TREMOR ENGINE: A Cross-Platform, Modular C++ Game Development Platform, Built for Those Experienced with Deprecated Development Platforms

Student Name: Caylee Morris
Student Number: [REDACTED]
Supervisor: [REDACTED]
Date Submitted: 06th April 2017
Module: UFCFS4-30-3 Creative Technologies Project

CONTENTS:

SUMMARY

Tremor Engine was born of the desire to create a modular C++ game engine platform, named after its design inspiration in the roots of id Software's *Quake engine* lineage. (id Software, 1996) Game development practices evolve at a rapid rate, and once a long-term project is completed, a developer may find themselves needing to re-learn an entire platform. Tremor is designed and written similarly to Valve Corporation's *Source Engine* (Valve Corporation, 2004) and id Software's *idTech engines* (Tei, 2013) in a consistent style, with the intention to release as open-

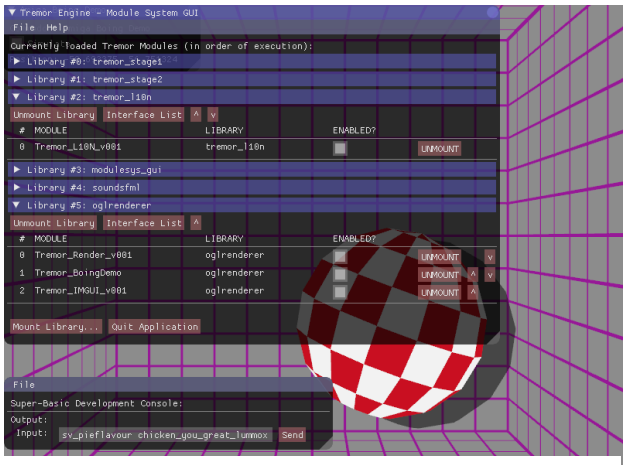


Figure 1 - Tremor running the Module System GUI and Boing demo, as well as showing the developer console simultaneously

source software once the project has developed to an adequate state of maturity, allowing something of a widely usable bridge from older development methodologies.

Over the course of development, the goals have been drastically pulled back from the initial intention due to the complexities of authoring the lower-level sections of the engine, including cutting the multi-platform compatibility; however, at the end of the allocated development time, a set of demo applications have been authored atop of the modular Tremor platform, demonstrating some of the capabilities as it currently stands.

Tremor's platform is still in the early stages of development and could not yet be used for a commercial project, however features a robust and modular framework, allowing for dynamic mounting and unmounting of libraries, with automatic memory clearing of Tremor specific classes when shutting down a module. Tremor's central framework also features a wide range of custom STL-style storage and utility classes, core application modules, and a priority-based filesystem.

VIDEO PRESENTATIONS

Video Presentation (January 2017 Prototype): https://www.youtube.com/watch?v=7pULr1Nx4aM
Video Presentation (April 2017 Submission): https://www.youtube.com/watch?v=eWakGJZRb8k

INTRODUCTION

Throughout the development of a long-term project, a developer will usually be locked to a single technology set in order to keep the project on track. While this has its advantages, such as avoiding a Duke Nukem Forever situation wherein the technology is constantly being upgraded and the development restarted, (3D Realms et al., 2011) it has drawbacks; most of which being the lack of familiarity with more recent development tools and trends that may have emerged over the course of their own project. While modern engines offer much in terms of ease of use, accessibility, and features, no single modern game development environment is built in a manner that would be familiar to developers emerging from a long term project making use of an engine derived from idTech or Source. To that end, Tremor aims to offer something of a familiar bridge between environments of the current day and environments of the past, taking the semi-modular design methodology of Valve Corporation's Source Engine to the logical conclusion by constructing the platform in an entirely modular fashion, allowing the dynamic loading and unloading of application libraries (.DLLs). Tremor would also be written with the intention of porting to other platforms in the future.

The final goal of Tremor is to release as an open-source game development platform, allowing both original game development and game modding, while avoiding the high licensing costs assosiated with many other game engines.

METHODOLOGIES

RESEARCH SUMMARY

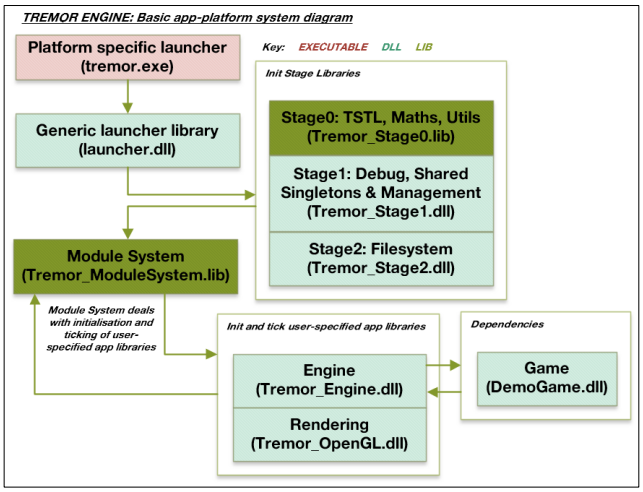


Figure 2 - Tremor's original structure, as seen in the Tremor Research Report. The final structure can be seen in *Appendix C.*

Research on how to build Tremor began by studying the reasons why many engines author their own memory management structures, and how this is undertaken, rather than using the OS-specific STL implementation. It was soon found that Electronic Arts, one of the largest development houses in the world, does use their own implementation of the STL which has been made open-source, with a complete reason list for their decision to author it. (Pedriana, 2007) The reasons were numerous, but several were more important than others, including; STL implementations varying between platform, compilers with weak-inlining, and lack of low-level debuggability. The OS-specific STL was passed over for these reasons, and EASTL was too complex for what was required in Tremor, so it was decided to author a new STL class set. Similarly, various maths libraries were also considered, but these were skipped over in favour of authoring a new one to enforce consistency of syntax.

Studying other game engines, it was soon realized that while the engine could be built in a modular fashion, it will require certain 'core' libraries that are mandatorily included in all applications built

using the platform. These included the Module System management library, application launcher, and a series of 'stage' libraries: Stage0 containing the Tremor STL, Stage1 containing management/debugging classes and internal modules, and Stage2 containing the filesystem module. Modules would make use of the "*Abstract Factory"* and *"Singleton"* design patterns, based on work presented in *"Design Patterns: Elements of Reusable Object-Oriented Software".* (Gamma, et al., 1994) Functionality of any one module is exposed to other modules via a pure-virtual interface. (Iqbal, 2000)

Game engine features such as entity management, rendering, and management of game-specific systems would be handled by their own modules. Rendering would make use of OpenGL, to allow for easier and faster cross-platform portability when compared to DirectX, which only natively supports Microsoft platforms (Geldreich, et al., 2014) While these components would be authored to a basic standard, the primary goal of development at this stage is to author a stable foundation to build future software.

Tremor was primarily inspired by the design of Valve's Source Engine and id Software's Quake Engine; taking numerous design elements from both, such as a priority based 'searchpath' system for locating game assets and the abstract interfaces for dynamically loaded code, both allowing for easily authored third-party modifications or expansions. Alongside these, several books were used as reference material throughout, including: *"Game Engine Architecture"* (Gregory, 2009) which was studied for its approach to engine structure, *"Ultimate 3D Game Design & Architecture"* (Sherrod, 2009) which featured numerous guides on how to author certain data structures, and finally *"Real-Time Rendering"* (Akenine-Möller, et al., 2008) and *"Introduction to 3D Game Programming with DirectX 11",* (Luna, 2012) both of which were studied to gain an understanding of different rendering frameworks in some game engine implementations. The open-source rendering framework OGRE was also studied for some time, providing a source code level insight into a large-scale rendering framework. (Torus Knot Software Ltd, 2016)

IMPLEMENTATION METHODOLOGY

Before development began, a code standard was authored to enforce consistency across the codebase. This was based on the idTech 4 standard published by id Software, and uses a form of Hungarian Notation. (ID Software, 2011) This was adjusted numerous times throughout development to account for new situations that arose or conventions that had not been considered in the initial document.

Development progressed incrementally, starting with implementing a basic rendition of the module system and application launcher. Key to this was the ability to load these dynamic modules, handled on Windows platforms via the `LoadLibraryEx` and `GetProcAddress` functions. Once a library is loaded, the module factories (defined by pre-processor macros) add themselves to a global factory list, which is then used by the module system to create and manage the module objects. This incremental approach required heavy refactoring mid-way through development, resulting in a rewrite of the entire module system and the removal of the unnecessary 'stage 1 modules' which were a separate modular system entirely for Stage1 owned modules. The original intention was to have the module system work via static linked library, rather than a dynamic library, however this added unneeded additional complexity and caused problems with shared memory spaces. While every care was taken to avoid doubling of efforts like this, it was a necessary part of the learning process that resulted in a far cleaner result overall.

Various open-source projects were made use of during development, either as a temporary place-holder for Tremor specific code, as reference material, or within components of the engine. These include: *RapidJSON* (a JSON text file parsing library), *Dear IMGUI* (a GUI framework), *SFML* (multi-media library used in the sound engine), *stb_image.h* (a single-file image loader library), *PCG* (a random number generator), *GLFW* and *GLEW* OpenGL libraries, *Dirent* (cross-platform directory/file navigation) and *TinyFiles* (another cross-platform filesystem navigation library). Some of these libraries are integrated into the core framework of Tremor, but some (such as Dear IMGUI) are included as their own modules with their own abstract interfaces. The open licenses for these elements allow their code to be distributed within Tremor.

While the majority of this code is following the terms of their respective licenses, certain elements of the maths library as it currently stands have been adapted from code found within publicly available code of Valve's Source Engine (specifically, this includes parts of the maths classes, such as Matrix). This placeholder code is preventing an open-source release of the codebase, but can be replaced in the future following similar syntax.

For a full list of source code attributions, see Appendix A.

Due to the modular structure of the platform, different components could be worked on, then dropped, with relative ease; for instance, a basic sound system was to be implemented, but became a lower priority as development progressed. Due to the modularity of the system, temporarily removing this component was as simple as disabling the '*tremor_sound*' DLL project in Visual Studio. In a team-driven project, this would allow several team members to work on engine systems at once with minimal clashing. Development on the sound module was picked up again towards the end of development.

Tremor's STL had a specific set of required templates to meet the goal of being a usable STL alternative, including String, Vector and LinkedList. Implementation on highly desired templates began early with continual improvements throughout, however several additional templates were added during development to add certain desired functionality that would not be viable to include in the already begun templates. A good example of this was the addition of StringTree, which acts as a binary-node tree allowing for fast searching for objects by string. StringTree follows the 'trie' structure, (Pitts, 2010) and was one of the more successful implementations, used in many various sections of the codebase, such as the localisation library. Many of Tremor's STL classes derive from an *IMemory* interface class, allowing for some degree of memory management (handled by the Stage 1 library), cleaning up after themselves when their dynamic library is unmounted from the engine. This was something of a compromise; it was previously intended for all *malloc/new* allocations to the tracked in this way, but this proved to be impractical when attempted.

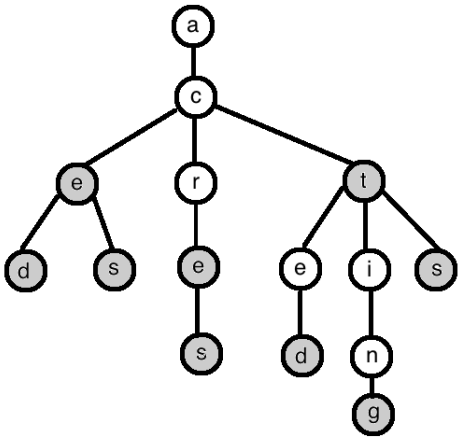


Figure 3 - Visualisation of a Trie structure, which was implemented in Tremor STL as 'StringTree' (Me and Mark Publishing, 2012)

RESULTS

While a genuine effort was made to author the long list of components initially conceived for Tremor, the project failed to reach the level of maturity that had been intended from the start. A key component currently missing from the project is a robust rendering system. This was mostly down to a lack of familiarity with OpenGL as a graphics platform, and never having written a renderer from scratch before; previous experience had already started with some sort of base, often on DirectX. OpenGL treats rendering differently than expected, which lead to a variety of issues that while not impassable, meant delays and spelled the end of a complex version of this component.

A basic OpenGL renderer was authored regardless, offering creation of render contexts, loading of shaders, models and textures, clearing of buffers, and rendering of `IBaseRenderable` interface objects. Unfortunately, what is assumed to be a compatibility problem between a wide range of OpenGL resources and the Tremor maths library has caused strange stretching and distortion of objects, rather than translation and rotation of them. This lead the 'boing' demo to be implemented using direct OpenGL access within the renderer DLL itself – somewhat bypassing the module system and ignoring the Tremor Renderer interface in a way that would not be suitable for production applications. Various other modules had their features cut throughout the project too; the sound module lacks many of the more advanced capabilities intended for it, the Tremor STL Map template is broken as it currently stands, some dependencies remain where they ideally wouldn't, and the math library still contains third-party code from another game engine that is preventing an open-source release. The filesystem module also never had the second version of its *TPAK* package file format implemented, which would have allowed for multi-part archives of files and improved patching of games, akin to Valve's *VPK2* format. (Valve Developer Community, 2016)

Late in development, it was discovered that there were major issues with the mathematical components of Tremor's STL library. Early on, it had been decided to take advantage of the parallel maths capabilities of modern processors, known as SIMD (*single instruction multiple data)* to quickly calculate matrix and vector math problems. Many math libraries take advantage of this, including Microsoft's DirectX Math library and OpenGL Math *(glm)*. Tremor's math library was developed from a combination of sources; following a similar syntax to the libraries found within the Source and Unity engines, whilst featuring functionality modelled after both a section in the book "*Ultimate 3D Game Design & Architecture"* and an article called *"How to Write a Maths Library in 2016"* (Mitton, 2016), which details the use of the `__vectorcall` calling convention to pass SIMD data between functions without shifting registers. Sections of the maths library were initially derived from other math libraries until replacements were written (several of these, namely the Matrix classes, are yet to be replaced). SIMD intrinsic functions generally require data to be byte aligned to 16 byte boundaries in 32 bit applications. Due to the dynamically allocation undertaken when creating an instance of a module, it is not guaranteed that memory will always be aligned in this way, leading to an access violation crash when performing actions using these. This is a major issue and requires the entire maths library to be refactored and redesigned internally, and was not noticed earlier due to a flaw in the testing methodology; these classes were not tested within a proper Tremor module, but rather the launcher DLL and therefore did not exhibit the alignment issues. This design flaw lead to severe problems implementing any game-esque software using the built-in maths library, or even just a basic renderer test. As previously mentioned, incompatibilities seem to exist between the Tremor maths library and a wide range of OpenGL resources as it is; as such, it would be best to undertake a complete rewrite of the maths library before developing any 3D or game-esque applications in the Tremor framework, or make use of an alternate library such as OpenGL Math.

The completed framework features a wide array of miscellaneous components, including a system for sharing variables and functions across libraries, which are also made accessible by string through a developer console, accessible in all Tremor applications by pressing the tilde key (~). These console objects can be saved and loaded from a configuration file, and basic scripts can be executed by running these commands in turn. Command-line switches can also be globally accessed and modified via the framework, alongside debug functionality such as assertions and logging, PCG random number generator, global time and application state variables and more.

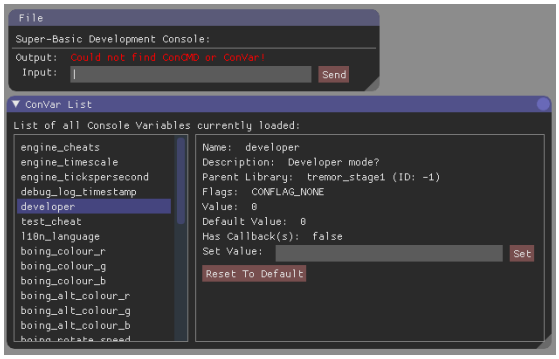


Figure 4 - Tremor's Command Console and ConVar list

Included within the Tremor codebase are various additional modules and demo applications. While several of these are not considered part of the core framework, they would almost certainly be used by a large number of developers making use of the platform; such as the localisation and sound modules. Sound has been implemented using the *SFML* media library, offering precaching of sounds and playback over a number of sound channels. These sounds can be adjusted during and before playback by holding a SoundRef object associated with the sound, which will delete the associated playing sound reference object once no references to it remain, using a simple refcounting system. This was eventually abstracted and adapted into a template class, which was added to the Stage0 library. The localisation library is a string bank database system making use of the StringTree template, allowing strings to be fetched by their associated token in a desired language. Some interoperability between the sound and language libraries is offered, wherein the language defined in the localisation library is used to pick which language to load in the sound library, if both are mounted at the same time. A base 'gameplay engine' module is included, though not complete, offering some basic functionality for the construction of game software. This includes entities with associated renderables, a 'thinking' system which allows for multiple threads of entity updates to occur at desired intervals, and object transforms with parentage support. While this is an entirely optional module to make use of, it was anticipated that many would use it as a point from which to begin work on a game project.

Tremor's STL storage templates are connected to a memory clean-up interface inside the Stage 1 DLL, which facilitates the deallocation of their associated data either on program shutdown or on module unmount. This offers some degree of protection against memory leaks and crashes associated with the dynamic library unmounting process, without the need to keep an entire memory database of all allocations (as various debuggers would) or run a full garbage collection routine akin to Unity. Garbage collection is something that was to be specifically avoided in the initial Tremor planning phase, due to the frustrations many Unity Engine developers suffer with GC. (Smuts, 2015)

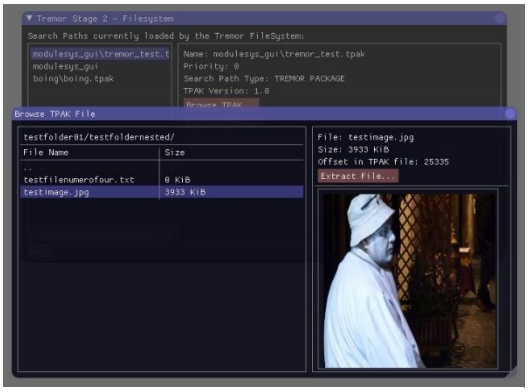


Figure 5 - Tremor's Filesystem GUI tool

For a breakdown of all the Demo Application Modules, please refer to Appendix B.

DISCUSSION

Despite the drawbacks of the project as it stands, it would be wrong to declare the project a failure. Tremor's core framework forms a stable base for future development, offering a high degree of modularity in a consistent style. Applications can easily be written that take advantage of the platform without modifying the core libraries, and this is demonstrated in the many demo applications that are included in the submitted project. Modules can be dynamically loaded and unloaded with some degree of memory leak prevention and clean-up. Various demo application modules have been created, many of which can either be run as separate applications, combined into one or more applications, or loaded dynamically using the Module System GUI.

While various components have yet to reach the maturity required of them to be used in a commercial project as-is, the Tremor codebase is a functional proof-of-concept for the construction of software, game or otherwise, consisting largely of dynamic and interchangeable modules. Modularisation of components promotes better decoupling of interfaces from their implementations, forcing programmers to properly consider exactly how other programmers will need to interact with their code.

The complexities of SIMD mathematics were underestimated throughout, which lead to a failure to produce true gameplay demos or workable renderer examples beyond the lower level Boing demo. As development on Tremor continues, the maths library will be replaced. The few open-source maths libraries available, such as OpenGL Math, will be studied far closer and a workable replacement will be authored, or a pre-existing solution will be adopted.

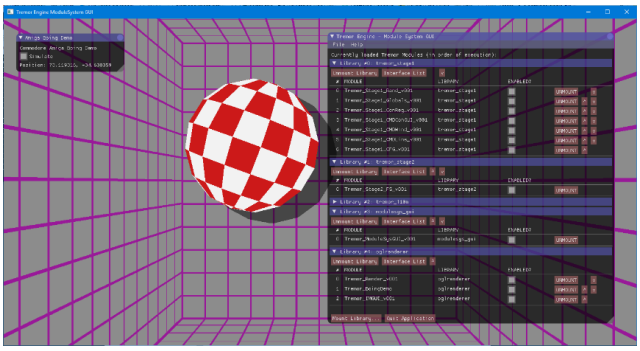


Figure 6 - Boing, currently the only working 3D rendering demo running alongside the Module System GUI

The memory management solution is one that could be improved from its current state. Late in the project, it was discovered that open-source garbage collection utilities for C++ are available, such as the *Boehm-Demers-Weiser conservative garbage collector*. (Boehm, 2016) While Tremor is intended to specifically avoid implementing garbage collection in the traditional sense, these libraries would be good resources to discover how to keep track of all dynamic allocations and clean them up when a library is unmounted.

Continuing onward with Tremor, the current renderer will be redesigned and rebuilt making use of the new graphics API Vulkan, (Khronos Group, 2017) offering a rare example of a game engine platform making use of the new technology. A full material API will be authored, allowing for full use of GLSL shaders. Enhancements would be made to all existing components, including implementing the spec for TPAK file version 2 (which allows data to be split across multiple archives, or altered by adding additional 'patch archives'). The entire codebase would undergo a refactoring effort to remove unnecessary or messy code, preparing the path to the next milestone; porting to another platform, such as Linux, Mac OS X or Sony PlayStation 4 via the ORBIS SDK – a goal that had been shelved earlier in Tremor's development.

CONCLUSION

Ultimately, the Tremor codebase meets its most major goal, offering a proof-of-concept modular development platform written in C++, with a style like that of deprecated platforms such as Source or Quake. The core of the framework is a solid one, and could be easily expanded further with enhanced functionality and a wider range of available modules. Issues with SIMD optimizations have left the maths library largely unusable, but this is not beyond rectification in the near future. Tremor forms a stable base for continued development and could easily be released as open-source software in the future for others to make use of or learn from.

REFERENCES

3D Realms, Triptych Games, Gearbox Software, Piranha Games, 2011. *Duke Nukem Forever,* s.l.: 2K Games.

Akenine-Möller, T., Haines, E. & Hoffman, N., 2008. *Real-Time Rendering.* 3rd ed. Natick: A K Peters, Ltd..

Amiga History Guide, 2007. *Bouncing Back- The origins of the Boing Ball.* [Online]
Available at: http://www.amigahistory.plus.com/boingball.html
[Accessed 04 April 2017].

Barrett, S. T., 2017. *stb_image.h,* s.l.: s.n.

Boehm, H.-J., 2016. *A garbage collector for C and C++.* [Online]
Available at: http://hboehm.info/gc/
[Accessed 04 April 2017].

Brooks, J., Geelnard, M., Luck, D. & Mical, R., 2016. *GLFW Boing,* s.l.: s.n.

Cornut, O., 2017. *dear imgui,* s.l.: s.n.

Epic Games, Inc, 2016. *Unreal Engine 4,* s.l.: Epic Games, Inc.

Gamma, E., Helm, R., Johnson, R. & Vissides, J., 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* 1st ed. s.l.:Addison Wesley.

Gaul, R., 2017. *Tiny Headers (Tinyfiles.h),* s.l.: s.n.

Geldreich, R., Ginsburg, D. & Mitchell, J., 2014. *Moving Your Games to OpenGL.* Seattle, WA, Steam Dev Days, Valve Corporation.

Gomila, L., 2017. *Simple and Fast Multimedia Library,* s.l.: SFML Team.

Gregory, J., 2009. *Game Engine Architecture.* 1st ed. Natick: A K Peters, Ltd..

id Software, 1993. *DOOM,* s.l.: GT Interactive.

id Software, 1996. *Quake,* s.l.: GT Interactive.

id Software, 2000. *id Technology Licensing Program.* [Online]
Available at:
https://web.archive.org/web/20001110100100/http://www.idsoftware.com/corporate/idtech/index.html
[Accessed 8 September 2016].

ID Software, 2011. *CodeStyleConventions.doc (mirror).* [Online]
Available at: http://ftp.ntua.gr/mirror/idgames/idstuff/doom3/source/CodeStyleConventions.doc
[Accessed 1 January 2016].

id Software, 2012. *Github - id-Software/Quake 2: Quake 2 GPL Source Release.* [Online]
Available at: https://github.com/id-Software/Quake-2
[Accessed 8 September 2016].

Iqbal, G., 2000. *Using Interfaces with DLLs,* s.l.: s.n.

Khronos Group, 2017. *Vulkan - Industry Forged.* [Online]
Available at: https://www.khronos.org/vulkan/
[Accessed 04 April 2017].

Luna, F. D., 2012. *Introduction to 3D Game Programming with Directx 11.* Dulles, Virginia: MERCURY LEARNING AND INFORMATION.

Me and Mark Publishing, 2012. *Using Tries to Store Word Lists in RAM.* [Online]
Available at: http://meandmark.com/blog/2012/07/using-tries-to-store-word-lists-in-ram/
[Accessed 05 April 2017].

Mitton, R., 2016. *How To Write A Maths Library In 2016.* [Online]
Available at: How To Write A Maths Library In 2016
[Accessed 2 April 2017].

O'Neill, M. E., 2015. *PCG (random number generator),* s.l.: s.n.

O'Neill, M. E., 2015. PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation. *(Submitted to ACM Transactions on Mathematical Software),* p. Under Review.

Pedriana, P., 2007. *EASTL -- Electronic Arts Standard Template Library,* s.l.: Electronic Arts.

Pitts, R. I., 2010. *Trie,* Boston: Boston University.

Rönkkö, T., 2016. *Dirent,* s.l.: s.n.

Sherrod, A., 2009. *Ultimate 3D Game Engine Design & Architecture.* Mason: Cengage Learning.

Smuts, B., 2015. Unity Garbage Collection Tips and Tricks. *Gamasutra*, 27 July.

StackOverflow, 2013. *How can I found out what is creating garbage? [closed].* [Online]
Available at: http://stackoverflow.com/questions/20413333/how-can-i-find-out-what-is-creating-garbage
[Accessed 11 March 2017].

Stewart, N., 2016. *OpenGL Extension Wrangler Library,* s.l.: s.n.

Tei, B., 2013. *Family Tree Of Quake Engines.* [Online]
Available at: https://en.wikipedia.org/wiki/Quake_engine#/media/File:Quake_-_family_tree.svg
[Accessed 2 January 2016].

The GLFW Development Team, 2016. *GLFW,* s.l.: s.n.

THL A29 Limited; Yip, Milo, 2017. *RapidJSON,* Shenzhen, China: Tencent.

Torus Knot Software Ltd, 2016. *OGRE - Open Source 3D Graphics Engine | Home of a marvelous rendering engine.* [Online]
Available at: http://www.ogre3d.org/
[Accessed 8 September 2016].

Valve Coropration, 2013. *thirdpartylegalnotices.txt.* [Online]
Available at: https://github.com/ValveSoftware/source-sdk-2013/blob/master/thirdpartylegalnotices.txt
[Accessed 3 January 2016].

Valve Corporation, 2004. *Half-Life 2,* Bellevue, Washington: Valve Corporation; Sierra.

Valve Corporation, 2004. *Source Engine,* Bellevue, Washington: Valve Corporation.

Valve Corporation, 2005. *Engine Licensing Information Sheet.* [Online]
Available at: http://www.valvesoftware.com/SOURCE_InfoSheet.pdf
[Accessed 2 January 2016].

Valve Developer Community, 2016. *VPK File Format.* [Online]
Available at: https://developer.valvesoftware.com/wiki/VPK_File_Format
[Accessed 05 April 2017].

Wasiak, P., 2013. Computer Dealer Demos: Selling Home Computers with Bouncing Balls and Animated Logos. *IEEE Annals of the History of Computing,* 35(4), pp. 56 - 68.

APPENDIX A: Source Code Attributions

List of source codes or libraries made use of within Tremor.

- RapidJSON (THL A29 Limited; Yip, Milo, 2017)
- Dear IMGUI (Cornut, 2017)
- Simple and Fast Multimedia Library (Gomila, 2017)
- stb_image.h (Barrett, 2017)
- PCG Random Number Generator (O'Neill, 2015)
- GLFW (OpenGL Framework) (The GLFW Development Team, 2016)
- GLEW (OpenGL Extension Wrangler Library) (Stewart, 2016)
- dirent.h (Rönkkö, 2016)
- tinyfiles.h (Gaul, 2017)

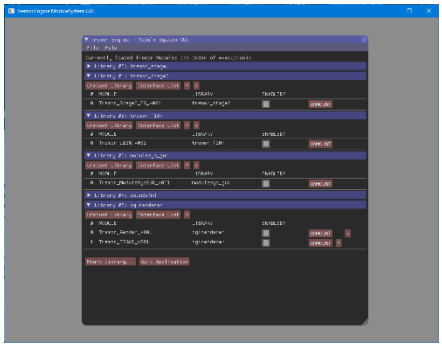
Finally, serving as placeholder matrix math code:

- Source Engine Mathlib (Matrix maths code) (Valve Corporation, 2004)

APPENDIX B: Tremor Engine Demo Applications

This is an overview of the demo applications included with Tremor, which can be loaded separately via their own app definition, or dynamically via the Module System GUI.

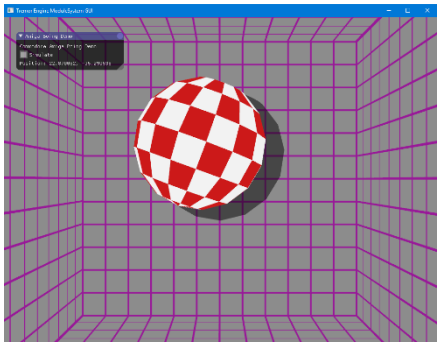
Tremor Module System GUI



The Module System GUI application most easily demonstrates the capabilities of the modular structure of Tremor as a codebase. It allows the user to experiment with changing the execution order of modules and libraries, unloading and loading libraries on the fly, or disabling some outright. It is possible to unmount core framework libraries using this interface, though this will crash the application once it attempts to access core functions and is best avoided. As this is a developer tool, rather than a user-facing application, it was decided not to disable the ability to modify operation of core modules in this way.

The Module System GUI makes use of many Tremor module features, including the localisation library, OpenGL renderer, filesystem and console variable registry.

Boing

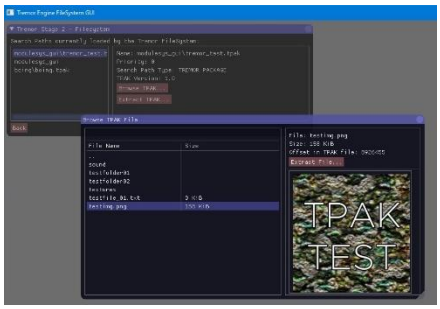


Boing is a classic render test originally authored for the Commodore Amiga, initially conceived to show how the system could draw a large complex 3D shape at a fast framerate. (Wasiak, 2013) Here, it is used to test OpenGL integration within Tremor, a working show of the sound playback module, as well as (when this module is loaded alongside the Module System GUI) how execution order of modules affects the order in which they are drawn. The Boing demo is based on code included with the OpenGL GLFW library, (Brooks, et al., 2016) makes use of draw functions that are deprecated in OpenGL 3 and above, and thus requires command line flags to specify an OpenGL version below 3 (`-ogl_major 2 -ogl_minor 1`), or specify compatibility mode (`-ogl_compat`). Boing does not make use of the renderer as it was authored at an earlier point, instead directly interfacing with OpenGL from within the renderer library itself.

VUtiful Joe

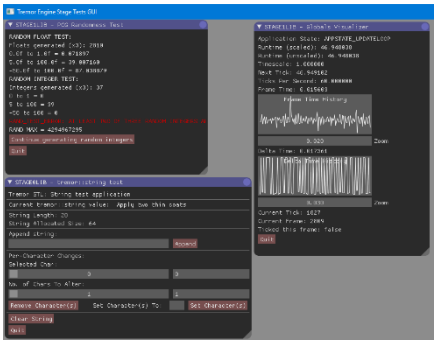
VUtiful Joe is a port of a C# Winforms application designed to control a VU Meters via an Arduino device over serial. This was authored as a demonstration of how fast the basic features of another application could be ported to IMGUI and used within Tremor, and could be applied to development tools that would previously have consisted entirely separate applications.

Filesystem GUI



Tremor's Stage 2 DLL contains a module which is not loaded by default with most applications that allows for the creation of TPAK files, browsing mounted TPAKs, exporting files from within a TPAK, or entirely decompiling a TPAK file. This tool shows how development tools associated with modules could be directly integrated into the library itself and either run as a separate program or run alongside other modules, as well as offering a useful tool for interacting with the filesystem.

Stage Library Tests



The final test applications were written as test applications for various components within Tremor's core Stage libraries. These consist of:

- *PCG Randomness Test*
- *Tremor String Test*
- *Global Variables Visualiser*

While these are intended to test various elements of Tremor's core libraries, the Globals Visualiser could be a helpful aid during the creation of a game application.

As previously mentioned, a renderer test was also authored, but this suffers from frequent crashes and does not function as intended when it does occasionally manage to load.

APPENDIX C: Tremor Engine Module Linking