeeking*	Interfaz para ubicar una posicion dentro del medio
m_pVentanaVideo	private : *IVideoWindow*	Interfaz para la ventana de Video
m_pAudioBasico	private : *IBasicAudio*	Interfaz para el audio básico
m_pVideoBasico	private : *IBasicVideo*	Interfaz para el video básico
m_pPosicionadorMedio	private : *IMediaPosition*	Interfaz para fijar el rango de reproducción
m_pPasoCuadros	private : *IVideoFrameStep*	Interfaz para la reproducción paso a paso
m_pNombreMedio	private : *CHAR*	Nombre del archivo de medio a cargar
m_bSoloAudio	private : *BOOL*	Indicador para saber si el medio es de solo audio
m_lVolumen	private : *LONG*	Nivel de volumen asociado con la reproduccion
m_dwGraphRegister	private : *DWORD*	No se para que sirve esto
m_enEstado	private : *EN_ESTADO_MEDIO*	Estado de la reproduccion
m_dRangoReproduccion	private : *DOUBLE*	Velocidad de reproduccion

m_stPosicion	private : *RECT*	Rectangulo que indica la posicion y dimensiones en la ventana

CMedio - Métodos

Método	Tipo	Descripción
CMedio (*HWND, HINSTANCE, CHAR*, RECT*)	public:	Constructor de inicialización de la clase.
~CMedio ()	public:	Se encarga de liberar todos los recursos de la clase.
IsSoloAudio ()	public: *BOOL*	Dice si el medio es de solo audio
GetVolumen ()	public: *LONG*	Obtiene el volumen de reproducción
GetEstado ()	public: *EN_ESTADO_MEDIO*	Obtiene el estado de la reproducción
GetRangoReproduccion ()	public: *DOUBLE*	Obtiene el rango de reproduccion
SetVolumen (*LONG*)	public: *VOID*	Fija el nivel de volumen de la reproduccion.
SetRangoReproduccion (*DOUBLE*)	public: *VOID*	Fija el rango de reproduccion para el medio.
Iniciar ()	public: *VOID*	Inicia las interfaces necesarias para la carga y manipulación del archivo medio.
Reproducir ()	public: *VOID*	Reproduce el contenido medio desde el archivo indicado en el cons- tructor del objeto CMedio.
Detener ()	public: *VOID*	Si hay un medio reproduciendose, lo detiene.
Pausar ()	public: *VOID*	Pausa la reproducción del medio o la activa si estaba en pausa.
VerificarSoloAudio ()	public: *VOID*	Verifica si el medio es de solo audio y actualiza el valor del atributo m_bSoloAudio.
IniciarInterfazPasos ()	public: *VOID*	Intenta obtener la interfaz IVideoFrameStep si el medio soporta la reproduccion de paso por cuadros.
IsFinReproduccion ()	public: *BOOL*	Verifica si la reproduccion llego al fin, de ser así la detiene, para que vuelva al inicio.
PasarCuadros (*INT*)	public: *VOID*	Reproduce una cantidad determinada de cuadros (frames).

- **CMedio::EN_ESTADO_MEDIO**

Enumeracion para los estados del medio

CMedio::EN_ESTADO_MEDIO - Atributos

Atributo	Tipo	Descripción
EEM_DETENIDO	public : *int*	Detenido
EEM_PAUSADO	public : *int*	Pausado
EEM_REPRODUCIENDO	public : *int*	Se está reproduciendo
EEM_INICIADO	public : *int*	Se ha iniciado

- **CModelo**

Clase base para la utilización de objetos en 3D, tal como modelos estáticos y/o animados

CModelo - Atributos

Atributo	Tipo	Descripción
m_vPosicion	protected : *D3DXVECTOR3*	Vector de posición del objeto
m_vRotacion	protected : *D3DXVECTOR3*	Cuaternion para guardar la rotación del objeto
m_vEscala	protected : *D3DXVECTOR3*	Vector para guardar la escala
m_vDiferencia	protected : *D3DXVECTOR3*	Vector diferencia entre el centro y la posición
m_vCentro	protected : *D3DXVECTOR3*	Centro de la esfera del objeto
m_fRadio	protected : *FLOAT*	Radio de la esfera del objeto
m_pControlAnimacion	protected : *LPD3DXANIMATIONCONTROLLER*	Control de animación del modelo
m_dUltimoTiempoAnimacion	protected : *DOUBLE*	Ultimo tiempo de animación
m_pAnimado	protected : *CAnimado*	Puntero a un modelo animado
m_pEstatico	protected : *CEstatico*	Puntero a un modelo estatico
m_enTipo	protected : *EN_TIPO_MODELO*	Tipo de modelo (animado o estatico)
m_dwAnimacionActual	protected : *DWORD*	Animación actual
m_dwNumConjuntoAnimaciones	protected : *DWORD*	Numero de conjuntos de animaciones
m_bRepetirAnimacion	protected : *BOOL*	Repetir animación?

CModelo - Métodos

Método	Tipo	Descripción
IsDentroDelObjeto (*DWORD, DWORD*)	protected: *BOOL*	Verifica que el rayo esté dentro de la esfera contenedora
CModelo ()	public:	Constructor por defecto
CModelo (*LPDIRECT3DDEVICE9, D3DXVECTOR3, CEscuchadorEventos**)	public:	Constructor de inicializacion
~CModelo ()	public:	Destructor
GetPosicion ()	public const query: *D3DXVECTOR3*	Retorna el vector de posición del objeto
GetRotacion ()	public const query: *D3DXVECTOR3*	Devuelve un vector con la rotación del objeto
GetEscala ()	public const query: *D3DXVECTOR3*	Retorna el vector de la escala
GetCentro ()	public const query: *D3DXVECTOR3*	Devuelve el vector con el centro de la esfera del objeto
GetRadio ()	public const	Retorna el radio de la esfera del objeto

	query: *FLOAT*	
GetTipo ()	public const query: *EN_TIPO_MODELO*	Retorna el tipo de modelo
GetAnimado ()	public const query: *CAnimado**	Puntero a un modelo animado
GetEstatico ()	public const query: *CEstatico**	Puntero a un modelo estatico
GetAnimacionActual ()	public const query: *DWORD*	Animación actual
GetNumConjuntoAnimaciones ()	public const query: *DWORD*	Numero de conjuntos de animaciones
IsRepetirAnimacion ()	public const query: *BOOL*	Repetir animación?
GetPeriodo ()	public query: *DOUBLE*	Devuelve el periodo para la animación actual
GetUltimoTiempoAnimacion ()	public const query: *DOUBLE*	Ultimo tiempo de animación
SetPosicion (*D3DXVECTOR3&*)	public: *VOID*	Asigna el vector de posición del objeto
SetRotacion (*D3DXVECTOR3&*)	public: *VOID*	Fija un vector con la rotación del objeto
SetEscala (*D3DXVECTOR3&*)	public: *VOID*	Asigna el vector de la escala
SetTipo (*EN_TIPO_MODELO&*)	public: *VOID*	Define el tipo de método de skinning
SetAnimado (*CAnimado**)	public: *VOID*	Fija el puntero a un modelo animado
SetEstatico (*CEstatico**)	public: *VOID*	Fija el puntero a un modelo estatico
SetAnimacionActual (*DWORD&*)	public: *VOID*	Animación actual
SetRepetirAnimacion (*BOOL*)	public: *VOID*	Animación actual
Pintar ()	public: *VOID*	Método para pintar el objeto
Pintar (*DOUBLE*)	public: *VOID*	Método para pintar el objeto animandolo primero
Animar (*DOUBLE*)	public: *VOID*	Se encarga de realizar la animación
FijarMatrices ()	public: *D3DXMATRIX*	Fija la tranformacion de mundo
operator = (*CModelo&*)	public const: *CModelo&*	Sobrecarga del operador de asignación

- **CModelo::EN_TIPO_MODELO**

Enumera los dos tipos de modelos

CModelo::EN_TIPO_MODELO - Atributos

Atributo	Tipo	Descripción
ETM_ANIMADO	public : *int*	Modelo tipo animado
ETM_ESTATICO	public : *int*	Modelo tipo estático
ETM_AMBOS	public :	Contiene ambos tipos de modelos

	int	
ETM_NINGUNO	public : *int*	No hay animación

- **CNodo**

Clase que define un nodo para ser usado dentro de una lista

CNodo - Atributos

Atributo	Tipo	Descripción
m_pDato	private : *void*	Puntero a la información
m_pSiguiente	private : *CNodo*	Puntero al siguiente nodo
m_bLiberarMemoria	private : *bool*	Dice si hay que liberar memoria o no

CNodo - Métodos

Método	Tipo	Descripción
CNodo ()	public:	Constructor por defecto
CNodo (*void*, bool&*)	public:	Constructor de inicialización
CNodo (*CNodo&*)	public:	Constructor de copia
~CNodo ()	public:	Libera recursos
GetDato ()	public: *void**	Devuelve un puntero al dato
GetSiguiente ()	public: *CNodo**	Retorna el puntero al siguiente nodo
SetDato (*void**)	public: *void*	Asigna un nuevo valor al dato.
SetSiguiente (*CNodo**)	public: *void*	Asigna un puntero al nodo siguiente
SetLiberarMemoria (*bool&*)	public: *void*	Dice si hay que liberar memoria o no
operator= (*CNodo&*)	public const: *CNodo&*	Sobrecarga del operador de asignación
operator== (*CNodo&*)	public: *bool*	Sobrecarga del operador de comparación
operator!= (*CNodo&*)	public: *bool*	Sobrecarga del operador de diferencia.

- **CNodoClaves**

Clase que maneja la información de un registro de una tabla de claves.

CNodoClaves - Atributos

Atributo	Tipo	Descripción
m_dwClave	private : *DWORD*	Clave del registro
m_pDato	private : *VOID*	Puntero al dato Asociado con la clave
m_pMenores	private : *CNodoClaves*	Puntero a registros con claves menores
m_pMayores	private : *CNodoClaves*	Puntero a registros con claves mayores
m_nPeso	private : *INT*	Contador de nodos del árbol

CNodoClaves - Métodos

Método	Tipo	Descripción

CNodoClaves (*DWORD, VOID**)	public:	Constructor de inicialización del nodo.
~CNodoClaves ()	public:	Destructor de la clase.
GetClave ()	public: *DWORD*	Obtiene la clave del nodo
GetDato ()	public: *VOID**	Obtiene el puntero al dato del nodo
GetMenores ()	public: *CNodoClaves**	Obtiene el puntero a los nodos con claves menores
GetMayores ()	public: *CNodoClaves**	Obtiene el puntero a los nodos con claves mayores
GetPeso ()	public: *INT*	Obtiene el contador de nodos del árbol
SetClave (*DWORD*)	public: *VOID*	Fija la clave del nodo
SetDato (*VOID**)	public: *VOID*	Fija el dato del nodo
SetMenores (*CNodoClaves**)	public: *VOID*	Fija el puntero a los nodos con claves menores
SetMayores (*CNodoClaves**)	public: *VOID*	Fija el puntero a los nodos con claves mayores
IsHoja ()	public: *BOOL*	Mira si el nodo es hoja o no.
Buscar (*DWORD*)	public: *CNodoClaves**	Busca un registro identificado con una clave en la jerarquía de registros
Buscar (*VOID**)	public: *CNodoClaves**	Busca un registro por si dato asociado en la jerarquía de registros.
BuscarPadre (*DWORD*)	public: *CNodoClaves**	Busca el padre de un registro en la tabla.
BuscarMayor ()	public: *CNodoClaves**	Busca el mayor de los registros en la tabla.
BuscarMenor ()	public: *CNodoClaves**	Busca el menor de los registros en la tabla.
BuscarPorPosicion (*INT*)	public: *CNodoClaves**	Busca un nodo según la posicion en el árbol.
Contar ()	public: *INT*	Cuenta el número de registros del árbol.
Insertar (*DWORD, VOID**)	public: *BOOL*	Inserta un nuevo nodo en la tabla ordenado por su clave.
Eliminar (*DWORD*)	public: *CNodoClaves**	Elimina recursivamente un registro de la tabla.
Liberar ()	public: *CNodoClaves**	Libera la memora de los subarboles recursivamente.

- **CObjetivo**

Esta clase se declara con el fin de servir de base para objetos que responderán a eventos de mouse y teclado

CObjetivo - Atributos

Atributo	Tipo	Descripción
m_pEscuchador	protected : *CEscuchadorEventos*	Puntero al escuchador de eventos de mouse y teclado
m_sPresionado	protected : *SHORT*	Variable que dice si se presionó un boton del raton sobre el objeto
m_bSeleccionado	protected : *BOOL*	Variable que dice si el objeto está seleccionado o no
m_enEstado	protected : *EN_ESTADO_OBJETIVO*	Define el estado del botón

CObjetivo - Métodos

Método	Tipo	Descripción
IsDentroDelObjeto (*DWORD, DWORD*)	«pure» protected abstract: *BOOL*	Verifica si el puntero del mouse está dentro del objeto
CObjetivo ()	public:	Constructor por defecto
CObjetivo (*LPDIRECT3DDEVICE9, CEscuchadorEventos**)	public:	Constructor de inicialización
IsRatonAbajo (*SHORT, WPARAM, DWORD, DWORD*)	public: *BOOL*	Se presionó un botón del ratón
IsRatonArriba (*SHORT, WPARAM, DWORD, DWORD*)	public: *BOOL*	Se liberó un botón del ratón
IsRatonMueve (*WPARAM, DWORD, DWORD*)	public: *BOOL*	Se movió el puntero del ratón
IsClick (*SHORT, WPARAM, DWORD, DWORD*)	public: *BOOL*	Se hizo click sobre un objeto
IsRuedaRatonGira (*WPARAM, DWORD, DWORD*)	public: *BOOL*	Se giró la rueda del ratón.
IsTeclaAbajo (*INT*)	public: *BOOL*	Se presionó una tecla
IsTeclaArriba (*INT*)	public: *BOOL*	Se liberó una tecla
IsCambio (*CHAR*)	public: *BOOL*	Se presiono una tecla de caracter
GetEscuchador ()	public const query: *CEscuchadorEventos**	Devuelve un puntero al escuchador de eventos de mouse y teclado
GetEstado ()	public: *EN_ESTADO_OBJETIVO*	Retorna el estado del botón
GetPresionado ()	public: *SHORT*	Retorna el boton presionado del raton sobre el objeto
IsSeleccionado ()	public const query: *BOOL*	Devuelve la variable que dice si el objeto está seleccionado o no
SetEscuchador (*CEscuchadorEventos**)	public: *VOID*	Fija el puntero al escuchador de eventos de mouse y teclado
SetEstado (*EN_ESTADO_OBJETIVO*)	public: *VOID*	Fija el estado del objetivo
ProcesarMensajes (*UINT, WPARAM, LPARAM*)	public: *VOID*	Se encarga de procesar los mensajes provenientes de Windows

- **CObjetivo::EN_ESTADO_OBJETIVO**

Enumeración de los estados del objetivo

CObjetivo::EN_ESTADO_OBJETIVO - Atributos

Atributo	Tipo	Descripción
EEO_LIBRE	public : *int*	Estado libre
EEO_ARRIBA	public :	No está seleccionado el objetivo

	int	
EEO_ABAJO	public : *int*	El mouse está sobre el objetivo
EEO_CLICK	public : *int*	Se presionó un botón del mouse sobre el objetivo

- **CObjeto**

Clase base que define el árbol de objetos gráficos que utiliza el motor

CObjeto - Atributos

Atributo	Tipo	Descripción
g_dwIdentificadorObjeto	private static : *DWORD*	Identificador global para los objetos
m_dwIdentificador	protected : *DWORD*	Identificador del objeto
m_pDispositivoD3D	protected : *LPDIRECT3DDEVICE9*	Puntero al dispositivo D3D

CObjeto - Métodos

Método	Tipo	Descripción
CObjeto ()	public:	Constructor por defecto
CObjeto (*LPDIRECT3DDEVICE9*)	public:	Constructor de inicialización
CObjeto (*CObjeto&*)	public:	Constructor de copia
GetIdentificador ()	public: *DWORD*	Retorna el identificador del objeto
GetDispositivoD3D ()	public const query: *LPDIRECT3DDEVICE9*	Devuelve el puntero al dispositivo D3D
SetDispositivoD3D (*LPDIRECT3DDEVICE9*)	public: *VOID*	Fija el puntero al dispositivo D3D

- **COpcion**

Clase que permite mostrar un cuadro de opciones con un texto 2D

COpcion - Atributos

Atributo	Tipo	Descripción
m_enEstado	private : *EN_ESTADO_OPCION*	Almacena es estado de la opción
m_pEtqEstado	private : *CEtiqueta*	Variable para la etiqueta que muestra el estado de la opcion

COpcion - Métodos

Método	Tipo	Descripción
COpcion (*LPDIRECT3DDEVICE9, RECT, CEscuchadorEventos*, CTexto&*)	public:	Constructor de inicialización
COpcion	public:	Constructor de inicialización

(*LPDIRECT3DDEVICE9, CTexto&*)		
~COpcion ()	public:	Destructor de la clase
GetEstado ()	public: *EN_ESTADO_OPCION*	Retorna el estado de la opción
SetEstado (*EN_ESTADO_OPCION*)	public: *VOID*	Asigna el estado de la opción
SetX (*DWORD*)	public: *VOID*	Asigna la posición en x del objeto
SetY (*DWORD*)	public: *VOID*	Asigna la posición en y del objeto
SetAncho (*DWORD*)	public: *VOID*	Asigna el ancho del objeto
SetAlto (*DWORD*)	public: *VOID*	Asigna el alto del objeto
SetPosicion (*DWORD, DWORD*)	public: *VOID*	Asigna posición del objeto
SetTamano (*DWORD, DWORD*)	public: *VOID*	Asigna tamaño del objeto
Pintar ()	public: *VOID*	Sobrecarga del método para pintar
IsRatonAbajo (*SHORT, WPARAM, DWORD, DWORD*)	public: *BOOL*	Se presionó un botón del ratón
IsRatonArriba (*SHORT, WPARAM, DWORD, DWORD*)	public: *BOOL*	Se liberó un botón del ratón
IsRatonMueve (*WPARAM, DWORD, DWORD*)	public: *BOOL*	Se movió el puntero del ratón
IsClick (*SHORT, WPARAM, DWORD, DWORD*)	public: *BOOL*	Se hizo click sobre un objeto

- **COpcion::EN_ESTADO_OPCION**

Enumeracion de los estados del cuadro de opciones

COpcion::EN_ESTADO_OPCION - Atributos

Atributo	Tipo	Descripción
EEO_LIBRE	public : *int*	Estado libre
EEO_SELECCIONADO	public : *int*	Estado seleccionado
EEO_DESHABILITADO	public : *int*	Estado deshabilitado

- **CRayo**

Clase para manipular rayos en el espacio 3D

CRayo - Atributos

Atributo	Tipo	Descripción
m_vOrigen	private : *D3DXVECTOR3*	Vector que define el origen del objeto
m_vDireccion	private : *D3DXVECTOR3*	Vector con la dirección del rayo

CRayo - Métodos

Método	Tipo	Descripción
CRayo ()	public:	Constructor por defecto
CRayo (*D3DXVECTOR3&*)	public:	Constructor de inicialización
GetOrigen ()	public const: *D3DXVECTOR3&*	Retorna el vector que define el origen del objeto
GetDireccion ()	public const: *D3DXVECTOR3&*	Devuelve el vector con la dirección del rayo
SetOrigen (*D3DXVECTOR3&*)	public: *VOID*	Fija el vector que define el origen del objeto
Calcular (*LPDIRECT3DDEVICE9, DWORD&, DWORD&*)	public: *VOID*	Realiza cálculos para determinar la dirección del rayo
Transformar (*LPDIRECT3DDEVICE9*)	public: *VOID*	Transforma el rayo de acuerdo a la matriz inversa de la vista
operator = (*CRayo&*)	public const: *CRayo&*	Sobrecarga del operador de asignación
operator == (*CRayo&*)	public: *BOOL*	Sobrecarga del operador de comparación
operator != (*CRayo&*)	public: *BOOL*	Sobrecarga del operador de comparación

- **CSecuencia**

Declaración una clase que permite la creación de un objeto 3D que permite hacer uso de una secuencia de texturas para crear una secuencia de imágenes.

CSecuencia - Atributos

Atributo	Tipo	Descripción
m_pVertices	private : *LPDIRECT3DVERTEXBUFFER9*	Puntero al buffer de vertices
m_pTexturas	private : *LPDIRECT3DTEXTURE9*	Puntero al arreglo de texturas
m_stPosicion	private : *D3DXVECTOR3*	Vector de posicion en el espacio
m_stRotacion	private : *D3DXVECTOR3*	Vector para guardar la rotación del objeto
m_fAncho	private : *FLOAT*	Ancho del objeto
m_fLargo	private : *FLOAT*	Largo del objeto
m_fAnguloInclinacion	private : *FLOAT*	Angulo de inclinación en grados con respecto al eje Z positivo
m_nNumTexturas	private : *INT*	Número de texturas
m_szRutaTexturas	private : *CTexto*	Guarda la ruta para llegar a los archivos de texturas
m_szExtensionTexturas	private : *CTexto*	Guarda la extension de los archivos de texturas
m_nTexturaActual	private : *INT*	Guarda el indice de la textura que se está usando actualmente
m_dTiempoCambio	private : *DOUBLE*	Tiempo para el cambió de cada textura
m_dTiempoUtlimoCambi	private : *DOUBLE*	Tiempo desde que se cambió de textura por ultima vez

o		
m_bOrdenAscendente	private : *BOOL*	Variable para saber el orden de cambio de texturas actual es ascendente
m_bSoloAscendente	private : *BOOL*	Variable para saber si la secuencia es ascendente únicamente
m_bTransparente	private : *BOOL*	Variable para saber si se debe aplicar transparencia a las texturas

CSecuencia - Métodos

Método	**Tipo**	**Descripción**
IsDentroDelObjeto (*DWORD, DWORD*)	protected: *BOOL*	Determina si un punto (x,y) está dentro de las coordenadas del objeto.
LiberarRecursos ()	private: *VOID*	Libera la memoria usada por los vétices y las texturas
CrearRecursos ()	private: *VOID*	Crea el buffer de vétices y el arreglo de texturas
FijarMatrices ()	private: *D3DXMATRIX*	Fija y retorna la matriz de transformación de mundo.
CSecuencia (*LPDIRECT3DDEVICE9, D3DXVECTOR3, FLOAT, FLOAT, FLOAT, INT, CTexto&, CTexto&, CEscuchadorEventos**)	public:	Constructor por defecto
~CSecuencia ()	public:	Destructor de la clase
GetPosicion ()	public const query: *D3DXVECTOR3*	Retorna el vector de posición del objeto
GetRotacion ()	public const query: *D3DXVECTOR3*	Devuelve un vector con la rotación del objeto
GetAncho ()	public const query: *FLOAT*	Devuelve el ancho del objeto
GetLargo ()	public const query: *FLOAT*	Devuelve el largo del objeto
GetAnguloInclinacion ()	public const query: *FLOAT*	Devuelve el ángulo de inclinación en grados con respecto al eje Z positivo
GetNumTexturas ()	public const query: *INT*	Devuelve el número de texturas
GetRutaTexturas ()	public const query: *CTexto&*	Devuelve la ruta para llegar a los archivos de texturas
GetExtensionTexturas ()	public const query: *CTexto&*	Devuelve la extensión de los archivos de texturas
GetTexturaActual ()	public const query: *INT*	Devuelve el índice de la textura que se está usando actualmente
GetTiempoCambio ()	public const query: *DOUBLE*	Tiempo para el cambió de cada textura
GetTiempoUtlimoCambio ()	public const query: *DOUBLE*	Tiempo desde que se cambió de textura por ultima vez
IsOrdenAscendente ()	public const query: *BOOL*	Variable para saber si el orden de cambio de texturas actual es ascendente
IsSoloAscendente ()	public const query: *BOOL*	Variable para saber si la secuencia es ascendente únicamente
IsTransparente ()	public const query: *BOOL*	Variable para saber si se debe aplicar transparencia a las texturas
SetPosicion (*D3DXVECTOR3&*)	public: *VOID*	Fija un vector con la posición del objeto
SetRotacion	public: *VOID*	Fija un vector con la rotación del objeto

(*D3DXVECTOR3&*)		
SetAncho (*FLOAT*)	public: *VOID*	Fija el ancho del objeto
SetLargo (*FLOAT*)	public: *VOID*	Fija el largo del objeto
SetAnguloInclinacion (*FLOAT*)	public: *VOID*	Fija el ángulo de inclinación en grados con respecto al eje Z positivo
SetTexturaActual (*INT*)	public: *VOID*	Fija el índice de la textura que se está usando actualmente
SetTiempoCambio (*DOUBLE*)	public: *VOID*	Fija el tiempo para el cambió de cada textura
SetTiempoUtlimoCambio (*DOUBLE*)	public: *VOID*	Fija el tiempo desde que se cambió de textura por ultima vez
SetOrdenAscendente (*BOOL*)	public: *VOID*	Fija si el orden de cambio de texturas actual es ascendente
SetSoloAscendente (*BOOL*)	public: *VOID*	Fija si la secuencia es ascendente únicamente
SetTransparente (*BOOL*)	public: *VOID*	Fija el estado de transparencia a las texturas
Pintar (*DOUBLE*)	public: *VOID*	Pinta el objeto

- **CTablaClaves**

Clase que permite organizar una tabla de claves con valores DWORD y un objeto de cualquier tipo asociado con cada clave.

CTablaClaves - Atributos

Atributo	Tipo	Descripción
m_nLargo	private : *INT*	Contador de registros de la tabla
m_pRegistros	private : *CNodoClaves*	Puntero a los registros

CTablaClaves - Métodos

Método	Tipo	Descripción
CTablaClaves ()	public:	Constructor por defecto de la clase
~CTablaClaves ()	public:	Destructor de la clase.
GetLargo ()	public: *INT*	
GetDato (*DWORD*)	public: *VOID**	Retorna el puntero al dato identificado por la clave dwClave.
GetClave (*VOID**)	public: *INT*	Retorna la clave del primer registro cuyo dato coincida con el dato proporcionado para la búsqueda
Insertar (*DWORD, VOID**)	public: *BOOL*	Inserta un nuevo registro en la tabla.
Eliminar (*DWORD*)	public: *VOID**	Elimina un registro de la tabla y retorna el puntero al dato asociado con la clave.
Vaciar ()	public: *VOID*	Destructor de la clase.
operator[] (*INT*)	public: *VOID**	Busca un registro por su posición en el árbol haciendo un recorrido en orden del árbol.

- **CTerreno**

Clase encargada del manejo de terrenos para un juego de estrategia.

CTerreno - Atributos

Atributo	Tipo	Descripción
m_pBufferVertices	protected :	Buffer de vértices

	IDirect3DVertex Buffer9	
m_pMiniMapa	protected : *IDirect3DSurface 9*	Superficie para conservar la superficie del minimapa
m_pColorTexturas	protected : *CTablaClaves*	Lista de colores para las texturas del terreno
m_pTexturas	protected : *CTablaClaves*	Texturas para el terreno
m_pArchivosTextura	protected : *CTablaClaves*	Nombre de los archivos de textura
m_pAnimados	protected : *CTablaClaves*	Tabla de animados
m_pModelos	protected : *CTablaClaves*	Tabla de modelos
m_pArchivosModelos	protected : *CTablaClaves*	Nombre de los archivos de modelos
m_pLosas	protected : *CLosa**	Puntero al vector para las losas
m_pAlturas	protected : *FLOAT*	Vector de alturas
m_nVerticesPorRegistro	protected : *INT*	Número de vertices por registro
m_nVerticesPorColumna	protected : *INT*	Número de vertices por columna
m_nEspacioLosas	protected : *INT*	Espacio entre losas
m_nLosasPorRegistro	protected : *INT*	Numero de celdas por registro
m_nLosasPorColumna	protected : *INT*	Numero de losas por columna
m_nAncho	protected : *INT*	Ancho del terreno
m_nProfundidad	protected : *INT*	Profundidad del terreno
m_nNumVertices	protected : *INT*	Numero de vertices
m_nNumTriangulos	protected : *INT*	Numero de triangulos
m_fEscalaAltura	protected : *FLOAT*	Escala de altura
m_nLosaSeleccionada	protected : *INT*	Indice de la losa seleccionada
m_bEditable	protected : *BOOL*	Indica si el terreno es editable
m_pMapa	protected : *DWORD*	Guarda la información acerca del mapa
m_bAmbientacionTransparente	protected : *BOOL*	Determina si la ambientación se debe pintar transparente

CTerreno - Métodos

Método	**Tipo**	**Descripción**
IsDentroDelObjeto (*DWORD, DWORD*)	protected: *BOOL*	Función heredada de CObjetivo que determina si un punto (x,y) esta dentro del objeto.
GenerarLosas ()	protected: *VOID*	Genera las losas dinámicamente a partir de las alturas del terreno.

Liberar ()	protected: *VOID*	Liberar los recursos del terreno.
CrearMapa ()	protected: *VOID*	Este método sirve para generar el mapa del terreno, a partir de la informaciòn actual de las losas
CTerreno (*IDirect3DDevice9*, CTexto&, INT, INT, INT, FLOAT, CTexto&, CEscuchadorEventos*, BOOL*)	public:	Constructor para cargar el terreno con un archivo de alturas y una textura por defecto.
CTerreno (*IDirect3DDevice9*, CTexto&, CEscuchadorEventos*, BOOL*)	public:	Constructor para cargar el terreno desde un archivo de terreno.
~CTerreno ()	public:	Destructor de la clase CTerreno.
GetMiniMapa ()	public: *IDirect3DSurface9**	
GetTextura (*INT*)	public: *IDirect3DTexture9**	Retorna el puntero a una de las texturas.
GetModelo (*INT*)	public: *CModelo**	Retorna el puntero a uno de los modelos del terreno.
GetPosicion (*INT, INT*)	public: *D3DXVECTOR3*	Retorna la posición del vértice en la fila y columna dadas.
GetAltura (*FLOAT, FLOAT*)	public: *FLOAT*	Retorna la altura del mapa en un punto (x, z).
GetVerticesPorRegistro ()	public: *INT*	Obtiene el número de vértices por registro
GetVerticesPorColumna ()	public: *INT*	Obtiene el número de vértices por columna
GetEspacioLosas ()	public: *INT*	Obtiene el espacio entre losas
GetLosasPorRegistro ()	public: *INT*	Obtiene el número de celdas por registro
GetLosasPorColumna ()	public: *INT*	Obtiene el número de losas por columna
GetAncho ()	public: *INT*	Obtiene el ancho del terren
GetProfundidad ()	public: *INT*	Obtiene la profundidad del terreno
GetNumVertices ()	public: *INT*	Obtiene el número de vertices de la malla
GetNumTriangulos ()	public: *INT*	Obtiene el número de triangulos del la malla
GetNumTexturas ()	public: *INT*	Obtiene el número de texturas que maneja el terreno actualmente
GetNumModelos ()	public: *INT*	Obtiene el número de modelos que maneja el terreno actualmente
GetEscalaAltura ()	public: *FLOAT*	Obtiene el factor de escala de la altura
GetLosaSeleccionada (*INT*, INT**)	public: *CLosa**	Retorna un puntero a la losa seleccionada.
GetLosa (*INT, INT*)	public: *CLosa**	Retorna un puntero a la losa de la posicion (nFila, nColumna).
GetLosa (*FLOAT, FLOAT*)	public: *CLosa**	Retorna un puntero a la losa de la posicion (fPosX, fPosY) en el plano (X, Z).
IsEditable ()	public: *BOOL*	Dice si el terreno es editable
GetMapa ()	public query: *DWORD**	Retorna la información acerca del mapa
IsAmbientacionTransparente ()	public: *BOOL*	Dice si la ambientacion se debe pintar transparente
SetTexturaLosaSeleccionada (*INT*)	public: *VOID*	Fija la textura actual de la losa seleccionada.

SetModeloLosaSeleccionada (*INT*)	public: *VOID*	Fija el modelo actual de la losa seleccionada.
SetAmbientacionTransparente (*BOOL*)	public: *VOID*	Fija si la ambientacion se debe pintar transparente
LeerArchivoAlturas (*CTexto&*)	public: *VOID*	Carga la información de alturas del terreno desde un archivo con formato RAW
GenerarNormales ()	public: *VOID*	Genera las normales para cada uno de los vértices del terreno.
Pintar (*DOUBLE, BOOL*)	public: *VOID*	Método que permite dibujar el terreno en la pantalla.
PintarAmbientacion (*DOUBLE*)	public: *VOID*	Método que permite dibujar la ambientación del terreno.
AdicionarTextura (*CTexto&*)	public: *VOID*	Crea una nueva textura para el terreno.
AdicionarModelo (*CTexto&*)	public: *VOID*	Adiciona un modelo a la lista de modelos.
Cargar (*CTexto&*)	public: *VOID*	Carga el terreno desde un archivo.
Guardar (*CTexto&*)	public: *VOID*	Guarda la información del terreno en un archivo.
EditarAltura (*BOOL*)	public: *VOID*	Este método permite incrementar o decrementar la altura de los vértices de la losa seleccionada dependiendo del valor bIncrementar, nCual y el atributo m_fEscalaAltura.
GenerarMiniMapa ()	public: *VOID*	Este método genera los datos de la superficie del minimapa.
DecubrirLosas (*INT, INT, FLOAT*)	public: *VOID*	Este método descubre algunas losas que no se encuentran visibles.
Nublar (*FLOAT*)	public: *VOID*	Este método aplica un factor de niebla a todas las losas del terreno.
MarcarCeldaMapa (*INT, INT, DWORD*)	public: *VOID*	Este método sirve para marcar una celda en el mapa, ya sea como ocupada (pasando el identificador de un modelo, por ejmplo) o libre

- **CTexto**

Clase que permite la manipulación de cadenas de texto

CTexto - Atributos

Atributo	**Tipo**	**Descripción**
m_pDato	private : *CHAR*	Puntero a la información
m_dwLargo	private : *DWORD*	Largo de la cadena de texto

CTexto - Métodos

Método	**Tipo**	**Descripción**
CTexto ()	public:	Constructor por defecto
CTexto (*CHAR**)	public:	Constructor de inicialización con un char
CTexto (*INT&*)	public:	Constructor de inicialización con un int
CTexto (*FLOAT&*)	public:	Constructor de inicialización con un float
CTexto (*DOUBLE&*)	public:	Constructor de inicialización con un double
CTexto (*CTexto&*)	public:	Constructor de copia
~CTexto ()	public:	Destructor
GetDato ()	public: *CHAR**	Retorna un puntero a la información
GetLargo ()	public: *DWORD*	Devuelve el largo de la cadena de texto

SetDato (*CHAR**)	public: *VOID*	Asigna nueva información a partir de un char
SetDato (*INT&*)	public: *VOID*	Asigna nueva información a partir de un int
SetDato (*FLOAT&*)	public: *VOID*	Asigna nueva información a partir de un float
SetDato (*DOUBLE&*)	public: *VOID*	Asigna nueva información a partir de un double
operator = (*CTexto&*)	public const: *CTexto&*	Sobrecarga del operador de asignación
operator = (*char**)	public const: *CTexto&*	Sobrecarga del operador de asignación
operator == (*CTexto&*)	public const: *BOOL*	Sobrecarga del operador de comparación
operator != (*CTexto&*)	public const: *BOOL*	Sobrecarga del operador de comparación
operator ! ()	public const: *BOOL*	Sobrecarga del operador de comparación
operator += (*CTexto&*)	public const: *CTexto&*	Sobrecarga del operador de adición
operator + (*CHAR**)	public: *VOID*	Sobrecarga del operador de adición
operator + (*INT&*)	public: *VOID*	Sobrecarga del operador de adición
operator + (*FLOAT&*)	public: *VOID*	Sobrecarga del operador de adición
operator + (*DOUBLE&*)	public: *VOID*	Sobrecarga del operador de adición
operator [] (*DWORD*)	public const: *CHAR*	Sobrecarga del operador []
operator LPCSTR ()	public query:	Sobrecarga del operador LPCSTR

- **CVentana**

Clase que permite crear ventanas de Windows, tal como formularios, botones, etiquetas, etc.

CVentana - Atributos

Atributo	Tipo	Descripción
m_hManejador	protected : *HWND*	Manejador de la ventana
m_dwEstilo	protected : *DWORD*	Estilo de la ventana
m_dwEstiloEx	protected : *DWORD*	Estilo extendido de la ventana
m_stRectangulo	protected : *RECT*	Posición y dimensiones de la ventana
m_szTexto	protected : *CTexto*	Texto de la ventana
m_pEscuchador	protected : *CEscuchadorVentana*	Puntero al escuchador de eventos de evenatan

CVentana - Métodos

Método	Tipo	Descripción
CVentana ()	public:	NOMBRE: CVentana Constructor por defecto
CVentana (*RECT&, CTexto&, CTexto&, CEscuchadorVentana*, DWORD&, DWORD&, HINSTANCE&*)	public:	Constructor de inicialización
~CVentana ()	public:	Destructor
GetManejador ()	public: *HWND*	Retorna el manejador de la ventana

GetEstilo ()	public: *DWORD*	Devuelve el estilo de la ventana
GetEstiloEx ()	public: *DWORD*	Devuelve el estilo extendido de la ventana
GetRectangulo ()	public: *RECT*	Retorna la posición y dimensiones de la ventana
GetTexto ()	public const: *CTexto&*	Devuelve una referencia al texto de la ventana
GetEscuchador ()	public const query: *CEscuchadorVentana**	Retorna el puntero al escuchador de eventos de ventana
SetEstilo (*DWORD&*)	public: *VOID*	Fija el estilo de la ventana
SetEstiloEx (*DWORD&*)	public: *VOID*	Fija el estilo extendido de la ventana
SetRectangulo (*RECT&*)	public: *VOID*	Fija la posición y tamaño de la ventana
SetTexto (*CTexto&*)	public: *VOID*	Fija el texto de la ventana
SetEscuchador (*CEscuchadorVentana**)	public: *VOID*	Fija el puntero al escuchador de eventos de ventana
Crear (*CTexto&, HINSTANCE&, HWND*)	public: *VOID*	Método que sirve para crear la ventana
Mostrar (*INT*)	public: *VOID*	Hace visible la ventana y actualiza el contenido
ProcesarMensajes (*UINT, WPARAM, LPARAM*)	public: *VOID*	Se encarga de procesar los mensajes de esta ventana
operator = (*CVentana&*)	public const: *CVentana&*	Sobrecarga del operador de asignaciòn

- **CVentanaPrincipal**

Permite crear una ventana principal para una aplicación Windows.

CVentanaPrincipal - Atributos

Atributo	Tipo	Descripción
m_bPantallaCompleta	private : *BOOL*	Indicador de pantalla completa

CVentanaPrincipal - Métodos

Método	Tipo	Descripción
CVentanaPrincipal (*RECT&, CTexto&, CTexto&, CEscuchadorVentana*, BOOL*)	public:	Constructor de inicialización
IsPantallaCompleta ()	public const query: *BOOL&*	Se muestra en pantalla completa?

- **CVertice**

Estructura para la definición de un vértice en D3D, de forma que se pueda usar en objetos 2D.

CVertice - Atributos

Atributo	Tipo	Descripción
m_vPosicion	public : *D3DXVECTOR3*	Vector que define la posición del vertice
m_vNormal	public : *D3DXVECTOR3*	Vector para las normales

m_fU	public : *FLOAT*	Posición de textura
m_fV	public : *FLOAT*	Posición de textura

CVertice - Métodos

Método	Tipo	Descripción
CVertice ()	public:	Constructor
CVertice (*D3DXVECTOR3, D3DXVECTOR3, FLOAT, FLOAT*)	public:	Constructor

- **IObjetivo**

Esta clase es una interfaz que declara los métodos que debe imple- mentar un objetivo.

IObjetivo - Métodos

Método	Tipo	Descripción
IObjetivo ()	public:	Constructor
IObjetivo (*LPDIRECT3DDEVICE9*)	public:	Constructor
IObjetivo ()	public: *IObjetivo*	Constructor
IsRatonAbajo (*SHORT, WPARAM, DWORD, DWORD*)	«pure» public abstract: *BOOL*	Se presionó un botón del ratón
IsRatonArriba (*SHORT, WPARAM, DWORD, DWORD*)	«pure» public abstract: *BOOL*	Se liberó un botón del ratón
IsRatonMueve (*WPARAM, DWORD, DWORD*)	«pure» public abstract: *BOOL*	Se movió el puntero del ratón
IsClick (*SHORT, WPARAM, DWORD, DWORD*)	«pure» public abstract: *BOOL*	Se hizo click sobre un objeto
IsRuedaRatonGira (*WPARAM, DWORD, DWORD*)	«pure» public abstract: *BOOL*	Se giró la rueda del ratón.
IsTeclaAbajo (*INT*)	«pure» public abstract: *BOOL*	Se presionó una tecla
IsTeclaArriba (*INT*)	«pure» public abstract: *BOOL*	Se liberó una tecla
IsCambio (*CHAR*)	«pure» public abstract: *BOOL*	Se presiono una tecla de caracter

ANEXO D: CONTEXTO HISTORICO

Se encontraba España en circunstancias muy difíciles, por razón de la guerra contra los africanos y de la ruptura de relaciones con Inglaterra, por lo cual necesitaba fondos para sostener las contiendas. Carlos III, afanado por recuperar a Gibraltar, la Isla de Menorca y la Florida, perdidas con Inglaterra, decide obtener los fondos de sus colonias en América, que para aquel entonces tenían una industria floreciente.

Se producía en la Nueva Granada aguardiente, tabaco y sal, además de los insumos básicos, centrándose una gran parte de la industria granadina en la Provincia del Socorro, centro manufacturero e industrial del país. Esta provincia comprendía, entre otros, los pueblos de Mogotes, Socorro, Simacota, Charalá, Barichara, Pinchote, Guadalupe, Vélez y San Gil.

El movimiento industrial en la Colonia fue bastante limitado, como consecuencia del monopolio comercial, de la restricción de las industrias, de las excesivas contribuciones y del menosprecio peninsular por el trabajo en las colonias. Las industrias más remuneradoras, en orden de importancia, a las cuales se dedicó más atención fueron la minería y la agricultura.

La industria minera se centraba principalmente en la explotación del oro y de la plata, mientras que la industria agrícola se centraba en la producción de trigo, cebada, arroz, café, frutas, entre otros productos, teniendo un gran impulso el cultivo de tabaco, algodón y caña de azúcar.

Había para entonces un gran descontento social liderado por los criollos, que buscaban reconocimiento para ocupar cargos de mayor jerarquía en el Nuevo Reino de Granada. Esto se había agudizado con eventos como la expulsión de los jesuitas a finales del siglo XVIII y se recrudecieron las arbitrariedades que llevaron a la revolución de los comuneros.

Se gestó desde entonces un caldo de cultivo expedito para la posterior Independencia de la ahora República de Colombia, surgiendo próceres de gran renombre y eruditos que allanaron el camino para los múltiples hechos que conformaron este proceso independentista y que no sólo tiene que ver con el 20 de julio de 1810 o con Simón Bolívar, sino que por el contrario incluye antecedentes como el florecimiento de las letras y la ciencia, el ejemplo tomado de revoluciones extranjeras, las confrontaciones contra los españoles, e incluso la Revolución de los Comuneros, entre otras.

Es por ser, la Revolución de los Comuneros, una de las primeras manifestaciones de descontento y a la vez de organización contra el reino español, que merece especial atención, porque exalta los valores de gente que supo oponerse a la tiranía pero que además se organizó para movilizar a un número tal de personas que incluso hoy es difícil lograr.

A continuación, entonces, se presenta una reseña histórica de la Revolución de los Comuneros, la cual fue el soporte para el guión del juego y dio las bases para la creación de los personajes, e incluso el mismo terreno donde se iban a desarrollar las acciones.

REVOLUCIÓN DE LOS COMUNEROS

El rey Carlos III, en su afán por salir bien librado de las contiendas belicosas que llevaba contra los africanos y los ingleses, decide enviar a la Nueva Granada al Visitador Juan Francisco Gutiérrez de Piñeres, con el fin de obtener más fondos para soportar su trifulca ya cazada.

Una vez llegado el Visitador Gutiérrez de Piñeres a la Nueva Granada, asumió el control total del gobierno, teniendo que, el virrey Manuel Antonio Florez, abandonar su cargo y radicarse en Cartagena.

Gutiérrez duplicó el precio del aguardiente y del tabaco, agravó el impuesto a la Armada de Barlovento[1], creó nuevos tributos y reorganizó los recaudos. Los viejos y los nuevos impuestos recaían sobre la gente pobre, pero se hacían más intolerables por la altanería de los guardas de rentas y los desmanes cometidos en la recaudación.

La gente no aguantó más y el 16 de marzo de 1781, día de feria semanal en la ciudad de Socorro, un puñado de hombres se dirige a la casa del alcalde en son de reclamo. Acto seguido Manuela Beltrán, grita intempestivamente: "¡Viva el Rey y muera el mal Gobierno!" – "Muera", responden todos en ímpetu solidario, después de lo cual, esta valiente mujer rompe el edicto de los nuevos impuestos.

El 16 de abril del mismo año se reunieron 6000 hombres (provenientes de regiones circundantes) en el Socorro, nombrando una junta a la cual se le denominó *Común*, siendo nombrado como general el señor Juan Francisco Berbeo, quien aceptó el cargo no sin antes declarar su fidelidad al Rey y diciendo que lo hacía debido a las amenazas de la plebe.

Berbeo, al mando de hombres muy mal armados (con lanzas, garrotes, hondas y algunas escopetas), ordenó la marcha hacia Santa Fe siguiendo la ruta Socorro, Oiba, Vélez, Puente Real, Chiquinquirá, Ubaté, Nemocón El Mortiño y terminando en Zipaquirá.

Al enterarse, la Real Audiencia, decide someter a los insurrectos utilizando la fuerza, por lo cual envía a 100 hombres bajo el mando de Joaquín de la Barrera, para que detengan la multitud. Estos son hechos prisioneros y pierden la contienda en Puente Real.

Ante esta noticia y en vista que la multitud revoltosa se acercaba a la capital, el Visitador sale huyendo a refugiarse en Honda, no sin antes enviar una delegación conformada por el Oidor Vasco, el Alcalde Eustaquio Galvis y el Arzobispo Antonio Caballero y Góngora, para que se reuniera con los revolucionarios.

[1] Llamado también *sisa* o *alcabala nueva,* este impuesto gravaba los artículos de mayor consumo. Tenía por objeto la creación y el sostenimiento de la flota con ese nombre, destinada a proteger contra piratas y corsarios el comercio entre España y los países del Caribe.

El 26 de mayo de 1781 el ejército revolucionario de casi 20000 hombres llega a Nemocón, desde donde Berbeo envía a José Antonio Galán a perseguir al Visitador Gutiérrez de Piñeres, siguiendo la ruta Zipaquirá, Facatativá, Villeta, Guaduas, Mariquitá y Ambalema.

Una vez reunida la comisión (enviada por el Visitador) y Berbeo, se acordó las capitulaciones donde se pedía, entre otras cosas, la abolición de los impuestos, la disminución de otros, la entrega de las salinas a los indígenas y la preferencia de los americanos para los empleos de primera. En primera instancia estuvieron a punto de fracasar, pero después de que el pueblo gritara "¡Traición! Guerra a la capital" y estuviera apunto de tomarse a Santa Fe, se firmaron las Capitulaciones y fueron suscritas por la real Audiencia de Santa Fe. Hecho esto, la revolución se dispersó.

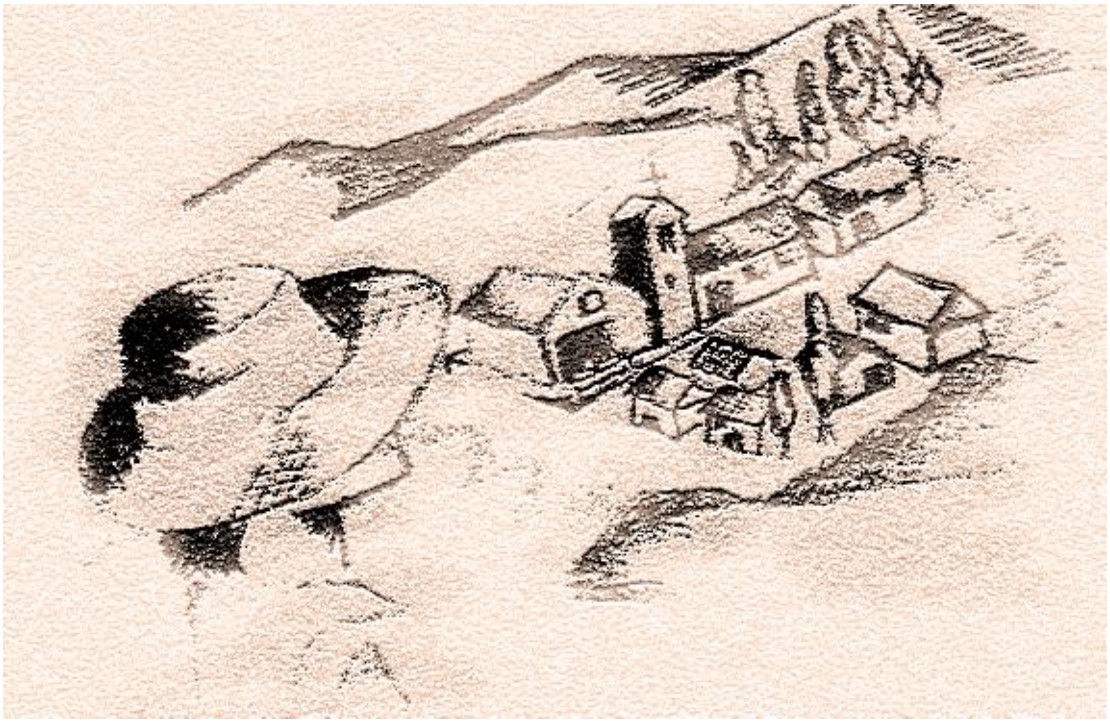
Entre tanto Gutiérrez de Piñeres ya había partido hacia Cartagena y Galán regresaba de su travesía. El primero, después de intrigas, hizo que se desaprobaran las Capitulaciones y por tanto quedaron sin fundamento, para posteriormente ya mejor armado volver a retomar el poder y controlar todo nuevamente.

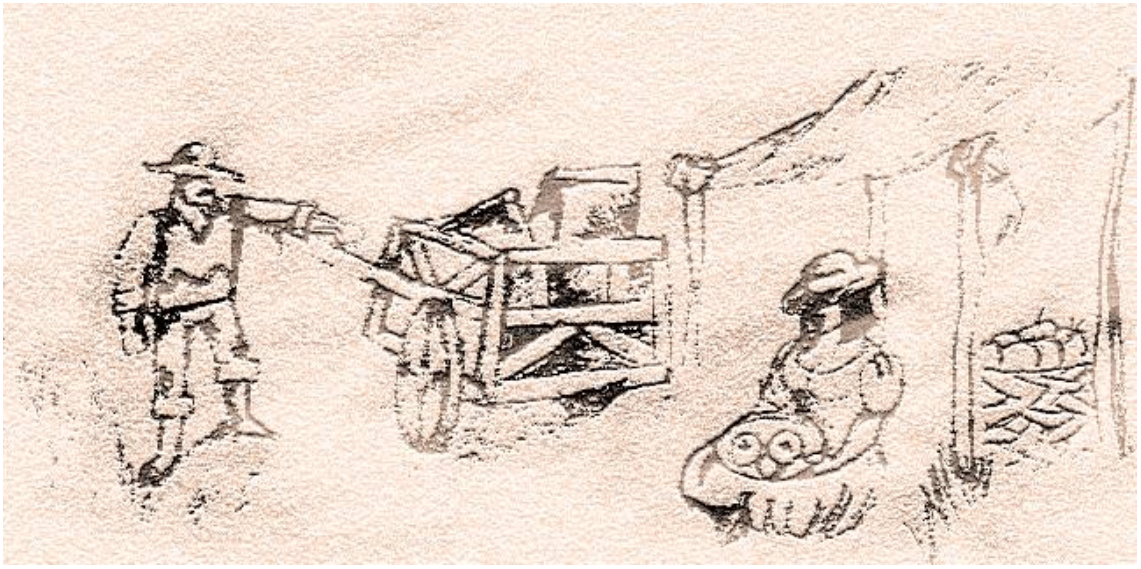
ANEXO E: STORY BOARD PRESENTADO EN IMAGINE CUP 2005

LA REVOLUCIÓN DE LOS COMUNEROS

The king Carlos III, in their desire to leave well liberated of the belligerent wars that took against the Africans and the Englishmen, decides to send to the "Nueva Granada" to the Visitor Juan Francisco Gutiérrez de Piñeres, with the purpose of obtaining more funds to support their already hunted muss. Once arrived the Visitor Gutiérrez de Piñeres to the "Nueva Granada", duplicated the price of liquor and tobacco. He increased the tax to the Armada of Windward ("Armada de Barlovento"), did new tributes and it reorganized the collects. The old ones and the new taxes relapsed on poor people, but these became more intolerable for the arrogance of the guards of rents and the outrages made in the collection.

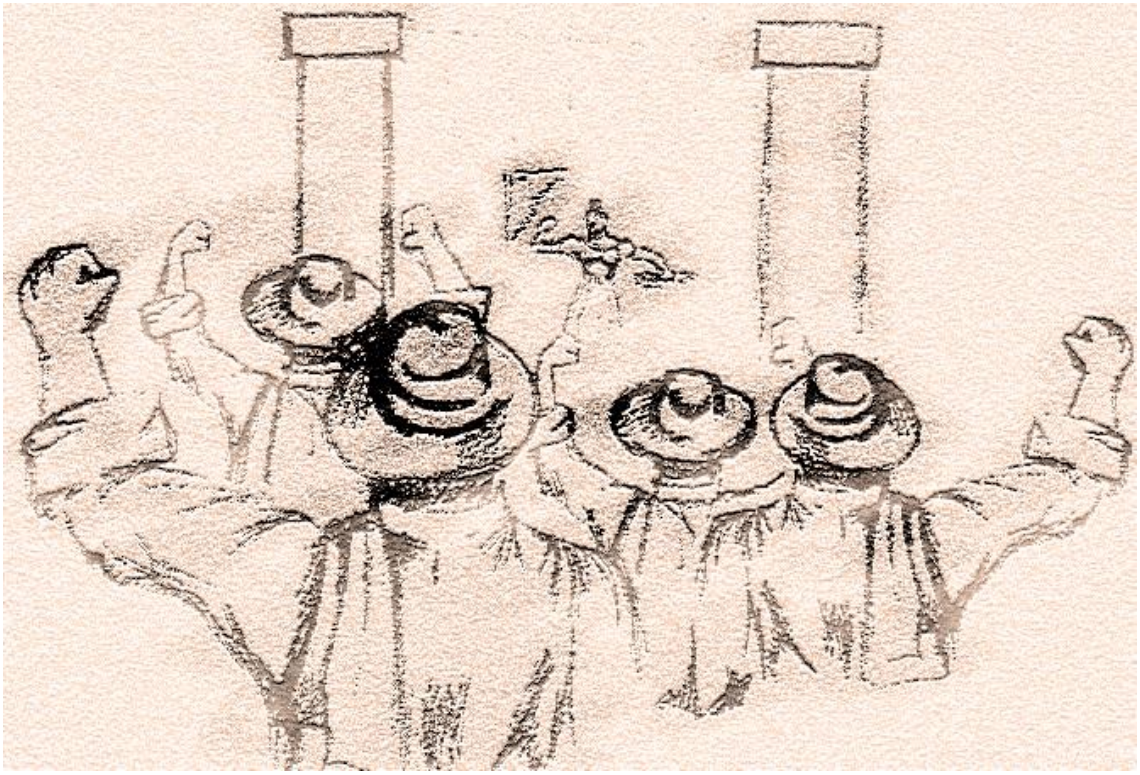
March 16, 1781, was day of weekly fair in Socorro's city.





That day people didn't tolerate more and a handful of men go to the mayor's house in a birdcall tone.

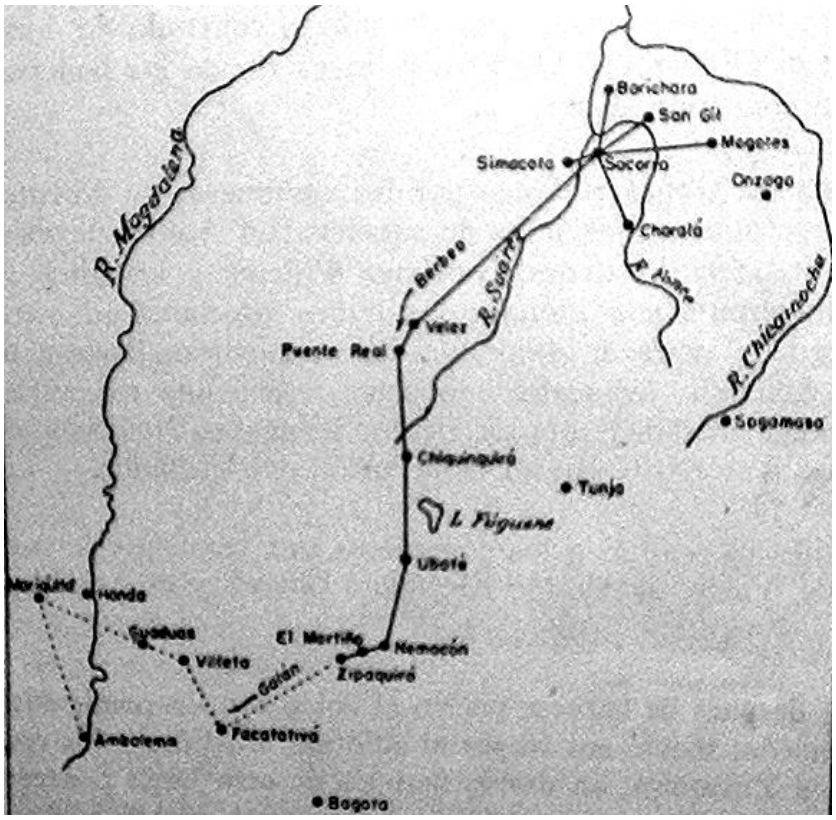
Followed act Manuela Beltrán, screams inopportunely: "Alive the King and die the wrong Government!" - "Die", all respond in solidary impulse, after which, this valiant woman breaks the proclamation of the new taxes.



April 16, 1781, one month after the break of the ban, It met in the Socorro (Santander) a group of men coming from surrounding regions, with the purpose of protesting for the recent increment in the taxes.

That day, a meeting is named which was denominated Common ("*Común*"), being designated as general Mr. Juan Francisco Berbeo.

Berbeo, at the control of very not well armed men (with lances, garrotes, "hondas", "machetes" and some shotguns), ordered the march toward Santa Fé following the route Socorro, Oiba, Vélez, Puente Real.



The colonial authorities decide to send a group of very armed men, to the control of Mr. Joaquín de la Barrera, to stop the march, and in Puente Real, a confrontation is unchained among the two decrees.

The fight was unequal, because the comuneros so alone was armed of "hondas" and "machetes", while the Spaniards were overcome in number, but they possessed rifles and combat swords. Only the exaltation of the spirits by means of its leader's continuous incentive, took to the Comuneros to win the battle.

ANEXO F: REFERENCIA DE MICROSOFT DIRECTX 9.0

Microsoft® DirectX® es un conjunto de Interfaces de Programación de Aplicación (APIs) para la creación de juegos y otras aplicaciones multimedia de alto rendimiento. Incluye soporte para gráficas en dos y tres dimensiones (2D y 3D), efectos de sonido y música, dispositivos de entrada y aplicaciones conectadas a la red como juegos multi – jugador.

Microsoft® DirectX® 9.0 está hecho de los siguientes componentes:

- DirectX Graphics: combina los componentes Microsoft DirectDraw® y Microsoft Direct3D® de versiones previas de DirectX en una simple interfaz de programación de aplicación que se puede usar para toda la programación gráfica. El componente incluye la librería de utilidad de extensiones de Direct3D (D3DX), que simplifica muchas de las tareas de programación gráfica.
- Microsoft DirectInput® provee soporte para una variedad de dispositivos de entrada, incluyendo soporte total para la tecnología force-feedback[2].
- Microsoft DirectPlay® provee soporte para juegos multi - jugador conectados en red.
- Microsoft DirectSound® puede ser usado en el desarrollo de aplicaciones de audio de alto desempeño que tocan y capturan audio en forma de onda.
- Microsoft DirectMusic® brinda un solución completa para bandas sonoras musicales y no musicales basadas en forma de ondas, sonidos MIDI, o contenido dinámico autorizado en el Productor de DirectMusic.
- Microsoft DirectShow® facilita la captura y reproducción de flujos multimedia.
- DirectSetup es una API simple que facilita la instalación de los componentes de DirectX.

[2] La tecnología Force-Feedback es la que permite que los dispositivos de entrada presenten reacción ante un tipo de acción ejecutada por el usuario.

- DirectX Media Objects da soporte para escritura y uso de objetos de flujo de datos, incluyendo video y codificadores de audio, decodificadores y efectos.

DIRECTX GRAPHICS

Un componente esencial en DirectX® es DirectX Graphics®, pues mucho de lo que se percibe en un juego es la parte visual y puede hacer que el mismo sea atractivo o no para un jugador.

Aunque DirectX incluye DirectDraw, que es una muy buena API, no es muy útil cuando se trata de programar juegos de vanguardia, pues la mayoría utiliza graficas tri – dimensionales y por tanto su uso es limitado a otro tipo de aplicaciones que requieren manejo de graficas en 2D. Por tanto a continuación se hará énfasis en Direct3D, pues constituye la base para la creación de juegos de última generación, por su compatibilidad con muchas tarjetas gráficas, su rendimiento y su completitud en cuanto a la API.

ARQUITECTURA DE DIRECT3D

Desde el punto de vista del hardware, Microsoft® Direct3D® brinda la facilidad de utilización de dispositivos independientes a través de la capa de hardware de abstracción (hardware abstraction layer - HAL). La HAL es una interfaz específica del dispositivo, proveída por el fabricante del dispositivo, que Direct3D usa para trabajar directamente con el hardware de despliegue. Las aplicaciones no interactúan directamente con la HAL, en su lugar, lo hace con la infraestructura que la HAL provee, esto es, con un conjunto de interfaces y métodos que una aplicación usa para mostrar las gráficas. La HAL que provee el fabricante puede ser parte del controlador de la tarjeta gráfica o puede ser una librería de enlace dinámico (DLL) separada.

Es de notar, entonces, que dada la sencillez en su organización, existe una integración del sistema que permite que las aplicaciones no accedan directamente al hardware, sino que se hace en primera instancia con la API, que a su vez se puede comunicar con

la HAL o con el controlador del hardware y este con el dispositivo de despliegue. En la *Figura 1* se muestra un esquema comparativo entre la utilización de Direct3D (implementada en DirectX) y la Interfaz de Dispositivos Gráficos (Graphics Device Interface – GDI) usada en Microsoft Windows (tradicional).

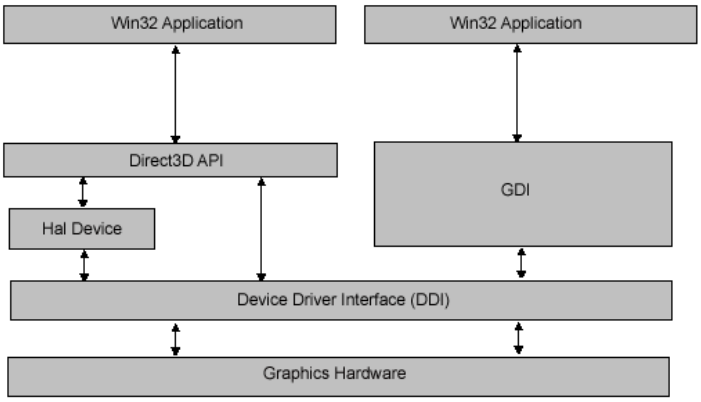


Figura 1. Integración del Sistema de Graficas entre Direct3D y el GDI

RENDERING PIPELINE EN DIRECT3D

Una vez entendido cómo se lleva a cabo la comunicación entre la aplicación y el hardware de despliegue, es conveniente observar de qué manera se lleva a cabo el proceso de dibujado en Direct3D, es decir, determinar la secuencia de dibujado (Rendering Pipeline), la cual se ilustra mediante la *Figura 2*.

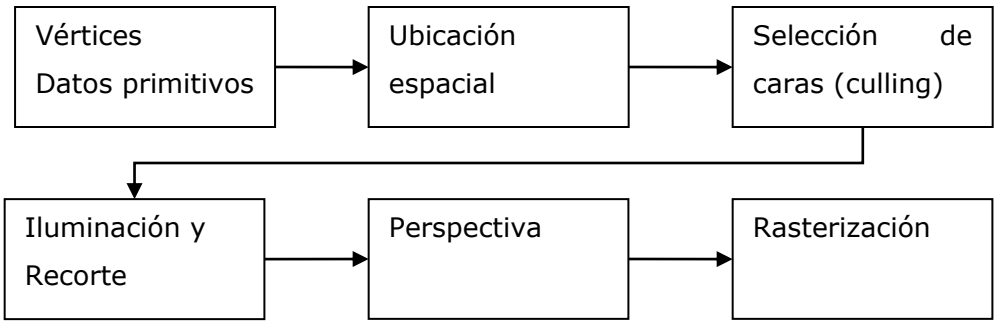


Figura 2. Secuencia de dibujado en Direct3D

En primera instancia, cuando se crea una aplicación que usa Direct3D, se define la representación del modelo, siendo esta una colección de objetos que utilizan triángulos, líneas, puntos entre otras, para dibujarse. Estos triángulos, líneas y puntos,

que los construye el programador, se consideran datos primitivos, pues a partir de ahí se obtienen todos los objetos tri – dimensionales imaginables. Adicionalmente, para poder definir estos datos primitivos, es necesario definir vectores, que indican la posición de cada punto final de los datos primitivos.

Después, y a grandes rasgos, lo que ocurre es:

- El sistema toma los datos proporcionados mediante las primitivas y los vectores, para ubicar al (a los) objeto (s) en las coordenadas apropiadas, según como se haya definido previamente.
- Luego el sistema hace un proceso de selección de caras (backface culling), es decir, para objetos 3D, se elige la cara que va al frente y la que va atrás, porque la segunda no se pinta.
- Posteriormente se realiza la iluminación (lighting) y el recorte (clipping), siendo en esta etapa (del clipping) donde se define que se pinta y que no, de acuerdo a lo que se va a presentar en pantalla.
- Finalmente, se realiza una transformación de proyección o perspectiva (es decir se ubican los objetos de acuerdo a la perspectiva que se haya definido) y se lleva a cabo el proceso de rasterización en el cual se le da color a la figura ya en un plano 2D.

EL DISPOSITIVO DE DIRECT3D

Para poder realizar muchas de las tareas que tienen que ver con el dibujado en Direct3D, se debe definir cuál va a ser el dispositivo sobre el cual se va a dibujar, es decir, se debe elegir el hardware apropiado para realizar las tareas de dibujado de los objetos. Para lograr esto, se debe referenciar el dispositivo a través de una Interface que provee la API de Direct3D y que es IDirect3DDevice9. Esta Interface, trae definidos unos métodos que permiten pintar figuras primitivas (como triángulos, puntos y líneas), fijar luces, definir texturas, cambiar las coordenadas del mundo, y muchas otras funciones que se utilizan al momento de realizar el proceso de dibujado.

Se puede apreciar, que el dispositivo es una parte muy importante en el proceso de rendering (dibujado) y por tanto se constituye en uno de los elementos centrales de la

API de Direct3D, y su conocimiento es de vital importancia para lograr potenciar muchas de las características que provee la librería gráfica de DirectX 9.

DIRECTSHOW

Otro componente muy versátil de DirectX es DirectShow que es una API para la plataforma Microsoft Windows y que sirve para dar soporte a la arquitectura de flujo de medios (o multimedia). Con DirectShow se facilita tanto la captura como la reproducción de flujos multimedia y se da soporte a una gran variedad de formatos, tanto de vídeo como de sonido, tal como Advanced Systems Format (ASF), Motion Picture Experts Group (MPEG), Audio-Video Interleaved (AVI), MPEG Audio Layer-3 (MP3), y WAV.

El bloque de construcción de DirectShow es un componente software denominado Filtro, el cual desarrolla una serie de operaciones en un flujo multimedia, dentro de las cuales se incluye: leer archivos, obtener el video de un dispositivo diseñado para tal fin, decodificar varios formatos de flujo como video MPEG-1, pasar datos a la tarjeta gráfica o de sonido, entre otras.

Para poder desarrollar las operaciones, la aplicación debe:

- Crear una instancia del Manejador de Filtros Gráficos.
- Utilizar el Manejador de Filtros Gráficos para construir un filtro.
- Usar el Manejador de Filtros Gráficos para controlar el filtro y los flujos de datos a través de los filtros.
- Liberar el Manejador y los filtros, cuando ya no se vayan a usar más.

ANEXO G: PREREQUISITOS MATEMATICOS

En este apartado se tendrá en cuenta los principales conceptos matemáticos que son de especial interés en la creación de juegos de computador en tres dimensiones. Dentro del estudio de estos temas se incluye vectores, matrices, rayos y superficies.

VECTORES EN R^3

Geométricamente se representa un vector en el espacio 3D, como un segmento de línea recta, que tiene una longitud y una dirección, de manera que se puede definir con dos puntos (inicial y final) tal y como se puede apreciar en la *Figura 3*.

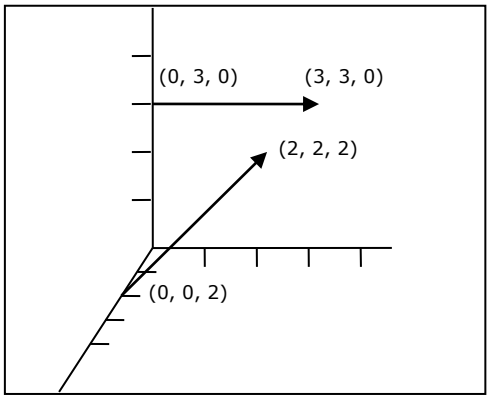


Figura 3. Ejemplos de vectores en R^3

Sin embargo, y debido a que dos vectores son iguales siempre y cuando tenga la misma magnitud y la misma dirección, lo más usual es que el punto inicial se haga corresponder con el origen (0, 0, 0), por lo que sólo es necesario definir el punto final para describir un vector.

Matemáticamente un vector se representa con letras minúsculas y tiene componentes que corresponden a los puntos en los ejes ***x***, ***y***, y ***z***. Ejemplos son: u = (2.5, 3.1, 4.2), v = (1.0, -2.0, 6.7).

Ahora bien, como se mencionó previamente, a un vector se le puede calcular la magnitud, pero además se pueden realizar operaciones con ellos, dentro de las cuales están: normalización, suma, resta y multiplicación. A continuación se muestra brevemente en que consiste cada una de estas.

Sea u y v vectores en R^3, k un escalar y $\hat{u}$ un vector unitario, se define:

- Magnitud de un vector: $\|u\| = \sqrt{u_x^2 + u_y^2 + u_z^2}$
- Normalizar un vector: $\hat{u} = \dfrac{u}{\|u\|} = \left(\dfrac{u_x}{\|u\|} + \dfrac{u_y}{\|u\|} + \dfrac{u_z}{\|u\|} \right)$
- Suma de vectores: $u + v = (u_x + v_x, u_y + v_y, u_z + v_z)$
- Resta de vectores: $u - v = (u_x - v_x, u_y - v_y, u_z - v_z)$
- Multiplicación por escalar: $ku = uk = (ku_x, ku_y, ku_z)$
- Producto punto: $u.v = u_x.v_x + u_y.v_y + u_z.v_z = s$

MATRICES

Una matriz de m x n es un arreglo rectangular de números con m filas y n columnas. Este número de filas y de columnas da la dimensión de la matriz. La notación de una matriz se hace mediante la utilización de letras mayúsculas (opcionalmente se escribe la dimensión de la matriz), mientras que sus elementos usualmente se nombran con la misma letra del nombre de la matriz, pero en minúscula y con un subíndice que identifica la fila y la columna en la cual se encuentra el elemento. Ejemplos de matrices (con la notación mencionada) son:

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \quad A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix}$$

Adicionalmente se puede decir que dos matrices son iguales siempre y cuando tengan la misma dimensión y cada uno de sus correspondientes elementos sean iguales. Así, sean A y B dos matrices de m x n, cada una, entonces A = B si todo $a_{ij} = b_{ij}$.

De la misma manera como se le definieron propiedades y operaciones a los vectores, se le pueden definir a las matrices. A continuaciones presentan brevemente las más importantes:

Sean A, B, I matrices y k un escalar, se define:

- Suma de matrices: $A+B=\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}+\begin{bmatrix} b_{11} & \dots & b_{1n} \\ \dots & \dots & \dots \\ b_{m1} & \dots & b_{mn} \end{bmatrix}=\begin{bmatrix} a_{11}+b_{11} & \dots & a_{1n}+b_{1n} \\ \dots & \dots & \dots \\ a_{m1}+b_{m1} & \dots & a_{mn}+b_{mn} \end{bmatrix}$

  si la dimensión de A es igual a la dimensión de B.
- Multiplicación por escalar: $kA=k\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}=\begin{bmatrix} ka_{11} & \dots & ka_{1n} \\ \dots & \dots & \dots \\ ka_{m1} & \dots & ka_{mn} \end{bmatrix}$
- Multiplicación de matrices:

  $$AB=\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & a_{ij} & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}+\begin{bmatrix} b_{11} & \dots & b_{1n} \\ \dots & b_{ij} & \dots \\ b_{m1} & \dots & b_{mn} \end{bmatrix}=\begin{bmatrix} a_{11}b_{11}+\dots+a_{1n}b_{m1} & \dots & a_{11}b_{1n}+\dots+a_{1n}b_{mn} \\ \dots & \dots & \dots \\ a_{m1}b_{11}+\dots+a_{mn}b_{m1} & \dots & a_{mn}b_{1n}+\dots+a_{mn}b_{mn} \end{bmatrix}$$

  pero hay que tener en cuenta que la dimensión de A debe ser *mxn* y la de B debe ser *nxp*, para dar una matriz resultante de *mxp*.
- Identidad de una matriz: $I=\begin{bmatrix} 1 & \dots & 0 \\ \dots & 1 & \dots \\ 0 & \dots & 1 \end{bmatrix}$ es una matriz cuadrada (*nxn*) donde todos los elementos son 0, excepto los ij donde i = j, que son 1. La matriz identidad tiene la propiedad de que al multiplicar otra matriz de la misma dimensión da la misma matriz, es decir $AI=IA=A$.
- Inversa de una matriz: $AA^{-1}=A^{-1}A=I$ si A es una matriz cuadrada (*nxn*), se denota A^{-1} como la inversa de la matriz. Cabe anotar que no toda matriz cuadrada tiene inversa.
- Transpuesta de una matriz: $Si\ A=\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}\Rightarrow A^{T}=\begin{bmatrix} a_{11} & \dots & a_{m1} \\ \dots & \dots & \dots \\ a_{1n} & \dots & a_{nm} \end{bmatrix}$

RAYOS

Un rayo puede ser descrito con un origen y una dirección como se muestra en la *Figura 4*.

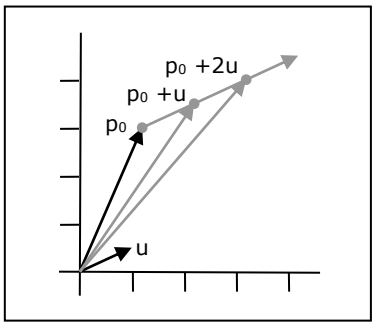


Figura 4. Un rayo descrito por un origen p_0 y una dirección u

La ecuación para un rayo está dada por $p(t) = p_0 + tu$, donde p_0 es el origen del rayo y *u* es la dirección del rayo.

ROTACION DE EJES

En ocasiones resulta útil hacer que una gráfica o función se pueda rotar según cierto ángulo con respecto a su origen. Esto es, cada punto en el plano (x*, y*) se gira en una cantidad determinada por el ángulo, aunque se conserva el origen del plano y la distancia del punto al origen. Luego de la rotación podemos decir que el sistema tiene dos sistemas de coordenadas: las del plano (x*, y*) y las del plano (u*, v*). Éste último es el plano generado luego de la rotación.

Se supone que *r* es la longitud de *OP* (distancia del punto al origen) y ϕ es el ángulo entre el eje positivo *u* y *OP*. Entonces al observar el triángulo *OPM*, se ve que:

$$\cos(\phi + \theta) = \frac{x}{r}$$

Entonces:

$$x = r \cdot \cos(\phi + \theta) = r(\cos\phi \cdot \cos\theta - sen\phi \cdot sen\theta)$$
$$x = (r \cdot \cos\phi)\cos\theta - (r \cdot sen\phi)sen\theta$$

Considerando el triángulo OPN se advierte que $u = r \cdot \cos\phi$ y $v = r \cdot sen\phi$. Por lo tanto,

$$x = u \cdot \cos\theta - v \cdot sen\theta$$

Y para *y* se tiene que

$$y = u \cdot sen\theta + v \cdot \cos\theta$$

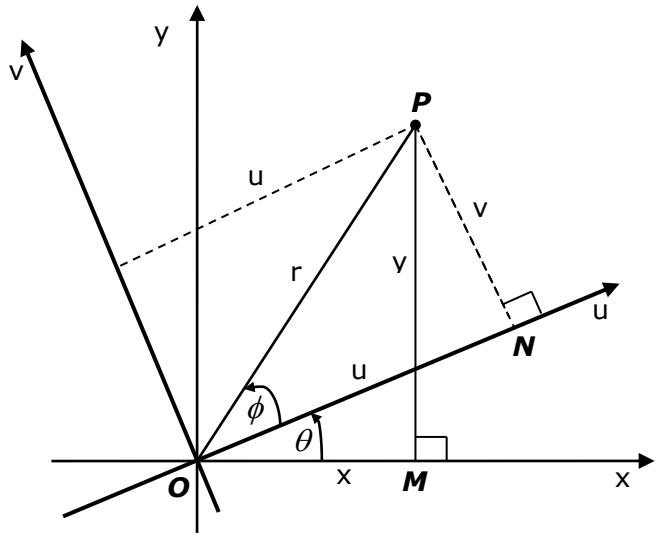


Figura 5. Rotación de ejes.

Estas últimas dos fórmulas definen la transformación denominada rotación de ejes