Kevin Loddewykx

Documentation v1.0

Shader Box

A HLSL library and Editor Tool

Contents

1 ShaderBox

Shader Box is an open-source tool that allows you to create HLSL shaders for the DirectX api, and is currently still a prototype. It supports the shader stages: Vertex, Hull, Domain, Geometry and Pixel. The UI is written in WPF with C#, and the engine which powers the rendering of the 3D viewport is custom written in C++. The application makes use of the Model-View-ViewModel (MVVM) pattern.

I, Kevin Loddewykx, started this project as a school project for the course Tool Development, where we were tasked to create a WPF tool, at Howest: Digital Arts and Entertainment. The project aims to be a replacement for FX Composer, which is no longer in development and only supports shaders up to DirectX 10 which doesn't include the Tessellation pipeline stage.

The UI is written in WPF with C#, and the engine which powers the rendering of the 3D viewport is custom written in C++. The application makes use of the Model-View-ViewModel (MVVM) pattern. The parsing and lexing of HLSL code is done by an external C# library namely HLSL Tools.

- Forward rendering
- Post Processing
- Vertex, Hull, Domain, Geometry and Pixel stages
- Toplogies: Triangle List, Triangle List with Adjacency, Patch List with 3 control points
- Cull modes: back, front and none
- Fill modes: solid and wireframe
- Include directives
- Wavefront .obj models
- Textures
- Properties panel for tweaking shader parameters
- Undo/Redo support
- snorm and unorm support

The app has three tabs each with their own subpanels: Library [1.3], Editor [1.4], Data [1.6].

1.1 Planning

Currently I am working on **rewriting** how the **3D viewport** is hooked up with the user interface. At the moment it is using the WPF DirectX Extensions project from Microsoft, which allows you to render with DirectX 11 to a image hosted inside WPF. The problem with this approach is that WPF handles internally when a new frame needs to be rendered and you need to use a Flush call on the DeviceContext to have the content appear, which is a non-blocking call and causes flickering. Also all the input needs to be captured inside the WPF application and manually send to the engine and the rendering needs to run on the UI thread.

Planned

- Docking manager + multiple text editors
- Improve integration of HlslTools: folding, code completion, ...
- C++/CLI project instead of marshalling

Backlogged

- Audio support: Fast Fourier Transform, Beat Detection as shader input
- Animation support
- File formats which support animations
- Deferred rendering
- Supporting arrays
- Supporting the row_major and column_major keywords
- Feature to make scenes which contains multiple shader projects
- Support for more topologies
- Parsing of initial values for parameters
- Blend states
- And many more...

1.2 Workings

When creating a new shader you can choose which type of shader you want to make, the passes it will need, the topology and set up the rasterizer. Afterwards Shader Box will initialize a new shader project with for each choosen pass a file. The user can always add extra header files which can be used locally so only the project can access it, or set it as a shared header so all the projects can include it. There is one built-in shared header which includes the samplerstates which the engine supports, two cbuffers one for the camera information which changes every frame and one for the per object information (currently only one object per scene). And a reserved slot for a texture, for when creating a post processing shader.

After writing your shader you can click compile or build. Compile will only compile the active shader pass and output the found errors, while build will compile the entire project to precompiled shader files and create a properties panel, for controlling the parameters. The look of the properties panel can be controlled by metadata, using the same syntax as in FX Composer. Finding all this information is done by calling the SyntaxFactory.ParseSyntaxTree(...) method of HLSL Tools on each pass seperatly while passing the shader content. Next Shader Box will go over each SyntaxTree and extract the necessary data for creating the properties, for each found cbuffer a byte array is created which stores the settings which needs to be passed to the GPU. Every variable gets a offset to where it value needs to be stored in the array, in conformance with the HLSL packing needs. Every time the user updates a variable this byte array, which is defined in C#, gets marshalled to the C++ engine.

1.3 Tab: Library

Panels:

- **Overview**: shows all the created shaders.
- **Shader Properties**: shows the variables which can be passed into the selected shader.
- **Viewport**: for visualizing the selected shader.

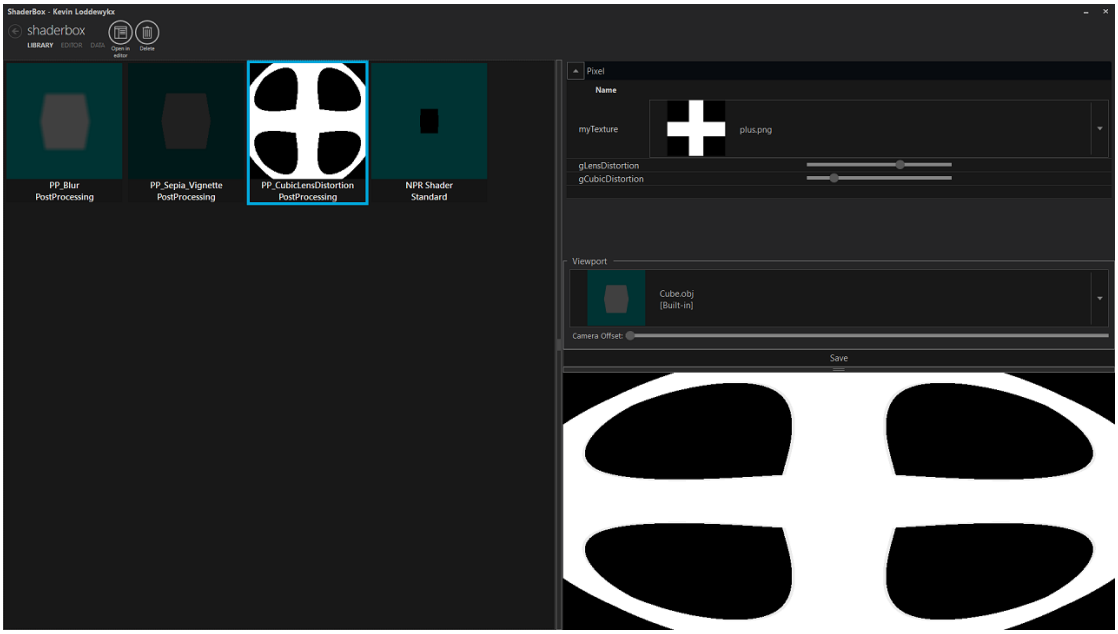

Fig. 1: Library tab: with a cubic lens distortion post processing shader selected.

1.4 Tab: Editor

Panels:

- **Explorer**: contains all the shader projects you have opened for editing inside the library overview
- **Code view**: a text editor showing the contents of the selected shader file. And an error window showing the found errors when compiling the shader.
- **Shader Properties**: shows the variables which can be passed into the selected shader.
- **Viewport**: for visualizing the selected shader.

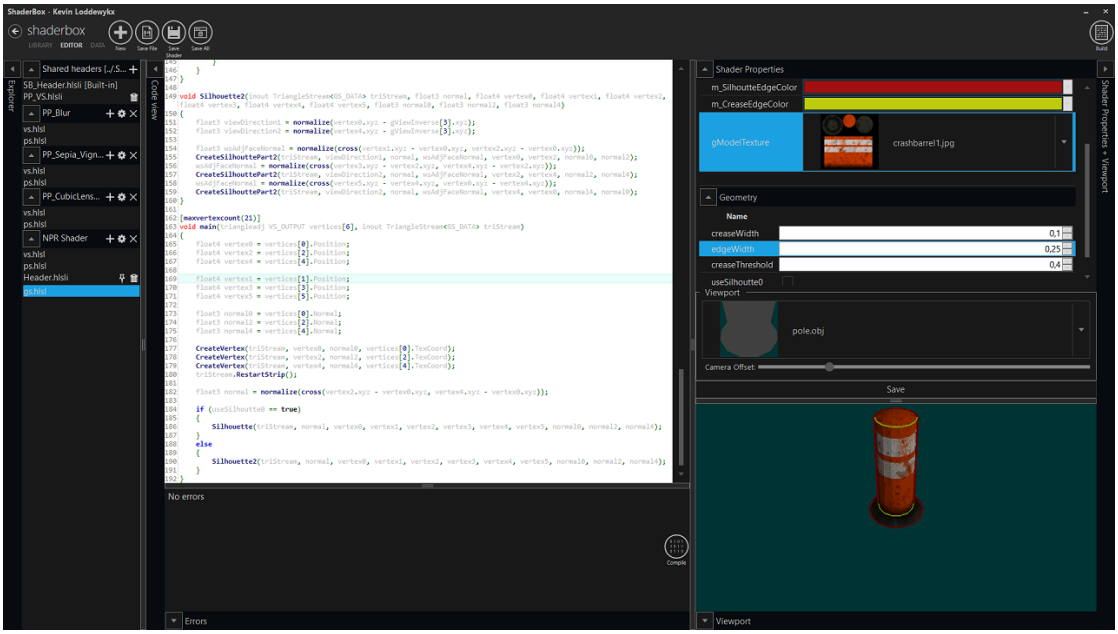

Fig. 2: Editor tab: with a silhouette and grease edge rendering geometry shader selected.

1.5 Tab: Data

Sub tabs:

- **Images**: Add and remove the image files which can be used as a Texture2D parameter.
- **Models**: The 3D model files between you can choose to display inside the viewport. Currently only Wavefront .obj are supported.

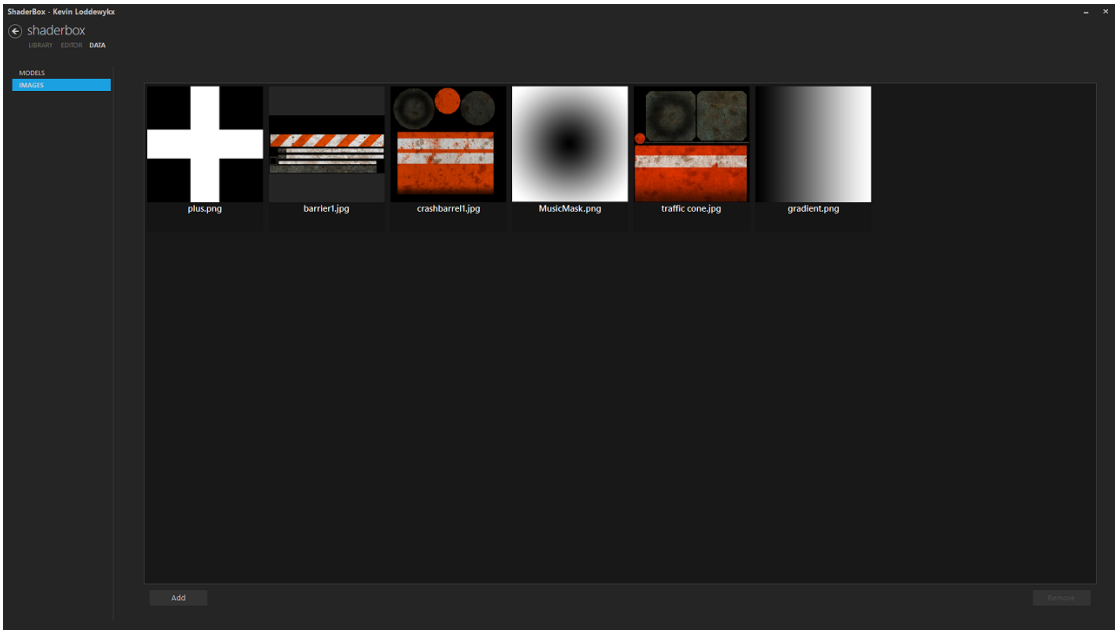

Fig. 3: Data tab

1.6 Sub window: New shader

Inside the popup window for adding a new shader project you can choose between two shader modes

- Standard

If you want to apply the shader to an object. Requires a vertex and pixel shader, and can have optionally a Hull and Domain shader and/or a geometry shader.

- Post processing

When you want to write a Post processing effect, requires a vertex shader and pixel shader, the other shader types are disabled.

Other settings you can set inside the window are:

- Topology to be used: Triangle List, Triangle List Adjacency or a 3 Control Point Patch List. Depending on the selected shader stages
- Cull mode of the rasterizer
- Fill mode of the rasterizer
- Name of the shader project
- Description of the shader project

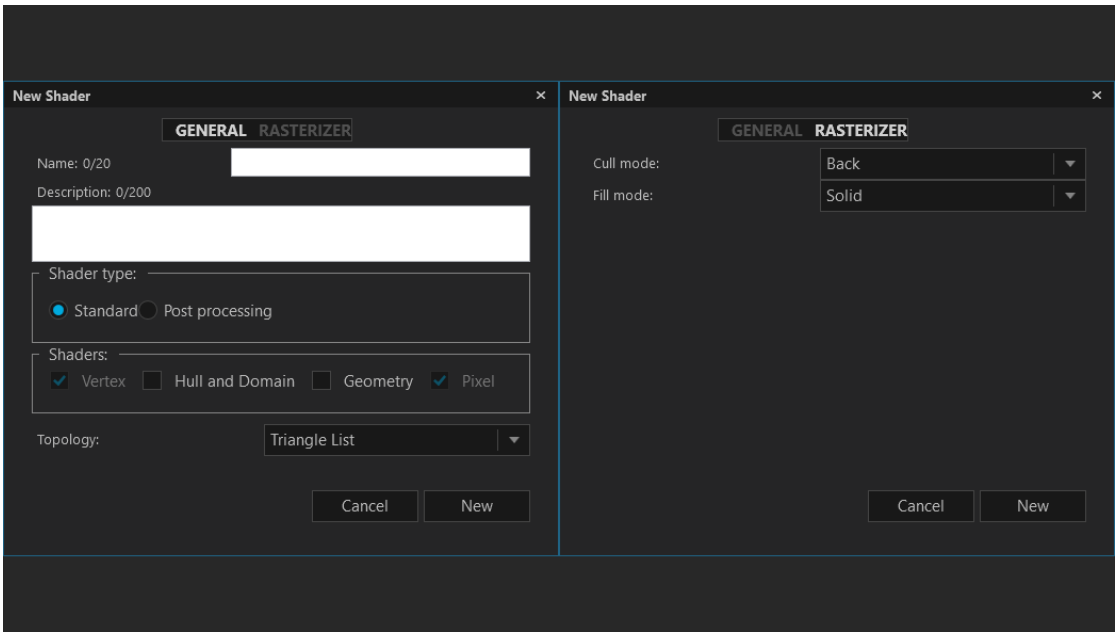

Fig. 4: New shader window: showing the two sub tabs side by side