

Manual

Shader–Evolver

Markus Reinhardt

August 3, 2004

Contents

1	Shader-Evolver	3
1.1	Installation	3
1.2	System requirements	3
1.3	Configuration files	3
1.3.1	settings.cfg	4
1.3.2	datatypes.xml	5
1.3.3	parameters.xml	5
1.3.4	operators.xml	6
1.3.5	template.cg	7
1.4	User interface	8
1.4.1	Command line (1)	8
1.4.2	Enter button (2)	9
1.4.3	Storage registers (3) und (4)	9
1.4.4	Evolve button (5)	9
1.4.5	Undo button (6)	9
1.4.6	Selected individual (7)	10
1.4.7	Fitness slider (8)	10
1.5	Command line options	10
1.6	Hot keys	10
1.7	Directory structure	10
2	Development environment	11
3	Credits	12

1 Shader-Evolver

1.1 Installation

No installation is needed to use this software. Just copy the entire directory structure to your hard disk. The folders must not be write protected. All needed DLLs are included in the bin folder. Feel free to remove them if you've got newer versions installed.

Required libraries: 'cg.dll', 'cgGL.dll' and 'glut32.dll'

1.2 System requirements

Minimal system:

- Intel Pentium/Celeron or AMD Athlon/Duron or m compatible with 1200 MHz
- 128 MB RAM
- Graphicsboard supporting the OpenGL extension GL_ARB_VERTEX_PROGRAM¹ and GL_NV_register_combiners² (Geforce 4x and higher) unterstützt

Recommended system:

- Intel Pentium/Celeron or AMD Athlon/Duron or m compatible with 2400 MHz
- 256 MB RAM
- Graphicsboard supporting the OpenGL extension GL_ARB_VERTEX_PROGRAM³ and GL_ARB_FRAGMENT_PROGRAM⁴ (Geforce FX / Radeon 9500 and higher)

1.3 Configuration files

This tool is designed to leave as much freedoms as possible for the composition of the experiments. Therefore exist a number of configuration files, which may be adapted to the respective requirements. The Shader-Evolver is a tool which does not claim to be bug free or user-friendly. Wrong usage of these files may lead to unpredictable program behavior.

¹ http://oss.sgi.com/projects/ogl-sample/registry/ARB/vertex_program.txt

² http://oss.sgi.com/projects/ogl-sample/registry/NV/register_combiners.txt

³ http://oss.sgi.com/projects/ogl-sample/registry/ARB/vertex_program.txt

⁴ http://oss.sgi.com/projects/ogl-sample/registry/ARB/fragment_program.txt

1.3.1 settings.cfg

This file contains key/value pairs. Some of these are defined by GALib others are needed for the tool to work properly.

- **population_size**
The size of the population in individuals (shaders). Needs to be in the range 1-200.
- **genome_size**
This value defines the size of one individual in integer values. Larger numbers lead to higher variation of the resolution shader programs. Too big programs may not work on all graphics hardware.

- **mutation_probability**
Defines the probability for mutations. Negative values get transformed as follows:

$$p_{mut} = \frac{|mutation_probability|}{genome_size}$$

For every individual $|mutation_probability|$ genes are mutated in this case.

- **crossover_probability**
Probability for crossover operation.
- **selection_method**
There are five possible values for this key.
See <http://lancet.mit.edu/galib-2.4/API.html#selection>
 - RANK
 - ROULETTEWHEEL
 - TOURNAMENT
 - UNIFORM
 - SRS
 - DS
- **vertex_shader**
If this parameter has the value [EVOLVE], the Shader-Evolver does evolution on vertex shaders. Otherwise the value has to be a valid vertex shader source file, which should be used.
- **pixel_shader**
If this parameter has the value [EVOLVE], the Shader-Evolver does evolution on pixel shaders. Otherwise the value has to be a valid pixel shader source file, which should be used.

- **Important**

Only one of the two keys `vertex_shader` or `pixel_shader` is allowed to have the value [EVOLVE].

- `vertex_shader_config_dir`

This is the directory in which files `datatypes.xml`, `parameters.xml`, `operators.xml` and `template.cg`, used for vertex shader evolution, are located.

- `pixel_shader_config_dir`

This is the directory in which files `datatypes.xml`, `parameters.xml`, `operators.xml` and `template.cg`, used for pixel shader evolution, are located

1.3.2 datatypes.xml

This config file is used to prevent, that undefined datatypes are used in the following files. Only values inside `<type>` elements are valid datatypes in `operators.xml` and `parameters.xml`. The datatype `<type>OPEN</type>` has a special meaning, which will be explained later on (siehe Kapitel 1.3.4, Seite 6).

Listing 1: datatypes.xml

```
<?xml version="1.0" standalone="yes"?>
<datatypes>
  <type>OPEN</type>
  <type>float</type>
  <type>float3</type>
</datatypes>
```

1.3.3 parameters.xml

This configuration file defines the parameter set (`<parameter>`). The scheme is intuitive.

Every parameter needs a name for the variable it represents (`<variable>`), a type (`<type>`, has to be defined in `datatypes.xml`) and one initial value (`<init>`). This is the value, that the parameter gets assigned at Cg program initialization. If initialization is not needed, because the variable is already defined in the template for example, `<init>NONE</init>` needs to be denoted.

One additional and optional field is `<access>`. To flag output parameters `<access>OUT</access>` is used. Output parameters are only used once at the end of the Cg program as lvalues.

Listing 2: parameters.xml

```
<?xml version="1.0" standalone="yes"?>
<parameters>
  <parameter>
    <init>NONE</init>
    <type>float3</type>
    <variable>v.xyz</variable>
  </parameter>
```

```

<parameter>
  <init>normalize( mul( modelViewInverse , IN.normal ).xyz );
  </init>
  <type>float3 </type>
  <variable>normalVec </variable>
</parameter>
<parameter>
  <init>dot( normalVec , halfVec );</init>
  <type>float </type>
  <variable>specular </variable>
</parameter>
<parameter>
  <init>lit( diffuse , specular , 32 );</init>
  <type>float3 </type>
  <variable>lighting </variable>
</parameter>
<parameter>
  <init>NONE</init>
  <type>float </type>
  <variable>animator </variable>
</parameter>
<parameter>
  <init>NONE</init>
  <type>float3 </type>
  <variable>OUT.color0.xyz </variable>
  <access>out </access>
</parameter>
</parameters>

```

1.3.4 operators.xml

The command set is saved in this XML-file. On singel command (<operator>) consists of a name (<name>) and exactly one (<syntax>) block containing one command block (<command>) and nothing else. This is the default syntax.

Listing 3 shows the operator Mov. It has the default syntax %1 = %2. This is a intuitive definition which means, that the first parameter (%1) gets assigned the value of the second one (%2).

This operator also has another syntax defined. In addition to the command block, it contains one type definition (<type>) for every parameter that this operator needs. Based on the Mov example it means the following:

If the first parameter is of type float and the second on of type float3, not the default syntax, but %1 = %2.x is used. The x component of a three dimensional vertex is assigned to a floating point variable. Additional syntax definitions may prevent type mismatches.

The third syntax of the Mov operator shows the contrary case. A vertex gets assigned the value of a one dimensional value. This shows, that placeholders for parameter may occur

several times. Every %2 gets replaced by the name of the second parameter. The syntax definitions are the place, where the type OPEN comes into play.

If one variable may be of arbitrary type, <type>OPEN</type> may be specified. This may save several syntax definitions in some cases.

Listing 3: operators.xml

```
<?xml version="1.0" standalone="yes"?>
<operators>
  <operator>
    <name>Mov</name>
    <syntax>
      <command>%1 = %2;</command>
    </syntax>
    <syntax>
      <command>%1 = %2.x;</command>
      <type>float</type>
      <type>float3</type>
    </syntax>
    <syntax>
      <command>%1 = ( %2, %2, %2 );</command>
      <type>float3</type>
      <type>float</type>
    </syntax>
  </operator>
  <operator>
    <name>Add</name>
    <syntax>
      <command>%1 = %1 + %2;</command>
    </syntax>
  </operator>
</operators>
```

1.3.5 template.cg

This is the template for every generated shader. The only important point is, that the tag %GENERATED_SHADER% is contained in this file. The generated vertex/pixel shader code is inserted at this position.

Listing 4: template.cg

```
struct appin {
  float4 position : POSITION;
  float4 color0 : COLOR0;
  float4 normal : NORMAL;
  float4 texcoord0 : TEXCOORD0;
};
```

```

struct vertout {
    float4 position : POSITION;
    float4 color0 : COLOR0;
    float4 color1 : COLOR1;
    float4 texcoord0 : TEXCOORD0;
    float4 texcoord1 : TEXCOORD1;
};

vertout main( appin IN, uniform float4x4 modelViewProj,
    uniform float4x4 modelViewInverse, uniform float animator ) {
    vertout OUT;
    float4 v = IN.position;
    // die folgenden Marke wird der
    // durch das Programmfragment ersetzt.
    %GENERATED_SHADER%
    OUT.position = mul(modelViewProj, v );
    OUT.texcoord0 = IN.texcoord0;
    OUT.texcoord1 = IN.texcoord0;
    return OUT;
}

```

1.4 User interface

Figure 1 shows a screenshot of the user interface. In the following all functions will be explained. Im Folgenden werden die einzelnen Funktionen beschrieben. Again attention is invited to the point that this software is not optimized for usability, it is only a tool.

1.4.1 Command line (1)

Commands may be entered here. Up to now, four commands are possible.

- Store a population:
The command `:w <filename>` stores the complete population (all visible individuals) to disk.
- Load a populationen:
The command `:e <filename>` loads a previously save population into the user interface. The actual population gets discarded.
- Store an individual:
The command `:ws <filename>` stores the selected individual to disk.
- Load an individual:
The command `:es <filename>` loads a saved individual and replaces the selected one.

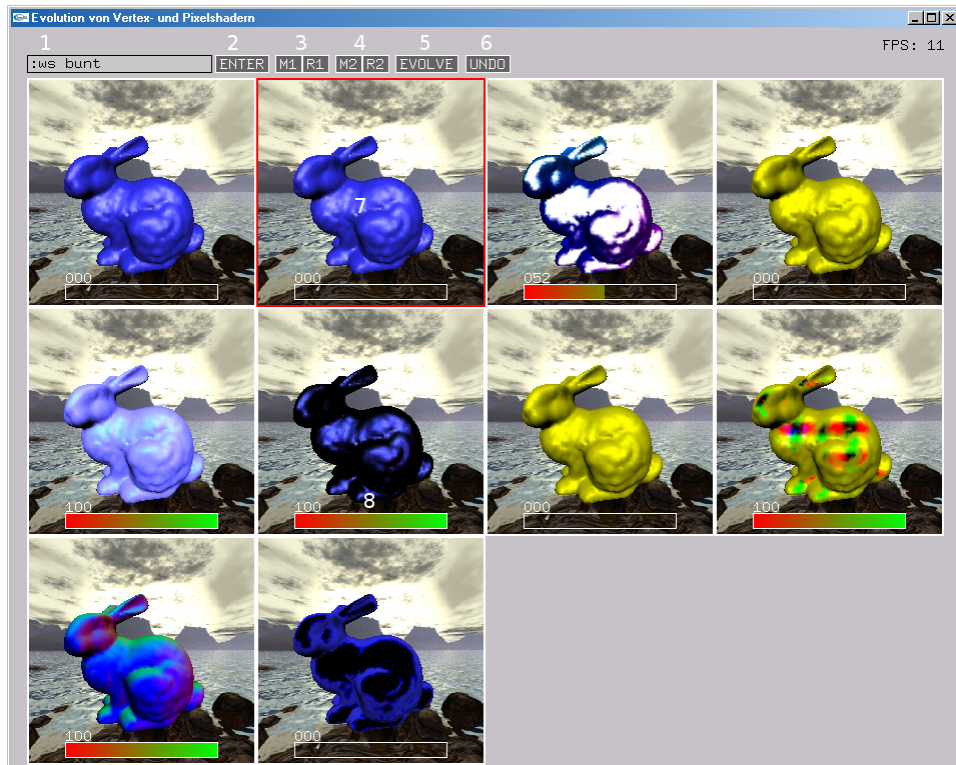


Figure 1: screenshot of the user interface

1.4.2 Enter button (2)

Submits the commands entered at (1). Alternatively the <ENTER> key may be used.

1.4.3 Storage registers (3) und (4)

There are two individual quicksave registers. M1 memorizes the selected individual. R1 replaces the selected shader by recalling the saved one. M2 und R2 provide the same functionality.

1.4.4 Evolve button (5)

This button performs one evolution step. This is not possible, if all individuals have a fitness value of zero. Alternatively the E key may be used.

1.4.5 Undo button (6)

The undo function refers to the evolution. Its possible to restore all previous population till the initial one.

1.4.6 Selected individual (7)

A red border marks the selected individual.

1.4.7 Fitness slider (8)

This slider allows to set the fitness value of one individual.

1.5 Command line options

Most of the options are defined in the configuration files. There are only two command line values.

- -v: shows some verbose runtime information
- -v2: even more verbose runtime information

1.6 Hot keys

The following shortcuts are defined

- W: switch from fullscreen to window display mode
- F: switch from window to fullscreen display mode
- S: switch shader usage on/off
- B: switch skybox rendering on/off
- O: switch through three possible objects (plane, sphere, bunny)
- C: switch command interpretation mode

1.7 Directory structure

The following shows the directory structure of Shader-Evolver

- **bin** – The runtime files and libraries are located here.
- **config** – Contains the main configuration file settings.cfg and additional directories for different predefined vertex- and pixel shader configurations.
 - **pixShader** – standard pixel shader which simply passes the color value
 - **pixShader_texture** – pixel shader which does some texturing
 - **pixShader_refract** – pixel shader which ought to be used with vertShader_refract. Calculates light reflections in glass objects.

- **vertShader** – standard vertex shader. Calculates ModelView transformation and passes texture coordinates and color values to the pixel shader.
- **vertShader_lighting** – vertex shader, which calculates lighting.
- **vertShader_refract** – vertex shader for glass objects.
- **doc** – contains this documentation
- **generated_shaders** – This folder contains temporary at runtime generated shaders. The filenames are shader0.cg - shaderXX.cg, where XX is the amount of actually displayed shaders. At each program start, old files get overwritten.
- **gfx** – Contains the needed textures.
- **save** – All user saved shaders/populations go to this folder.
 - **populations** – Populations as .pop file. The subdirectory **runtime** contains the populations needed by the undo function.
 - **shaders** – All saved shaders. Every shader is represented by one .ind file, which stores the integer values and one .cg file, which contains the Cg sourcecode and may be used directly in other applications. The subdirectory **runtime** contains shaders, which are stored in the quicksave registers.

2 Development environment

Compiler:

gcc⁵ for Linux respectively Windows.

IDE:

Dev-C++⁶ for Windows, make for Linux.

Additional modules:

- OpenGL and GLUT libraries
- CenterPoint - XML⁷
- GAlib⁸ for evolutionary computing
- Cg libraries⁹

⁵<http://gcc.gnu.org/>

⁶<http://www.bloodshed.net/dev/>

⁷<http://www.cpointc.com/XML/>

⁸<http://lancet.mit.edu/ga/>

⁹http://developer.nvidia.com/object/cg_toolkit.html

3 Credits

Special thanks go to Dr. Marc Ebner and Prof. Dr. Jürgen Albert at "Lehrstuhl für Informatik II" at the University of Würzburg, Germany. The idea of using evolutionary computing to generate vertex and pixel shaders originated by Dr. Marc Ebner. The software and composition was worked out by me, Markus Reinhardt, during a three month practical.

The software for this work used the GALib genetic algorithm package, written by Matthew Wall at the Massachusetts Institute of Technology.

-><http://lancet.mit.edu/ga/>

Shader-Evolver uses the free CenterPoint - XML library,

-><http://www.cpointc.com/XML/>

For further information, new program versions and comments feel to contact me.

-><http://shader-evolver.bitmap-friends.de>.

-><mailto:m.reinhardt@bitmap-friends.de>