Game Engines: How do they work? (/profile/michaelenger/blog/game-engines-how-do-they-work/101529/)

By michaelenger June 20, 2013

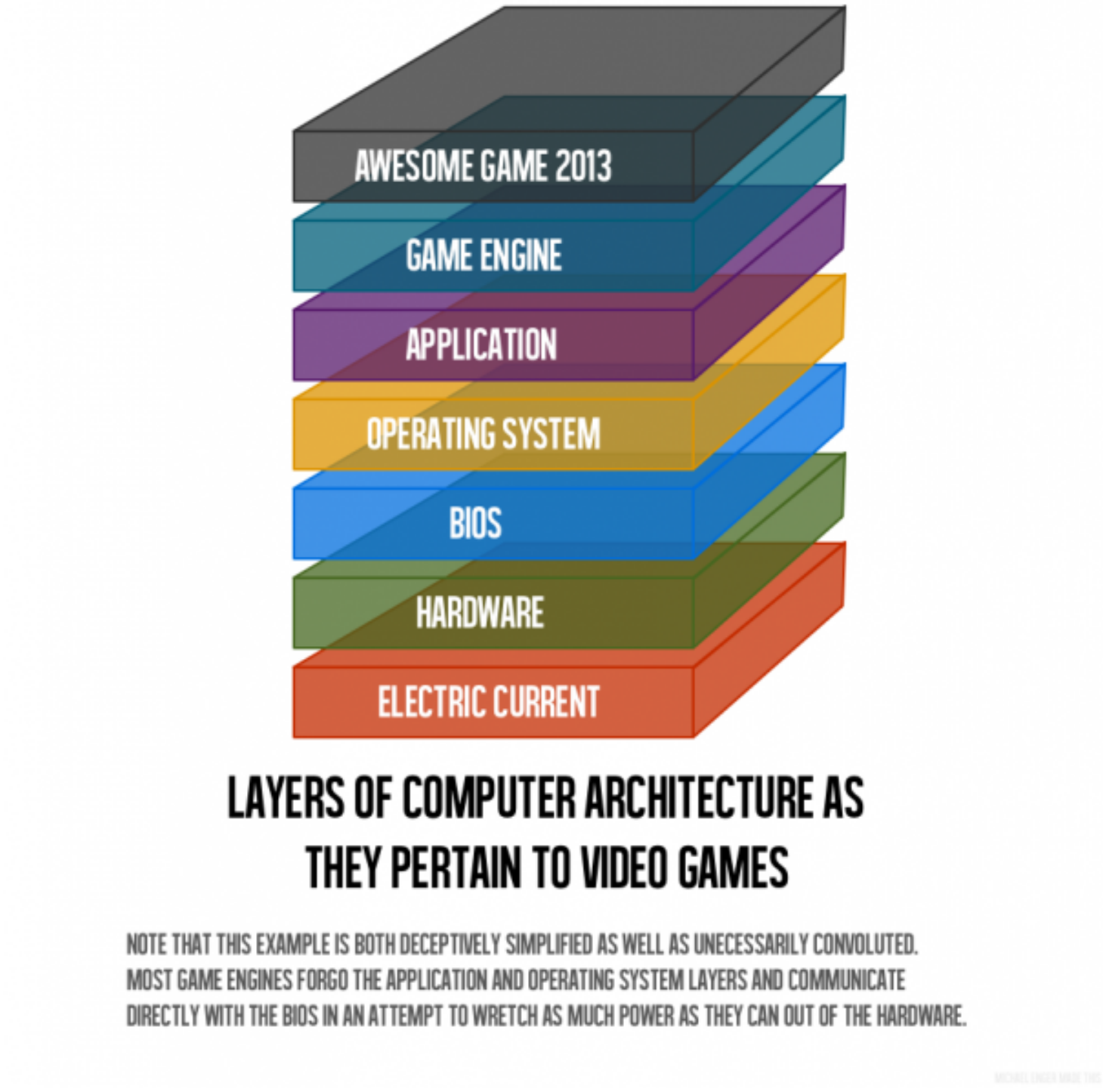
2 Comments (/profile/michaelenger/blog/game-engines-how-do-they-work/101529/#comments-block-1441684)

Game engines are great things, able to take the weight off developing a game idea to let you focus on the idea itself. Powerful engines like the Unreal Engine (//www.giantbomb.com/unreal-engine-3/3015-86/), Source Engine (/source-engine/3015-751/) and indie-darling Unity3D (/unity-engine/3015-5683/) are examples of great tools built by people who want to make games bigger and better. Game engines provide developers with a slew of components and helpers they can use to build their games faster and with less hassle, but the most important factor games engine provide are interoperability between the various gaming systems available. Game engines are amazing, awesome things, but how do they work?

*Note that this article will focus on 3D game engines as they are the most proliferated and technically impressive. 2D game engines are for indie game hipsters who don't know how to use orthographic projection.**

**That was a joke, albeit a terrible one.*

Computers: How do they work?



Disclaimer: this is not a technical diagram

To understand how game engines work, we first need to look at how computers in general work as a set of systems. Computers work on a principle which can be described as "layers of abstracted complexity", which just means that everything in a computer is built atop something complicated which has been abstracted to be easy to work with. In its barest sense, a computer is a machine which uses patterns of fluctuating voltages in an electrical signal to do arithmetic, but it would be impossible to get anything done on a computer if you needed to think about this every time you wanted to build something.

To give you an example of how stupidly complex a computer is, I'll try to describe the various layers in the most basic terms: Down in the depths of a computer's hardware is a powered circuit which is manipulated so that the voltage in the circuit changes. This circuit passes through transistors which interpret the changing voltages into a predictable signal of either off or on (0 or 1). This way the

voltage, which would look more like a wave if measured directly, turns into a pattern of 0s and 1s which are interpreted in the layer above as binary patterns. These binary values can be strung together in "words" to form commands that the computer needs to understand. In turn, those commands are then bunched together in processes which can do even more complicated things like manipulating memory storage and sending/receiving signals from peripherals connected to the computer system. Combine enough processes and you have an operating system capable of being programmed to perform wondrous tasks to entertain and educate, illuminate or obliterate.

These layers are the foundation of computer science, with each layer having its own fields of specialization (and archetype of geek), each building on top of each other and working together to make the modern all-purpose computer function. Standing atop this pillar of complexity are the high-level languages, the easy to read and write scripting languages which power the likes of web browsers and game engines. Thanks to the layers of abstractions you can tell the game engine to draw a 3D character inside a room without having to worry about what electrical signal you need to send to the screen to draw the correct pixels.

If you want to know more, the people over at Computerphile post regular videos on how computers work: Computerphile on YouTube (https://www.youtube.com/user/Computerphile)

Components of the Modern Game Engine

Game engines are complicated sets of components which provide a lot of useful features for making games. Unlike general development frameworks, like Cocoa Touch (https://developer.apple.com/technologies/ios/cocoa-touch.html) (for building iOS applications) or .NET (http://www.microsoft.com/net) (for building Windows applications), game engines are made specifically for creating games and have all of their components organized to do just that, to the detriment of other forms of applications. To compensate for lacking easy tools for building menu bars and widgets, game engines have graphics engines optimized to be as fast as possible and instead of using default popup windows and system sounds they contain sound engines which place sounds in 3D space.

Input

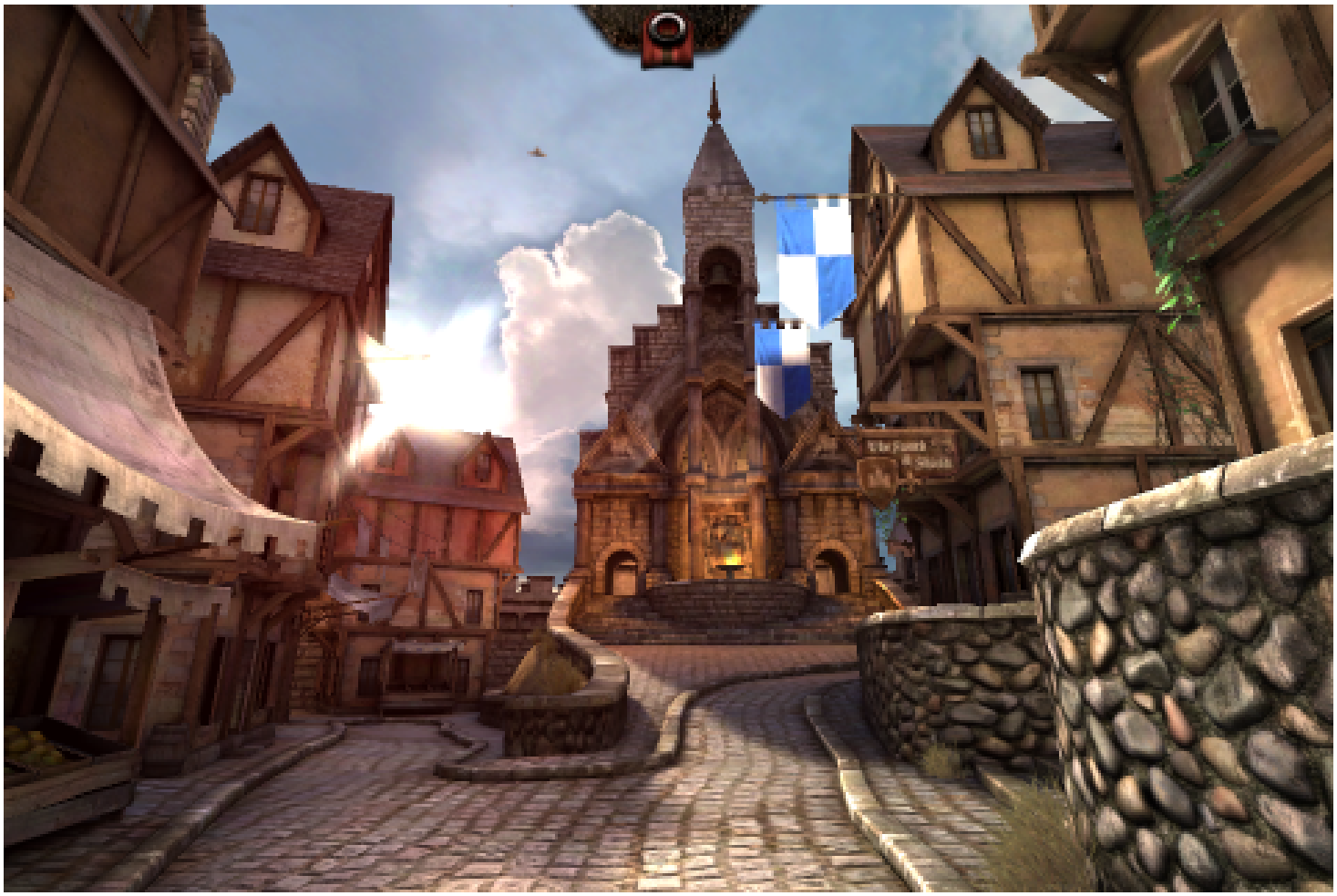
One of the most important aspects of a game is the means to play it, so game engines usually support an array of input types: keyboard, mouse, gamepad and touch are the mains ones and any less-common input methods (joystick, steering wheel, rollerball, multi-touch) being subsets thereof. There are many different ways to handle input, but there are two common means: events and polling.

Input events work by the computer listening for some form of input (mouse button pressed, keyboard key released, joystick axis changed, touch pressed) and triggering your custom code. This can be combined with a "mapping table" which will connect keyboard/controller/mouse buttons to named actions, such as "jump" or "shoot", so that you can build your code without having to worry about the user wanting to play using a different layout than the one you build your game around.

Polling is usually done when it comes to position values, such as the x/y coordinates of the mouse or the amount of tilt of a gamepad's analog stick. The game engine provides the means to retrieve these values whenever the developer wants to and it's up to the developer to react to changes in these

values, whether it be moving a character or changing the position of the custom mouse cursor.

Graphics



An example of what the Unreal Engine 3 is capable of

Also, what would a game be without cutting edge graphics? A major selling point of game engines (especially high-profile ones like the CryEngine (/cryengine/3015-1724/)) is the impressive graphics that they can power, usually combined with the ease of production. 3D games are built around 3D assets which are usually created in an external 3D rendering program, like Maya (http://www.autodesk.com/products/autodesk-maya/overview) or Blender (http://www.blender.org/), and imported into the game engine. Game engines which support a lot of import formats wear the fact proudly, allowing game developers to work in the program they are familiar with and import it to a functioning game without having to jump through hoops.

Once the asset has been imported, you can add it to the game you're building together with bump maps, specular/translucent materials and shadows to create a believable object. Game engines also feature a slew of lighting technologies and effects, which give life to the assets you've added, as well as handling the animation of said assets, including crazy things like blending animations to transition between running/jumping/shooting in a believable manner.

Describing all the graphical features that game engines provide would take thousands of words, but in essence game engines are all there to make your task as simple and straightforward as possible. Developers don't want to deal with converting their carefully crafted 3D models to cryptic formats, or manually building meta-data to show them properly. Game engines do their best work when they take

your creative output and spits it out on the screen without (too much) hassle. This, combined with post-processing effects, terrain building and particle effects means that you can create an entire game world inside the game engine by combining assets from various sources.

Sound

Sound is also an integral part of games, despite being overlooked most of the time (unless the sound is terrible, in which case it's game-breaking). Adding sound effects to games isn't as straightforward as one would think, especially with the advent of 3D games.

Sound effects usually don't just come out of your speakers as they were recorded, but most game engines have the means to place sounds inside the 3D world which will modify the volume depending on where your character is relative to the sound. There are also a lot of ways to improve a sound's realism by adding pitch modulation and reverberation to make it seem like the sound is bouncing off the walls of its surroundings. Take for example the sound of clashing swords out in an open field versus down in the depths of a dungeon and how it adds to the atmosphere if the sounds reflect the world around them.

Music and GUI sounds work differently, as they are added without respect to a 3D position, but rather are played as if the sound comes from inside the players head. Obviously, the engine needs to provide the means to adjust the music to fit the mood of the game, rather than just blaring it out on full volume.

Networking

It has been decades since games started featuring online multiplayer and in we are in the midst of a social gaming phenomenon which wants to connect all your gaming adventures with your friends. This requires as lot of logic revolving around communicating with different servers or even other client computers, which is a complete nightmare to handle manually. Thankfully, most modern game engines provides a lot of pre-built components and helper scripts which do most of the heavy lifting, allowing you to work on the responsiveness and fun of the multiplayer rather than worry about the intricacies of TCP/UDP traffic.

Physics

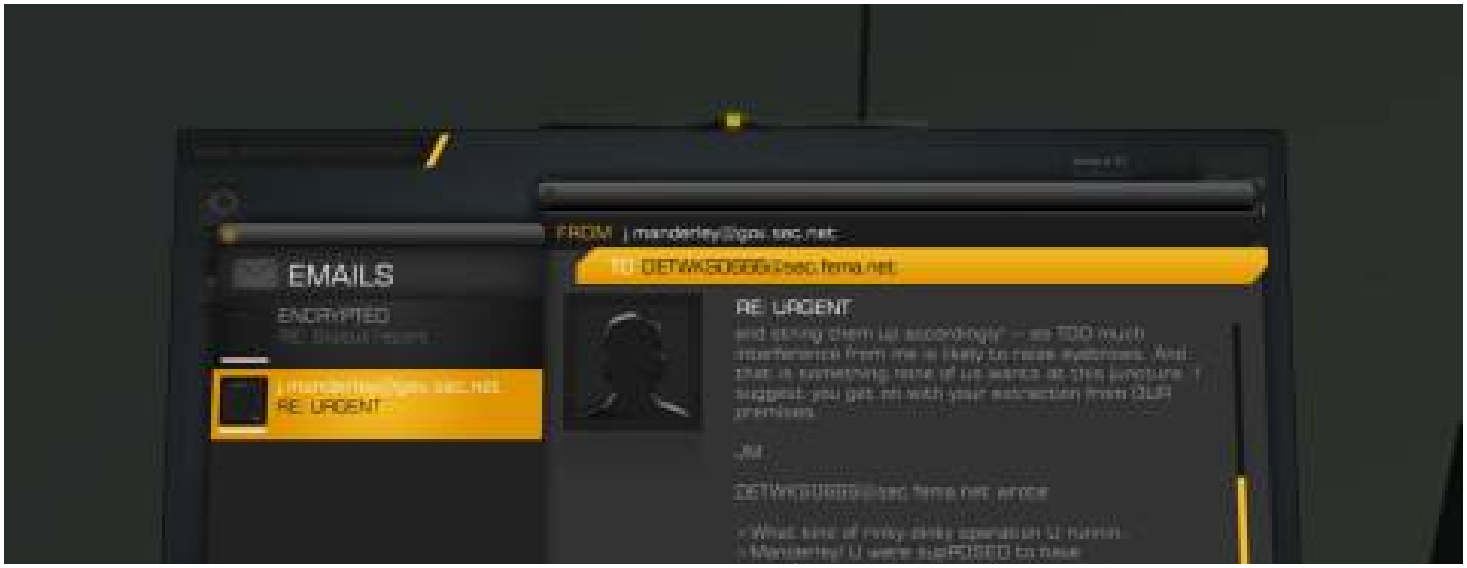
Valve Software's Source Engine had "realistic physics" as one of its major selling points when it was first released. Now all modern game engines come with some means to simulate real-world physics.

It wasn't long ago that realistic physics wasn't really a selling point of games, with most of them relying on their own crazy interpretation on how the laws of physics works (anybody remember rocket-jumping?). Now there are more physics engines than you can shake a stick at, the biggest ones being Havok (/havok-physics/3015-502/), Box2D (//www.giantbomb.com/box-2-d/3015-7935/) and PhysX (/physx/3015-1923/), which are interwoven into game engine to handle the complicated math needed to realistically simulate the real world.

One thing that needs to be understood is that physics isn't an integral part of rendering 3D game worlds, despite modern game engines combining the two so that the end-user doesn't have to be aware of the distinction. When you render a cube in a game, it's just a visual effect, perhaps combined with light refraction and bump maps to give it a sense that it really existed in the world, but there is nothing inherent in the cube which says that it has to adhere to physical laws. Physics needs to be added to the cube for it to react to gravity or being pushed/shot by the user. The cube is given a physical shape, which may not be the same as the visual shape, as well as mass, friction, bounciness and other properties to create an object which can interact with the world around it.

Handling physics is a costly process ("costly" in this case referring to the amount of processing power needed) and only adding physical bodies to the objects that need to react, as well as making their physical shape less complicated than the visual one, is a simple way to make games run faster. This is the reason why you can slide along a row of trees as if they were a flat wall, because the computation needed to determine your collision with every single tree trunk is a waste of time for the game when you should just be following the corridor rather than exploring anyway.

Graphical User Interfaces



The GUI isn't just the main menu.

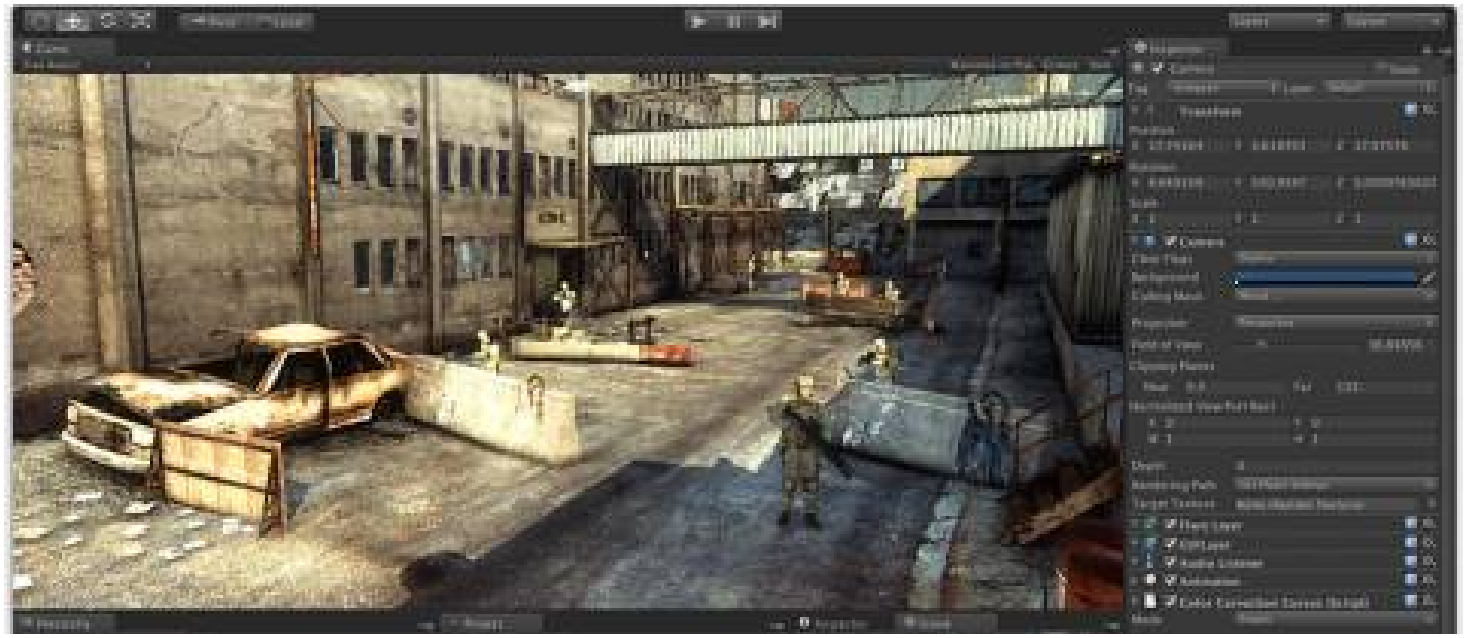
Despite not really being capable of building the typical user-interface of a Windows program, with a menu bar and floating windows, game engines tend to feature at least rudimentary GUI capabilities. Games tend to have their own custom GUI to fit with the style of the game, so providing a standard UI isn't really as important as giving developers the means to build their own custom buttons, drop-downs, sliders and such by combining textures, colors and events.

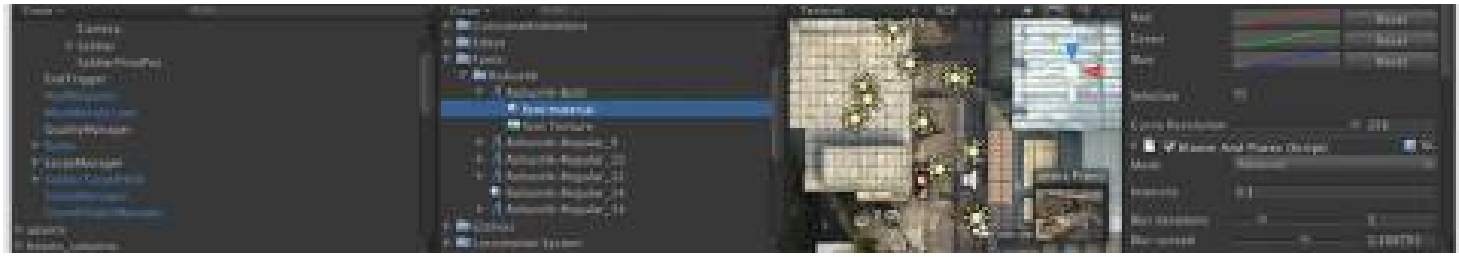
Different game engines handle the problem of GUI differently, with some ignoring the issue altogether, requiring the developers to build the functions manually. This isn't exactly the hardest thing to do, as a GUI is pretty much just a list of text/images which can be clicked on or selected using the keyboard/gamepad.

Scripts

Another **huge** part of game engines is pre-build scripts which can be attached to objects in the game world. The terminology differs from engine to engine, some using the term "behaviors" while others talking about "game objects" differing from "visual objects", but in the end it boils down to one simple thing: someone else did the work for you.

Most game engines come with scrips for initiating a game with the player in a specific position, adding and moving cameras, starting/stopping particle generators, manipulating lights, triggering events when a player moves into an area and a whole slew of other features. Some cases it can be as simple as adding a 3D model in the game world and designating it as a 3rd-person player character, and the game engine will have pre-build scripts for moving and animating the character, rotating the camera and having the world react to the player's position.





The Unity Engine is built around the idea of attaching components to objects in the world to adjust their behavior, whether it be giving the player control over them to adding complicated AI

Artificial intelligence is a big part of the scripts/modules/behaviors/whatever that game engines provide, with the most complicated ones giving you the means to dictate how characters react by building a tree of behavior nodes rather than having to write hundreds of lines of code. The "scripts" that a game engine provides comes down to how much it does for you, which can be a great help to get a game finished quickly, but can also create a problem if you want to do something outside the provided functions.

Benefits and Drawbacks

One of the biggest benefits of game engines is how they provide developers with tools for building games so that they don't have to reinvent the wheel. From handling the low-level graphical optimizations necessary to get a good FPS rate to importing common asset formats, game engines essentially do the "grunt work" of game developing so developers can focus on the atmosphere, story and other factors important to creating a good game.

This is also one of the biggest drawbacks of game engines, as they homogenize the games which are built. A game engine built for first-person shooters may not be the best to use for a racing RPG and your choice of engine may in the end hamper your creative expression. One typical example is how all Unreal Engine 3 (//www.giantbomb.com/unreal-engine-3/3015-86/) games have the same visual "feel", despite wildly different styles and they all struggle with texture pop and a style of sound effects due to the technical limitations of the engine itself.

However, in the end this may not matter because game engines provide something that is an absolute necessity in the modern world of desktop/mobile/console gaming: platform interoperability. The idea is that you build the game atop an engine and can export your game to a variety of platforms, depending of how many the game engine supports. The benefit to developers is hard to down-play, as you'll be building a game once and then having them available on multiple platforms with the press of a button. Whether the game is optimized for that platform (touch vs controller vs keyboard) is another question, so ubiquitous interoperability is a double-edged sword which can result in terrible ports if left in uncaring hands.

Conclusion

Game engines are great! They give developers tools to build games quickly and efficiently while hiding the hard parts involved in building games for the various platforms. Even if you're only aiming to build a game for one specific platform, game engines can give you a boost which will get your game out faster and with more features than if you were to build the whole thing from scratch.

There are a whole myriad of game engines available for the picky game dev nowadays, with more being added every time someone decides that the ones provided just doesn't scratch their particular itch, so there is no chance of running out of options. They differ in quality, feature set and price, so it's just a matter of picking your preference and getting to work on your game.

2 Comments (/profile/michaelenger/blog/game-engines-how-do-they-work/101529/#comments-block-1441684)

2 Comments Refresh (/profile/michaelenger/blog/game-engines-how-do-they-work/101529/?
First to Last (/profile/michaelenger/blog/game-engines-how-do-they-work/101529/?comment_sort=m.dateCreated&comment_direction=ASC) Latest (/profile/michaelenger/blog/game-engines-how-do-they-work/101529/?comment_sort=m.dateCreated&comment_direction=DESC)
comment_page=1&comment_sort=m.dateCreated&comment_direction=ASC)

Posted By igorchernakov (/profile/igorchernakov/) - 1 year ago

> **game engines essentially do the "grunt work" of game developing so developers can focus on the atmosphere, story and other factors important to creating a good game**

Well, there are some developers, who know what they're doing, yes - but mostly those are newcomers with **no skills whatsoever** who are using *Unity in pair with Asset Store* to quickly ship another turd to the market, usually involving a deceived investor and false promises.

Posted By michaelenger (/profile/michaelenger/) - 11 months ago

@igorchernakov (/profile/igorchernakov/): It's funny to think about that when I wrote this blog post two years ago Unity Asset Flipping wasn't a thing. Now it is and it's a damned shame. The Unreal Engine is free now too and has its own asset store, so I wonder if any of those sleazy developers will just ditch Unity now that they've poisoned the well and instead move onto an engine which doesn't have the same stigma.

Please Log In (https://auth.giantbomb.com/login/) to post.

michaelenger's blog stats

BLOGS CREATED:

19 blogs

MOST COMMENTED BLOG:

Has the Xbox One Already Lost in Europe? (/profile/michaelenger/blog/has-the-xbox-one-already-lost-in-europe/105342/)

TOTAL COMMENTS ON BLOGS:

70 comments

michaelenger (Michael Enger)

Send michaelenger a private message (/pm/compose/?to=9456)

michaelenger has not linked a Twitter account.