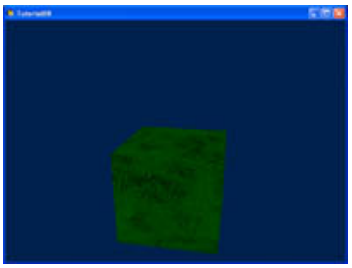
Tutorial 8: Introduction to DXUT

Collapse All



Summary

This tutorial introduces DXUT. DXUT is a layer that is built on top of Direct3D to help make samples, prototypes, tools, and professional games more robust and easier to build. It simplifies Windows and Direct3D APIs, based on typical usage.

The result of this tutorial is a sample that looks just like the previous tutorial. However, it is implemented differently.

Source

(SDK root)\Samples\C++\Direct3D10\Tutorials\Tutorial08

DXUT

DXUT provides a simplified process for creating a window, creating (or selecting) a Direct3D device, and handling Windows messages. This allows you to spend less time worrying about how to perform these standard tasks.

DXUT for Direct3D 10 is also highly componentized. The core component of DXUT contains the standard window creation, device creation, and management functions. Optional components of DXUT include functions such as camera manipulation, a GUI system, and mesh handling. This tutorial explains the DXUT core component. Optional components are explained in later tutorials. Features introduced in this tutorial include device creation, the main loop, and simple keyboard input.

DXUT exposes a wide range of callback functions that the user can hook into. The callbacks are called by DXUT at logical points during program execution. You can insert custom code into the callbacks to build the application, while DXUT manages the window and device management implementation requirements.

DXUT supports applications that contain Direct3D 9 and/or Direct3D 10 code paths. This allows DXUT to pick the best code path for the system the application is running on. However, this tutorial focuses only on Direct3D 10 rendering. For an example of a dual Direct3D 9 and Direct3D 10 application built on DXUT, see the BasicHLSL10 sample in the DirectX SDK.

Setting Callback Functions

Modifying the WinMain function of an application is the first step. In previous tutorials, WinMain invoked the initialization function, and then entered the message loop. In the DXUT framework, WinMain behaves similarly.

First, the callback functions are set by the application. These are the functions that DXUT calls during specific events when the application is run. Notable events include device creation, swap chain creation, keyboard entry, frame move, and frame rendering.

```
// Direct3D10 callbacks
DXUTSetCallbackD3D10DeviceAcceptable( IsD3D10DeviceAcceptable );
DXUTSetCallbackD3D10DeviceCreated( OnD3D10CreateDevice );
DXUTSetCallbackD3D10SwapChainResized( OnD3D10ResizedSwapChain );
DXUTSetCallbackD3D10SwapChainReleasing( OnD3D10ReleasingSwapChain );
DXUTSetCallbackD3D10DeviceDestroyed( OnD3D10DestroyDevice );
DXUTSetCallbackD3D10FrameRender( OnD3D10FrameRender );

DXUTSetCallbackMsgProc( MsgProc );
DXUTSetCallbackKeyboard( KeyboardProc );
DXUTSetCallbackFrameMove( OnFrameMove );
DXUTSetCallbackDeviceChanging( ModifyDeviceSettings );
```

Then the application sets any additional DXUT settings, as in the following example:

```
// Show the cursor and clip it when in full screen
DXUTSetCursorSettings( true, true );
```

Finally, the initialization functions are called. The difference between this tutorial and the basic tutorials is that you have to deal only with application specific code during initialization. This is because the device and window creation is handled by DXUT. Application specific code can be inserted during each of the associated callback functions. DXUT invokes these functions as it progresses through the initialization process.

```
// Initialize DXUT and create the desired Win32 window and Direct3D
// device for the application. Calling each of these functions is optional, but they
// allow you to set options that control the behavior of the framework.
DXUTInit( true, true ); // Parse the command line, handle the default hotkeys, and show msgboxes
```

```
DXUTCreateWindow( L"Tutorial8" );
DXUTCreateDevice( true, 640, 480 );
```

Debugging with DXUTTrace

DXUTTrace is a macro that can be called to create debug output for the application. It functions much like the standard debug output functions, but it allows a variable number of arguments. For example:

```
DXUTTRACE( L"Hit points left: %d\n", nHitPoints );
```

In this tutorial, it is placed next to function entry, to report the state of the application. For instance, at the beginning of OnD3D10ResizedSwapChain(), the debugger reports "SwapChain Reset called".

```
DXUTTRACE( L"SwapChain Reset called\n" );
```

Device Management and Initialization

DXUT provides various methods for creating and configuring the window and the Direct3D device. The methods that are used in this tutorial are listed below. They are sufficient for a moderately complex Direct3D application.

The procedure from the previous tutorial inside InitDevice() and CleanupDevice() has been properly managed by splitting it into the following functions.

- IsD3D10DeviceAcceptable Callback
- ModifyDeviceSettings Callback
- OnD3D10CreateDevice Callback
- OnD3D10DestroyDevice Callback

Because not all resources are created at the same time, we can minimize overhead by reducing the number of repeated calls. We achieve this goal by recreating only resources that are context dependent. For the sake of simplicity, previous tutorials recreated everything whenever the screen was resized.

IsD3D10DeviceAcceptable Callback

This function is invoked for each combination of valid device settings that is found on the system. It allows the application to accept or reject each combination. For example, the application can reject all REF devices, or it can reject full screen device setting combinations.

In this tutorial, all devices are acceptable, because we do not need any advanced functionality.

ModifyDeviceSettings Callback

This callback function allows the application to change any device settings immediately before DXUT creates the device. In this tutorial, nothing needs to be done in the callback, because we do not need any advanced functionality.

OnD3D10CreateDevice Callback

This function is called after the Direct3D10 device is created. After the device is created, an application can use this callback to allocate resources, set buffers, and handle other necessary tasks.

In this tutorial, most of the InitDevice() function from tutorial 7 is copied into this callback function. The code for device and swap chain creation is omitted, because these functions are handled by DXUT. OnD3D10CreateDevice creates the effect, the vertex/index buffers, the textures, and the transformation matrices. The code has been copied from tutorial 7 with minimal alterations.

OnD3D10DestroyDevice Callback

This callback function is called immediately before the ID3D10Device is released by DXUT. This callback is used to release resources that were used by the device.

In this tutorial, OnD3D10DestroyDevice releases the resources that were created by the OnD3D10CreateDevice function. These resources include the vertex/index buffers, the layout, the textures, and the effect. The code is copied from the CleanupDevice() function, but it has been changed to use the SAFE_RELEASE macro.

Rendering

For rendering, DXUT provides two callback functions to initiate rendering for your application. The first, OnFrameMove, is called before each frame is rendered. It advances time in the application. The second, OnD3D10FrameRender, provides rendering for DXUT.

These two functions split the work of the Render() function into logical steps. OnFrameMove updates all the matrices for animation. OnD3D10FrameRender contains the rendering calls.

OnFrameMove Callback

This function is called before a frame is rendered. It is used to process the world state. However, its update frequency depends on the speed of the system. On faster systems, it is called more often per second. This means that any state update code must be regulated by time. Otherwise, it performs differently on a slower system than on a faster system.

Every time the function is called in the tutorial, the world is updated once, and the mesh color is adjusted. This code is copied directly from the Render() function in tutorial 7. Note that the rendering calls are not included.

OnD3D10FrameRender Callback

This function is called whenever a frame is redrawn. Within this function, effects are applied, resources are associated, and the drawing for the scene is called.

In this tutorial, this function contains the calls to clear the back buffer and stencil buffer, set up the matrices, and draw the cubes.

Message Processing

Message processing is an intrinsic property of all window applications. DXUT exposes these messages for advanced applications.

MsgProc Callback

DXUT invokes this function when window messages are received. The function allows the application to handle messages as it sees fit.

In this tutorial, no additional messages are handled.

KeyboardProc Callback

This callback function is invoked by DXUT when a key is pressed. It can be used as a simple function to process keyboard commands.

The KeyboardProc callback is not used in this tutorial, because keyboard interaction is not necessary. However, there is a skeleton case for F1, which you can experiment with. Try to insert code to toggle the rotation of the cubes.

Summary

The functions covered in this tutorial can get you started with a project on DXUT. Tutorials 9 and 10 cover more advanced DXUT features.

To start a new project using DXUT, you can use the Sample Browser to copy and rename any sample project in the DirectX SDK. The EmptyProject sample is a good blank starting point for DXUT applications, or you can choose another sample.