GAME CAREERGUIDE

PRINT

What is a Game Engine?

By Jeff Ward [04.29.08] If you follow the game industry, you'll hear the term "game engine" thrown around a lot. And I bet sometimes, late at night when everyone else has gone to sleep, you sit and wonder, "What is this nebulous thing I keep hearing about?"

You would expect that the answer would be as simple as being shown a car's engine: "Yeup, thar she is." After all, the game engine, much like a car's engine, is what makes the game go. Unfortunately, sometimes there's a fuzzy line between where a game's engine ends and where the content of a game begins, as if there were a fuzzy line between whether a car's air conditioner is part of its engine.

Generally though, the concept of a game engine is fairly simple: it exists to abstract the (sometime platform-dependent) details of doing common game-related tasks, like rendering, physics, and input, so that developers (artists, designers, scripters and, yes, even other programmers) can focus on the details that make their games unique.

Engines offer reusable components that can be manipulated to bring a game to life. Loading, displaying, and animating models, collision detection between objects, physics, input, graphical user interfaces, and even portions of a game's artificial intelligence can all be components that make up the engine. In contrast, the content of the game, specific models and textures, the *meaning* behind object collisions and input, and the way objects interact with the world, are the components that make the actual game. To use the car analogy again, think of how the body, CD player, in-dash navigation system, and leather seats make the actual car. That's the content.

APIs and SDKs
Two other terms you hear in the game industry that are closely related to game engines are "API" (application programming interface) and "SDK" (software development kit). APIs are the software interfaces that operating systems, libraries, and services provide so that you can take advantage of their particular features. An SDK is a collection of libraries, APIs, and tools that are made available for programming those same operating systems and services. Most game engines provide APIs in their SDKs. The Unreal Engine, for example, provides an interface for programmers to create their games, both through a scripting language called UnrealScript, and through libraries, which are provided to anyone who licenses the engine, and which come in the same package as their other tools, like the editor UnrealEd.

But where do game engines come from?

Birth of a Game Engine

For a long time, many game companies made their own game engines and kept that technology in house, iterating on it as computers improved and more advanced versions were needed. Engines like SCUMM by LucasArts and SCI by Sierra, for example, powered most of the adventure games that those companies released in the late 1980s and into the mid 1990s. More recently, engines like id Tech (the engine that powers the *Quake* series of games) and the Unreal Engine started as in-house technologies, though they have recently evolved into middleware technologies as well.

Over the past several years, the cost of making an in-house engine has grown significantly, and more and more companies have begun to specialize in making either full game engines or game engine components to sell to other companies, rather than make games. We call these kinds of companies middleware providers. The middleware providers can offer these products at very reasonable prices, and, for most game development studios, this creates a very clear "build versus buy" decision. Why pay six programmers for a year to build an engine when you can buy 90 percent of the features you want from a proven technology for less money -- and have it immediately? As a result, almost all components of a game engine are purchasable at a variety of prices, or downloadable in the form of open source projects.

Types of Game Engines
Game engines come in many different flavors and at many levels of programming expertise. To get a feel for how different they can be, I'll explain three kinds of engines: the roll-your-own version, the mostly-ready version, and the point-and-click engine.

Roll-your-own game engines (lowest level). Despite the cost, many mainstream companies (as well as indie game makers) will still attempt to roll their own engines. This means they use publicly available application interfaces, such as APIs like XNA, DirectX, OpenGL, the Windows and Linux APIs and SDL, to create their own engines. In addition, they may use other libraries, both commercial and open source, to help them along the way. These libraries might include physics libraries like Havok and ODE, scene graph libraries like OpenSceneGraph, and GUI libraries like AntTweakBar.

I include XNA and SDL here because, although they make creating an engine much easier by abstracting away some of the more nasty platform implementation issues, they still require a lot of programming to get a game off the ground. They're not really engines so much as good starting points for creating your own engines.

Generally, these home-rolled systems give programmers the greatest amount of flexibility, letting them pick and choose the components they want and integrating them exactly how they want. But they also take the longest amount of time to build. Additionally, programmers frequently will have to build the tool chain from scratch, since they can rarely rely on all these libraries to work together straight out of the box. This makes rolling your own engine less attractive to most game developers, even the professional ones.

Mostly-ready game engines (mid level). I consider most game engines to be "mostly ready." These engines are ready for prime time right out of the box, with rendering, input, GUI, physics -- you name it. Many of them even have mature tool chains so you don't have to roll your own. Engines in this category include OGRE and Genesis3D, which are both open source, low priced engines like Torque, and even really high priced ones such as Unreal, id Tech, and Gamebryo.

To varying degrees, all these engines still require a bit of programming to get them up and running into a complete game. They might call for some scripting or sometimes even low-level coding to get a real game working. Mostly-ready game engines are a bit more limiting than roll-your-own engines and are frequently optimized for the general case. That said, many of these engines are the product of dozens of people's work over hundreds of long hours, and will provide better performance with less effort than most roll-your-own engines, even if they don't do exactly what you want.

Point-and-click engines (highest level). Point-and-click engines are becoming more and more common these days. They include a full tool chain that allows you to point and click your way to creating a game. These engines, which include GameMaker, Torque Game Builder, and Unity3D, are built to be as friendly as possible, and are made to require as little coding as possible. That's not to say knowing a little coding doesn't help, but it isn't really a necessity the way it is for the mostly-ready and roll-your-own engines.

The problem with many point-and-click engines is that they can be extremely limiting. Many do one or two types or genres of game well, or one or two types of graphics modes. This is not to say they're useless. Even faced with the restrictions of these tools, it's possible to make highly creative games or even find creative ways around those restrictions. The best thing about these engines is that they allow you to work quickly, and play your games quickly, without too muck work. If you're just starting out in game design, you could do worse than these tools.

For More Information
There are many resources for finding game engines. Wikipedia is generally the best resource for engine information, and it lists all the known open source engines. The pages are frequently updated and frequently give feature comparisons:

- http://en.wikipedia.org/wiki/Game_engine
- http://en.wikipedia.org/wiki/List_of_game_engines

These pages also provide a list of games that use more common engines.

Jeff Ward is a co-founder of and lead programmer at Orbus Gameworks, a provider of metrics and data-gathering tools for game developers.