Introduction

This is a series of tutorials where we'll be building a simple real-time physics engine, and a simpler 3D engine, in C# Visual Studio 2005 Express.

This is mostly a tutorial on OO design with eXtreme Programming ideas rather than a detailed tutorial on C# programming or implementing a physics engine, but it does cover a bit of everything.

I'm writing these tutorials just because I thought it would be good for me to share the knowledge and experience I've picked up from implementing my own very simple engine. I'm not working in the games industry, this is very much from an enthusiast point of view. Of course, if you spot any errors or anything I could be doing better than I'd be very glad to receive feedback.

There are lots of tutorials on the internet, and good books, about physics engines, but most of them only include code snippets and don't provide full working examples. This is good as you should really implement an engine to fit your application, but for the beginner getting to grasp with learning about building an engine, having a simple working example would make understanding the concepts much easier.

Who is this aimed at?

These tutorials assume that the reader has some programming knowledge, but not necessarily object-oriented – it's aimed at the most typical users of Visual Studio 2005 Express, so they're probably going to be new to C# as well. It won't be written quite like a programming tutorial though – I won't hand-hold the reader through e.g. what a delegate is; this is more of a learning resource by demonstrating where the language features would be used.

Similar for the physics – the user could just ignore the physics and use this as a programming tutorial, or it could be that they're interested as a reference Physics implementation. At least the unit tests should be useful when implementing your own engine!

What will we be doing?

First, I'd best explain what the aim is. I want to produce an engine that calculates the movement of rigid (and maybe later soft) bodies with force acting on them, in real time. This is pretty vague – I'm not aiming to get a million bodies moving at 120fps on a 486, but on quite modern hardware I want to get an engine that will demonstrate the physics, and maybe good enough to write a very simple game against.

The main engine's goal is to be simple. The code should be structured so that it's easy to understand, which gives the benefits of being easier to maintain and extend.

Different to many other tutorials out there, I'll be including much of the *boring stuff* like exception handling, the folder structure of the projects, and following best practices. Many tutorials don't include this stuff so as not to distract the reader's concentration from the topics they're demonstrating. However, I want to build a proper working engine and illustrate how a full application fits together.

I'll be following eXtreme Programming (XP)/Agile practices, with much refactoring (and may include a lot of back-tracking – I haven't written much of this upfront, so I'm not sure

where it'll go either). I'll also be writing lots of unit tests, and try to demonstrate why they're so useful.

I'm also keen to explore multi-threading, to see what effect this has on performance (on multi-core processors, it'll be a good excuse to upgrade my computer!)

A physics engine on its own isn't very interesting, so I'll be building a very simple 3D engine to go along with it, implementing it in OpenGL using the TaoFramework.

The rough outline of the steps we're going to take are:

- Build a very very very simple 3D *engine*. A basic 3D app so we can actually see some physics.
- Get a sphere moving under force. Introduce unit tests.
- Get a sphere moving with torque.
- Maybe here look at different integrators (might do this later though).
- Sphere collision.
- Implement other objects (collide with themselves).
- Objects of other types colliding.

And after that I'll see what looks interesting.

Why bother?

Why a physics engine? Why not? Seriously, to me it's one of the more 'fun' uses of a computer – real-time 3D following physical laws is just cool! Why C#? Aren't most engines written in C++? Well, yes, but this is more of a learning engine – C++ looks much more hideous for the beginner. It will be good to investigate just how C# performs for something like this. Also, I prefer C# - I'm much more productive in it, and don't generally like spending hours searching for intermittent crashes due to memory corruptions.

Why OpenGL? Well, it's cross platform. I've never wanted to learn Direct3D as it seems to be too much of a lock-in to Windows. This may seem to conflict with using C#; I'll generally be taking advantage of features of the .NET framework, but the algorithms in and the structure of the engine could be quite easily converted to C++/Java. And hopefully Mono (the open source .NET Framework implementation) will become more widespread.

Background knowledge

Physics

There are lots of resources on the internet for creating a physics engine. Have a read of:
http://www.pixar.com/companyinfo/research/pbm2001/notesg.pdf
http://www.d6.com/users/checker/dynamics.htm#articles

- Physics For Game Developers (David M Baraff, O'Reilly, ISBN 0596000065) may be a good guide if your basic physics knowledge needs a boost – but it's a bit too basic for creating an engine.
- Game Physics (David H Eberly, Morgan Kauffman, ISBN 1558607404) is much more comprehensive, and does provide enough info for building an engine (there is an engine somewhere on his website). The problem I have with this book is that it's written much too formally, where I already understood topics that he was covering I was amazed how complicated he made the explanations, and where I came across something new to me, I just used its presence in the book to figure out what I needed to learn, and went elsewhere to have a read up on it.

- Numerical Methods For Physics (Alejandro L. Garcia, Pearson US Imports, ISBN 0139067442) is an excellent book. Well recommended for anyone studying physics, or interested in numerical methods.

Much of the OpenGL code comes from:

- NeHe's excellent tutorials, at http://nehe.gamedev.net/
- OpenGL Superbible (Richard S. Wright, Michael Sweet, Waite Group, ISBN 1571691642).

Programming

Agile Software Development (Robert C. Martin, Pearson, ISBN 0135974445) is an excellent book, the examples of refactoring to patterns are more useful than in some other refactoring and test driven development books that I've read.

And now enough of me waffling, let's get hold of any dependencies we need and get prepared to code!

Getting Started

Now we'll go ahead and get the dependencies we require to start writing the engine.

From http://mono-project.com/Tao download Tao-1.2.0.2-win32.zip

Extract the contents of the zip file to a folder somewhere on your system.

Testing the installation.

First we want to test one of the examples to see that everything works OK on our machine. Attempt to run one of the examples in the examples folder e.g. NeHe.Lesson02.exe.
You'll get an exception:
File not found – Tao.Platform.Windows.dll or one of its dependencies.

So go ahead and copy Tao.Platform.Windows.dll into the examples folder from the bin folder.

Then you'll get a file not found for Tao.OpenGL.dll, so go ahead and copy this into the folder. Similarly for Tao.OpenGL.Glu.dll.

Everything should now be running ok, so in the next tutorial we're able to go ahead and do some coding.