Live Coding of a VR Render Engine in VR

Markus Schütz* TU Wien

Michael Wimmer† TU Wien

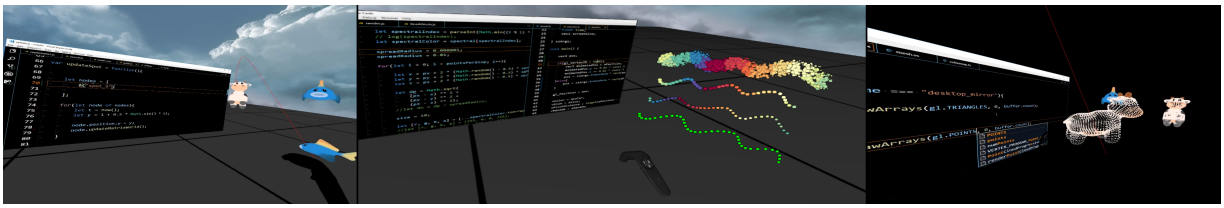

Figure 1: Left: Drag&Drop scene objects into source code to create selector snippet. Middle: Change brush generation and rendering code at runtime. Right: Modify OpenGL calls, e.g. draw gl.POINTS instead of gl.TRIANGLES.

Abstract

Live coding in virtual reality allows users to create and modify their surroundings through code without the need to leave the virtual reality environment. Previous work focuses on modifying the scene. We propose an application that allows developers to modify virtually everything at runtime, including the scene but also the render engine, shader code and input handling, using standard desktop IDEs through a desktop mirror.

Index Terms: Human-centered computing—Human computer interaction (HCI)—Interaction paradigms—Virtual Reality;

1 Introduction and Related Work

Programming environments with notebook interfaces such as Mathematica [6], Jupyter [2] and MATLAB Live Editor [7] provide live coding environments where results are displayed directly within the code. The code is split into sections that can be evaluated separately, and the result of a section can be shown to the user as static text output and images, but also interactive widgets such as 2D or 3D scenes that react to input from sliders, buttons or mouse.

Other live coding environments such as Shadertoy [5] recompile the code, in this specific case the code of fragment shaders, at runtime and immediately display the updated results. This allows developers to quickly and iteratively tweak and refine their code.

RiftSketch [4], CodeChisel3D [3] and Rumpus [1] already provide live coding environments for virtual reality. The former two are browser-based applications using three.js [1], the latter is an application that is available on Steam. All of them are developed as live coding for VR scenes, e.g., creation and animation of objects in a scene, and they are tied to specific basic text editors.

To the best of our knowledge, our framework is the first live coding framework for VR that gives developers the ability to tweak the underlying render engine at runtime. RiftSketch and CodeChisel3D have a similar ability at least in theory, but the underlying render engine, three.js, is not built to be modified at runtime. For example, many functions in the renderer are private and cannot be modified at runtime without replacing the whole renderer.

*e-mail: mschuetz@cg.tuwien.ac.at

†e-mail: wimmer@cg.tuwien.ac.at

[1] https://threejs.org/

2 Live Coding

Our C++ & Javascript live coding application allows developers to modify and tweak the underlying render engine directly in VR. Developers can add, remove and modify OpenGL calls, which allows them to fix errors, tweak existing rendering logic and even add new functionality without the need to take off their head-mounted display. This is possible because of our custom rendering engine, which is written mostly in Javascript and with modifications at runtime in mind. Rendering logic is distributed in various files and whenever one of these files is modified and saved, it will be executed immediately. Upon execution, the executed code will replace respective parts of the old render engine with new functions due to statements like *renderBuffers = function(view, proj, target){ ... }*, which replaces the old renderBuffer function with a new one. In case of classes, functions can be overridden for all existing instances by modifying the prototype of the class. For example, *BrushNode.js* defines the update function of Brush scene nodes as *BrushNode.prototype.update = function(){ ... }*. The first time that this code is executed, it will add an update function to the BrushNode class. Each consecutive time it is executed, it will replace the current update function with a new one. This architecture allows developers to rewrite most parts of the render engine at runtime by modifying and then saving files, where saving of a file automatically triggers its execution.

Users can use any IDE or text editor of their choice to live code, since a 1280x720 pixel large region of the upper-left part of the desktop monitor is mirrored in VR. Our IDE of choice for the live coding framework is Visual Studio Code [2] because it is relatively slim with a low amount of visual clutter, thereby leaving more available space to the code, which is especially important due to the low resolution of the desktop mirror.

We currently limit the resolution of the desktop mirror to 1280x720 pixels due to the relatively low resolution of currently available HMD displays. The HTC Vive Pro, for example, has a display resolutions of 1440x1600. Desktop resolutions larger than that consequently produce resampling artifacts on display devices with a lower resolution. The main problem, however, is that large desktop resolutions lead to smaller UI elements, which won't be visible or appear blurry, especially text. In our tests, 1280x720 has shown to be a good trade-off between amount of content that fits on screen and legibility when viewed in VR.

2.1 OpenGL Shaders

Live coding of OpenGL shaders is a basic functionality that is easily integrated even in rendering applications that otherwise need compi-

[2] https://code.visualstudio.com/

lation. This is usually limited to changing the computations within the shader, however, and does not allow changing shader inputs without respective changes to the compiled code that feeds these inputs. Live coding of shaders is also a core part of our framework, with the additional possibility to modify shader inputs such as uniforms and uniform blocks, because the Javascript code that feeds these inputs can also be changed accordingly at runtime.

2.2 OpenGL

Coding an OpenGL application regularly involves trial & error, especially for beginners. Developers may have to test multiple configurations of state-changing calls, such as *gl.enable* and *gl.disable*, or experiment with parameters to calls such as *gl.blendFunc*. Our live coding framework allows developers to do these experiments at runtime without recompilation, and even inside VR in order to judge the impact to the renderings within the virtual environment, which can be a very different experience compared to viewing the renderings on a desktop monitor.

2.3 OpenVR

While live coding, developers have access to the pose matrices and state of the virtual reality devices, including HMD and controllers. The default controller behavior, a direct mapping of physical pose to virtual pose, is one of the first things that we developed during a live coding session. We used our live coding framework to tweak the mapping from input pose to their representation of the controllers inside the scene graph, and also to map the XY location of the finger on the controller's trackpad to a certain functionality.

2.4 Drag & Drop

We provide a drag & drop feature for live coding where developers can pick one of the scene objects at runtime, and drag it towards the desktop mirror. Dropping the object on the desktop mirror creates a selector code-snippet at the respective location in the source code, for example *$('house.window_1')*, which can be used to retrieve an instance of this object and then modify it programmatically.

3 Implementation

Our live coding application is implemented in C++ and Javascript. We use Google's V8 engine to map C++, OpenGL, and OpenVR calls to Javascript. The MS Windows desktop duplication API is used to obtain a DirectX texture of the desktop, and the NV_DX_interop OpenGL extension is employed so that this texture can be used in our OpenGL live coding application.

During startup, C++ generates the OpenGL window and executes the *start.js* script file once. During the render loop, it repeatedly calls the *update* and *render* script functions, which handle all the input and rendering, and which can be modified at runtime by the developers.

Our application comes with a basic Javascript rendering engine that is developed in a way so that re-executing source files at runtime will replace already running parts with new code.

The desktop mirror is a quad with a texture of the desktop monitor. We render the quad with mipmapping enabled, at least 4x multisample anti-aliasing, and 16x anisotropic filtering in order to improve readability and reduce discomfort from aliasing artifacts under motion in VR.

Drag & drop of scene objects into the desktop mirror is implemented by invoking windows API commands to simulate the mouse moving to the drop location, a left click, pressing the *end* key, pressing the *enter* key, and finally, pressing *ctrl + v* to paste the code from the clipboard.

4 Performance

Our test system consists of an AMD Ryzen 5 1600 CPU and an NVIDIA GTX 1060 GPU. Performance was measured with VR turned off to illustrate the overall performance achievable with our system. The performance impact of VR on our system is the same as for native C++ applications.

The duration from starting the application to the first execution of the rendering loop is 1.37s. The full timeline is: 0.66s until the OpenGL windows is created, 0.69s until the V8 Javascript engine and all C++/JS bindings are set up, and finally 1.37s until start.js has been executed. Start.js creates a default scene consisting of a skybox, a ground plane, loads respective images and materials, and compiles various shaders for meshes, point clouds, and post processing.

During each iteration of the render loop, the script functions *update* and *render* are called. In their most basic form, with an empty update script and a render script that sets the clear color and then clears color and depth buffer, the application needs about 0.196ms to render a frame (5,100 frames per second). A simple scene with a skybox, a ground plane, and a desktop mirror takes about 0.666ms to render (1,500 frames per second).

5 Limitations and Future Work

Live coding environments have quirks that developers need to be aware of. For example, restarting the application may lead to unexpected behaviour if the developer did not produce code with a restart in mind. A simple example would be a live coding session with a variable that is already initialized and in use. The developer then removes the code that initializes the variable, but not the code that uses it, which will keep working for the current session, but result in an error after a restart.

Users of this application also have to take care about resource loading and generation in code that is frequently executed during a session. For example, a source file that creates scene objects and loads textures from disk should make sure to account for existing instances, either by updating existing instances, doing nothing if they already exist, or by removing and unloading them first.

We currently don't visualize keyboard or mouse inside the VR environment. This hasn't shown to be an issue for the mouse, but users of our application need to be able to use their keyboard blindly. In the future, we would like to experiment with see-through via the cameras on the HTC Vive. We omitted it for now because initial tests have resulted in severe impacts on the performance if the cameras were enabled.

6 Conclusion

We have shown a live coding application for VR that allows developers to modify the underlying render engine without leaving the virtual environment. It is neither intended nor possible to be used during all development stages, but it promises to be a significant productivity boost in some, especially during prototyping and tweaking phases, but also partially during debugging. This application was already successfully used by ourselves to develop, evaluate and benchmark rendering algorithms for ongoing research at our institute.

The source code of our live coding framework is available at https://github.com/m-schuetz/Fenek.

References

[1] L. Iannini. Rumpus: A livecoding playground for room-scale vr. http://rumpus.land/.
[2] P. Jupyter. Jupyter. https://jupyter.org/.
[3] R. Krahn. Codechisel3d: Live programming with three.js and webvr. https://robert.kra.hn/past-projects/live-programming-with-three-and-webvr.html.
[4] B. Peiris. Riftsketch: An html5 live-coding environment based on webvr. https://github.com/brianpeiris/RiftSketch.
[5] I. Quilez and P. Jeremias. Shadertoy. https://www.shadertoy.com/.
[6] W. Research. Mathematica. http://www.wolfram.com/mathematica/.
[7] I. The MathWorks. Matlab live editor. https://www.mathworks.com/products/matlab/live-editor.html.