Microsoft's XNA Framework

March 5

2007

A CS 525
Presentation

Michael Schuld - 3/5/2007

CONTENTS

BACKGROUND

XNA Game Studio Express is a new game development solution targeted primarily at students, hobbyists, and independent game developers. XNA Game Studio Express is based on Visual C# Express 2005 and lets developers create games for both Windows and Xbox 360. (Microsoft Corporation, 2007)

Using the XNA Framework and XNA Game Studio Express allow for extremely fast development of games by any of these groups of individuals without all the hassle of needing to know about graphics programming and theory. The framework also supplies a simple to use content building and management component, keeping the art and graphical side of games separated from the code.

The XNA Framework in essentially a wrapper over the DirectX 9 runtime environment, which allows for all of the functionality of programming directly in the DirectX SDK, without as much of the overhead of some of the common tasks that every DirectX application has to deal with. The reasons XNA was built on top of DirectX 9 runtime instead of the much more powerful and optimized version 10 SDK are varied, but the most important reason was to allow permanent backwards compatibility with Windows XP and the Xbox 360.

Since DirectX 10 is only going to be released for Windows Vista PCs, basing the Framework off of those components would have crippled its use greatly, largely due to the fast that XNA was released months before Vista. The adoption rate of the XNA Framework has also been much higher than that of Windows Vista, and keeping all of these users from XNA would have been a major mistake. Also, to ensure Xbox 360 compatibility (and ease of implementation for Microsoft), the DirectX 9 runtime was chosen as the base because the 360 does not support anything newer. Thanks to smart planning in this area, the ability to port code from PC to Xbox can be as simple as changing a pull down menu.

As a project, the XNA Framework is managed and developed by a subset of individuals that have been in charge of major component of the DirectX and especially the Managed DirectX (MDX) world for a long time. This allowed the team to put together a versatile and fully functioning framework in only 6 months (from its inception in one of countless team meetings to final release date).

NEW FEATURES

The XNA Framework supports new integration of some tools that were previously only included as add-ons to other related frameworks (like DirectX and MDX). These tools include the Microsoft Cross Platform Audio Creation Tool (XACT) and the XInput framework.

Along with these tools, the XNA Framework also includes a completely new Content Pipeline that automatically imports and processes many of the basic types of game content that almost every game uses including sounds, shaders, models (in .x and .fbx formats), and also allow for complete extensibility which lets the developer create their own custom content types and their importers and processors.

TOOLS

XACT

The Cross Platform Audio Creation Tool goes to great lengths to make adding sound to games incredibly easy. XACT lets you add files to a sound bank and to organize and control almost every aspect of the sounds and music used. The breakdown of the organization capability is as follows:

WINDOWS AND XBOX 360 SETTINGS

The separation between Windows and Xbox 360 Settings allows the author to create completely different schemes for every setting based on the platform that the content will be run on. This allows for a high level of customization on PCs while still keeping settings more standard for situations involving the Xbox which makes the transition between platforms even smoother.

WAVE BANKS

Wave banks allow for organization of the actual .wav files that will be used in creating Sound Banks later on. This level simply allows for easier use in creating Sound Banks.

SOUND BANKS

Sound Banks are directly loaded in the code of the game, and allow for organizing what content the developers want loaded at what time. They also allow for the creation of cues based on single sounds or combinations of them in tracks. This not only keeps the sounds themselves in manageable groups in memory, but also allows the developers to choose between multiple versions of a sound without having to remember which sounds are similar using complex naming schemes, because they can simply call the different tracks of a single sound.

CATEGORIES

Categories allow for even more specific grouping of sounds into types. There is a Default and Music group in every new project, but custom categories can be added as well.

VARIABLES

Variables are used by every sound in the project and can be set up as either global or specifically applied to single sounds (which affect the actual code use of them.) The

default global variable is the SpeedOfSound which, when changed in the application, will cause all sounds to be played faster or slower. Cue specific sounds include orientation, pitch, instances, and position. These variables allow each sound to control its Doppler shift, volume, and direction to incorporate 3D sound easily in a game and to manage how many times the sound can be playing at once (which helps keep frame rates high.)

RPC PRESETS

RPC Presets allow for control hooks to be created in a sound which let changes to much of the underlying properties of the sound wave itself be changed.

DSP EFFECT PATH PRESETS

Along with RPC Presets, DSP Presets work by setting a large number of physical properties of where a sound is played (ex: In a cave, in a large room, in a hallway) so that different effects like reverb and echo can easily be achieved.

COMPRESSION PRESETS

These allow for different default compression types on the Xbox and PC to keep file size and memory use in check with the available resources on each type of system.

XINPUT

XInput is an API that allows applications to receive input from the Xbox 360 Controller for Windows. Previous uses of the controller on PCs allowed for most of the functionality provided, but there was no interface for accessing the vibration features and some buttons were also unable to be polled for input (Microsoft Corporation, 2007). XInput organizes each of the controllers' input signals into an easily accessible object that allows for full access to all of the available features.

Also included in XInput is the ability to tell whether a controller is still connected, and mouse and keyboard hooks that are very similar to their Managed DirectX counterparts.

CONTENT PIPELINE

Together with the new Content Pipeline, XACT allows users to create and manage sound libraries for their games without the hassle of setting up a lot of application code to process and deal with storage and sound management during game play. This keeps the coders coding and the sound guy making sounds and allows their workflow to no longer be based on one another's progress. On the same note, updates to models and their textures and the shaders that render them to the screen can all be done without affecting any code. The Content Pipeline actually consists of four separate components:

CONTENT IMPORTER

The Content Importer is the first piece of the pipeline that files are run through. It is defined as a generic class that can return any type of data needed later on in the pipeline and can consist of anywhere from a simple file loading operation, to a complicated conversion to some other custom built data type. The content importer allows for the setting of a default processor that it passes its newly imported data in to.

My version of the content importer simply passes out an XMLReader object that reads the file.

CONTENT PROCESSOR

The processor takes the imported data and turns it into an intermediate data type that can be converted to a file format for use on the PC and the Xbox 360. These data types are usually custom components of the game or engine being developed and in some cases are easily converted to a serialized format for writing out to the file system.

My version of the processor converts the XML content being processed by the passed XMLReader into a collection of classes that are used in my game engine. These classes are later read directly into the engine itself in their final form.

CONTENT WRITER

The writer takes the processed data and writes it out to an .xnb file that is recognizable by both PC and Xbox versions of XNA code. These files can be built externally to the main code project and simply copied over old versions if any content files are ever updated.

My version of the writer takes each of the data objects (recursively) created by the processor and serializes them into a binary format that can be written out to file.

CONTENT READER

The reader is what is actually called at runtime from the content pipeline itself. It is automatically invoked based on the type of content being imported.

My version of the reader simply takes the file that was previously written and deserializes (recursively) all of the data contained within.

This process is incredibly simple compared to the old coding methods used by engine writers in the past and it allows for much more flexibility in the way content is handled since different components can be used in combination with each other as long as their input and output data types are the same. For example if one modeling tool outputs meshes at 100 times their needed size, a separate processor could be set up to automatically find and rescale those types of files.

BETTER EXTENSIONS TO OLD FEATURES

Some of the older methods of coding DirectX and even MDX applications were much too complicated to be easily understood and quickly implemented by hobbyist and independent game developers, which is exactly who the XNA Framework was designed to target, so Microsoft took a new stance on how certain parts of the game application would be handled.

THE GAME LOOP

One of the major issues that caused much heated debate in DirectX development was the best way to implement the game loop. This little piece of code was sometimes one of the most important aspects of putting together a game project, and so it required a lot of forethought and experience to know what type of game loop was needed in the right situation. Since no one could really even agree on what was better and when different types of game loops were best to implement, the XNA Team simply included the type they felt most easily allowed for fast and easy game creation without too many complicated features for developers to deal with (which had previously been created by Tom Miller of MDX and used since the June 2005 DirectX SDK.) Here is the basic idea of how this game loop works:

```
public void MainLoop() {
        // Hook the application's idle event
        System.Windows.Forms.Application.Idle += new EventHandler(OnApplicationIdle);
        System.Windows.Forms.Application.Run(myForm);
}

private void OnApplicationIdle(object sender, EventArgs e) {
    while (AppStillIdle) {
         // Render a frame during idle time (no messages are waiting)
         UpdateEnvironment();
         Render3DEnvironment();
    }
}

private bool AppStillIdle {
     get {
        NativeMethods.Message msg;
        return !NativeMethods.PeekMessage(out msg, IntPtr.Zero, 0, 0, 0);
     }
}

[StructLayout(LayoutKind.Sequential)]
public struct Message {
    public IntPtr hWnd;
    public WindowMessage msg;
    public IntPtr wParam;
    public IntPtr lParam;
    public uint time;
    public System.Drawing.Point p;
}

[System.Security.SuppressUnmanagedCodeSecurity] // We won't use this maliciously
[DllImport("User32.dll", CharSet=CharSet.Auto)]
public static extern bool PeekMessage(out Message msg, IntPtr hWnd, uint messageFilterMin, uint
messageFilterMax, uint flags);
```

“The Idle event fires when there's no messages in the queue, and then the handler keeps looping continuously until a message does appear, in which case it stops. Once all the messages are handled, the idle event is fired again, and the process starts over.” (Miller, 2005).

This type of game loop allowed the XNA team to create a very simple Game class that could be extended by anyone that could simply be run with a call to a function. Even though the great debate of the game loop was taken care, the looming problem of device handling was still on the list.

DEVICE MANAGEMENT

The device handling in the XNA Framework is all done entirely by the Game class itself and is no longer something the developer needs to worry about at all (unless of course they want to, then it is all still accessible to them.) Previously, a device would need to be set up based on a scan that DirectX would perform at the beginning of the application, and a set of presentation parameters was constructed. These options were often what were changed when video settings menus appear in games. Parameters would also have to be changed if the device was ever supposed to be able to change from windowed to full screen mode. Now, all of these setup issues are completely handled by the framework, and only have to be changed or set if the developer needs them for an advanced feature later on.

One other big issue with device handling was hat to do when the game window as resized or the resolution of a full screen application was changed. This usually required a resetting of all of the device parameters based on the new screen or window resolution and a reset and reloading of all the graphics content that relied on the device itself to work. Now, the Game class has a simple LoadGraphicsContent function that is automatically called any time these events occur, and the developer only needs to pass the new device to its content objects to be reloaded within that function, and going full screen is reduced to a simple function call. All of the extra features of the device are available through the GraphicsDeviceManager class that is easily created with a single function call as well.

MATH

The math handling in the XNA framework is very similar to that of MDX, but with a few important additions. The old MDX framework added support for many sizes of Vector and Matrix objects, and even had things for complicated Quaternion conversion and rotation representations, but it lacked any built in game based functions and only allowed for the mathematical manipulation of these objects. This allowed for any number of math based features to be included in graphics applications, but only if the developer had previous knowledge of the equations involved. The XNA Framework added in functionality for bounding box collision and intersection math and many programming guides for implementing picking, collision detection, and different camera styles based on the allowed math functionality. Static values for math constants such as pi/4 were also included. These features and all of those allowed by the MDX Framework were included in the first release version of XNA.

STORAGE

The new storage classes in the XNA Framework allow for easy creation, loading, manipulation, and storage of game files by essentially being the entire file system as far as the XNA application is concerned. Using this model, Microsoft was able to create a very secure storage framework that would not allow the Xbox 360 to be compromised by an XNA application and also made cross platform compatibility much easier.

XBOX 360 DEVELOPMENT

The most exciting aspect of the XNA Framework is its ability to be run on the Xbox 360. There are a few things involved in the process of moving code over to the Xbox that are not usually mentioned that are very important for developers to be familiar with before they try to port their games.

The major change that causes most people to move away from XNA development for the Xbox and to stick strictly to PC development is the charge involved. Microsoft has created an application on Xbox Live that can be downloaded to any 360 that will connect to XNA Game Studio Express (the XNA extension to the normal Visual Studio Express) and allow for deploying of the game code to the Xbox hard drive. Downloading this program is free, but running it requires an active subscription to the XNA Creators Club, which can be purchased over Xbox Live as well. This $100 per year charge is enough to keep many hobbyist developers completely away from developing their games for the 360.

Another drawback of XNA on the Xbox is that currently there is no method of sharing games with friends. It is possible to move your games over to another Xbox as long as it has the XNA Launcher program installed, but unless the friend who uses that box has a Creators Club membership, they will not be able to play the game. There are of course rumors of many different plans that Microsoft has under their belt that may be implemented later on, but for now we simply pay the money to work on our own box and get no other benefit from it.

In terms of coding for the Xbox, not much changes from the normal development of a PC game. The only things that the Xbox is specifically missing are support for mouse devices and for shader models above 2.0. You can actually use the keyboard input provided by XInput, because the Xbox allows for USB keyboards to be plugged in. As long as the developer stays within those two boundaries there are no real issues with moving over to the Xbox.

CONCLUSION

Microsoft's XNA Framework was created to allow hobbyist and independent developers get their hands into real game development without needing years of experience and expertise. The inclusion of simple tools and extensible API components has accomplished this task beautifully, and the added ability to move code to the Xbox 360 brings a rush to those groups that was previously unattainable.

BIBLIOGRAPHY

Microsoft Corporation. (2007). *XInput and DirectInput*. Retrieved March 2, 2007, from http://msdn2.microsoft.com/en-us/library/bb173051.aspx

Microsoft Corporation. (2007). *XNA Frequently Asked Questions*. Retrieved March 3, 2007, from XNA Developer Center: http://msdn2.microsoft.com/en-us/xna/aa937793.aspx

Miller, T. (2005, May 5). *Tom Miller's Blog : My last post on render loops (hopefully)...* Retrieved March 3, 2007, from Tom Miller's Blog: https://blogs.msdn.com/tmiller/archive/2005/05/05/415008.aspx