Lowering the entry barrier for students programming Virtual Reality applications

M. Lambers

Computer Graphics Group, University of Siegen, Germany

Abstract

In Computer Graphics, it is common practice to accompany lectures with hands-on tutorials and/or project assignments that allow students to write and run their own interactive graphics applications. In the special case of Virtual Reality courses, this approach is difficult to maintain since the software requirements pose a high entry barrier to students.
In this paper, we propose a technique to significantly simplify Virtual Reality application programming, and implement it in an easy-to-use framework that supports the full range of typical Virtual Reality hardware setups, from head-mounted displays to multi-node, multi-GPU render clusters. The framework lowers the entry barrier for students and allows them to focus on course goals instead of fighting software complexities.

Categories and Subject Descriptors (according to ACM CCS): K.3.2 [Computers and Education]: Computer and Information Science Education—Computer Science Education I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual and Augmented Reality

1. Introduction

Virtual Reality (VR) courses at universities often target postgraduate students because the entry barrier regarding hardware and software has traditionally been high. Recently, the hardware situation improved significantly with the availability of affordable sensor, display, and interaction devices. Sousa Santos et al. [SDM15] documented a course taking advantage of this development.

However, software requirements for VR application programming are still complex, and finding the right software to base a course on is difficult [SDSS14, SDM15]. To cover the full range of common VR setups, applications must be able to handle multi-GPU and multi-node systems [EMP09]. The complexity of distributed graphics applications is usually handled by a specialized framework, but available frameworks come with their own complexities and steep learning curves.

Consistent with observations made by Boers et al. [BDHB08] and Anderson and Peters [AP10], our experience with past courses shows that students lose time and motivation fighting software complexities instead of focusing on course goals. Therefore, the hands-on tutorial part of our VR course was limited in comparison to other computer graphics courses, and student projects in VR typically had a difficult and time-consuming start phase.

In this paper, we focus on reducing the software-side complexity of VR courses, with the goals of increasing hands-on tutorial content and enabling more student projects. We define the following requirements for a VR software framework:

1. The framework must be free and open-source. We agree with Sousa Santos et al. [SDM15] that in academic contexts paying for VR software licenses is problematic, especially when students can use their own devices.
2. The framework must allow the use of external rendering software, but also plain OpenGL. High-level rendering engines are useful for advanced student projects and thesis works, but simple tutorial assignments should not require the students to learn about such an engine when plain OpenGL is sufficient.
3. The framework must support the full range of common VR graphics hardware, including head-mounted displays and multi-GPU/multi-host setups. Applications based on the framework must run in CAVE-like VR environments and other hardware available in a VR lab, and also on the students' own devices.
4. The framework must be easy to set up and work with. Students must be able to quickly reach a productive familiarity with the framework so that they can focus on course goals.

To arrive at a VR framework that fulfills these requirements, we propose a technique to simplify multi-window, multi-GPU, and multi-process handling in Sec. 3, and describe its implementation in Sec. 4. Sec. 5 shows example results, and Sec. 6 discusses the current state and future work.

2. Related Work

Software frameworks used in VR courses at universities include in-house frameworks that are not publicly available [Sta05] and/or are

DOI: 10.2312/eged.20161024

```
while application is running do
    preRenderProcess();
    foreach window w do
        preRenderWindow(w);
        for i ← 1 to stereoRenderPasses(w) do
            M_i ← viewMatrix(w, i);
            F_i ← frustum(w, i);
            T_i ← texture(w, i);
            render(M_i, F_i, T_i);
        end
        postRenderWindow(w);
    end
    postRenderProcess();
    foreach window w do
        updateDisplay(w, T_1, T_2);
        asyncBufferSwap(w);
    end
    processEvents();
    updateApplication();
    waitForBufferSwaps();
end
```

Algorithm 1: The main loop in the single-context single-thread approach. Multi-process handling is omitted for brevity.

```
class VRApplication {
  /* Mandatory functions */
    // Update scene state (animations etc).
      // Called once before each new frame.
      void update(...) = 0;
    // Render the scene into texture T using
      // frustum F and view matrix M.
      void render(M, F, T) = 0;
  /* Optional functions */
    // Event handling, using Qt conventions
      void keyPressEvent(...) {}
      void mouseMoveEvent(...) {}
      void mousePressEvent(...) {}
    // Special per-process/per-window actions
      void preRenderProcess(...) {}
      void preRenderWindow(...) {}
      void postRenderWindow(...) {}
      void postRenderProcess(...) {}
    // Multi-process support
      void serializeDynamicData(...) const {}
      void deserializeDynamicData(...) {}
};
```

Figure 1: *Summary of the interface that a VR application needs to implement. Note that most functions are optional.*

not actively developed [Ant09, DK11], frameworks that are coupled to a particular rendering, scene graph, or game engine [Tra99, vRKG*00, LCC*12, SDSS14] and scene graph engines with multi-GPU or multi-host support [Rei02], and web-oriented techniques such as VRML and WebGL [Zar06, SDM15]. None of the solutions from these categories fulfill the requirements defined in Sec. 1.

The only two frameworks we could find which fulfill the requirements 1 – 3 are VR Juggler [BJH*01] and Equalizer [EMP09]. Both fail to fulfill requirement 4. As noted by Sousa Santos [SDM15], VR Juggler is complex to install and set up, and support for recent hardware is absent. We also note that development seems to have slowed down significantly, or even stopped.

We have therefore based our VR courses, tutorials, and projects on Equalizer in the last years. While it is a very powerful and flexible framework, capable of much more than just VR application scenarios, in our experience it is also very hard for students to work with. We have observed the following major obstacles:

- Install-and-setup obstacle: Equalizer is split into many sub-libraries and requires several external libraries that a custom build script partly tries to download and install during the Equalizer build. This process fails regularly on relevant platforms.
- Hierarchy-level obstacle: Distributing program logic over the Equalizer hierarchy levels (see Sec. 3) runs counter to the students' previous experiences with object-oriented programming, which is to group program logic according to its purpose.
- Multi-context obstacle: Equalizer uses multiple OpenGL contexts, and contexts on the same GPU share objects such as textures. Students are typically not familiar with OpenGL contexts, context sharing, and context/thread binding. As a result, they struggle to understand which context is active at which time and in which thread, and how OpenGL objects should be managed across multiple contexts and/or multiple GPUs.
- Multi-thread obstacle: Equalizer uses multiple rendering threads within each process on multi-GPU systems. Many students struggle to grasp all consequences of this approach, especially when integrating third-party software, and consequently run into multithreading pitfalls that are notoriously hard to debug.

Like Sousa Santos et al. [SDM15], we prefer simple software that allows "actually understanding the whole process necessary to create a virtual environment". Furthermore, we agree with Boers et al. [BDHB08] and Anderson and Peters [AP10] that students need to focus on course goals instead of losing time and motivation fighting with framework setup and usage complexities.

3. Simplified VR Application Programming

A central function of a VR framework is the handling of multi-GPU and multi-node render systems. Challenges associated with multi-GPU programming are listed in the Parallel OpenGL FAQ [Eil07]. The main aspects relevant to VR are the following:

- An OpenGL context can only be bound to one thread at a time, and a switch of that thread is expensive. Therefore, all rendering to a context should happen from only one thread.
- Access to a GPU is serialized by the driver. Rendering into different contexts on the same GPU therefore best happens in a serial manner, to avoid unnecessary costly context switches.
- The swapping of back and front buffers of a context is typically synchronized with the refresh rate of the connected display. The function that triggers the swap blocks the calling thread until that swap happens.
- OpenGL contexts can share objects such as textures if they live on the same GPU.
- Multi-GPU systems need to provide a way for an application to differentiate between GPUs. This is system dependant.

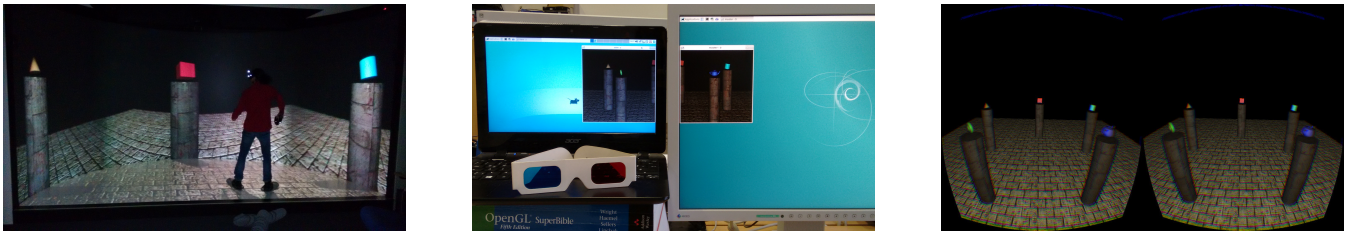

Figure 2: *Example VR application running in a CAVE-like VR lab (left), across a laptop and a desktop PC (middle), and on the Oculus Rift DK2 head-mounted display (right).*

The Equalizer framework handles these challenges using a hierarchy of processes, GPUs, windows, and channels. Each process can handle multiple GPUs, each GPU can drive multiple windows, and each window can be divided into multiple channels. Each window has its own context, and contexts on the same GPU share objects. Each GPU has a dedicated rendering thread.

While very flexible and powerful, this *multi-context multi-thread* approach introduces a lot of complexity: an application needs to split operations and data over four hierarchy levels, it has to manage multiple contexts that share objects only if they live on the same GPU, and it has to manage concurrent access to process-level resources by different rendering threads.

We propose the following simplified *single-context single-thread* approach. There are only two hierarchy levels: process and window. Each process handles only one GPU, and maintains a master OpenGL context on that GPU that is not connected to any visible window. This is the only context a VR application process sees. All windows of a process have their own private contexts that share objects with the master context. The windows are represented by textures in the master context. The private window contexts are driven by private rendering threads that display these textures and wait for the buffer swap, while the main thread is free for other work such as application scene state updates. The corresponding render logic is summarized in Alg. 1.

In this approach, the application programmer only handles a set of textures in a single OpenGL context, and renders different views into these textures sequentially from the main thread. The multi-context and multi-thread obstacles are eliminated.

Multiple processes, both for multi-GPU and multi-host setups, are handled by serializing relevant VR application data (mostly scene state) on the master process, writing it via local or network pipes to the slave processes, and deserializing it there. The application only needs to implement serialization and deserialization logic.

One potential drawback of our approach affects multi-GPU hosts. We use separate processes for each GPU, and inter-process communication via pipes. Equalizer uses separate threads, thus avoiding communication overhead. However, we found that the complexities and pitfalls of multi-threaded rendering often drove our students to switch to multi-process but single-threaded Equalizer configurations, thereby eliminating the advantage. Furthermore, we expect that the communication costs are tolerable when shared memory or similar techniques are used for processes on the same host.

4. A Simple VR Application Framework

We based our implementation of the single-context single-thread approach on the C++ language and the Qt library because our students are already familiar with both from other courses, and because both offer widely used and easy-to-setup development tools. This eliminates the install-and-setup obstacle.

The interface that a VR application has to implement is summarized in Fig. 1. Since only a single interface needs to be implemented, the hierarchy-level obstacle is removed. To allow students to quickly become familiar with the framework, the interface is kept minimal, and optional components have an empty default implementation.

The configuration file concept of Equalizer has been very useful in our experience, and we copy it in our framework. A configuration file defines the VR application processes, one for each GPU, and their properties. For each process, it defines windows with attributes such as stereo mode, size, position, and projection area geometry. When starting the VR application, the running process is assumed to be the master process defined first in the configuration file. Slave processes are started automatically by the framework. The application can run unmodified on different display hardware setups by using different configuration files.

5. Results

The simplifications proposed in Sec. 3 are made possible by graphics hardware and API capabilities that are now ubiquitous (render-to-texture via framebuffer objects, off-screen contexts, context object sharing). All platform dependent aspects are handled by Qt, keeping the code base of the framework small.

To demonstrate the versatility in terms of display hardware setups, we show a simple example application using different configuration files for different hardware setups in Fig. 2. The first setup (left) is a CAVE-like VR lab. Its render system has four GPUs, each connected to three projectors for a total of six passive stereo projection areas. The second setup (middle) demonstrates multi-host support using a laptop and a desktop PC. The third setup (right) is an Oculus Rift DK2 head-mounted display.

While the simple example application is written in plain OpenGL, more complex applications often require external rendering engines. The integration of such engines is simplified by our single-context single-thread approach since this corresponds well

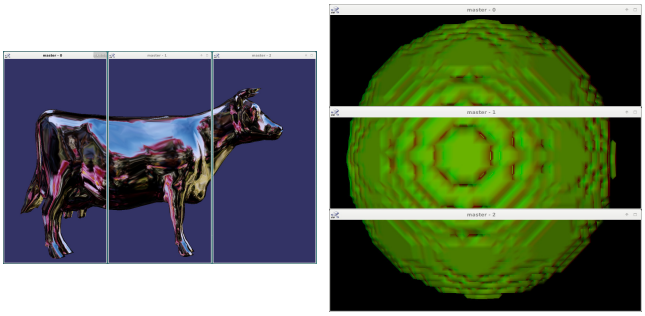

Figure 3: *Left: OpenSceneGraph viewer running in a three-window configuration with monoscopic views. Right: a VTK visualization pipeline running in a three-window configuration with stereoscopic views for red-cyan anaglyph glasses.*

to the common usage scenario of embedding a rendering engine into an application-managed graphics window.

The versatility in terms of rendering approaches is demonstrated in Fig. 3. The left side shows a scene rendered by a full-featured OpenSceneGraph viewer. The integration of OpenSceneGraph with our framework requires only few lines of code, while its integration with Equalizer (osgScaleViewer; included in the Equalizer source code) is much more complex and still does not provide full functionality. The right side of the figure shows a VTK example that renders an isosurface extracted from a voxelized sphere. The integration of VTK is equally simple and does not impose any restrictions on VTK visualization pipelines.

Full source code is available under the MIT/Expat license at `https://github.com/marlam/qvr`. This includes the framework, called QVR, and all examples shown in this section.

6. Discussion and Conclusions

The four complexity obstacles we observed while using Equalizer are removed: the single-context single-thread approach removes the multi-context and multi-threading obstacles, and the framework implementation removes the install-and-setup and hierarchy-level obstacles.

The framework handles distributed graphics only. Applications typically use third-party libraries for other VR-related tasks, e.g. VRPN [THS*01] for tracking devices. On the one hand, this requires teachers to provide additional software depending on their lab equipment and course topics. On the other hand, it keeps the framework lightweight and easy to set up, and lowers maintenance burden.

The framework is currently in a testing stage. Initial feedback from selected students is encouraging to the point that we decided to base our future VR courses on it. However, further tweaks will likely become necessary once the framework is in wider use. For example, we plan to measure the impact of the potential communication overhead in multi-GPU systems for real-world applications, and implement appropriate countermeasures such as the use of shared memory instead of pipes if required.

References

[Ant09] ANTHES C.: *A Collaborative Interaction Framework for Networked Virtual Environments*. PhD thesis, Institute of Graphics and Parallel Processing at JKU Linz, Austria, August 2009. 2

[AP10] ANDERSON E. F., PETERS C. E.: No more reinventing the virtual wheel: Middleware for use in computer games and interactive computer graphics education. In *Eurographics - Education Papers* (2010). doi:10.2312/eged.20101013. 1, 2

[BDHB08] BOERS J., DOBBE J., HUIJSER R., BIDARRA R.: From a light CG framework to a strong cannibal experience. In *Eurographics - Education Papers* (2008). doi:10.2312/eged.20081002. 1, 2

[BJH*01] BIERBAUM A., JUST C., HARTLING P., MEINERT K., BAKER A., CRUZ-NEIRA C.: VR Juggler: a virtual platform for virtual reality application development. In *IEEE Proc. Virtual Reality* (March 2001), pp. 89–96. doi:10.1109/VR.2001.913774. 2

[DK11] DOERR K.-U., KUESTER F.: CGLX: A scalable, high-performance visualization framework for networked display environments. *IEEE Trans. Visualization and Computer Graphics 17*, 3 (March 2011), 320–332. doi:10.1109/TVCG.2010.59. 2

[Eil07] EILEMANN S.: Parallel OpenGL FAQ. `http://www.equalizergraphics.com/documentation/parallelOpenGLFAQ.html`, 2007. Accessed: 2015-12-20. 2

[EMP09] EILEMANN S., MAKHINYA M., PAJAROLA R.: Equalizer: A scalable parallel rendering framework. *IEEE Trans. Visualization and Computer Graphics 15*, 3 (May 2009), 436–452. doi:10.1109/TVCG.2008.104. 1, 2

[LCC*12] LUGRIN J.-L., CHARLES F., CAVAZZA M., LE RENARD M., FREEMAN J., LESSITER J.: CaveUDK: A VR game engine middleware. In *Proc. ACM Symp. on Virtual Reality Software and Technology* (2012), pp. 137–144. doi:10.1145/2407336.2407363. 2

[Rei02] REINERS D.: *OpenSG: A scene graph system for flexible and efficient realtime rendering for virtual and augmented reality applications*. PhD thesis, Technische Universität Darmstadt, 2002. 2

[SDM15] SANTOS B. S., DIAS P., MADEIRA J.: A virtual and augmented reality course based on inexpensive interaction devices and displays. In *Eurographics - Education Papers* (2015). doi:10.2312/eged.20151022. 1, 2

[SDSS14] SOUZA D., DIAS P., SANTOS D., SANTOS B. S.: Platform for setting up interactive virtual environments. In *Proc. SPIE 9012, The Engineering Reality of Virtual Reality* (2014), pp. 90120O–1–90120O–9. doi:10.1117/12.2038668.7. 1, 2

[Sta05] STANSFIELD S.: An introductory VR course for undergraduates incorporating foundation, experience and capstone. In *Proc. 36th SIGCSE Technical Symposium on Computer Science Education* (2005), SIGCSE, pp. 197–200. doi:10.1145/1047344.1047417. 1

[THS*01] TAYLOR II R. M., HUDSON T. C., SEEGER A., WEBER H., JULIANO J., HELSER A. T.: VRPN: A device-independent, network-transparent VR peripheral system. In *Proc. ACM Symp. on Virtual Reality Software and Technology* (2001), pp. 55–61. doi:10.1145/505008.505019. 4

[Tra99] TRAMBEREND H.: Avocado: a distributed virtual reality framework. In *IEEE Proc. Virtual Reality* (Mar 1999), pp. 14–21. doi:10.1109/VR.1999.756918. 2

[vRKG*00] VAN REIMERSDAHL T., KUHLEN T., GERNDT A., HENRICHS J., BISCHOF C.: ViSTA: a multimodal, platform-independent VR-toolkit based on WTK, VTK, and MPI. In *Proc. 4th International Immersive Projection Technology Workshop* (2000). 2

[Zar06] ZARA J.: Virtual reality course - a natural enrichment of computer graphics classes. *Computer Graphics Forum 25*, 1 (2006), 105–112. doi:10.1111/j.1467-8659.2006.00921.x. 2