Unit 70

Mark Woods

South Essex College

Unit 70

Computer game engines

Contents

1. Introduction
2. Types of game engines
3. 2D game engines
4. 3D game engines
5. Mobile game engines
6. Game mods
7. Engine examples
8. Engine examples
9. Components: Rendering: Shadows
10. Components: Rendering: Ambient occlusion, reflections, Global illumination, Anti-aliasing
11. Components: Rendering: Anisotropic filtering, Tessellation, Water
12. Components: Collison detection, Scripting, Sound/Video, Animation, Shading/Lighting
13. Components: Networking, Physics, AI, Level editor
14. Conclusion, References

**Understanding Game Engines**

**Introduction**

This report is going to explain and expand on video game engines, as well as giving examples of different game engines. I will discuss the functions of components, purpose and comparison of game engines.

A computer game engine is the building block of a video game and they allow the developer to create and edit video games. For almost as long as video games have been popular a game engine has been necessary to the creation of a video game, the elements and features that a game engine provides to a developer include physics, rendering, sound and AI. One of the main benefits of game engines is that there are multiple types of games that can be created with the same engine.

**Types of game engines**

There are three main types of game engines that are used in the industry and they allow for the development of different types of games for different types of platforms, these game engines include:

* 2D Engines
* 3D Engines
* Mobile Engines
* Game modifications

**2D game engines**

2D engines are designed to develop 2D games like **platformers** and **top down shooters**, usually these engines are used to create games for the mobile platforms but there are still plenty of 2D games that are on PC and Consoles. The performance that you can get from a game made on a 2D engine is usually better because the engine is not as intense on the **hardware**, because the 2D engine does not have to worry about the third dimension and axis it means that designing and coding the game is much easier which allows 2D engines to be a great place for new developers to learn about game design.

2D engines incorporate different components including:

* Algorithms
* Artificial intelligence (AI)

A popular 2D game engine is Gamemaker which was used to create games like Hotline Miami, Angry birds and Flappy bird. The benefit of using Gamemaker so that it is quite simple to learn and it gives developers the option to focus on **sprite** creation and use **drag & drop** coding or they have the option to use the Gamemaker language (GML) which is fairly simple compared to other **game engine languages** but allows developers to learn the basics of coding including **Arguments**, **functions, loops** and **statements**. One of the algorithms used in Gamemaker is called the **A-star (A\*) algorithm** which allows for objects to avoid walls and dangers and seek out goals and enemies.

**3D game engines**

3D engines are designed to develop 3D games like First person shooters **(FPS)** and Third person adventure games, 3D engines have become the most popular type of game engines because of the vast amount of games that are able to be developed using them, some 3D game engines like the Unreal Engine are capable of making 3D, 2D and mobile games which helps them reach a wider audience but other 3D game engines don’t support 2D games. These type of engines can be tougher on hardware but unlike 2D engines they allow for high quality rendering, animations and physics which are necessary for most triple A titles nowadays.

3D engines incorporate different components including:

* Real-time graphic rendering
* 3D Sound emulation

A popular 3D game engine is the Source engine developed by Valve in 2004 using the C++ language, when developed it was known for its physics capabilities which incorporated weight and gravity to give the game the first most accurate physics engine of its time. This engine helped in the development of popular games like Half-life 2, Counter Strike, Dota 2 and Left 4 Dead 2. Another benefit of the source engine is that it was developed to not be tough on hardware allowing it to have good performance on most consoles and PCs. 3D sound in games has become something that almost all games have but when it was first introduced it was very unique, in the game engine it simply consists of having the audio be similar to the players and objects in the way that they can be moved around 3D environments freely to create the ambient, close and distant sounds which can specifically in creating horror games.

**Mobile Engines**

Mobile game engines are most commonly 2D game engines but the main difference between other engines is that they have built in components and capabilities that allows for specific mobile controls like touch screen, shaking and tilting. These features within the engine allow for mobile games to be made distinctly different to other games which is good for the uniqueness of the platform but can also be bad because the games cannot easily be ported between different engines because the control schemes change drastically. To allow for tilting and shaking controls the mobile uses an accelerometer which checks which way the phone is facing and moving as well as how fast it is moving, this data is used by the game engine to control the game by checking if the player is tilting or shaking the phone in a certain way.

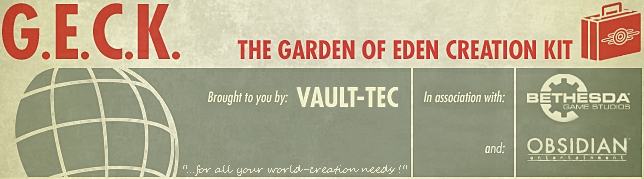
Mobile engines incorporate different components including:

* Touch screen
* accelerometer tilt controls

Mobile engines are the newest type of engines of the three which means that the feature are quite unique, an example of a mobile game that uses the accelerometer well is super monkey ball. The aim of the game is to control a monkey in a ball through various levels which rely on the players’ balance of the phone to get through successfully. Engines made for mobile games are often 2D based but sometimes a game might include 3D game features but this can come at accost of the game performance.

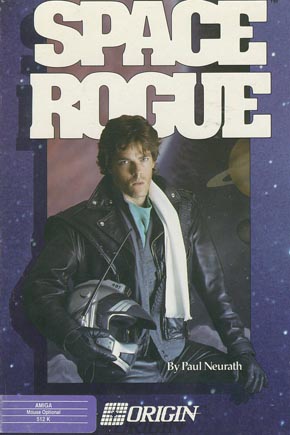
**Game mods**

Game modifications are not one of the three engine types but they are closely tied to them because they allow for the base code of a game to be changed, this is often done by players that feel like adding, changing or fixing the game to make it more enjoyable. Good examples of this are games like Minecraft and Fallout.

* Ever since Fallout 3 the franchise has decided to develop a kit alongside their game which the player can freely use to alter their game files how they please and even share them online with other players. Bethesda chose to do this because they thought it would help give the players a reason to keep playing the game and they wanted the player to have the opportunity to show what they can do with the tools provided.
* Unlike Fallout Minecraft was not initially developed with a kit to help the player alter the game files instead the modifications added t Minecraft started because people thought that it would be a simple and rewarding game to mod. Since Minecraft was built using Java it was very easy for players to access the code and alter for themselves, eventually the developers noticed the mods being made for Minecraft and decided not to stop people from doing it instead they helped by making the code and files of the game easier and simpler for people to access as well as creating User Interface (UI) to help players configure mods.

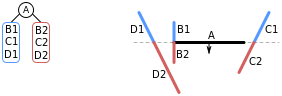
**Engine examples**

**Space rouge**

****

When the first video games were being designed for game consoles they did not use game engines instead they would be made from scratch which meant that a very difficult part of game creation was implementing the base code that allowed for sound, **sprites** and possible **AI**. The first 2D video game engine was the Space Rogue engine released in 1989 which was named after the game that Origin Systems developed, The main feature of the Space Rogue engine was that it could **texture map** which allowed developers to have the engine do more work by itself and not have to write code to allow other games to have the same elements.

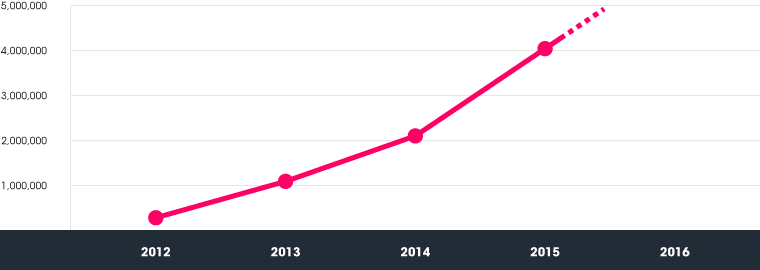
**Quake**

****The Quake engine was a popular 3D game engine that was named after id Software’s game Quake, it was released in 1996 and allowed for a lot of new features to be implemented into the Quake game. It was popular for its focus on running smoothly so that it would not have problems with a computer’s processing units, A popular way that it made games easier to run was a technique called **Binary Space Partitioning** (BSP) that rendered maps that purged where the player could not see which resulted in less polygons being used, Quake also took advantage of **Gouraud shading** and a **static lightmap** for non-moving objects within the game.

**Unreal**

Unreal is a popular video game engine and a variation of it is still used today for a range of **triple A** titles, When first developed it was called UT (Unreal Tournament) and was given a build number for each version. This specific engine is free to use which allows up and coming developers to learn about their engine, they do this because it helps to grow the general game development community and it means that if they are employed then they already have a good grasp of unreal and may even understand features of other engines from using unreal. As well as being widely available it is also designed to for use with 3D, 2D and mobile games.

**Unity**

Unity is a popular game engine that allows for 3D, 2D and mobile games, it is often used as a mobile engine for many developers and it includes optimization features like occlusion culling, asset bundling and build size stripping. Unity was developed in 2005 by Unity technologies and since then has climbed to over 4 million users, popular mobile games that have been developed using unity include Beacon, Strikers edge and polyball.

**Registered Unity developers - 2012-2015**

**Understand the function and purpose of components of game engines**

**Graphic rendering**

Graphic rendering is basically how the computer generates the image on the screen, there are two types of rendering; real time rendering and Offline or pre-rendering, game engines use real time rendering whereas pre-rendering is used in films and animation. Real time rendering means that the computer cant guess what the player is going to do so it must render everything on screen as the player is playing the game, this can be hard on some systems and often requires a good Graphics processing unit (GPU) because the game must be rendered at least 20-30 Frames per second (FPS) otherwise it can look choppy but most games these days render at 60 FPS.

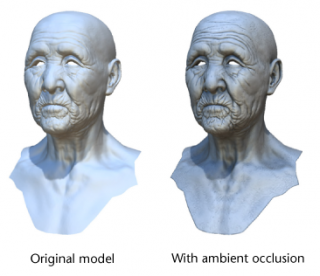


Games like Crysis are made to look realistic and appealing to customers and this is achieved using the graphic rendering component of the engine which is regarded as the most important part of the game engine as a whole, Two popular Application programming interfaces (API’s) that are used are OpenGL and DirectX, these are known for their capabilities for graphic rendering in games. Graphic rendering incorporates many features including:

* Shadows

Graphic rendering controls the shadows within the game, shadows are made by having a "second camera" where the light source is which takes images called "Shadow maps" in each pixel of this image the distance between the light source and the object that the light is hitting is recorded. This shadow map is then used to calculate where the shadows should be drawn which is the job of graphic rendering.

* Ambient occlusion

Ambient occlusion allows for more detailed lighting, specifically between two objects that come in come in close contact with each other e.g. clock and a wall. It works by calculating the brightness of a pixel on an object in relation to environmental light, if two objects are close enough they will have smooth shadows rendered onto them which adds depth to an environment or object.

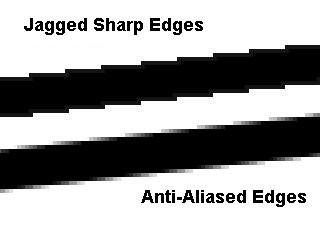
* Real time reflections

Reflections in video games are calculated using ray trace renderers which follow a ray which is sent from the players' camera to the reflective surfaces, the renderer then calculates where it bounces to and it continues to bounce until it no longer finds any surfaces or any reflective surfaces. The surfaces that the rays hit can be different which in turn changes the way the ray bounce off the surface, these types include: Polished, Blurry, Metallic. This data is then used in the rendering process to give the game realistic reflections.

* Global illumination

The goal of global illumination (GI) is to imitate real life lighting, similar to reflections its function is to follow light as it bounces and disperses from objects that are in contact with a direct light. The algorithm works by calculating the brightness, colour and direction that light comes off of objects, this means that in a scene where a green ball in a white room is hit by light it can reflect light as well as the faint colour of the ball onto the walls.

* Anti-aliasing

Anti-aliasing is the process of smoothing out angles that look jagged or have the "Staircase effect". For example when an image appears on the screen of an angled wall the pixels will be diagonal to each other which is what gives it the jagged look, what anti-aliasing does is attempt to fill in the gaps between the pixels to make the line or wall seem smoother and more realistic. Types of anti-aliasing include Multi-sample anti-aliasing (MSAA) and Fast Approximate Anti-Aliasing (FXAA).

* Anisotropic filtering

When textures in video games are not given a filter it means that a texture that is close up to the player can appear fine but a texture a little far off can appear blurry and badly detailed. Anisotropic filtering is the process used to counter this, it does this by reducing the "jagged" edges in textures and making texture that are further away appear more detailed and less blurry. Other filtering techniques include bilinear and trilinear filtering which do a similar job to anisotropic filtering but are less advanced and less powerful.

* Tessellation

Tessellation is similar to bump maps in the way that it can give an object the look of depth without taking away much performance. It works by increasing the amount of quads in the model which allows for more detail with the curvature, bumps and gouges in the model. An example of a popular game that has used tessellation is Grand Theft Auto 5, it was used to increase the detail of the games textures without taking too much performance, and the amount of performance taken by using tessellation was an average of 2 FPS.

* Water

Water has come a long way in video games, it started out in 3D video games as a transparent texture with moving waves which ran across the top giving the illusion of waves, the only real problem was that it seemed the waves had no depth and they didn’t interact with walls or objects that they ran into. Nowadays water is much more advanced and in high end video games small particles are used to simulate the waves which collide and interact with each other to create realistic waves in the water.

**Collision detection**

Collision detection refers to the engine function which detects the intersection of two given objects like spheres, planes, tubes and polygons. The response of a Collision can be used to implement actions into the game, for example a player could collide with an enemy which can then in turn cause an action which kills the player and removes one of their lives, this same collision detection could be used for a ball to rebound of a wall or floor after colliding with it as well as losing some momentum. 2D engines like Gamemaker often have simpler collision detection which is easier to make collision precise. An example of bad collision detection is The Elder scrolls V Skyrim, Collisions in a 3D game are a little more difficult to get precise so problems can occur, in Skyrim NPCs can often clip into objects and enemies or characters that have died do not have collisions with their ragdoll so it means that the player can walk through them.

**Scripting**

Usually the scripting uses an interpreted language like Python or C++ but the engine can be implemented with custom code and scripts to help with algorithms or just add necessary functions to make game development easier. Does not require the source code and does not change any engine code, is also is Game Specific.

**Sound/video**

This engine is usually used in conjunction with an event for example if an enemy is hit or killed a sound will be played to represent that. The video part of the engine is often used for cut-scenes or possibly as part of an in-game object, this component of the engine also handles the 3d sound in the game.

**Animation**

This handles all the animations within the game and tells the renderer how to display the assets, it also handles in-between animation where frames are added in-between animations to make the transitions look better and smoother overall, and rig based animation where a character is rigged with joints and limbs which are then controlled on an axis to make the animation.

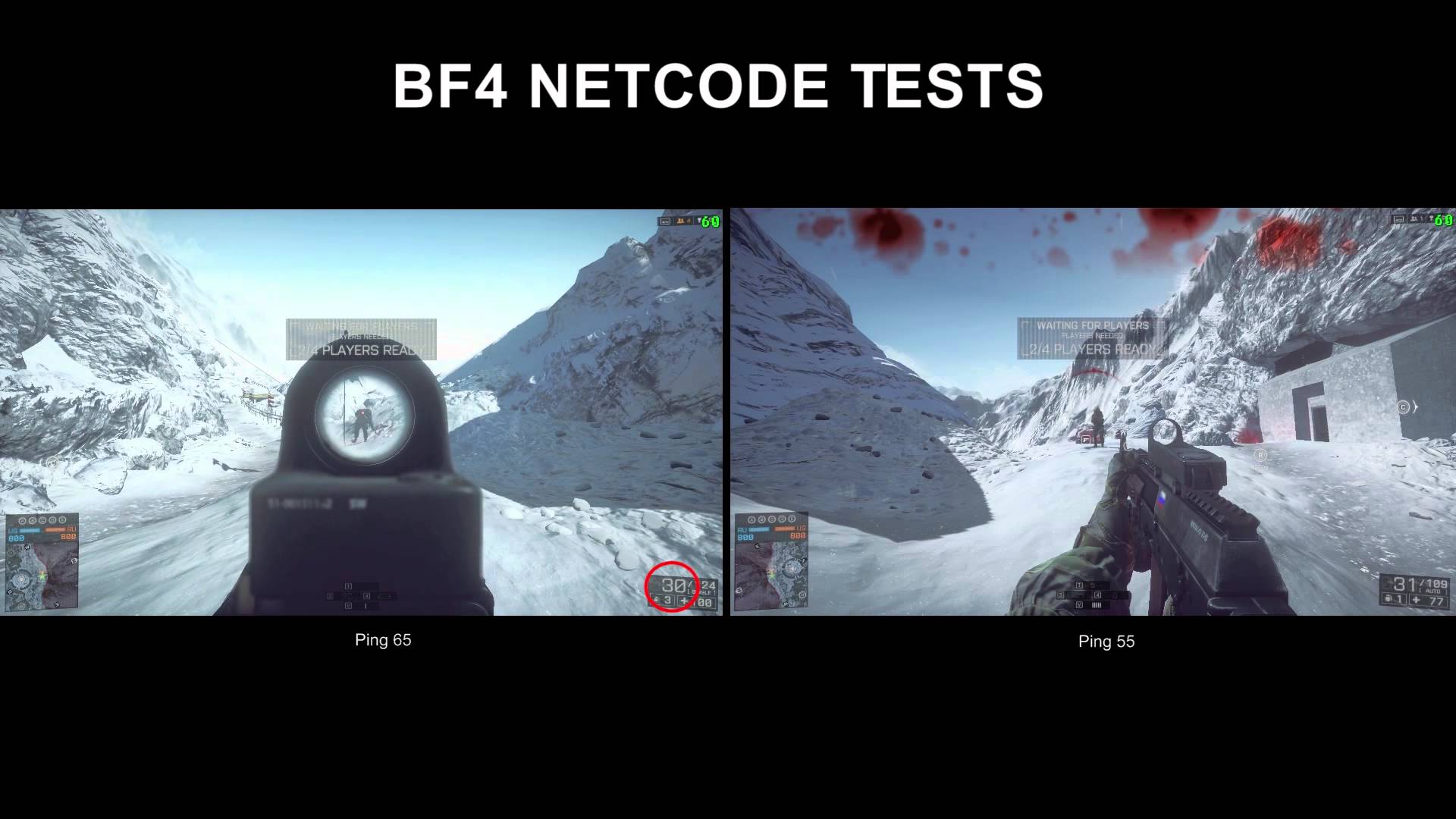
**Shading/lighting**

Special effects within the engine are controlled by this component including shadows, Normal maps and Particle effects. This is an important component of the engine because it gives the game a more realistic look if that is needed but it also allows for textures that may otherwise be bland and plain to be very different and more effective at attracting the player.

**Middleware**

Middleware is software used to connect two separate applications, this type of software is very useful for many real world applications but within games development it relates to game engines as a whole. The purpose of a games engine is to bring other software, applications and data together so by definition it is middleware. Each component of a game engine that I have previously discussed comes together to form the middleware.

**Networking**

Networking components within game engines are most commonly used with online multiplayer games and don’t apply to offline or single player focused games because they do not need have good networking capabilities. Games like CS: GO and Battlefield are a good example because they require good networking code (Net-Code) to have an enjoyable experience for players online otherwise things can happen that don’t seem realistic or fair for the player for example in previous Battlefield titles the Net-Code was mocked by players because when playing they could often be shot from behind cover because the Net-Code was behind what the player sees making the game feel unresponsive and the overall game look bad.

**Physics**

Not all game engines need physics components but most modern engines have built-in physics capabilities, this component allows for games to have realistic gravity, collisions and ragdolls. Games like GTA 5 which are very open and “Sandboxy” require good physics components within the engine to allow for the vehicles and NPCs to properly interact with the world with things like crashes, jumps and rag dolling. Having good physics components can be tough on the hardware because simulating physics for multiple objects is difficult but some games need that ability to be more immersive for the player.

**Artificial intelligence**

AI or Artificial intelligence is a huge part of all popular engines used because it gives the opportunity for games to have intelligent NPC’s or enemies. Games like Alien isolation use Artificial intelligence very well by having the enemy track the player in a smart and unpredictable way which adds challenge to the game and make it more immersive for the player. Algorithms are often used to calculate an AI’s movement, a good example is Fallout 3’s way of implementing information about the surroundings of the AI, and this is done by having a Navigation mesh or NavMesh covering the floor of game environments that the AI then read to tell them where obstacles and walls are. Artificial intelligence uses other components like collision detection to help with understanding the situation of the game like the player position and the obstacle positions.

**Level editor**

The level editor of a games engine has become a more and more necessary thing for game engines as they have advanced, they are used to map out and create environments for 2D and 3D games and they allow for easy and quick creation of an environment. A good example of this is the Unreal engine which has a built-in level editor that allows the user to drag and drop objects, material and “Actors” straight into the environment, Level editors also allow for the developer to see the environment in ways that are similar to what it will look like in game which helps with time management because they do not need to compile and run the whole game just to test a environment that they are working on.

**Conclusion**

Overall engines have become a very important part of game design over the years and the types of engines have expanded and will continue to expand with the arrival of Virtual Reality and Augmented Reality hardware along with new motion control methods. New components and features will be invented and implemented beside all the already available ones that will continue to improve driving technology and gaming in new and exciting directions.

**References**

|  |  |
| --- | --- |
| **Game engines collection** | <http://eact-tech.com/> |
| **Gamemaker studio logo** | <http://store.steampowered.com/app/214850/> |
| **Half-Life 2 screenshot** | <http://gametronic.net/half-life-2-the-orange-box.html> |
| **Accelerometer demonstration** | <http://www.indodaily.ga/news/accelerometer-in-phones> |
| **Fallout G.E.C.K** | <http://wiwiki.wiwiland.net/index.php/GECK_:_Aide_aux_moddeurs> |
| **Space Rouge Cover** | <https://upload.wikimedia.org/wikipedia/en/1/14/Space_rogue_cover.jpg> |
| **BSP Diagram** | <https://upload.wikimedia.org/wikipedia/commons/thumb/8/84/Example_of_BSP_tree_construction_-_step_2.svg/282px-Example_of_BSP_tree_construction_-_step_2.svg.png> |
| **Unreal Logo** | <https://twitter.com/unrealengine> |
| **Unity users graph** | <https://unity3d.com/public-relations> |
| **Crysis Graphics** | <http://www.techspot.com/review/642-crysis-3-performance/> |
| **Ambient occlusion** | <http://www.giantbomb.com/ambient-occlusion/3015-7660/> |
| **Anti-aliasing** | <http://forum.ls-rp.com/viewtopic.php?f=4&t=191806> |
| **Water example** | <http://www.worldtvpc.com/blog/nvidia-make-waves-to-show-off-water-simulation-technology/> |
| **Sound mixer** | <https://davereiddesign.wordpress.com/2015/09/15/the-psychology-of-sound-in-video-games/> |
| **Net-Code** | <https://www.youtube.com/watch?v=cVuHzSx5zSo> |
|  |  |