Write once, play everywhere? – write games in XNA for PC and XBOX 360

Learn how to create a 3D graphics engine with XNA 4

Fruit Ninja Games Theory

Developing a cross platform game using XNA

Guidelines for Your First Game Design

6 Lessons on Becoming a Game Producer

Lua as a Script Language for Native Games

Developing CarneyVale: Showtime for Xbox 360 and PC

Python Programming with Great Flexibility and Strong Conventions

Scalability and performance with PHP
GAME DEVELOPMENT GEMS

010 Write once, play everywhere? – write games in XNA for PC and XBOX 360
by Marlos Karagiannis

As a game developer, and especially an indie game developer, the resources needed to target more than one platform are often not available. Although such a possibility would allow games to be available to a much larger audience, it is at difficult to manage and maintain different parts of the same game consistently and in a timely manner. Fortunately, Microsoft has broken this general rule with the availability of XNA, a set of tools that allows just that.

020 Can RTS Ever Truly Work on Consoles?
By Alexander Hinkle

Real-time strategy games, often referred to simply as RTS games for short, have been and still are a staple in the PC video game market. Franchises such as Command & Conquer, StarCraft, Age of Empires, and Sid Meier’s Civilization are all multi-million unit sellers and highly recognizable names for any gamer “in the know.” If game developers can manage to get past the many hurdles that have thus far prohibited console RTS games to be successful, the RTS genre on consoles represents an untapped market with a lot of potential. This article explores the reasons behind why real-time strategy games haven’t been a success on consoles such as the Xbox 360 as they have on PC and whether or not developers should bother porting PC versions to consoles at all. After reading this article you will know the challenges that must be overcome for RTS games to be converted from PC to console and learn ways around them based on examples of previously successful transitions.

030 Developing CarneyVale: Showtime for Xbox 360 and PC
by Bruce Chia

[...] In our case, we developed CarneyVale: Showtime in XNA for the Xbox 360 as one of the projects in the Singapore-MIT GAMBIT Game Lab, a collaboration between the Massachusetts Institute of Technology and the Media Development Authority of Singapore. CarneyVale: Showtime was the winner of Microsoft’s Dream Build Play 2008, a finalist for the Seamus McNally Grand Prize for IGF 2009 and it was also selected as one of the PAX 10 in 2009. Following its success, we ported it to PC for the Games For Windows LIVE service in 2010, using C++ for reasons mentioned in this article.

036 Learn how to create a 3D graphics engine with XNA 4
by Lance Zimmermann

Welcome to my tutorial, with this I hope that you will learn some aspects of creating a video game. I’m Lance Zimmerman and this tutorial was written for someone who has a passion for video games, and would enjoy learning and creating their own. I use XNA 4 mainly because I think it is the best free game creation library you can get for making video games on and for Windows. In this tutorial I will go into some game making and 3D theory, but mostly I will use practical application for teaching. By the end you will have the information needed to create a video game using a generic graphics engine. We build the engine together, and then use it to create a classic arcade game clone. The game will be much like Asteroids by Atari in 1979. If you need more information on XNA 4 go here to see what you need for it. I will supply the 3D models, so all you need is Windows. You may want to keep up to date at my XNA web site.

070 Developing a cross platform game using XNA
by limea tabori

With the quick development of new technologies and the rise of a new culture of gamers, being present on multiple platforms and markets is becoming increasingly important. Both established developers and upcoming indies face the challenges of an ever evolving industry landscape and it’s important to reach out to a wider market by being present on multiple platforms.

In this article I would like to talk about game conversion between PC and Xbox and focus on using XNA in particular. After discussing key points and design considerations to keep in mind I would like to offer a reflection on a personal experience working on a cross platform title, in hope of providing knowledge and inspiration to those who wish to undertake a similar project.

076 Developing for xBox360
by Mikhail Mukhin

After overview of setting up for X360 development and X360 SDK tools, we will go over high level issues with systems (memory, multithreading), certification issues, XBox Live, Marketplace, briefly over graphics, audio, networking. At the end, some miscellaneous (and possibly controversial) practices that I found useful.
Developing CarneyVale:
Showtime for Xbox 360 and PC

Being able to deploy your games to many platforms is always a good thing since the games can reach a larger audience. However, most games are built to target a certain platform and porting it to another platform can be tedious. Even though the Xbox 360 and PC are both Microsoft related, there are still important differences to take note. Depending on your type of game, even through the use of the Unity3d game engine or Microsoft’s XNA Framework, converting the game between the Xbox 360 and PC may not necessarily be simple.

What you should know:
- Basic knowledge on Xbox 360 and PC

What you’ll learn:
- Navigating pitfalls about screen resolutions on Xbox 360
- Converting game input controls from Xbox 360 to PC
- Comparing hardware specifications on PC and Xbox 360
- Understanding XNA for Xbox 360 better

In our case, we developed CarneyVale: Showtime in XNA for the Xbox 360 as one of the projects in the Singapore-MIT GAMBIT Game Lab, a collaboration between the Massachusetts Institute of Technology and the Media Development Authority of Singapore. CarneyVale: Showtime was the winner of Microsoft’s Dream Build Play 2008, a finalist for the Seamus McNally Grand Prize for IGF 2009 and it was also selected as one of the PAX 10 in 2009. Following its success, we ported to PC for the Games For Windows LIVE service in 2010, using C++ for reasons mentioned in this article.

Screen Resolutions

When porting your game to a different device, one of the first things you would probably think about is the screen resolutions. We had to learn this the hard way. While we were developing the Xbox360 version of CarneyVale: Showtime, we started testing our game first with our own PC monitors since we wanted to prototype quickly.

When we actually tested the game on an actual Xbox 360 and a high-definition television set, the first issue we noted was that the screen borders were cut off. We learned that we should place all texts within the Title Safe Area, which is an area that is 80% of the screen resolution, as different television sets have different overscans (Figure 1) (next page).

We continued development with the widescreen high definition television sets we had in the office and made the mistake of creating all the art assets for that and only that television set. Furthermore, we wanted some fonts to have fancy effects so we pre-rendered certain texts as image assets to be used in our game.
Rather late in development, we found out that our game could not be played well on standard-definition television sets because when you play a widescreen game on a non-widescreen television, the XNA framework will letterbox the game. We had not thought about letterboxing before and many of the user interface layouts and crucial game information was designed to be at the edges of the screen, so we also did not have the option to crop the view and the end result was that everything was scaled down in the letterbox. As the standard-definition television sets were already much smaller in size, certain small texts could no longer be seen due to the scaling. On top of that, we had pre-rendered these texts so changing their font size meant exporting all of the assets again, not to mention we had to readjust all the layouts when we changed the font sizes (Figure 2).

Having learnt our lesson about screen resolutions, moving the game to the PC was smoother but posed several questions. We wanted the game to be playable on as many PCs as possible and we did a bit of research through sites like Steam’s Hardware Survey and found that there were still quite a number of people using 1024x768 monitor resolutions. Our game also had good feedback from the casual players who had older computer hardware so we decided we should build the game for a non-widescreen format in order to cater to as many players as possible.

This compatibility issue is something to think about for any conversion between one platform and another. Several PC games deal with it by offering different resolutions for the game where different assets or settings could be loaded depending on the resolution selected. In our case, we were a small development team and our game was a 2D game so we chose not to develop for different resolutions and instead we chose to pillarbox, adding vertical matts to the sides of the game, if it was played on widescreen. Finally, the other consideration we had to decide on was whether the game could be played in non-fullscreen mode and for that, we chose to allow it but not allow the window size to be changed.

**Input Controls**

CarneyVale: Showtime for the Xbox 360 had been designed for the hand-held controller and it utilized the directional pad and two buttons. When we were developing the game, it had been a conscious decision to use a simple control scheme as we wanted to allow people to pick up the game easily. That decision did pay off and the game was even accessible to many casual players.

Furthermore, having a simple control scheme did help when we wanted to port the game over to the PC. The two buttons could be easily mapped onto the mouse left and right click and the directional pad could be mapped to the mouse movement. Sounds easy, right? At first glance, it definitely seemed straightforward but as we developed the game further and started testing it, we realized it was not that simple as it seems due to the differences between how a directional pad or analog stick works and how a mouse works.
The first important difference to realize is that a directional pad or analog stick always returns to the center when it is released after pressing, whereas when you move your mouse, since it’s physically moving, it never returns to its original spot once you when you stop moving.

The second difference was that on a directional pad or analog stick, you could hold down the button for as long as you wanted, but if you mapped that with the mouse movement, you cannot get the same effect because a mouse has a limited space to move. For example, if you were to control a character in a platformer game with mouse movement, you would have to move your mouse and then lift it up and bring it back to other the edge and then move your mouse again and repeat that in order to get to other end of the screen. And on top that, the character would stop when the mouse is lifted.

The third difference was that every player can have a different mouse setting on their operating system whereas a directional pad or analog stick will always produce the same values by any two players as long as their hand movement is the same. On the PC, even if the two different player’s hand movements are the same, the output values would be different if the setting was different (Figure 3).

These issues were not game breaking for us but it did affect the level design and the game balance greatly as they frustrated players if they had to keep lifting up their mouse. Furthermore, the third issue was a double whammy for us since we could not just tweak it to fit one mouse setting. We had to spend a significant amount of time just tweaking the game and comparing various ways to use the input, such as choosing between the relative motion or the absolute position of the mouse cursor. We ended up choosing the relative motion and we also had to lock the mouse to the center of the game screen.

One final note about converting input controls from PC to Xbox 360 is that your game should support multiple controllers on the Xbox 360. Even if you are making a single player game, you still need to check which controller is connected on the Xbox 360 because when controllers are synchronized with the Xbox 360, they are assigned a controller index and that index may remain even after all the devices are switched off. In other words, when a player picks up a controller, they might not necessarily be assigned to the first controller index even if only one controller is turned on. So in your Xbox 360 version, you cannot assume that people are using the first controller index.

**Minimum Specifications & Dependencies**

PC developers are aware that you have to consider a large range of hardware, ranging from different CPUs, RAM, GPUs, and even hard disks, as well as considering the software such as the operating systems and libraries such as DirectX, OpenGL, .NET Frameworks. Therefore when developing for PC, it is always important to set the minimum specification from the start of development and test on that configuration throughout development to help you discover issues early.

Console developers on the other hand, only have one hardware configuration to deal with and a limited set of software libraries to use. In the case of Xbox 360, its CPU is a custom triple-core IBM designed Xenon as its CPU, each core able to run two hardware threads, for a total of six hardware threads available to games. Its GPU is an ATI Xenos which supports for a superset of DirectX 9.0c API and Shader Model 3.0+. It also has 512 MB of GDDR3 RAM clocked at 700 MHz.

One of the reasons why we chose to redo the game in C++ on PC was because the XNA framework has quite a number of requirements. The first was that it would only run on Windows XP Service Pack 2 and above, the second was it required the user to install

![Figure 3: Mouse controls – The difference between sliding the mouse and pressing a key](image)


.NET Framework, the third was the XNA Framework installer itself. Being new developers on the PC, one mistake we made was that we were not aware of the silent install option, where these installers could just run by itself without user input, so we thought it was a very tedious process if the user had to click through so many installers.

We also started development just before Windows 7 was released and many people were still using Windows XP back then due to Windows Vista unpopularity, so we thought it would be a good idea to develop the game to support Windows XP. However, by the time we were done with the game, not only did we have to fulfill the Technical Checklist Requirements for Windows 7, we also realized many gamers had moved on to it already. Perhaps the lesson there is that the minimum specifications you decide could also factor in the development time you will take, because the average computer specifications will be different by the time you are done, nevertheless predicting the future is always hard.

Another reason why we chose to redo our game in C++ when porting the XNA version to PC was that we had agreed with Microsoft that we would integrate their Games For Windows LIVE service. At that time, the service was brand new and it was only available through C++ code and not XNA. It is uncertain if this has changed by now, but at that time, that meant we could not use the service in XNA, so developing in C++ made more sense (Figures 4, 5).

For development on the Xbox 360, knowing the exact hardware capabilities and limits is very helpful as a developer because you can optimize your software to fit it well. For example, you can write shaders or parallelize your game code without worrying about the player not being able to run them. Debugging is also much easier since you do not need to own multiple different hardware configurations to test. Yet, debugging can be harder if problems only occur on the Xbox 360, especially only in release builds since the debugging symbols are not available. Finally, console makers impose many restrictions in place to prevent hackers. For example, the Xbox 360 will not allow you to modify the folder containing the executable, called the Title Storage, once your game has been installed so you cannot create a malicious executable that modifies itself.

**XNA tips for Xbox 360**

XNA is based on the .NET Compact Framework, which is a subset of the .NET Framework and therefore you may need to redo code that uses the APIs which were not included if you are converting your game from PC to Xbox 360. Furthermore, some virtual function calls may be expensive and can be optimized by manual inlining code. In addition, keep in mind that you are only allowed to execute pure managed code on the Xbox 360 and you cannot invoke a non-managed DLL.

When using threads on the XNA, it is important to note that out of the six hardware threads available, the first and third are reserved for the XNA Framework. This means that you are effectively left with only four threads for your game.
The Xbox 360 has a relatively large amount of RAM and many small games can actually fit entirely on the RAM so loading everything beforehand could be useful if you are facing optimization issues.

Being in a managed environment does help to simplify things but it does not mean you can ignore allocations and deallocations of memory for objects. Most games would want to run at a smooth frame rate so triggering the garbage collector as infrequently as possible is always a good idea and that means you would have to avoid allocations of memory at run-time. It is also crucial to understand that conversion from value types (C# struct) to reference types (C# class instance or Object) will result in allocations on the heap in a process called boxing and converting the other way is known as unboxing. These operations can be costly if it occurs often enough.

The Xbox 360 supports many different peripherals such as guitars, arcade sticks and many more and all Xbox 360s come with a hard drive for downloading games and storing user files. Although most developers probably would not bother with most of the accessories, one of the accessories to keep in mind is the memory unit. The memory unit is not a standard item that comes with the Xbox 360, however it is one that deserves your attention especially if your game has save files. The reason being that unlike other peripherals which might just give very strange input values to your game, your game may crash if the memory unit was not handled. There are two main ways to deal with a memory unit that is plugged into an Xbox 360. One way is to allow the player to select it and another way is to choose to ignore it and only support the hard drive. Either way, you will need to add the code to handle the scenarios accordingly. As a best practice, it is recommended that all games should give the player the option of selecting the memory unit or the hard disk, however considering the number of people who actually use memory units, perhaps it may or may not be worth the trouble.

**Conclusion**

Developing a game for Xbox 360 and PC has its own set of challenges. On the console, you will want to push the hardware to its limits but for the PC, generally you need to cater to a wide number of hardware configurations so you will want to avoid using the latest technology or providing multiple options for different users. Keeping in mind the pitfalls from the start will help you develop better games faster.

---

**About the author**

Bruce is the co-founder, programmer and audio designer of Ludochip, a company putting together interesting and different game mechanics. Prior to starting his own company, Bruce was the lead programmer in the Singapore-MIT GAMBIT Game Lab and he led the teams of programmers for the Xbox 360 and PC versions of the game CarneyVale: Showtime.

**Contact** bruce@ludochip.com