GRATIN 0.3.1 DOCUMENTATION

October 6, 2015

Romain Vergne Pascal Barla

Contents

1 Gratin

1.1 Summary

Gratin is a programmable node-based system tailored to the creation, manipulation and animation of 2D/3D data in real-time on GPUs. It is written in C++ and uses Qt for the interface, Eigen for linear algebra, OpenGL for renderings and optionally OpenExr for loading and saving high dynamic range images. It is free and open source (licensed under MPL v2.0) and relies on OpenGL and GLSL to ensure wide OS and GPU compatibility. Source code and installation packages are available at http://gratin.gforge.inria.fr/.

1.2 Contribute!

You implemented a paper in Gratin, or you designed a useful node that would benefit the graphics community? Please, send it to us, either as a node (.grac file) or as a pipeline (.gra file). We will be pleased to integrate it in the next release if possible. The nodes should come with the author names, the citation of the paper (if there is a paper) and a small description explaining how to use it.

2 Compilation

Before all, make sure you have a GPU/graphics drivers capable to run (at least) OpenGL 4.1 applications. Otherwise, Gratin will not work.

2.1 Download sources

Sources can be downloaded from the web site: http://gratin.gforge.inria.fr/, or via the svn repository:

```
svn checkout https://scm.gforge.inria.fr/anonscm/svn/gratin/branches/gratin-v0.3.1 gratin
```

2.2 Compiling on Debian, Ubuntu and Linux Mint

Install external packages

```
$ sudo apt-get install cmake qtbase5-dev libqt5svg5-dev libeigen3-dev libopenexr-dev
```

Building Gratin

```
$ cd gratin
$ mkdir build
$ cd build
$ cmake ..
$ make -j4
```

Launching Gratin

```
$ ./gratin [-in path/to/pipeline.gra]
```

Remark: Gratin will not compile if QT version is less than 5.2. On old Linux distributions, QT5 might not be available natively. In that case, download and install from the web site: http://www.qt.io

Other Linux distributions: apart from the way external packages are installed, the installation should be equivalent.

2.3 Compiling on Mac OSX

We provide an installation package for Mac OSX (Yosemite and later releases) on our Website: http://gratin.gforge.inria.fr/. Otherwise, Gratin may be compiled using the following steps.

Install external packages

Install required packages (Qt, OpenEXR, Eigen3) from the packaging system of Mac OS.

Building Gratin

```
$ cd gratin
$ mkdir build
$ cd build
$ cmake .. -DCMAKE_CXX_FLAGS='-DOPENGL_MAJOR_VERSION=4 -DOPENGL_MINOR_VERSION=1'
$ make -j4
```

Launching Gratin

```
$ ./gratin [-in path/to/pipeline.gra]
```

2.4 Compiling on Windows

We provide a binary package for Windows 64 bits on our website:
http://gratin.gforge.inria.fr/.

Otherwise, the simplest to compile Gratin is to use qtcreator: http://www.qt.io/download/. You should download and install all the required packages and open gratin/CMakeLists.txt via qtcreator. Once done, you should provide the paths to the libraries in the cmake options before building the solution.

The main problem with the Windows compilation comes from the (optional) OpenEXR library that is quite difficult to obtain. It requires several weird steps that we do not describe here.

2.5 Cmake options

A cmake error might be due to missing paths for the external libraries. In that case, you must provide the good paths using "-DLIBNAME_{INCLUDE/LIBRARY}_PATH", where LIBNAME is the name of the missing library and INCLUDE or LIBRARY specifies if the target is a source or a lib file. For instance, to specify the path to eigen, use:

```
cmake -DEIGEN3_INCLUDE_DIR="path/to/eigen3" ..
```

If cmake does not find Qt, use the following option:

```
cmake -DCMAKE_PREFIX_PATH="path/to/qt" ..
```

for changing the OpenGL version (default=4.2), use:

```
cmake -DCMAKE_CXX_FLAGS='-DOPENGL_MAJOR_VERSION=4 -DOPENGL_MINOR_VERSION=1' ..
```

In that case, the version is set to OpenGL 4.1.

3 Interface

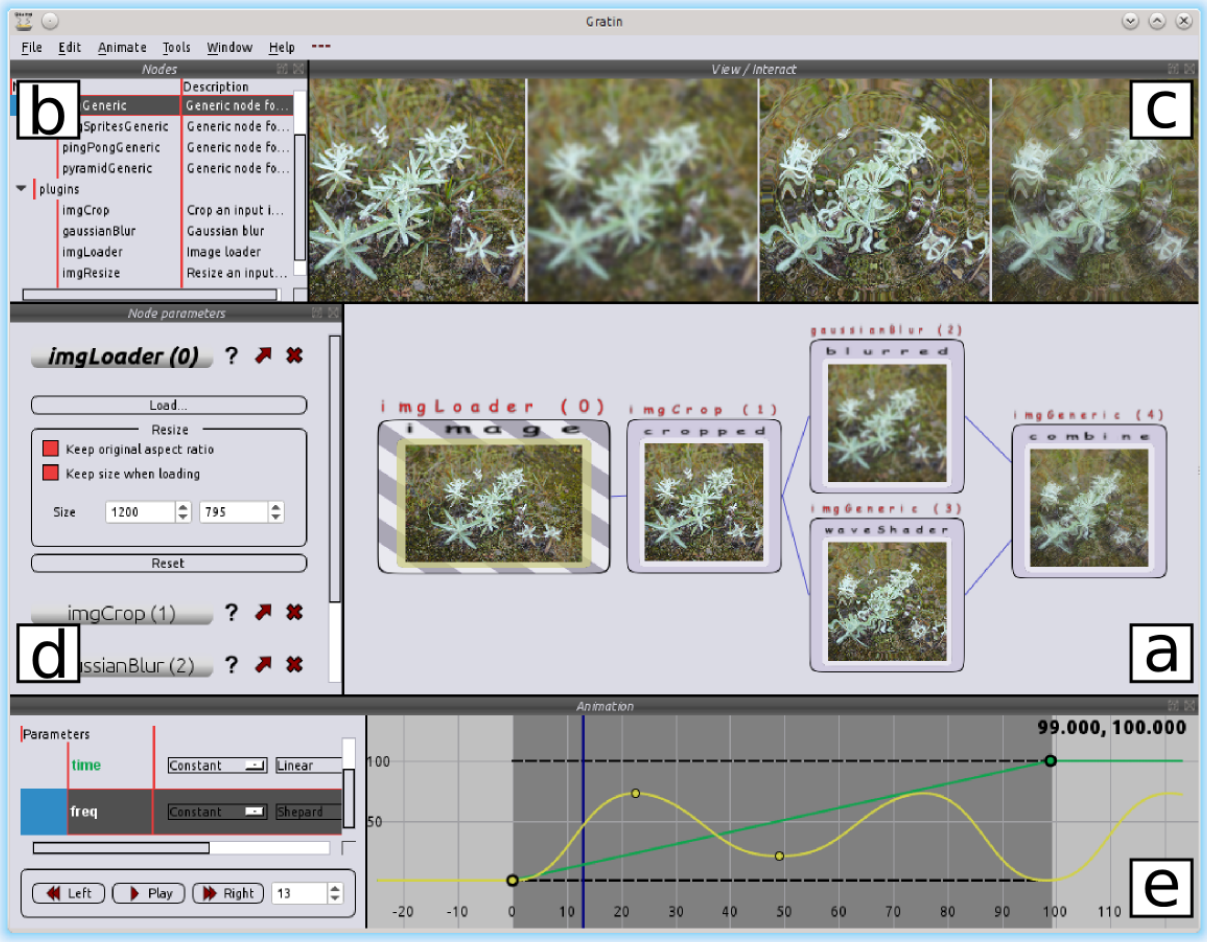

Figure 1: Interface of Gratin. (a) Graph visualization. (b) List of available nodes. (c) Node viewer. (d) Node interfaces. (e) Animation parameters and curves.

As shown in Figure 1, the interface of our system is composed of five main panels. The pipeline panel (a) where one can interactively add, connect, disconnect, copy or paste nodes. Node outputs are visualized in real-time inside the interface. All available nodes are stored in user-defined directories and automatically loaded in the node tree (b) at initialization. The viewer (c) permits to display particular node outputs and manipulate them via keyboard or mouse events. Each node has its own user-interface that can be displayed and manipulated via a list of widgets (d). Any node parameter may be keyframed and interpolated via control curves in the animation panel (e).

3.1 Menu and global shortcuts

File menu

- New (ctrl+n): clear everything.
- Open (ctrl+o): open an existing pipeline.
- Save (ctrl+s): save/resave the current pipeline.
- Save as: save the current pipeline in a new file.
- Exit: quit the application.

Edit menu

- Copy (ctrl+c): copy selected nodes inside the pipeline.
- Paste (ctrl+v): paste selected nodes inside the pipeline.
- Select all (ctrl+a): select/unselect all the nodes of the pipeline.
- Reload: does nothing (only for debugging new implemented nodes).

Animate menu

- Play (p): start the animation.
- Stop (shift+p): stop the animation.
- Next frame (shift+right): compute and show the next frame.
- Previous frame (shift+left): compute and show the previous frame.
- First frame (shift+up): compute and show the first frame.
- Last frame (shift+down): compute and show the last frame.
- Anim settings: choose the number of frames and the framerate.

Tools menu

- Group (ctrl+g): group selected nodes (only if they form some directed acyclic graphs).
- Ungroup (ctrl+shift+g): ungroup the selected node (only if the node is a group).
- Export node output (ctrl+w): save a node output image (if an output is selected).
- Export node output animation (ctrl+shift+w): save all frames of a node output image (if an output is selected).
- Add node to list: if a node has been customized, this option allows the user to add the node inside the list of available nodes (panel b). The user has to choose a unique ID, a version, a name, a description and a help string that will be used for the new node. He also has to choose, in the list of available directories, the one in which the node will be saved.
- Manage node paths: allows the user to add or remove directories in which Gratin will look for new nodes when launched. This action needs a restart of Gratin.

Tools menu

- Window/show-hide: show or hide panels (b), (c) and (d).
- Window/zoom: zoom/resize pipelines and images in panels (a) and (c).

Help menu

- Help: show the help widgets for all available nodes.
- About: display some information about Gratin.

3.2 Pipeline panel (a)

Summary of controls

- Left click on a node: select it. If the click was done on an image, this action also selects the corresponding output inside the node.
- Double left click on a node: display its interface (panel d).
- Left click on the background : unselect all.
- Left click, drag, drop on the background: select multiple nodes.
- Right click, drag, drop on the background: move the scene.
- Left click on an output, drag and drop on an input: create a connection.
- Wheel/middle click: zoom in/out.
- Left click on a selected nodes, drag and drop: move selection.
- Press space: add/remove a selected node output inside the viewer (panel c)
- Press suppr: remove selection from the graph.

3.3 Node panel (b)

The node panel contains the tree of available nodes that can be added and combined inside the pipeline. A small description is also shown for each node. A double left click on a node will automatically insert it inside the graph.

3.4 Viewer panel (c)

Summary of controls

- Left click on an image: select the corresponding node.
- mouse/keyboard on top of a selected node: interact with the node (depends on the node).
- Left click on the background / press esc: unselect everything.
- Right click, drag, drop on the background: move the scene.

- Wheel/middle click: zoom in/out.
- Press suppr: remove selected node from the viewer.
- Press ctrl+right: move selected node to the right.
- Press ctrl+left: move selected node to the left.

3.5 Node interface (d)

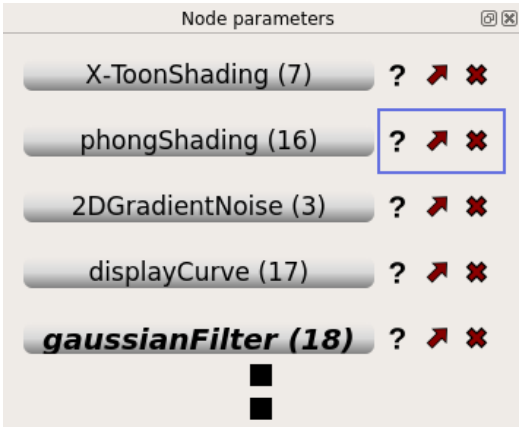

Figure 2: Each node interface is displayed as a list of widgets in the panel d.

The panel d contains a list of node interfaces opened by the user (via a double click on a node), as shown in Figure 2. Each widget thus contains parameters specific to each node as well as 3 buttons (as shown in the blue square):

- a help button (showing a small widget containing the documentation of this particular node),
- a detach button (detach the window - useful when programming inside the interface),
- a close button that removes this interface from the list (simply double-click again on the node to show it again).

Figure 3: A keyframable parameter.

Each parameter contained in the node interface might be animated using keyframed curves. A keyframed parameter can be easily recognized by its associated 2 buttons, as shown in Figure 3:

1. create a keyframe at the current frame with the chosen value inside the interface.
2. Add the parameter inside the animation panel (d) in order to control its curve(s).

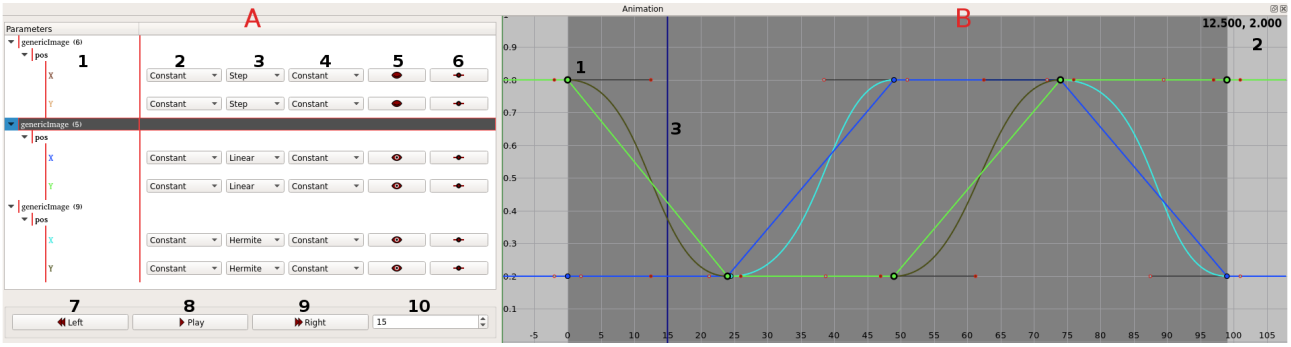

Figure 4: The animation panel

3.6 Animation panel (e)

The animation panel is composed of 2 main widgets, as shown in Fig. 4. The first one (A) contains the list of parameters currently edited by the user, their options and the player. The second one (B) shows the corresponding keyframed curves that can be edited directly.

A: list of parameters

1. parameters are displayed in a tree widget, containing the node name (root), the parameter name and each of its components.
2. control the behavior of the curve before the first control point (see Fig. 5).
3. control the type of interpolation used in-between control points (see Fig. 5).
4. control the behavior of the curve after the last control point (see Fig. 5).
5. show/hide the curve in widget B.
6. clear all control points for this curve.
7. (shift+up) place the current frame at the beginning of the animation.
8. (p and shift+p) play/stop the animation
9. (shift+down) place the current frame at the end of the animation.
10. show and select the current frame.

Available curve behaviors and types are summarized in Fig. 5.

B: curve edition

All selected curves can be visualized and edited in widget B. Control points are displayed with small dots (1). Their positions are displayed on the top right corner (2). The current frame is also displayed with a blue line (3). Here is a summary of the controls in this widget:

- Left click on the frame bar and drag: modify the current frame.
- Left click on a control point: select the corresponding curve.
- Left click on a control point and drag: modify the position of the control point.
  - +ctrl: use big steps to control the position.
  - +shift: use small steps to control the position.

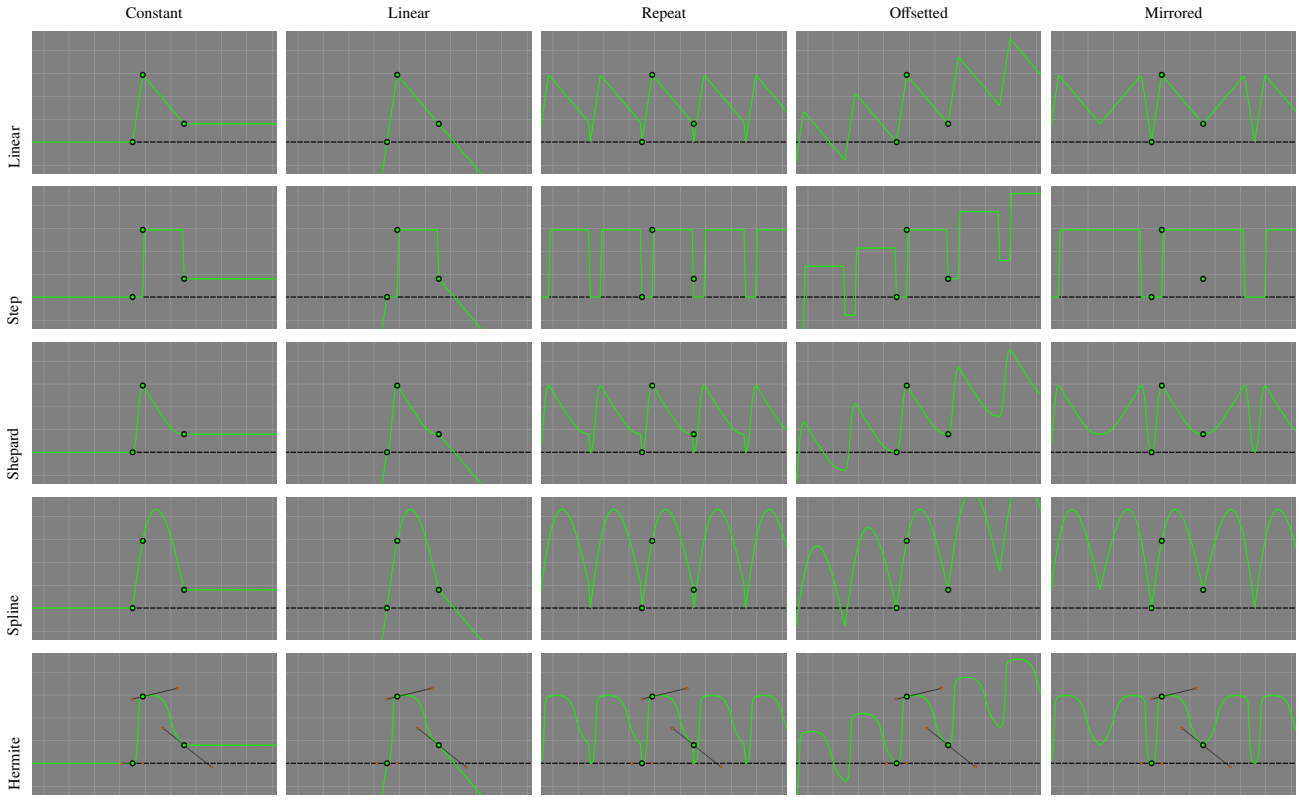

Figure 5: Curve types and behaviors obtained with three control points. The y-axis shows the types of interpolation implemented in our system. The x-axis shows the modes controlling the behavior of the curves before and after the first and last control points.

- Ctrl+right click on the background: add a control point to the selected curve.
- Ctrl+middle click on a control point: remove the control point from the selected curve.
- Right click on the background: move the scene.
- Wheel: zoom in/out.
- Ctrl+wheel/middle click and drag horizontally: horizontal scaling.
- Shift+wheel/middle click and drag vertically: vertical scaling.

4 Programming generic nodes

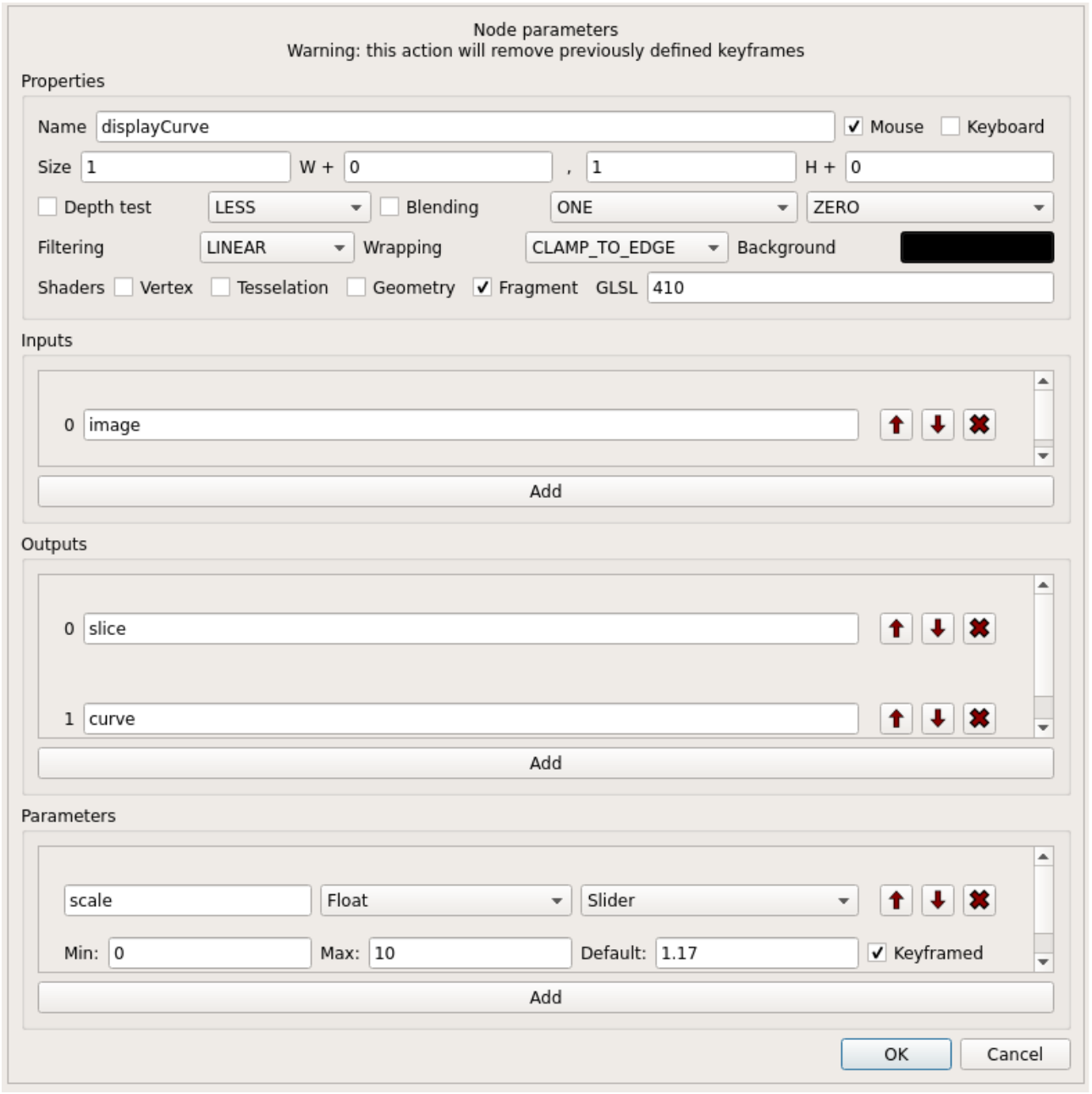

Figure 6: Settings of a generic node.

Generic nodes provide users with the ability to precisely customize processes to their needs by writing GLSL shaders directly from inside the user interface. Most of the nodes available in Gratin are in fact designed with generic nodes and their code might be directly modified by users. The creation of a generic node begins with the filling of a dialog box where users specify important parameters of a GLSL shader (Fig. 6):

- Properties: contains the name and other general parameters for the node. Check mouse and/or keyboard will grant access to mouse positions and keyboard keys in the shaders. The size of the output images are chosen via a function of the input texture sizes W and H (0 if no input). The user may also enable the depth test (with commonly used OpenGL functions), the blend-

ing, the filtering and the wrapping used for the output textures (see the OpenGL documentation - https://www.opengl.org/ - for more information on how these options will affect the resulting images). Finally, the user can decide which shaders to activate (vertex/tesselation/geometry/fragment), choose the GLSL version and a default background color.

- Inputs/outputs: choose the number of input (resp. output) images and modify their names. Chosen names will be directly used inside the shaders.
- Parameters: add/remove necessary parameters. For each parameter, one may choose a name, a type (int/float/vec2/etc) and how it should appear in the node interface (slider/spin/etc). Min, max and default values should also be provided. Finally, checking the keyframed checkbox will allow the parameter to be controlled by keyframe curves.

Once the settings have been chosen, users may customize the node behavior by writing GLSL code and observing results in real-time. Input and output names are used to automatically generate the head of the shaders. All the parameters are also available via generated uniform variables. In the following, we present the different types of generic nodes currently available in Gratin. For more information about how to program using GLSL, visit: https://www.opengl.org/documentation/glsl/. The provided pipeline samples also illustrate how each of these generic node has been used to produce different effects.

> Remark: modifying the settings of a node will remove all previously defined keyframes for this node

4.1 Generic image node

This node permits to analyze, manipulate and visualize input textures (e.g., colored images, g-buffers). As commonly done for image processing in OpenGL, a simple quad is drawn in the viewport so that input textures can be easily accessed and mapped to create outputs containing specific effects. This node may be used to create one-pass custom complex shaders (as in Shadertoy for instance), to mix input images, to create various patterns and so on. Implementation-wise, user-specified GLSL version, input and output texture names are automatically used to generate the header of the shader. By default, the fragment shader simply makes a copy of the first input texture.

4.2 Generic buffers and grid nodes

They let users apply any effect to 3D meshes by customizing vertex, tessellation, geometry and fragment shaders. The main difference between object and grid node types is that the former loads a mesh in OBJ format, while the latter creates a planar grid. In the case of the grid node, tessellation is chosen by the user in the dedicated interface; vertices are then typically displaced according to input GLSL code. As with other generic nodes, any number of textures might be provided as input (such as color or normal maps for instance). Outputs are typically in the form of g-buffers or renderings for further 3D or 2D processing respectively. A trackball camera is associated to this node so that users may manipulate their object in the viewer panel. In both cases, mesh positions, normals, tangents

and texture coordinates are directly sent as vertex attributes. Consequently, the header of the vertex shader is adapted to grant access to these attributes.

4.3 Generic splat node

This node permits to manipulate point sprites and is useful to control particles, visualizations or even image warpings. The particularity here is to be able to modify splat sizes and locations (possibly using overlapping and blending) to obtain specific effects. The interface permits to control the number of rendered sprites, and their behavior is controlled through GLSL code. In practice, it works by sending a set of point sprites to the GPU, with one splat per pixel by default. Shader headers for the generic splat node contain the position of the splat (as an attribute in the vertex shader), as well as uniform variables. The remainder of the shader can be freely modified.

4.4 Generic pyramid node

This node creates one or more mipmaped textures where one can control how each level is computed. It might be used for the creation of usual mipmaps, but also for multiscale analysis (Gaussian or Laplacian pyramids) or to compute global information such as the mean and variance of an input image. The shader header is automatically generated and contains variables such as the number of levels and flags to know whether the top or bottom of the pyramid have been reached. In addition, the previously computed level of each input texture is automatically added to the GPU program as a uniform sampler. Users thus only have to describe per-level operations in a specific order (top-down or bottom-up) in the GLSL code. The resulting pyramids are stored as mipmaps. Further connected nodes may thus easily access any texture level via GLSL built-in functions such as *textureLod()*.

4.5 Generic ping-pong node

This node provides another useful feature commonly used in GPU programming to implement iterative processes. The same process is applied at each internal pass, and repeated a number of times specified by the user. Such a type of node may be used to iteratively accumulate or propagate some information in the output textures. In practice, it uses a pair of textures for internal multiple passes, with one texture being read and the other written on even passes, and the opposite on odd passes. However, this is not apparent to the user: we automatically generate a header that gives access to the resulting texture of the previous internal pass, as well as the current pass number. Note that such a ping-pong architecture could not be created manually by connecting simpler nodes, since our graph is acyclic.

> Remark: when opening a pipeline containing a ping-pong node, it will by default be initialized and run for the default number of passes. This might lead to important lags when each pass is a time-consuming process.

5 Pipeline samples

A good start for designing new pipelines is to have a look at and take inspiration from the provided samples available in "data/pipes". They describe how to use generic nodes and show how to easily and quickly obtain multiple effects. This section presents the samples.

5.1 image-operators.gra

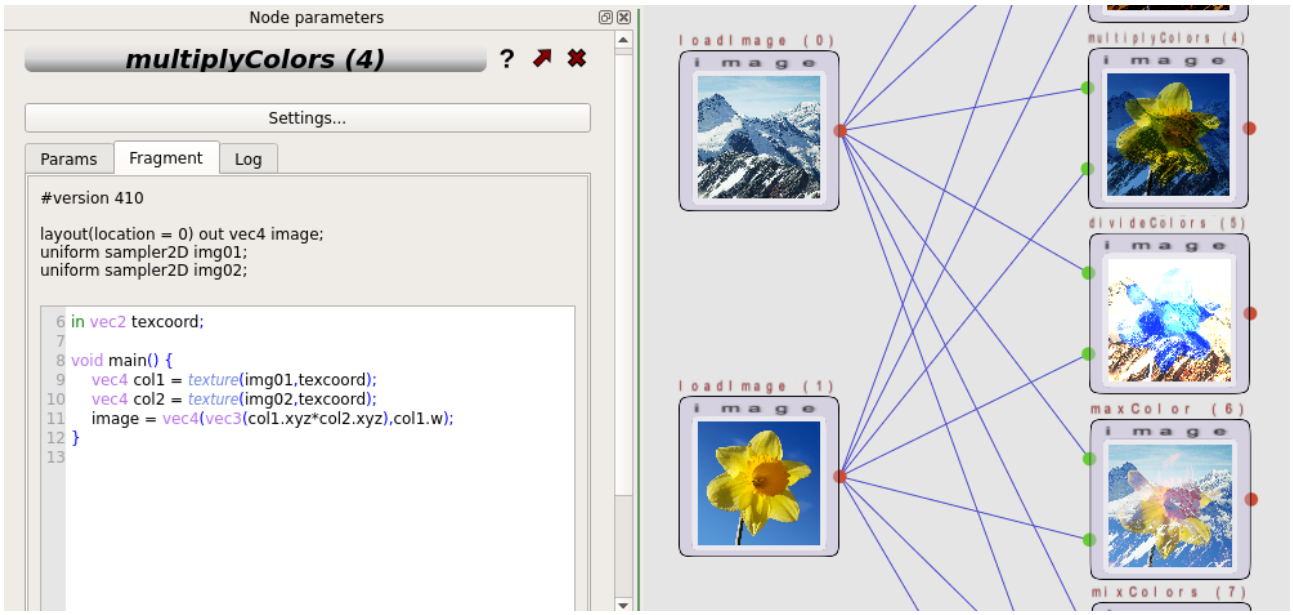

Figure 7: The image operator pipeline example.

image-operators is the simplest pipeline and shows how to combine two input images using simple image operators ($+, -, *, /, etc$). Images are loaded with a "loadImage" node. All the combining operations are done with some "genericImage" nodes (Fig. 7). In these nodes, the fragment shader is executed independently on each image pixel. One can visualize the code by double-clicking on a node and choosing the "fragment" tab in the node interface. Of course, you can modify the settings and the code to obtain your own desired effect. The only node containing a parameter in these examples is the "mixColors" node that linearly interpolates between 2 images based on a user-chosen value (in the "Params" tab of the node interface).

5.2 color-conversion.gra

This pipeline illustrates the use of color conversion nodes to manipulate image colors (Fig. 8). Except for the loader, all nodes were created from the genericImage node. The top row first convert an input RGB image into the HSV color space. A node is then provided to control and manipulate HSV parameters (double click on the "modifyHSV" node to try it via the interface). The last node convert the HSV color back to the RGB color space, so that the image can be displayed properly. The bottom row is similar, except that colors are converted into the $L\alpha\beta$ color space.

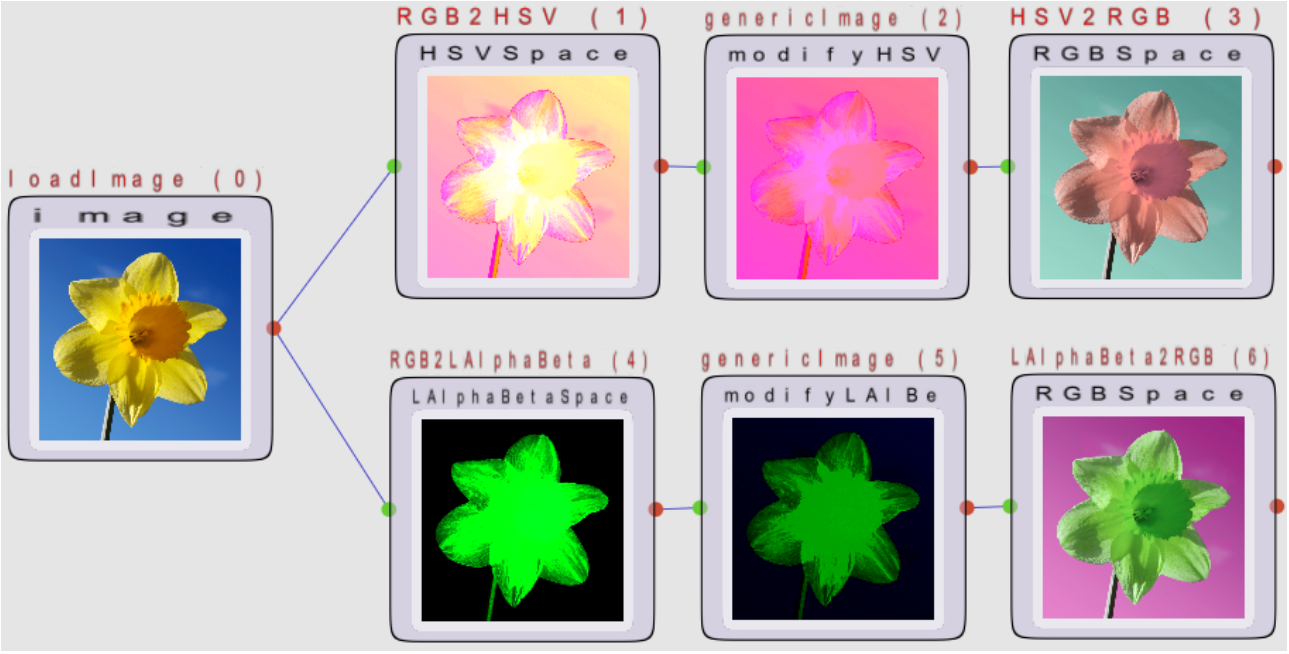

Figure 8: The color conversion pipeline example.

5.3 image-filtering.gra

This pipeline (Fig. 9) illustrates the use of currently available filtering nodes. The gaussian filter (left) was not implemented in a generic node because it was optimized with a two-pass convolution kernel. The bilateral filter (second) is based on a genericImage node. The intensity and spatial sigma parameters can be controlled in the node interface. The third example shows how a genericPingPong node was used to obtain an anisotropic diffusion that iteratively diffuse colors everywhere except on edges. The fourth example uses the same process to obtain the rolling guided filter. The fifth example is based on a genericImage node and detects edges using a Sobel filter.

Remark: citations for implemented papers are provided inside the node descriptions/helps.

5.4 curves-surfaces.gra

This pipeline shows how to use nodes to visualize isolines, curves and surfaces. The node "displayCurve" is based on a genericImage node. It allows the user to easily modify a function inside the GLSL code and visualize the resulting curve. The top row of Fig. 10 shows some variations of this

Figure 9: Image filtering pipeline example.

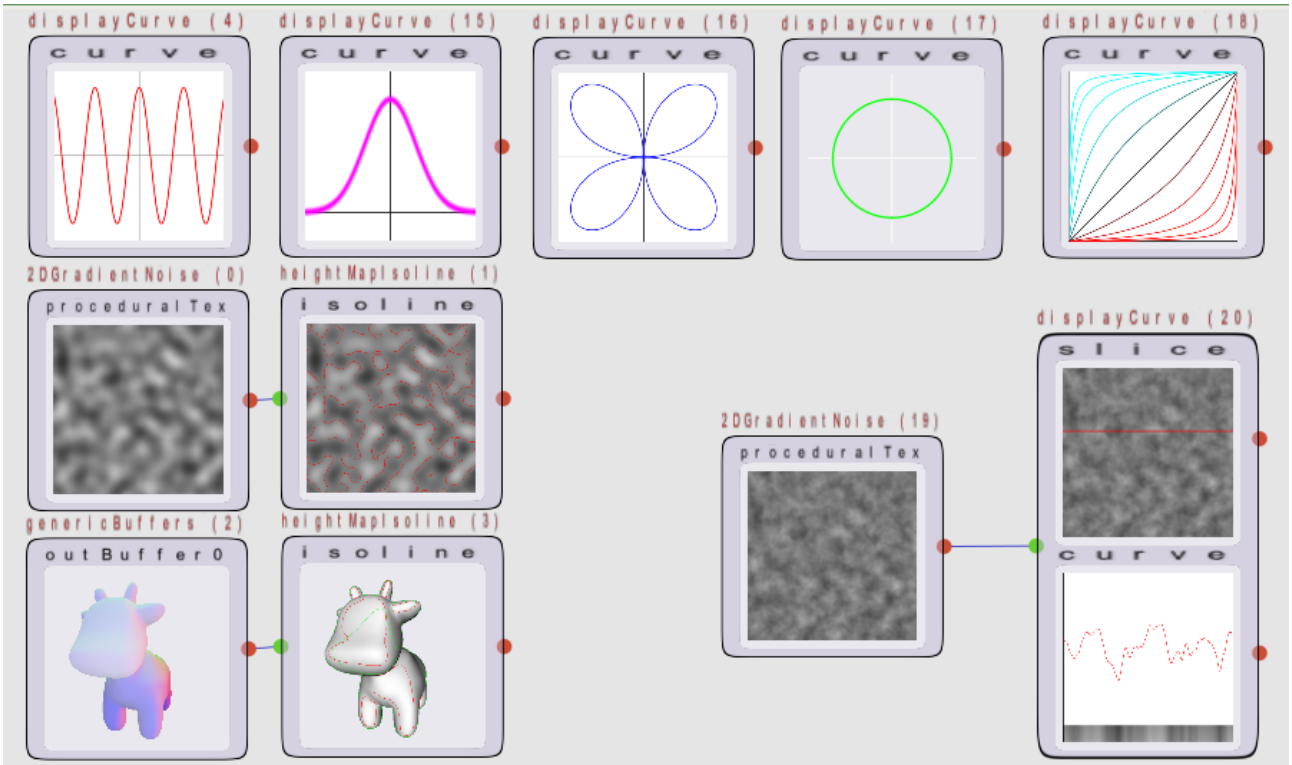

Figure 10: The curves examples.

node, with different functions, thicknesses and colors. To set your own curve, open the “fragment” tab inside the node interface and modify the equation inside the “evalFunc” function. Any parameter may be added to the interface to control the curves in real-time, as shown in the sine function node.

In the bottom left of the figure, 2 examples of isoline visualization are given. Again, this node is based on a genericImage node and might be modified depending on user expectations. The first example shows the isoline on a 2D noise texture. The isovalue can be controlled inside the node interface. In the second example, the GLSL code was slightly modified to simultaneously show 2 isoline curves, based on 2.5D information as input (surface slant in red and depth in Green).

Finally, the bottom right example is a variation of the “displayCurve” node that was modified to display a slice of an input image. This slice can be interactively changed in the viewer panel: click on the slice and associated curve image and press “space” on both of them. The two node outputs should thus appear in the viewer panel. In this viewer, click on the slice image to select it and click and drag vertically to control which slice to display in the curve image. This pipeline also contain a set of nodes for visualizing surfaces, as shown in Fig. 11. All these nodes are actually based on variations of the “displaySurface” node, itself based on a “genericGrid” node. The user may modify the surface function by editing the “evalFunc” function inside the vertex tab of the node interface. From left to right, surfaces are visualized with their normals (surface orientation), height and in wireframes. The top row shows a quadratic function that can be manipulated with 2 parameters. The second row is a sinewave surface. The third one used a heightfield as input to generate the 3D surface. The scale parameter of the interface allows the height to be scaled manually. The camera can be controlled in the viewer for each of this node.

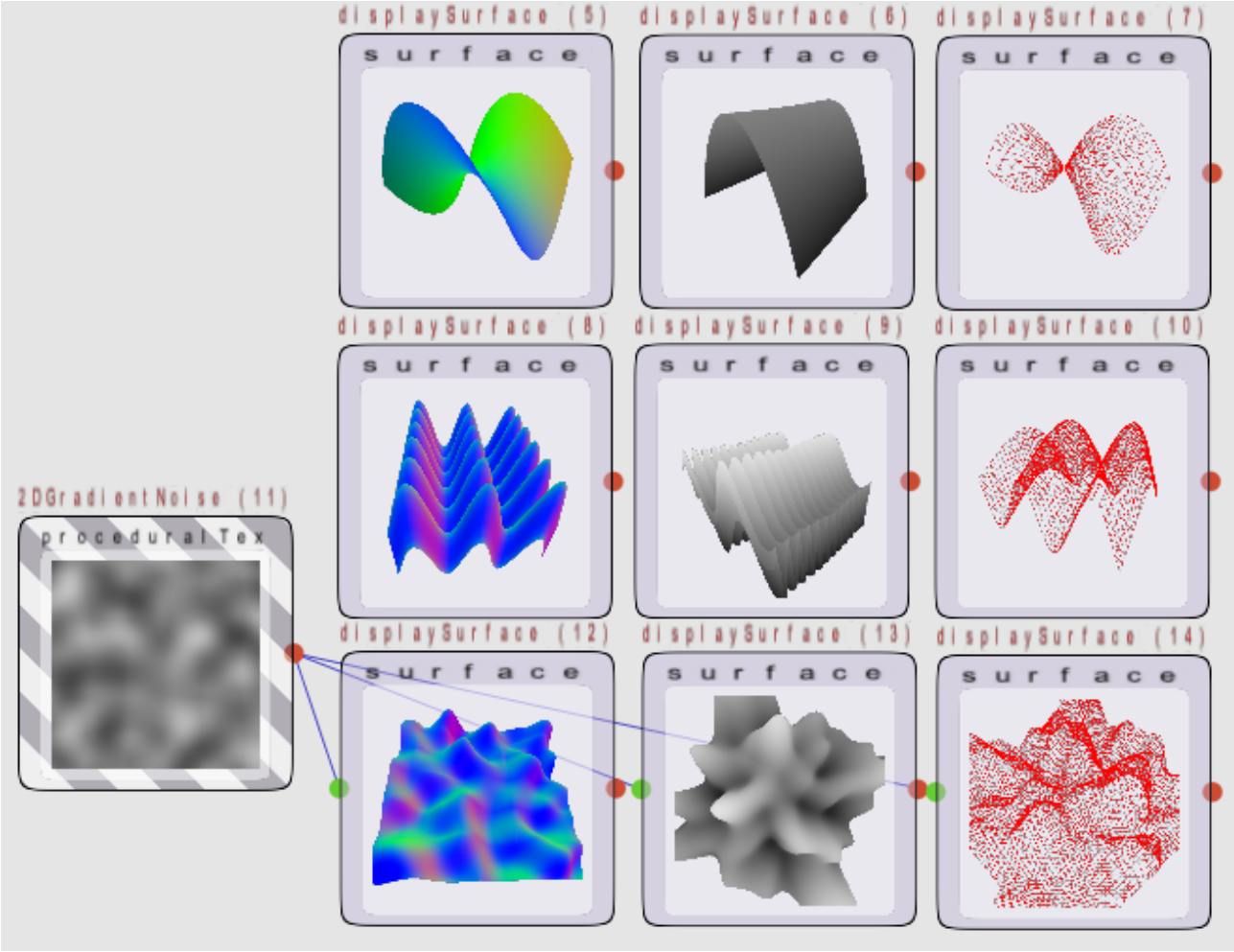

Figure 11: The surfaces examples.

5.5 flow-visualization.gra

This pipeline (Fig 12) illustrates the use of the “lineIntegralConv” and the “HSVColorFlow” nodes (based in generic image nodes) to visualize a flow field. In this example, the flow is obtained by computing the gradient flow of the surface. The line integral convolution (LIC) blurs a texture (the noise in that case) in the direction of the flow. This process is commonly used to visualize flow field. The HSV color node has the same goal, but the input flow direction is visualized with different hues and magnitude is related to the color value. The last node on the right combines the flow to obtain an original visualization. The pipeline contains a second example where an image is abstracted by applying a LIC, based on its gradient flow.

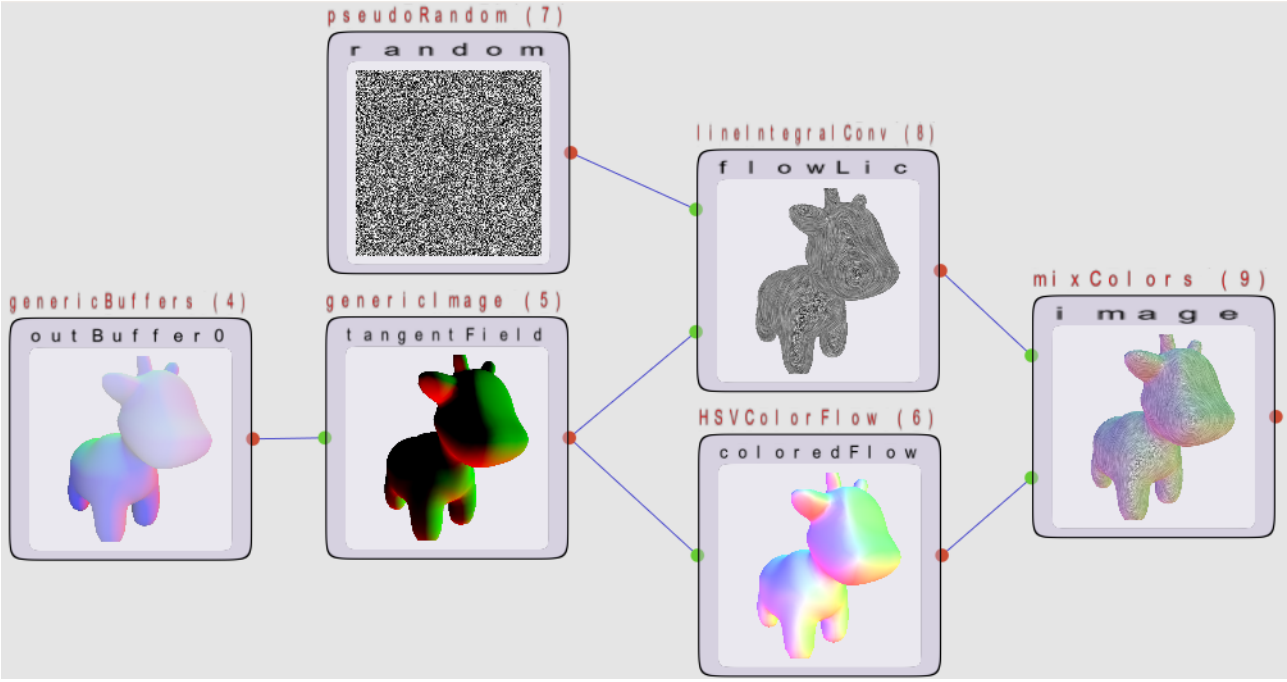

Figure 12: Flow visualization pipeline example.

5.6 laplacian-pyramid.gra

The laplacian example (Fig. 13) shows how to use the gaussian and laplacian pyramid nodes to decompose and modify the frequency components of an image before rebuilding it. These pyramids are all based on the "genericPyramid" node. In this example, small scale details are enhanced before reconstructing the image. The last node shows the resulting enhanced image by selecting the finest scale of the pyramid (in a generic image node).

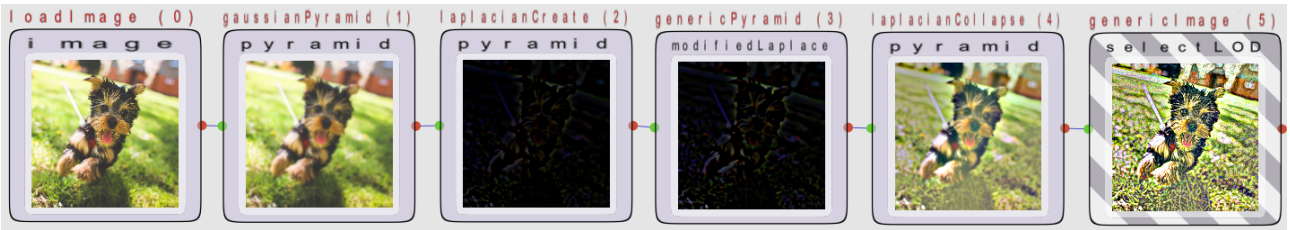

Figure 13: Laplacian pyramid pipeline example.

5.7 global-values.gra

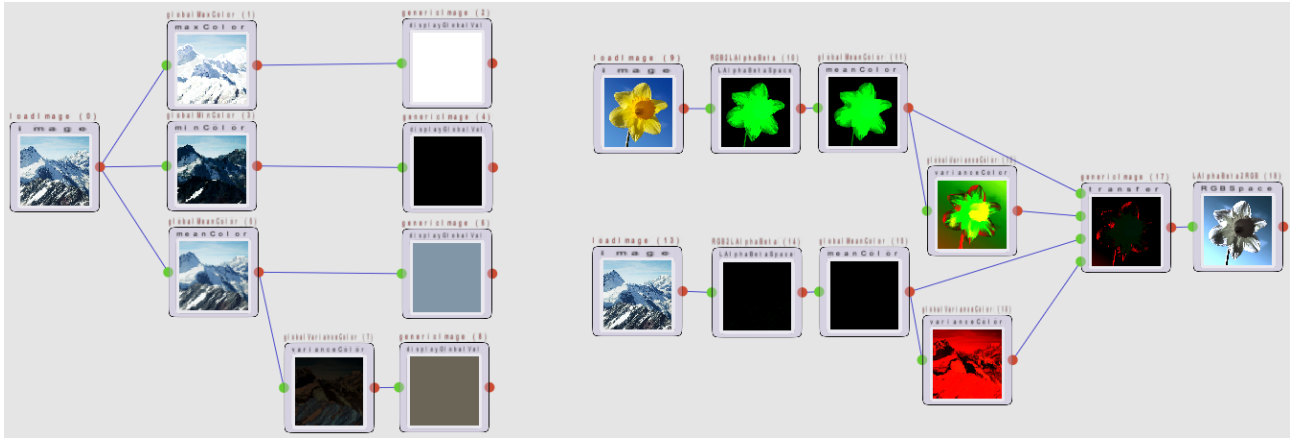

Figure 14: Global values pipeline example.

The pipeline shown in Fig. 14 also makes use of pyramid nodes to compute global values on images such as the minimum/maximum/mean/variance colors. This is shown on the left, where generic images nodes are used to access the last levels of the pyramids and visualize the corresponding colors. On the right side, the mean and standard deviations are used to reshape the histogram of a source image based on a target one (a simple example-based color transfer function).

5.8 histograms.gra

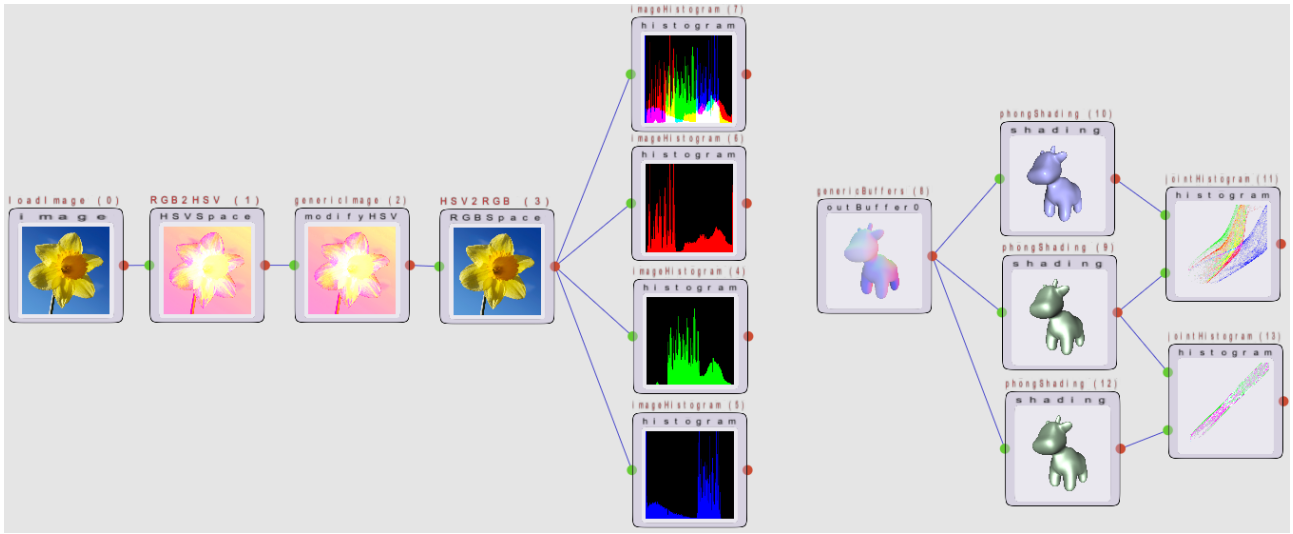

Figure 15: Histograms pipeline example.

Histograms (Fig. 15) are based on generic splat nodes. The left side of the figure shows the original node as well as some variations that select only one particular channel to display. The geometry shader was slightly modified for this purpose. Histograms are obtained in real-time. Try to play with the parameters of the "modifyHSV" node to visualize their effects. On the right side, a joint histogram is used to visualize the correlations between 2 rendered images.

5.9 poisson-diffusion.gra

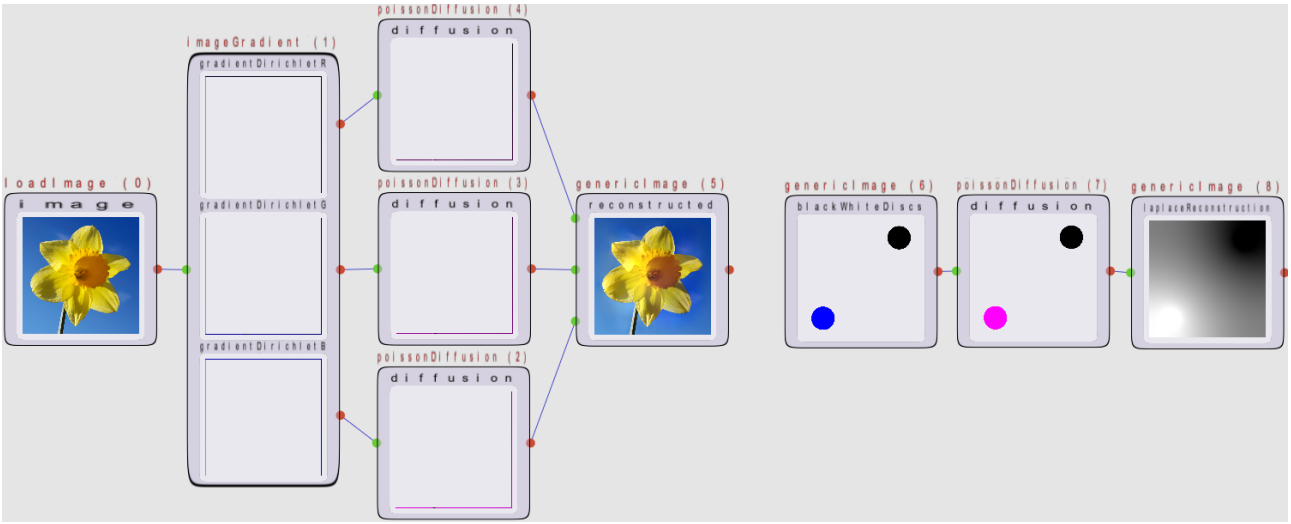

Figure 16: Diffusion pipeline example.

The pipeline shown in Fig. 16 illustrates the use of the poisson diffusion node to reconstruct an image from its gradients. On the left, 3 diffusions are computed (one for each channel) to compute the image. The "gradientDirichlet node" is based on a generic image node and computes the image gradient for each channel, as well as some Dirichlet constraints on the borders (to avoid offset differences in the result). The second example on the right shows how to use the poisson node to compute a laplacian diffusion. In that case, the gradient is set to 0 everywhere. Only Dirichlet constraints are placed inside the 2 black/white discs. The node then diffuses the data by minimizing the gradient. The positions of the discs can be controlled in real-time in the viewer: click on the blackWhiteDiscs image, press "space" and modify the position of the discs with the mouse.

5.10 renderings.gra

The rendering pipeline (Fig. 17) shows the available shading nodes on the left. They can all be controlled via the mouse (for moving the light direction) or via some parameters inside the interface. The right example starts from a generic buffer node for loading the cow model, and uses the texture coordinates as well as the tangents in order to perturb normal at the surface (normal-mapping technique). The model texture is also loaded and combined with a simple shading to obtain the final result on the right. Parameters for the normal perturbation, shading colors and textures might be modified inside the corresponding node interfaces.

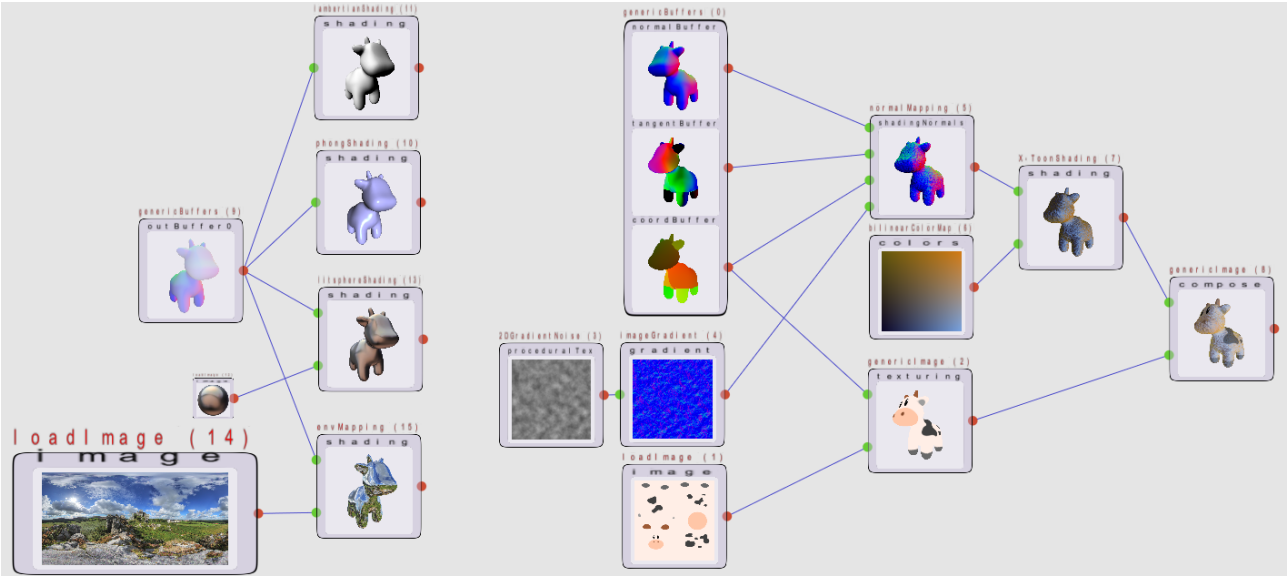

Figure 17: Renderings pipeline example.

5.11 displacement-mapping.gra

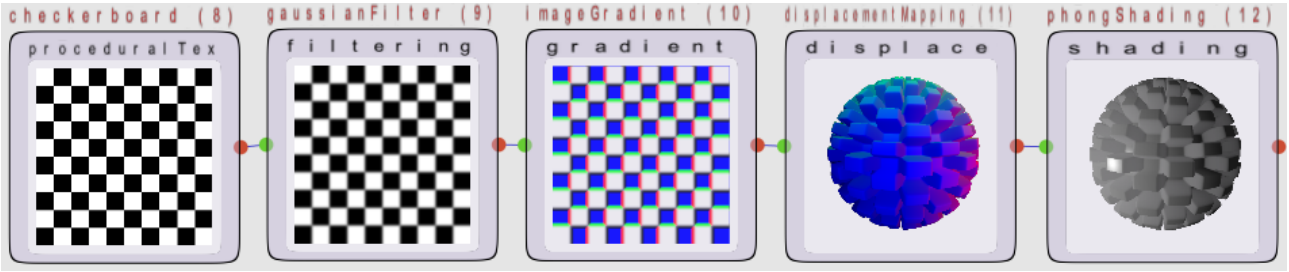

Figure 18: Displacement mapping pipeline example.

The displacement mapping technique (Fig. 18) displaces mesh vertices based on an input height map. Here, the input texture also contains the associated normals, based on the gradient of a blurred checkerboard texture. The displacement is applied on a sphere. The "disp" parameter inside the interface of the displacement node controls how the sphere is deformed. "T" controls how the sphere should be tesselated (how many triangles needed) via the tesselation shader to displace all the details.

5.12 animate.gra

The animation example illustrates basic animation behaviors. You will have to open the pipeline to follow the remainder of this section. The five nodes on the top row show a disc passing through 4 control points, using five different interpolation types: linear, step, shepard, spline and hermite. The second row of nodes shows a rotated sunFlower illustrating the effect of the curve behavior on a small number of control points: no behavior (all linear), constant, repeat, mirrored repeat and offseted repeat. Press the "p" key to start the animation and see the effects (press "shift+p" to stop it). To check and modify the curves, click on the editing button of the parameter inside the interface of the

node.

5.13 fourier-transform.gra

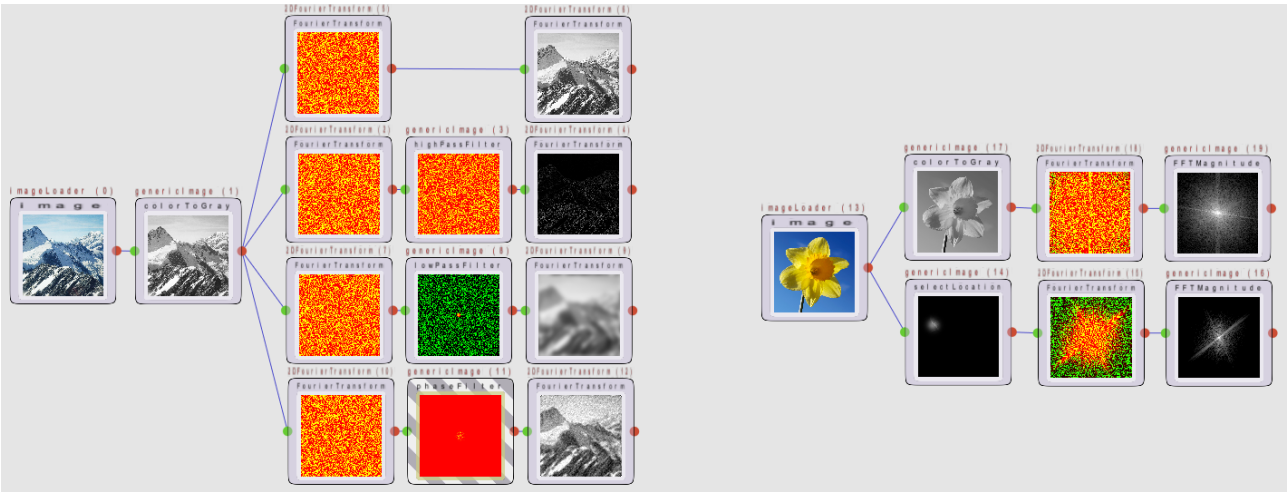

Figure 19: The Fourier pipeline example.

The Fourier transform example (Fig. 19) shows how to use the node to manipulate the frequency content in images. On the left, forward and backward Fourier transforms are successively applied. From top to bottom: the original image is reconstructed; a highpass filter is applied (using a genericImage node that multiplies the frequency magnitude with a user-defined gaussian function); a low-pass filter is applied (on the magnitude - same approach); a low-pass filter is applied (on the phase - same approach). On the right, the magnitude is rescaled using a generic node to visualize the spectrum. (top) the full spectrum is visualized; (bottom) the user can interact with the select node (click + space on the node and interact in the viewer) to visualize the spectrum of a part of the image only.