Development of CAVELib Compatible Library for HMD-type VR Devices

Shintaro Kawahara¹* and Akira Kageyama²

¹Japan Agency for Marine-Earth Science and Technology (JAMSTEC)
²Department of Computational Science, Graduate School of System Informatics, Kobe University

* kawahara@jamstec.go.jp

Received: November 7, 2018; Accepted: April 4, 2019; Published: May 30, 2019

Abstract. Head-mounted display-type VR devices (HMDs) are becoming a practical platform for three-dimensional scientific visualization. To reuse software assets that have been developed for CAVE-type VR systems (CAVEs), this paper presents a C++ library for porting CAVE application software to HMDs. Our library emulates the function calls of CAVELib, which is a commercial library for developing application software executable on CAVEs, and it enables us to easily port CAVELib application software to HMDs with minor modifications to the original source code. Sharing the source code also leads to an improvement in the software development efficiency, which is executable on both CAVEs and HMDs.

Keywords: Scientific visualization, Virtual reality, Software development

1. Introduction

To visualize and analyze three-dimensional (3-D) simulation data, including data that simulates complex natural phenomena, the Japan Agency for Marine-Earth Science and Technology has been employing a CAVE-type virtual-reality (VR) system [1] since 2003. We also developed application software for this system in C++ and CAVELib [2] with OpenGL as an application programming interface (API) for the two-dimensional (2-D) and 3-D computer graphics.

Recent advances in head-mounted displays (HMDs) make them a practical new platform for 3-D scientific visualization. To make use of the software assets that have been developed for CAVE-type VR systems (CAVEs), we developed the CAVELib compatible library (CLCL), which can generate an executable file for an HMD with only slight modifications to source code written in C++ and CAVELib.
2. HMD-type VR devices

In this work, we targeted three types of HMDs: The Oculus Rift [3], the HTC VIVE [4] and a Windows Mixed Reality (WMR) [15] device. The three requirements that enable us to employ them are as follows.

1. The headset and controllers have six degrees-of-freedom (6DoF) tracking.
2. The software development kits (SDK) for C++ have already been released by vendors.
3. The SDKs support OpenGL as a 2-D and 3-D graphics API.

VR experiences of almost the same quality as that of CAVE can be realized with inexpensive systems, even ones that just use a computer to generate stereoscopic images if the first requirement is satisfied. The second and third requirements are most important for porting existing CAVElib programs to HMDs because CAVElib programs are written in C++ and use OpenGL as the graphics API. In addition, Table 1 shows the hardware specifications of the HMDs targeted in this paper.

Table 1: Hardware specifications of the HMDs.

<table>
<thead>
<tr>
<th></th>
<th>Oculus Rift</th>
<th>HTC VIVE</th>
<th>WMR device (Lenovo Explorer with Motion Controller)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>2160 × 1200 pixels (1080 × 1200 pixels per eye)</td>
<td>2880 × 1440 pixels (1440 × 1440 pixels per eye)</td>
<td></td>
</tr>
<tr>
<td>Refresh Rate</td>
<td>90 Hz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field of View</td>
<td>110°</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tracking</td>
<td>6DOF (headset and controllers)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connection with PC</td>
<td>HDMI × 1, USB 3.0 × 3</td>
<td>HDMI × 1, USB 3.0 × 1</td>
<td></td>
</tr>
<tr>
<td>Power Supply</td>
<td>USB</td>
<td>Outlet</td>
<td>USB</td>
</tr>
<tr>
<td>Development</td>
<td>Game engine (Unity, Unreal engines)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SDK for C++ (Oculus SDK, OpenVR)</td>
<td></td>
<td>SDK for C++ (OpenVR)</td>
</tr>
</tbody>
</table>

3. Development method of application software for HMDs

Currently, the use of game engines such as Unity [5] and Unreal Engine [6] is a mainstream software development method for general VR devices including non-HMDs. These engines enable us to generate an executable file suitable for specific HMDs. It is the fastest approach
for developing very simple VR applications for displaying polygonal data, which is frequently used for the data visualizations, with fly-through in the VR world using handheld controllers. However, it is not practical to reuse the application software assets for CAVEs directly in a game engine because this would require the re-implementation of routines such as data loading, visualization algorithms, and graphical user interfaces that are implemented in the existing application software for CAVEs. In contrast, some vendors are also releasing the SDKs for developing application software for their HMDs, and many of them are being provided as C++ libraries. Therefore, they are better than the game engines for porting source code written in C++ and CAVELib. The Oculus SDK [7] and OpenVR [8] are employed for the HMDs targeted in this paper. Figure 1 shows the relationships among the SDKs for HMDs, the platforms that are needed for launching an executable binary built using the SDKs, and the HMD devices. Their details are described in Sections 3.1 and 3.2.

![Diagram of SDKs, platform software, and HMD devices](image)

Figure 1: Relationships among the SDKs, platform software, and HMD devices. The arrows indicate the flow of execution when an executable binary built using an SDK for an HMD is launched.

### 3.1. Oculus SDK

The *Oculus SDK* is an SDK provided by Oculus VR, LLC. It is used for developing application software for Oculus devices (Rift and Development Kit 2). Currently, the Oculus SDK is provided as pre-built libraries for Microsoft Visual Studio and executable files built using it run only on Microsoft Windows. To launch the executable files, it is necessary to go through the *Oculus App*, which is a platform software for Oculus devices. When using Oculus SDK, programmers are able to use C++ function calls to control HMD devices (i.e., the headset and handheld controllers) and perform functions such as initialization, termination, and acquisition of the device’s position and orientation. OpenGL and DirectX are supported as graphics APIs.

### 3.2. OpenVR

The *OpenVR* is an SDK provided by Valve Corporation. It is used for developing application
software that is able to execute through SteamVR, which is the platform software of Valve Corporation. Compatible HMD devices include not only the HTC VIVE, but also Oculus devices and WMR devices. When an OpenVR application is executed on the Oculus device, it is also necessary to launch the Oculus App in addition to SteamVR. When an OpenVR application is executed on a WMR device, it is also necessary to go through Windows Mixed Reality for SteamVR, which is the software to execute an OpenVR application on WMR devices, in addition to SteamVR. Currently, OpenVR is provided as open source software that includes pre-built libraries for multiple platforms (Microsoft Windows, Mac OS X, and Linux). In this paper, we only describe the use of Microsoft Windows because we did not test OpenVR on other operating systems. When using OpenVR, programmers are able to use C++ function calls for controlling HMD devices similar to the Oculus SDK. OpenGL and DirectX are supported as graphics APIs. These functionalities are similar to those of the Oculus SDK. As mentioned above, OpenVR also supports Oculus devices. However, OpenVR is not optimized for these devices because it is compatible with various HMD devices. To develop software optimized for Oculus devices, it is better to use the Oculus SDK than OpenVR.

4. Development of CLCL

4.1. Overview of CAVElib programming

Figure 2 shows a simplified example of a CAVElib program. In this example, function names starting with "CAVE" are functions of CAVElib. In CAVElib programming, specific routines such as initialization and scene rendering are registered as callback functions before the main loop is entered. In this example, CAVEInitApplication() and CAVEDisplay() are functions for registering the callback functions. Once CAVEInit() has been called, the registered callback functions are processed in the background while the main loop is active. CAVElib also automatically manages processes such as window management, the generation of rendering buffer, and generation of parallax images based on head tracking information. Therefore, programmers do not need to implement these processes from scratch. CAVElib programming is similar programming using the GLUT and GLFW libraries [9] for OpenGL. Programmers with experience using these libraries will find CAVElib programs easy to create.
**Journal of Advanced Simulation in Science and Engineering**

```c
#include <cave_ogl.h>

void initialization() {} /* callback function */
void scene_render() {} /* callback function */
void navigation() {} /* function for checking state of controllers */

int main(int argc, char **argv) {
    CAVEConfigure(&argc, argv, NULL);
    CAVEInitApplication(initialization, 0); /* register callback function */
    CAVEDisplay(scene_render, 0); /* register callback function */
    CAVEInit();

    while (!CAVEgetbutton(CAVE_ESCKEY)) { /* main loop */
        navigation();
        CAVEUSleep(10);
    }

    CAVEExit();
    return 0;
}
```

Figure 2: Simplified example of a CAVELib program.

### 4.2. Re-implementation of CAVELib function calls

Figure 3 shows the processing flow for emulating the internal process of CAVELib programs in CLCL. To re-implement function calls of CAVELib, we used the SDKs for HMDs described in Section 3. However, neither SDK has any function calls other than those for device control such as initialization, termination, and the position and orientation tracking of a headset and controllers. Therefore, in addition to the SDKs for HMDs, we used GLFW to re-implement the function calls of CAVELib. Figure 4 shows a part of the CLCL source code that contains a sub-thread that launches when CAVEInit() is called. The callback functions registered in the main thread are executed in this thread. The initialization of OpenGL is also performed in this thread because the OpenGL context cannot be shared between the main thread and sub-thread. Figure 5 shows simplified CLCL code for initializing an HMD device and OpenGL. Similarly, we also re-implemented other function calls that are commonly used in the CAVELib program for CLCL. Networking functions for multiple CAVEs have not yet been implemented. The structure of a CLCL program is almost same as that of a program written using SDK and OpenGL directly. Therefore, the use of CLCL does not lead to performance degradation in the development of HMD application software.
Figure 3: Processing flow for emulating CAVELib programs.

```c
void SubThread()
{
    InitHMD();
    InitGL();
    CreateGLBuffers();

    while (m_IsThreadRunning) {
        ExecInitCallback(); /* exec if initialization is not done */
        UpdateTrackingData();
        ExecIdleCallback();
        PreProcess();
        for (int eyeIndex = 0; eyeIndex < ovrEye_Count; eyeIndex++) {
            SetMatrix(eyeIndex);
            ExecDrawCallback();
        }
        PostProcess();
    }

    ExecStopCallback();
    Terminate();
}
```

Figure 4: Simplified CLCL code that is launched when CAVEInit() is called.
void InitHMD() /* Oculus SDK version */
{
    ovrResult result = ovr_Initialize(nullptr);
    if (OVR_FAILURE(result)) exit(EXIT_FAILURE);
    ovrGraphicsLuid luid;
    result = ovr_Create(&m_HmdSession, &luid);
    if (OVR_FAILURE(result))
    {
        ovr_Shutdown();
        exit(EXIT_FAILURE);
    }
    m_HmdDesc = ovr_GetHmdDesc(m_HmdSession);
    m_WindowWidth = m_HmdDesc.Resolution.w / 2;
    m_WindowHeight = m_HmdDesc.Resolution.h / 2;
}

void InitHMD() /* OpenVR version */
{
    vr::EVRInitError eError = vr::VRInitError_None;
    m_HmdSession = vr::VR_Init(&eError, vr::VRApplication_Scene);
    if (!vr::VRCompositor())
    {
        vr::VR_Shutdown();
        exit(EXIT_FAILURE);
    }
    m_HmdSession->GetRecommendedRenderTargetSize(
        &m_FrameBufferWidth, &m_FrameBufferHeight);
    m_WindowWidth = m_FrameBufferWidth / 2;
    m_WindowHeight = m_FrameBufferHeight / 2;
}

void InitGL()
{
    glfwSetErrorCallback(ErrorCallback);
    if (!glfwInit()) exit(EXIT_FAILURE);
    m_Window = glfwCreateWindow(
        m_WindowWidth, m_WindowHeight, "CLCL", NULL, NULL);
    if (!m_Window) {
        glfwTerminate();
        exit(EXIT_FAILURE);
    }
}

Figure 5: Routines for initializing the HMD device with OpenGL called in the sub-thread.
4.3. Generation of the pre-built library

Table 2 shows C/C++ libraries used for generating the pre-built CLCL library. To build CLCL using these libraries, we used Visual Studio 2017 as the C++ compiler. Currently, we have developed two versions of CLCL, which are for Oculus devices developed using the Oculus SDK and for OpenVR compatible devices developed using OpenVR. As the SDK for HMD, either the Oculus SDK or OpenVR is needed to build CLCL according to the targeted HMD device. The ZED SDK and the CUDA Toolkit are for the external camera employed in the examples in this paper (as described in Section 5) and do not need to be linked if the external camera function is not used.

Table 2: C++ libraries used to generate a pre-built CLCL library.

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oculus SDK</td>
<td>1.26.0</td>
<td>SDK for HMD</td>
</tr>
<tr>
<td>OpenVR</td>
<td>1.0.16</td>
<td></td>
</tr>
<tr>
<td>GLFW</td>
<td>3.2.1</td>
<td>OpenGL framework</td>
</tr>
<tr>
<td>GLEW</td>
<td>2.1.0</td>
<td>C++ library for using OpenGL extension</td>
</tr>
<tr>
<td>GLM</td>
<td>0.9.9</td>
<td>C++ mathematics library for OpenGL</td>
</tr>
<tr>
<td>ZED SDK</td>
<td>2.5.0</td>
<td>SDK for ZED Mini</td>
</tr>
<tr>
<td>CUDA Toolkit</td>
<td>9.1</td>
<td>Necessary for using ZED SDK</td>
</tr>
</tbody>
</table>

When the CAVELib programs are compiled, a generated pre-built CLCL library file (“CLCL.lib” for the Oculus SDK version of CLCL or “CLCL_openvr.lib” for the OpenVR version of CLCL) is linked instead of the original CAVELib library file. To reduce the changes needed to the CAVELib program, the header file of CLCL is intentionally given the same name as in CAVELib (“cave_ogl.h”). Figure 6 shows a conceptual diagram of the use of CLCL.

Figure 6: Conceptual diagram of the use of CLCL.
5. Building a CAVELib program using CLCL

Before describing the porting procedure with CLCL library, here we briefly explain a video-through function that we have added to the HMD devices. Our paper [10] provides further details. In this work, we implemented the video-through using an external stereoscopic camera (Stereolabs ZED Mini [11]). Figure 7 shows a snapshot of a Rift with a ZED Mini installed on the front face. A feature of this camera is the generation of depth maps from captured parallax images. This function can be used via an SDK for C++ provided by the vendor (ZED SDK). Using this function within CLCL, we implemented the ability to provide occlusion between real and virtual objects rendered as 3-D computer graphics. If the ZED SDK and the CUDA toolkit are linked when generating the pre-built CLCL library file described in Section 4.3, the connection status of the ZED Mini is automatically checked by CLCL when an executable file built with CLCL is launched. The user can dynamically switch the active or inactive state of the camera input while an application program is running. When the camera input is activated, the occlusion between real and virtual objects rendered as 3-D computer graphics is calculated and displayed. However, the frame rate for rendering stereoscopic images decreases because a high-performance GPU is needed to generate depth maps from captured parallax images. Programmers are able to use this function without changing the original CAVELib program, because it is embedded in the CAVELib functions re-implemented in CLCL. Currently, this function is implemented in only the Oculus SDK version of CLCL.

Figure 7: Rift with a ZED Mini installed on the front face.

5.1. Porting simple programs

To evaluate the compatibility between CAVELib and CLCL, we attempted to use CLCL to build the very simple programs listed in [12], which is a lecture note on how to write CAVELib programs and contains five sample programs. Figure 8 shows the simplest one, which draws a white triangular object in VR space. To build this code using CLCL, we modified several lines of the original code. The modified lines are shown in bold italic. The original code was written for IRIX, which was often used as the operating system of graphics workstations for CAVEs. Therefore, a UNIX-derived header file ("unistd.h") and an IRIX-derived C function ("sginap") are used in the original code, and they must be changed to alternative ones ("CAVEUSleep"). In this example, these changes are shown using the
“IRIX” directive.

```c
#include <cave_ogl.h>
#ifdef IRIX
#include <unistd.h>
#endif

void init_gl(void)
{
    glClearColor(0., 0., 0., 0.);
}

draw(void)
{
#ifdef USE_CLCL_OCULUS_SDK
    glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
#endif
    glPushMatrix();
    glColor3f(1.0, 1.0, 1.0);
    glTranslatef(0.0, 5.0, -5.0);
    glBegin(GL_TRIANGLES);
    glVertex3f(-2.0, 0.0, 0.0);
    glVertex3f(2.0, 0.0, 0.0);
    glVertex3f(0.0, 4.0, 0.0);
    glEnd();
    glPopMatrix();
}

int main(int argc, char **argv)
{
    CAVEConfigure(&argc, argv, NULL);
    CAVEInit();
    CAVEInitApplication(init_gl, 0);
    CAVEDisplay(draw, 0);
    while (!CAVEgetbutton(CAVE_ESCKEY)) {
#ifdef IRIX
    sginap(10);
#else
    CAVEUSleep(10);
#endif
    }
    CAVEExit();
    return 0;
}
```

Figure 8: A sample CAVE code, triangle.c, from [12].
Additionally, "glClear" in the callback function, which is used for rendering scenes, must be skipped when using the CLCL-linked Oculus SDK for building this code. In the Oculus SDK, the images for both eyes are drawn in an OpenGL render buffer using viewport division. If this change is not applied, the image for the left eye, which is rendered first, will be erased by "glClear" when the image for the right eye is rendered. In contrast, it is unnecessary to apply this change when using the CLCL-linked OpenVR, because the OpenGL render buffers for both eyes are generated independently in OpenVR programming. In this example, this change is shown using the “USE_CLCL_OCULUS_SDK” directive.

Using the pre-built CLCL library instead of CAVELib, an executable file for an HMD device was generated and we executed it on a laptop computer (Razer Blade 2016, GeForce GTX 1060) connected to an HMD. The results confirm that not only can parallax images be displayed on the HMD device but headset tracking also works properly. We also applied similar modifications to the other programs listed in [12] and confirmed that all of them can be built using CLCL. Two of them (“snowfall_nav.c” and “sword.c”) could be operated using a handheld controller called WAND when executed on a CAVE. When using CLCL, it was confirmed that the same interactive WAND operations can be performed using the handheld HMD controller (Oculus Touch and VIVE Controller). Figures 9 and 10 show screen captures taken when executing the generated files.

Figure 9: Screen captures when executing the sample programs listed in [12]. (with the external camera disabled)
5.2. Porting of VFIVE

To demonstrate the application of CLCL to an advanced CAVELib program, we also applied it to VFIVE [13], which is our original visualization software for CAVEs. When using VFIVE in a CAVE, a user can visualize scalar/vector fields in the immersive environment of VR with the interactive control of visualization-related objects with WAND.

In this experiment, we used *vfive3.72Amt*, which can be downloaded from [14]. It includes a C/C++ header file named "glext.h" for using OpenGL extensions. In contrast, CLCL uses GLEW for using OpenGL extensions. This may cause conflicts depending on the inclusion order of the header files. To avoid this problem, we modified the original VFIVE code so that "glext.h" is not used. Other code modifications were similar to those of the examples described in Section 5.1. Figure 11 shows snapshots of VFIVE executing on CAVE, a Rift, and a VIVE. Each executable file has been built with CAVELib, CLCL (Oculus SDK version), and CLCL (OpenVR version). This figure also shows the interactive visualization operations using the handheld controller that is WAND on a CAVE, Oculus Touch on a Rift, or VIVE Controller on a VIVE. In this case, a fluid-type simulation for geodynamo is being visualized. The user can operate the input data interactively using the 6DoF handheld controller like the CAVE WAND. Figure 12 shows a screen capture of VFIVE executing on a Rift with a ZED Mini. This figure also shows how the occlusion of objects in the camera image and virtual objects was implemented using the ZED Mini. For the executable files linked with CLCL (OpenVR version), we confirmed that all of the programs described in Section 5.1 and VFIVE also worked with a Rift and a WMR device (Lenovo Explorer with Motion Controllers). Table 3 shows controller inputs for each HMD's controller in CLCL. Currently, CLCL supports only for the right-handed controller.
Figure 11: VFIVE executed on three types of VR devices.
The parentheses indicate the C++ library used for building VFIVE.
6. **Summary**

In this paper, we outlined the development of a CAVELib-compatible library. Source code of CLCL is available on our GitHub page [16]. This library enables us to easily port CAVELib application software to HMDs with minor modifications to the original source code. The source code compatibility improves the efficiency of the development of software that is executable on both CAVEs and HMDs. Currently, networking functions for multiple users and devices have not yet been implemented to CLCL. As future work, we are planning to implement these functions and to connect multiple CAVEs and HMDs for virtual space sharing.

**Acknowledgment**

The authors would like to thank Dr. Takeshi Yoshinaga from Institute of Systems, Information Technologies and Nanotechnologies (ISIT) for evaluating our library on a Windows Mixed Reality device. This work was supported by JSPS KAKENHI Grant Number JP16K00178.
References


