About how to create a virtual pet

## Preface

First, it is very sorry to say that I am not able to finish the project completely at the last moment, but I want to show my efforts and what I have learned during this process up to yesterday.

There is a very easy toy example in the file attached, which are purely text, with no picture or other visible things. What’s more, there is a 2D version of a little cute dragon which cannot move at the moment. Together there are the framework for the 3D pet which I didnot realize.

## Direct X

To begin with the topic, the DirectX is very useful tool for such things.

The components may be useful for a virtual pet program:

Direct3D: for drawing 3D graphics

Direct2D: for 2D graphics

DirectInput: for interfacing with input devices include keyboards, mice and other game controllers.

DirectPlay: for communication over a local-area or wide-area network.

DirectSound: for the playback and recording of waveform sounds.

## 2D virtual pet

A virtual pet is actually an irregular window which can change its shape based on the time eclipsed or other principles designed by the programmer.

It can be realized by using the API function UpdateLayeredWindow, together with some resource of Flash or bitmaps.

The key part of a 2D pet is how to make part of the context picture transparent. First I know that we can use SetLayeredWindowAttributes function to do this.

### SetLayeredWindowAttributes API function

BOOL SetLayeredWindowAttributes( HWND hwnd, COLORREF crKey, BYTE bAlpha, DWORD dwFlags);

hwnd is the handle to the layered window.(While a pointer literally contains the address of the item to which it refers, a handle is an abstraction of a reference which is managed externally)

crKey specifies the transparent color key to be used when composing the layered window. All pixels painted by the window in this color will be transparent.

bAlpha value is used to describe the opacity of the layered window. It regions from 0 to 255. When bAlpha is 0, the window is completely transparent. When bAlpha is 255, the window is opaque.

dwFlags can be LWA\_ALPHA (0x00000002) or LWA\_COLORKEY (0x00000001). LWA\_ALPHA means that use bAlpha to determine the opacity of the layered window, while LWA\_COLORKEY means that use crKey as the transparency color. If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

But it can only deal with the overall form transparent, partially transparent form cannot be achieved. What’s more, if we use maskColor to realize transparent will cause serrate on the boundary. So if we can set Alpha channel for the window then that would be perfect. (The alpha channel controls the transparency or opacity of a color. Its value can be represented as a real value, a percentage, or an integer: full transparency is 0.0, 0% or 0, whereas full opacity is 1.0, 100% or 255, respectively.) That led us to the UpdataLayerWindow WIN32 API.

### UpdataLayerWindow API function

BOOL WINAPI UpdataLayeredWindow(HWND hwnd, HDC hdcDst, POINT \*pptDst, SIZE \*psize, HDC hdcSrc, POINT \*pptSrc, COLORREF crKey, BLENDFUNCTION \*pblend, DWORD dwFlags);

hwnd is a handle to a layered window.

hdcDst is a handle to a DC for the screen.(A Device Context is used to define the attributes of text and images that are output to the screen or printer. A handle to DC is obtained before output is written and then released after elements have been written.)

pptDst is a pointer to a structure that specifies the new screen position of the layered window. If the current position is not changing, pptDst can be NULL.

Psize is a pointer to a structure that specifies the new size of the layered window. If the size of the window is not changing, psize can be NULL.

hdcSrc is a handle to a DC for defines the layered window.

pptSrc is a pointer to a structure that specifies the location of the layer in the DC.

crKey is a structure that specifies the color key to be used when composing the layered window. We can use a RGB macro to generate a COLORREF.

pblend is a pointer to a structure that specifies the transparency value to be used when composing the layered window.

dwFlags will be one of the three values: ULW\_ALPHA(0x0000002, which means use pblend as the blend function. If the display mode is 256 colors or less, the effect of this value is the same as the effect of ULW\_OPAQUE), ULW\_COLORKEY (0x00000001, which means to use crKey as the transparent color), ULW\_OPAQUE(0x00000004, which means to draw an opaque layered window). If the function succeeds, return true. Else return false.

Considering all the above, to realize a 2D pet, we just need to transfer a different DC to the UpdateLayeredWindow everytime. The function will change the shape of the window according to the bitmap and keycolor of the DC. What we need to do is to change the DC at regular time, then call the UpdateLayeredWindow function.

## 2D vs 3D

For the programmer, the first difference between 2D and 3D is how the data is stored. When considering 2D, the main method to present picture is to store the pre- painted picture into files, and transfer them to the game window when they are needed. But 3D technology regards the body as many 3D objects composed by many multilateral geometry, what stored in the files describes that what are these multilateral and their relative position. To present a body, the program needs to do real time compute due to the description.

The advantage of 3D is that the player can view the world at any angle, the motion pictures need less space to store, the object is easier to modify. The advantage of 2D is that the picture can be very nice in details, and the process can be done at high speed, and a game engine is easy to make.

## 3D virtual pet

The key point of a 3D pet is that we must obtain the rendered picture first, then we can use the function above (UpdateLayeredWindow) to paint it to the computer desktop. First we can create a new RT(render target). Then set the RT of the 3D device to be the new one. The device will clear our render target using a specified color. Notice the color can be specified as D3DCOLOR\_ARGB (0, 0, 0, 0), by doing this we don’t need to create BITMAP and set its ALPHA after we obtain the rendered picture. Create OffscreenPlainSurface, introduce the RT into Surface using GetRenderTargetData. Obtain the DC of the Surface, then show it on window using UpdateLayeredWindow. That will be fine.

The difference between 3D and 2D pet is that 3D render commands a higher change rate of the shape (that depends on the fps). What’s more, the RT to obtain the 3D render is limited to the feature of the hardware. So the first thing to solve 3D render is to solve the problem of change rate.

One method is that we can use software alone to render, not concerning the hardware. Because ordinary virtual pet will not demand too much about the quality of render. And the speed of totally software rendering is acceptable. What’s more, there are some 3D framework realized by software on the internet.

The other method is that to optimize the speed to retrieve RT from hardware. (1)create a RT, render all the pictures to RT. (2)create a offscreen surface, introduce the RT to surface by GetRenderTargetData. (3.1) if we need to combine with the ALPHA of the computer desktop, then lock surface, create a 32bit bitmap, select to mem dc.(3.2) if not(then there might be serrate or not transparent), we can directly use surface->getdc. (4) call UpdateLayeredWindow use the DC above, then we are done.

## Source code and comment:

There is a very easy toy example in the file attached, which are purely text, with no picture or other visible things. What’s more, there is a 2D version of a little cute dragon which cannot move at the moment. Together there are the framework for the 3D pet which I didnot realize.

## other features：

We can use DirectDlay to realize the network feature, to build a pet lobby, to send message to other peoples or other things, which I haven’t learn sufficiently.

## Problems encountered and solved in the past few days.

1. The source code of the books borrowed from the library can be obtained online from media.lib.tsinghua.edu.cn
2. VC++ and visual studio have some difference. For example, the return value of functions must be declared in Visual Studio, or there will be errors, while in VC++ there may not be. What’s more, some header files like the “dinput.h” of DirectX are not included in the visual studio, so we need to add them into the lib and include by ourselves.

“tools”->”option”->”project and solutions”->”VC++ directory”->”lib”/”include”

1. There are some edition difference between the series of DirectX, such as the “dxerr8.h” changed to “dxerr.h”, “d3d.h” changed into “d3d9.h”
2. After we download the DirectX components, we also need to add it into the project we build. We can copy the files we need to the directory where the project is. What’s more, we also need to do like followings: “Project”->”properties”->”linker”->”routine”->”additional library catalog” to write something.
3. The forum on CSDN.net and stackoverflow.com is very useful.
4. To cooperate with someone powerful is of vital importance, especially when you are a fresh man. ~~~~(>\_<)~~~~

## Reference :

1. Visual C++ 游戏开发技术与实例。丁展 编著。人民邮电出版社

2. <http://en.wikipedia.org/wiki/DirectX>

3. <http://msdn.microsoft.com/en-us/library/windows/desktop/ms633540(v=vs.85).aspx>

4. <http://msdn.microsoft.com/en-us/library/windows/desktop/ms633556(v=vs.85).aspx>