Programming 3: Tutorial

Introduction

There is a lot to do if you are to learn how to produce exciting DirectX games. There are notes available from my website and there are copies here for you to take home.

Please read them and practice programming DirectX in your spare time, feel free to alter and play around with the code available from the website. Hopefully with the comments to direct you it will help you develop a greater understanding of DirectX.

Although DirectX is not massively difficult to program, it is massive, there is lot of methods that need to be used. This can cause to code to quickly spiral out in many lines.

In this we hope to start to implement a more OO approach to our programs.

To begin this we are going to define some of our structures and classes to modularise some of the development process.
This should help us when we develop different applications as we can "plug in" the different files, reusing them in our different applications.
It should also make more readable, by cutting down the amount in one place.

Remember though, the best way to actually learn is to actually program a Direct3D based game. I would therefore recommend looking at what you need to do to complete a game, trying not to be too adventurous and then having a go. Having a go at this development process will enable you to gain insight in how to complete your assessments and possibly even become your assessment.

For today, it is on with the development:

Header File

First off we are going to code a header a file that will contain much of the basic functionality we always need and that will provide a method of storing information we need without resort to global variables.

Lets `typedef` a `struct` called `_D3DCurrentSettings`
Or if you prefer we can create a class, complete with functions that set and return the information. The structure is easier.

We want the structure to hold certain information that many of the Direct3D methods need.
The information we often need is the height of our device, the width of our device, are we windowed or not, are we going to multisample, our D3DFORMAT and the D3DPRESENT_PARAMETERS.
Remember when coding this that structure will just define the types, it will not actually hold anything yet.

We can then initialise an instance of this structure in our code files.

The next thing to do is to declare functions prototypes (or headers) as seen below:

```
//-----------------------------------------------------------------
// Function Name: InitDirect3DDevice
// Purpose: Encapsulates the setup of D3D
//-----------------------------------------------------------------

int InitDirect3DDevice(HWND hWndTarget, D3DCURRENTSETTINGS
   &d3dcurrentsettings, LPDIRECT3D9 pD3D,
   LPDIRECT3DDEVICE9* ppDevice);

//-----------------------------------------------------------------
// Function Name: ValidateDevice
// Purpose: Ensures we have control of the display adapter in the
//          multitasking win32 environment.
//-----------------------------------------------------------------

HRESULT ValidateDevice(LPDIRECT3DDEVICE9 &pDevice, LPDIRECT3DSURFACE9
   &pBackSurface, D3DCURRENTSETTINGS &d3dcurrentsettings)
```

`InitDirect3DDevice` should perform many of the functions that InitFullScreenDirect3D in the Direct3D_1.cpp code provided from activehelix.com.
`ValidateDevice` should call the TestCooperativeLevel and then perform relevant checks to determine the failure. Appropriate actions should be taken, whether that means exiting, displaying a message or returning the relevant error.
We then need to define the functions for these prototypes in a relevant file. For instance if your header file is named: D3DFunction.h then your source file will be named D3DFunction.cpp

After you have completed that then you should incorporate these into a basic Direct3D program that displays a full screen window in a colour of your choice.

After you have finished that you have a few options:

1. Modify the Looper program so that it only uses one sprite interface pointer to display all the bitmaps. Also, call the sprite Begin( ) function before, and the sprite End( ) function after the calls to the sprite Draw( ) function. (This will improve the rendering speed of the program.)

2. Combine parts of the Explosions program with the modified Looper program so that when a key is pressed the plane is hidden and one of the explosions is shown in its place. Then, when the explosion has finished, start the plane again, just off-screen on the right.

   When the Sound Explosion example is available, add sound to the explosion in the program.

3. Add a gun sprite on the grass background of the Looper program and move it left and right when the left and right arrow keys are pressed, respectively. Position the gun in the layers of sprites so that it appears to move behind the tree on the right and in front of the tree on the left, like the plane.

   Make the gun fire a bullet vertically upwards when the up-arrow key is pressed. If the bullet appears to hit the foliage of a tree, but not the trunk of a tree, the bullet should disappear but the tree should remain. If the bullet hits the plane, both the bullet and the plane should disappear and an explosion should be shown at the position of the collision. When the explosion is finished, the plane should start again, just off-screen on the right.

   Only one bullet may be shown at any one time. If a bullet goes off-screen at the top, or hits a tree or the plane, it ceases to exist and another bullet may then be fired.

4. Look at incorporating a bitmap or sprite onto the screen, either online or the code file provided. The MSDN that houses the SDK documents provide some tutorials.