12-2009

Anaglym: A Graphics Engine Providing Secure Execution of Applications

Josh Holtrop
Grand Valley State University

Anaglym: A Graphics Engine Providing Secure Execution of Applications

2009-12-17
Josh Holtrop
Grand Valley State University

There are many uses for graphics-intensive applications in our world today. One of the most prevalent uses is that of video games. Other uses include physics demos and simulations, various forms of computer-aided design, virtual walk-throughs of three-dimensional environments, and more. These types of programs benefit immensely from, and sometimes even require hardware-accelerated graphics cards.

With the invention of the Internet and the increasing number of web sites available, developing and distributing software applications has become cheaper and easier. Various sites have been created that are dedicated to distributing games or other types of graphical applications. FTP servers house public repositories of content available for both upload and download to their users. With such ease of transport available for software came ease of distribution of malware such as viruses and spyware.

This type of malware can be detrimental to unsuspecting users who simply download an application because they are interested in it. In some cases, the user may be unaware that what they downloaded came with additional hidden functionality. This is a big security risk, as software running directly on the user's computer can scan his or her personal files or monitor keystrokes entered to capture passwords, credit card numbers, or social security numbers.

To combat this security risk, various technologies were created that allow running of web-based applications in a “virtual machine.” Java applets have security levels so that Java code can be safely executed within a web browser. Adobe Flash allows running hosted applications within a sandbox as well. The web browser itself usually supports JavaScript which also executes web-based code in a virtual machine.

Each of these technologies “sandbox” the hosted application by prohibiting it from interacting with the host environment in dangerous ways. For example, the hosted code is normally unable to open, read, or write arbitrary files from the host file system. The code cannot spawn child processes to carry out certain actions either. Most of the time, the virtual machines impose memory limits and can also limit the amount of CPU consumed by the hosted code so that the applications being executed do not interfere with the normal execution of the user's native operating environment.

In general, these technologies for secure execution of web-based content do not allow any type of access to the hardware in the physical machine. This includes display adapters that may provide support for hardware-accelerated graphics rendering. Therefore, it is not possible with traditional technologies such as these to provide real-

time, interactive 3D graphical applications.

To provide access to graphics APIs such as OpenGL and DirectX, some modern scripting languages such as Perl and Python provide their own API which “wraps” the native API provided by the graphics card vendor for performing graphics operations. These languages often execute their code in a virtual machine similar to technologies like Java applets and Flash applications.

These languages are not suitable for executing web-based content, however, because they are not focused on security by default. They are built to be general-purpose programming languages and as such they expose potentially dangerous functionality to the script being executed.

There has been some effort to provide Java applications and applets with the ability to utilize hardware-accelerated graphics cards. Java is still a general-purpose programming language, however, so it was not designed from the ground up to be secure enough to operate in a web-based environment where any arbitrary code may be encountered.

The Lua programming language was designed to be an embedded scripting language rather than a stand-alone, general-purpose programming language. As such, it is suitable for environments where the set of functions available to a hosted application must be strictly controlled.

The Anaglym graphics engine uses the Lua programming language as the high-level language in which to interpret Anaglym applications. Applications running inside Anaglym are Lua scripts that have a particular environment exposed to them. By default, when Lua is embedded inside a C or C++ host application, no Lua functions are available to the hosted script. Lua functions must be exported by the host environment to be callable by a Lua script.

There are a few standard libraries of Lua functions that are available with the default Lua engine. These libraries are simply collections of related functions or objects. The Anaglym engine exports three of these standard libraries to Lua scripts being interpreted in its environment. These three libraries are the math, string, and table libraries.

These libraries are included because they only contain “safe” functionality. The string library contains functions dealing with common operations on Lua strings, one of the Lua data types. The math library contains common math functions such as trigonometric functions and constants such as pi. The table library includes helper functions for dealing with Lua tables, another of Lua's data types.

The Anaglym engine allows hosted Lua scripts to make use of hardware-accelerated graphics libraries available in the host system. However, it does not simply expose the API of the graphics library to the Lua scripts directly. Instead, the Anaglym engine

includes a 3D graphics engine with concepts of solid objects, collision detection, and physics operations.

To provide physics and collision detection functionality, the Anaglym engine relies on ODE, the Open Dynamics Engine. ODE is an open-source package that provides an API to create and maintain “worlds” of solid objects that can be collided together and interacted with according to physics rules. ODE maintains attributes of the solid objects such as their position, rotation, velocity, torque, and masses. However, ODE does not provide any functionality for drawing the objects it knows about to the screen in any manner.

All of the functionality for drawing objects utilizing hardware-accelerated graphics operations is provided by the core Anaglym engine. The engine allows loading objects from model files or by creating them programmatically using Lua functions. Once created, various attributes about the objects can be controlled through the interface exported from the Anaglym engine to the Lua environment.

There are various events that the engine captures such as mouse movements, button presses, keyboard input, and screen update events. Anaglym exposes these events to the hosted script by invoking Lua callback functions that have been registered as event handlers for a particular event. When a script is initially loaded, functions with names corresponding to known engine events (with “_event” appended) are automatically registered as callback functions for those events so that the script developer does not have to specifically register them as such.

The Anaglym engine attempts to do as much for the user as possible. By default, physics processing and drawing objects to the screen are enabled. This means that a Lua script executing in the Anaglym engine can perform one operation to create a box object, for example, and it will immediately appear on the screen, be affected by gravity, and interact with any other previously created Anaglym engine objects.

Various utility functions are exported from the engine to the Lua environment to make development of applications utilizing the Anaglym engine easier. For example, there is a clone() function available that allows Lua to make a copy of an engine object in a single call rather than recreating it in whatever manner was used to create the first one. There are functions available to add forces and torques to objects, change their masses, colors, or textures, and to remove all objects from the world.

Anaglym applications can make use of the engine functions provided for them in this manner instead of having to repeat many low-level interface calls. Since Anaglym provides this high-level interface for interacting with its engine, applications can be developed much more quickly and debugged more easily. This allows more rapid prototyping to take place.

Another feature useful to Anaglym script developers that the engine provides is the ability to dynamically reload the executing script. The Lua environment accepts blocks

of Lua code known as “chunks.” Lua is indifferent as to where these chunks originate. They could be files loaded in or strings of code dynamically generated by the host application. They are simply loaded into the current Lua environment and executed as Lua code.

What this means is that a script can be loaded which defines certain functions, and then later loaded again. If the same functions are defined, they simply overwrite the previous function definitions. The Anaglym engine captures presses of the F5 key and reloads the currently executing script into the same Lua environment that was previously executing. This benefits a script developer as he or she can make modifications to callback functions present in the script and reload it without having to exit the Anaglym engine and start the script over from scratch.

When engine functions such as loading a texture or model file are invoked, the files accessed are limited to those within the directories containing the script being executed and the library folder for the engine. The library folder contains a set of common textures and model files that all Anaglym scripts can make use of. In this way, file access is strictly controlled by the engine to not allow access to arbitrary files in the user's file system.

Since no functions are exported to the Lua environment which allow reading or writing files, or spawning processes, or other such potentially dangerous operations, the hosted script is effectively “sandboxed” from the host environment of the machine. The only other way that a hosted script could potentially adversely affect the machine is by performing some sort of denial-of-service attack, whether maliciously or without intent. The Lua script could enter some sort of infinite loop, where all of the engine's processing time went into executing the Lua code and it never got around to processing timer or input events.

The Anaglym engine uses the “hook function” feature of the Lua virtual machine to guard against Lua code misbehaving in this manner. The engine registers a callback hook function with the Lua environment. The hook function is called after a certain number of Lua instructions have been executed. The number of instructions between hook function invocations is set relatively high so that the hook function is only invoked every few seconds. This incurs only a very minor performance penalty.

If the hook function is called and it has been longer than a certain amount of time since the last engine event was processed, then the engine knows that the Lua script has been blocking event processing and it can take action. In this way, the engine offers a fully secure environment in which to execute applications. The application does not necessarily need to be trusted. It can be downloaded from a web site and executed without worry that it may adversely affect the environment.

Because the Anaglym engine offers access to hardware-accelerated graphics in a completely secure environment and utilizes a high-level language that allows for rapid development of applications, it has a unique offering. The engine is geared toward

game developers who need more graphics power than a simple web-based Flash environment would give or creators of 3D content who want a way to quickly develop and distribute virtual environments, etc... Since it provides security for the end-user, he or she can execute content from any source without worrying about what effect it might have on their operating environment.