BUTTLEOFX

OPEN SOURCE COMPOSITING SOFTWARE

Clément CHAMPETIER
Xochitl FLORIMONT
Aurélien GREFFARD
Elisa PRANA
Arthur TOURNERET

Fabien CASTAN

ESIPE
imac
ingénieur

INTRODUCTION

As part of the engineering school IMAC's tutored projects, five of us worked on the ButtleOFX project. The aim was to create an Open Source compositing software based on TuttleOFX, a framework using the standard plugin OpenFX.

At the beginning of the project, our tutor Fabien Castan told us that he had a pretty clear idea of the important features he wanted to include in the software. But he also said that the project was up to us and that we were the ones who would decide on how ButtleOFX would look like. So, after getting to know everything about the project, we had to make decisions on the future functionalities of the application. The three main modules we worked on are the following :

- An image viewer/video player
- A graph editor
- A parameters editor

When the time came to define precisely each of these three modules, we all had plenty of ideas to suggest, such as a fully customizable interface with the possibility of resizing and merging the different panels. Of course, we soon realized we would not be able to set up all these brilliant ideas. This is why we started by defining a scope statement in which we ordered our suggestions by importance, and it proves to have been a really useful idea for the development of the application.

TABLE OF CONTENTS

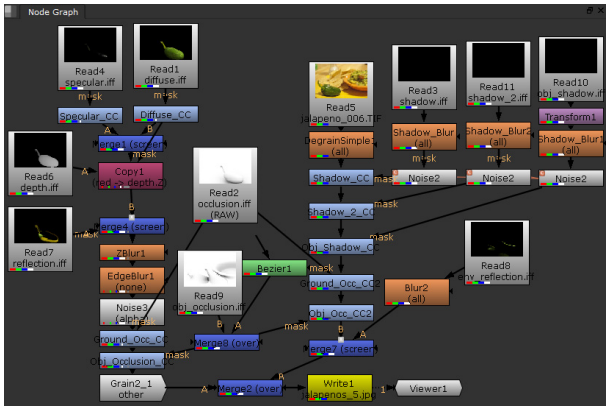
1. THE PROJECT

A COMPOSITING SOFTWARE

Compositing is the combining of visual elements from separate sources into single images, often to create the illusion that all those elements are parts of the same scene. It is mainly used for visual effects or in television studios. A famous example is the use of a green screen to put another scene behind an actor or a speaker.

Compositing with green screen

In order to compute new images, we use plugins to apply effects on different types of images and videos. The effects are managed in compositing softwares with what we call a graph. It contains nodes that represent effects and can be connected to create a composition.



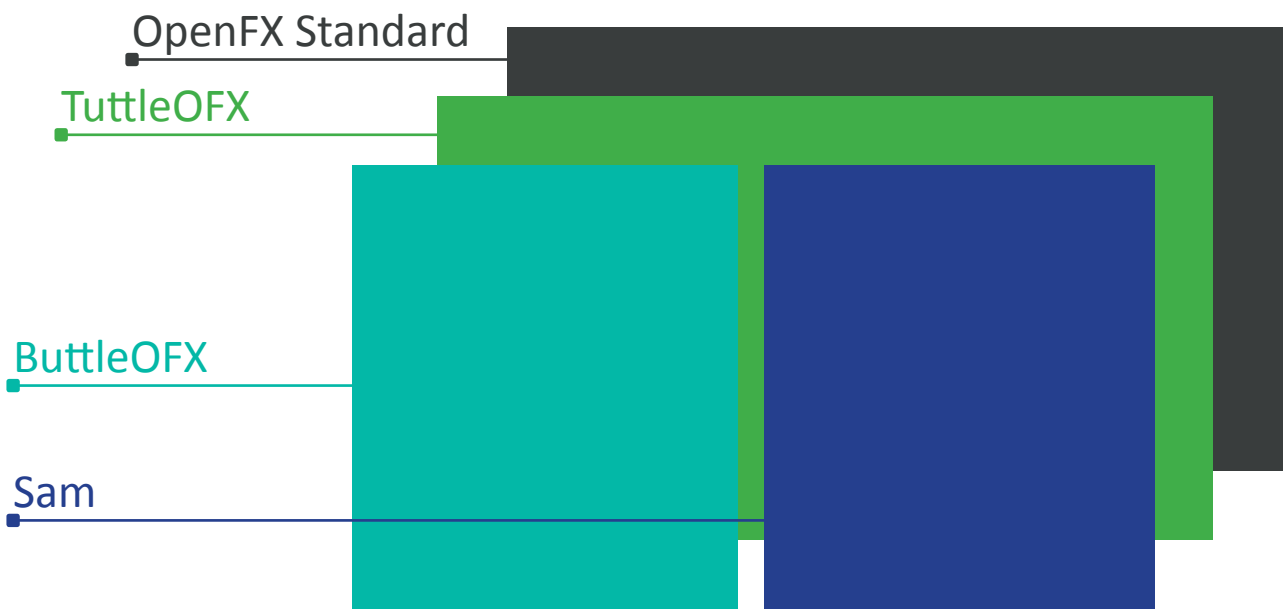
An example of graph, from the compositing software Nuke

FROM TUTTLEOFX TO BUTTLEOFX

ButtleOFX is based on TuttleOFX, an open-source image processing framework based on the OpenFX standard. It contains a set of basic plugins, the Tuttle plugins, that can be used in softwares such as Nuke and Fusion.

But the main strength of TuttleOFX is Sam : it enables to use the Tuttle plugins on images with command line tools. And better still, it is compatible with all OpenFX plugins, even commercial ones like Sapphire, Furnace or Keylight.

Thanks to Sam, image conversion and batch processing operations is very easy. The single disadvantage of Sam is that there is no interface to use it, but only command lines : it is not very convenient and it is quite tedious to manage a long chain of effects. And this is the reason why our tutor decided to create the ButtleOFX project.



From TuttleOFX to ButtleOFX

THE TARGET

TuttleOFX was initiated by HD3D-iio R&D project. It has been developed and maintained by Mikros Image and other famous companies in the multimedia industry, so ButtleOFX should be used by them. At present, they exploit TuttleOFX especially for batching processes. We already talked to them about our project and since it is an open-source project, they can keep on the development.

ButtleOFX should also be used by those who practice compositing. As our application is open-source, they do not have to pay for using it. And it is the only open-source application that enables the use of all the plugins based on the OpenFX standard. That should be a sufficient argument for them to try ButtleOFX !

RULES OF THE GAME

Some features are essential to a compositing software. As we were not completely familiar with the concept of compositing, our tutor presented us some important features.

- **A viewer**
  This module enables to display computed images or to play videos. We have the possibility to zoom in and out, and move around in the viewer with the mouse middle button. It is useful to see the result of an effect on an image.

- **A graph editor**
  This module is the center of the interface : by using several nodes, new images and videos can be obtained and displayed in the viewer. The nodes correspond to OpenFX plugins and can be connected together in order to create a succession of effects.

- **A parameters editor**
  This is the third important module of the application. It is generated according to the selected node and displays its parameters. We can play with the value of each node's parameter in order to obtain customized images.

- **The undo/redo management**
  Before starting the development, it was important to make a choice about implementing undo/redo or not. And, as it is an essential option even in basic softwares, we obviously decided to do it !

- **OpenGL interaction with plugins**
  We had two possibilities about how to display computed images : either just display a static image (with no possibilities of interactions), or set up an openGL context (and with it, it is easy to get a lot of information about the image, about all pixels, etc...). That is why an openGL context was more interesting for us.

Once this basics features presented, our tutor explained to us that since there was no client, the project was ours. It means that there was no definitive bill of specifications and we were up to chose the features that we wanted to implement.

2. THE PROJECT MANAGEMENT

Such an important project could not have been achieved without some management methods. At the beginning, we were not used to project management nor working as a team, so we had to get organized and learn how to use some new tools.

ORGANIZATION

At the beginning of the project, we had to think about our organization. Indeed, we were five persons to collaborate and being so many on a single project was new for everyone.

- **Election of a chief**
  First, we decided to choose a chief for our team. We elected Xochitl Florimont at unanimity. Her role was to ensure the smooth running of events and keep an eye on the team.

- **Regular meetings with our tutor**
  In order to have a regular feedback on our work, we decided to meet twice a month with our tutor. During these meetings we used to show him the advancement of the project but also and above all ask him questions from a list we would specially draw up in order to discuss the most important items. He would then answer our questions, help us with the project when we were stuck on something and tell us if we were going the right way.

- **Weekly common working sessions**
  In this same context, we decided to have a weekly common working session. This was really useful to review the situation : each of us could explain the work he had done, or ask for help when he was stuck on a problem. In this way, the working sessions let us help each other easily and make progress on the project. This was the time to communicate on the parts of the project that would not work and to agree on the decisions we had to make.

RISK MANAGEMENT

During one of our first working sessions, we decided to set up the risk management. The aim was to find a solution to every issue that could occur.

Here is the list of risks and solutions we envisaged :

- **Material risks**

  A computer breaks down
  The owner can work on the school's computers or can find another computer waiting for repairs or repurchase.

Something is not working on Mac
If the tool is not developed by everybody, Clément (who is the only one to have a Mac) does not develop it. If he has to work on it, he develops with his binome or install Linux.

- **Human risks**

Someone fall sick or is in the inability to work
The others divide up the work together and make report in order that he keeps being informed. We look after/comfort him with candies or chocolates.

Disagreement in the team
We first talk to the project manager. If the issue is not solve, we talk to Fabien, and then to Vincent Nozick.

Someone wants to give up the project
We try to motivate him, but we must not be aggressive : the key is to never get angry ! The person has to talk before being in that state !

Someone gives up the project
We warn all the team, plus Fabien and Vincent Nozick. We divide up the task in consequences.

Someone wants to change his role
He tells the other during a meeting, where Fabien is attending if needed. We discuss to see if someone is up to change.

Someone is lacking time or important sleep
He has to talk to the team and ask for help. The features will be reviewed in order to keep the essential.

Someone is not working or very few
The project manager has to make him understand he has to work more. If necessary, we talk about it during a meeting.

Someone has big difficulties to achieve a task
He has to ask help to the other or to Fabien if no-one has a solution. If necessary, talk to the project manager or during a meeting in order to divide the tasks better.

Someone doesn’t have Internet anymore
If he is at Champs-sur-Marne, he can go to Copernic or at someone’s place.
If it is during a week-end, he searches for a solution. If he cannot, the team can work without him for two days.

Someone is not available for a meeting
We maintain the meeting but we have to write a report (at least verbal) to the him. Everybody has to be aware all the time : nobody can be lag behind. Try to avoid multiples absences.

Fabien is unreachable
Ask for Farchad Bidgolirad or contact Adrien Herubel or Serge Taillandier. In last resort talk to Vincent Nozick.

Fortunately, we did not encountered many issues during the project. So we were able to deal with them.

DISCOVERING AGILE METHODS

No one in the team was aware of the agile methods. Slow but steady, we began to use some tools of the Scrum method, sometimes without knowing that is was part of it.

- **Dividing up tasks**
  In order to divide the work, we drew up a list of tasks for each of the three modules : viewer, graph editor and parameters editor. At the beginning, we used to work in little groups (two or three people) since we had to implement modules from scratch and we did not know very well the languages. Our tutor suggested that we create many groups at a time so none of us would be restricted to a specific part of the application, and try instead to have the most general view of the application development.

  When the project was more advanced and we were more comfortable with the technologies, we divided the tasks in order to be able to work alone. During our weekly working sessions, we used to pick up the five most important tasks and each one of us would chose a task among the five to work on it. This is the way we divided up task.

- **Agile method : sprints of two weeks**
  Our tutor made us aware of project management strategies such as Scrum but we did not want to be too constrained and just kept some ideas of it. One of them was the “sprint” : it consists of scheduling the implement of tasks for a milestone. Every milestone had the same duration in order to learn to evaluate difficulty and time spent on tasks. Thus, every two weeks we reviewed what we had done since the last sprint and could establish the aims of the next two weeks.

- **Improving step by step**
  At the beginning of the project, the modules (viewer, graph editor and parameters editor) were not linked. We worked on them in order to have separated functional tools. When they were sufficiently developed, we decided to link them. When the time came to add TuttleOFX to our application, we worked in the same way. We waited for a long time because we wanted the application to be efficient enough before adding another layer.

- **Mutual help**
  One of the key of agile methods is to help one another. So when somebody had some difficulties in developing a feature, he could ask help to the others. We did not hesitate to swap tasks if someone was more comfortable with a task than another. Conversely, if someone had difficulties with a specific subject, he could work with someone else in order to understand it. When a task was more important or thought than others, we often worked in pairs in order to help each other and progress faster. This way of working is inspired by the XP (eXtrem Programming), where programmers work in pair, and switch regularly.

THE TOOLS WE USED

As our project was more important than those on which we were used to work, we had to find specific tools to work, for sharing files and publishing reports from our meetings.

- **From Dropbox to Github**

  Working on ButtleOFX has been the opportunity for us to discover Github, a web-based hosting service using Git. This allowed us to share code easily, review the code and post comments. Later, we learned how to work with true openSource projects. We discovered how to create issues and assign them in order to plan the future of the project. We also used the milestone tool in order to schedule the issues. It turned out to be a little complicated at the beginning, but now that we have better command of it, it has become an indispensable tool. And we have to admit that using Dropbox for sharing code (yes, this is how we did at first, until our tutor pointed out it was not the correct way) is not appropriate.

- Our github repository :
  https://github.com/buttleofx/ButtleOFX

News for all : a blog

The aim of the blog was to post news about the progress of the project.
The students and people interested in ButtleOFX could read the news and be informed about the new developed functionalities. We could, above all, sum up the main achievements which is important to have an overview of how the project evolved, especially concerning the architecture of the application which changed a lot since the first stage.

Our blog :
http://buttleofx.wordpress.com/

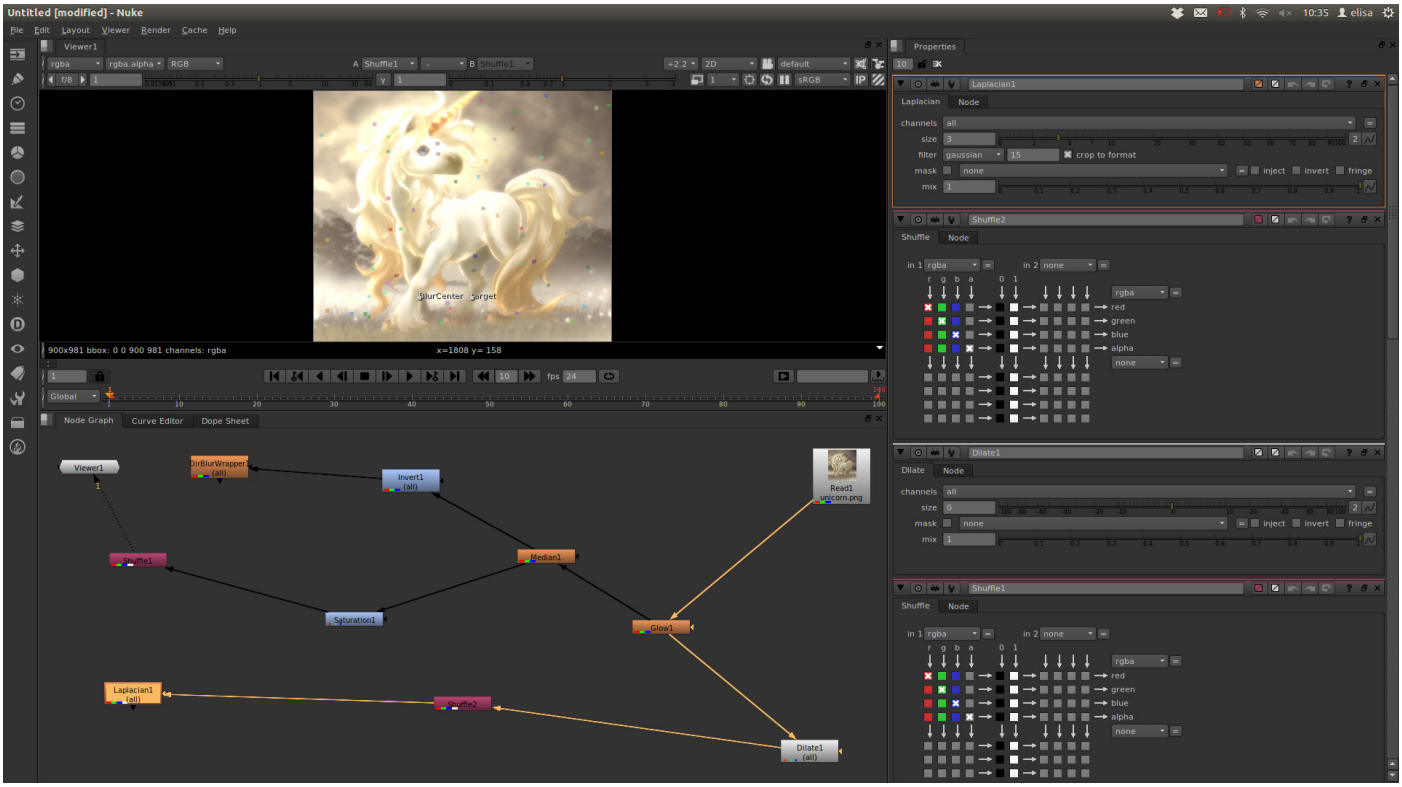
3. BUTTLEOFX

CHOICES AND GUIDELINE

At the beginning of the project, none of us really knew what a compositing software was and how it looked like. So we studied the most used compositing softwares to understand how this type of software was organized and how it worked. An important part of our project was to develop the graphic interface of the application. That is why we spent some time to think about the appearance of ButtleOFX.

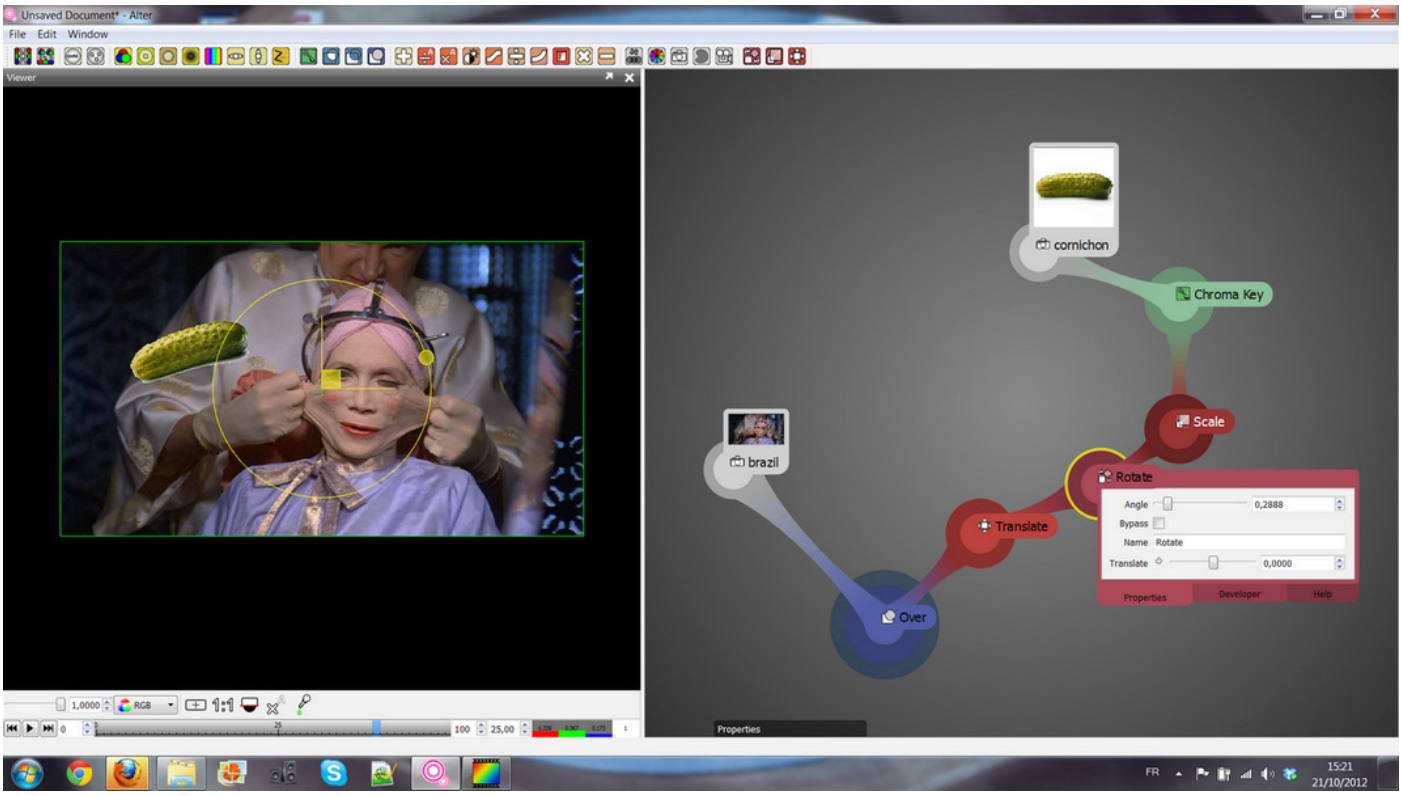
- **State of the art**

  In order to create our interface, we first checked the state of the art. We analysed the functioning and the interface of different compositing softwares, both open-source and commercial ones : **Nuke**, **Synapse**, **Blender**, **Toxik** and **Scratch**. It appeared quickly that the three modules that our tutor told us about (viewer, graph editor and parameters editor) were essential. In all the concurrent softwares, the interface is totally resizable. It means you can hide or show every panel of the application and also pop them out of the main window. We also noticed that the viewer was often at the top of the application while the graph editor was at the bottom.



Default layout in Nuke

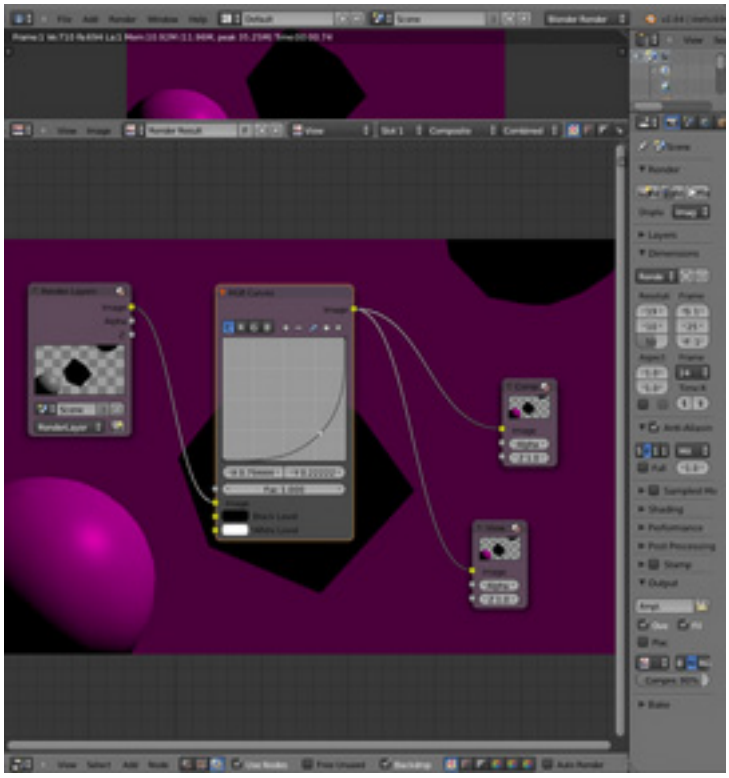
Nuke is the most famous compositing software based on a graph system. It is the most complete in term of functionalities and user choices. But it is not open source and you need a license to use it.



Default layout in Synapse

Synapse is an open source software, it is less famous than **Nuke** and there are less possibilities but we have appreciated the original design of the application.

Some of these softwares, such as **Blender**, have a feature that we really liked : the merge between the graph editor and the viewer. The nodes are then displayed over the image :



Graph editor and viewer merged in Blender

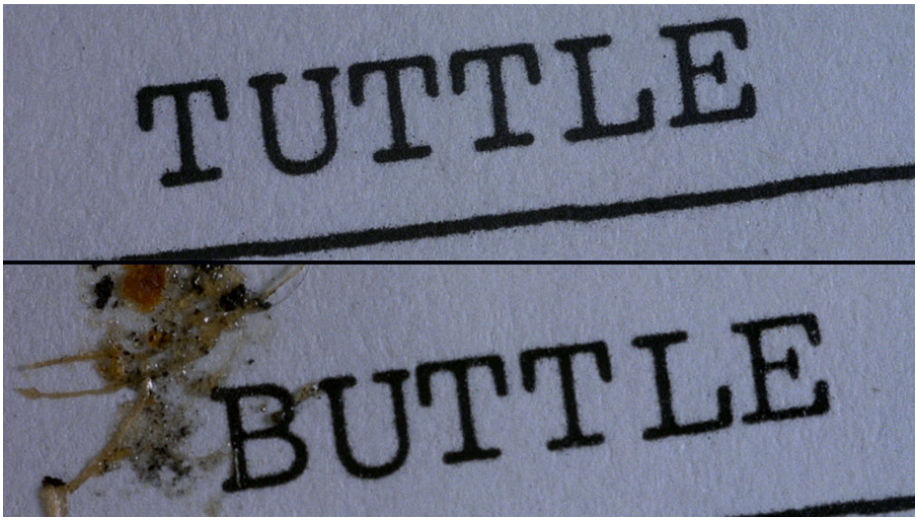
We also payed attention to the colors of the application, which are very often grey, as we can see in the different images above. This choice is made in order to have a better view of the images on which the user works.

Concerning the nodes and the connections between them, the shapes are very different in each software. We chose round rectangles for the nodes and curved connections because we found it nicer.

- **The universe of Brazil**
  The name of Buttle and Tuttle are references from the movie of Terry Gilliam **Brazil**.

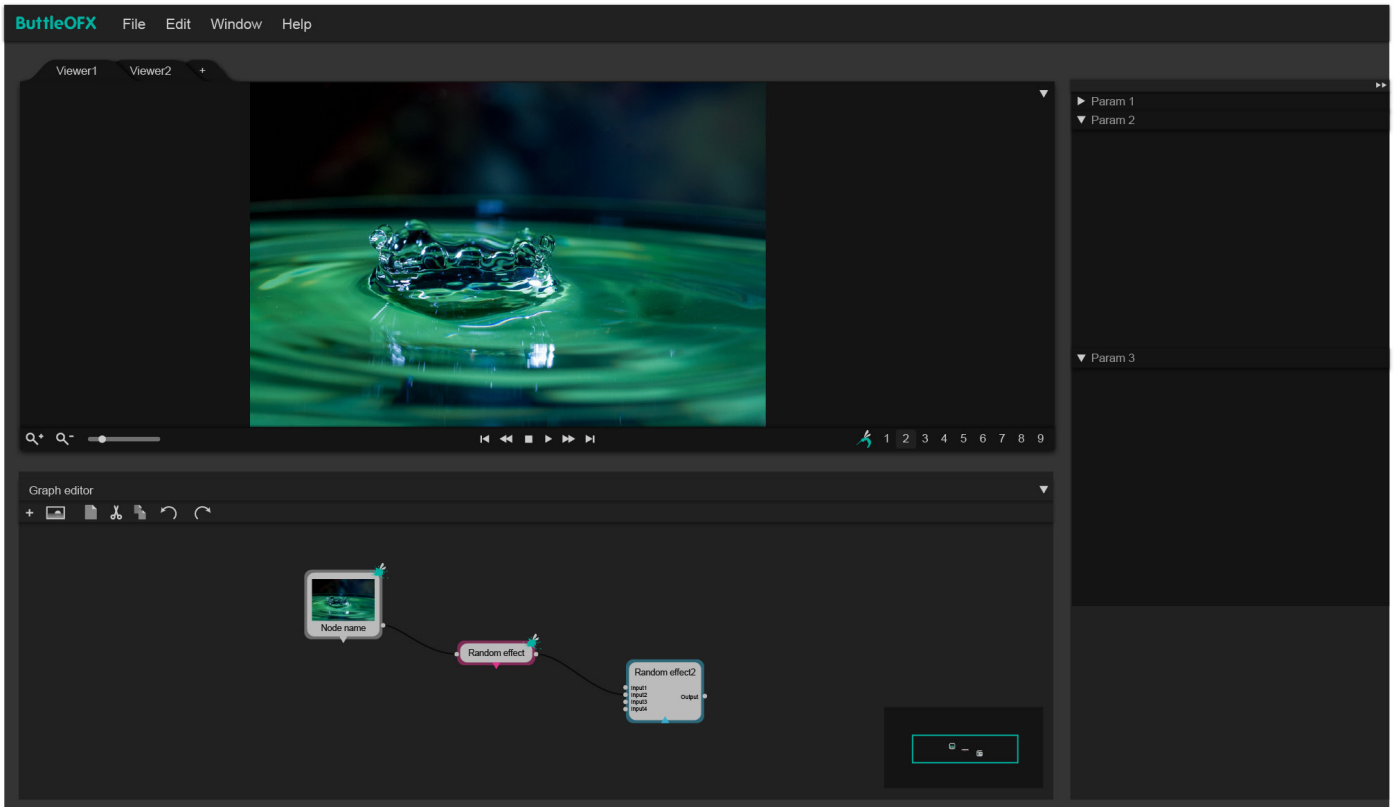
Brazil DVD cover

We wanted to keep this connection with the film in our interface. That is why we decided to use a mosquito. It is the key of the confusion between the two characters in the movie (we realized later that it is actually not a mosquito, but a sort of beetle. Anyway, we chose to keep the mosquito for the application). We decided to use it to display the image of a node in the viewer and to put it in our logo.

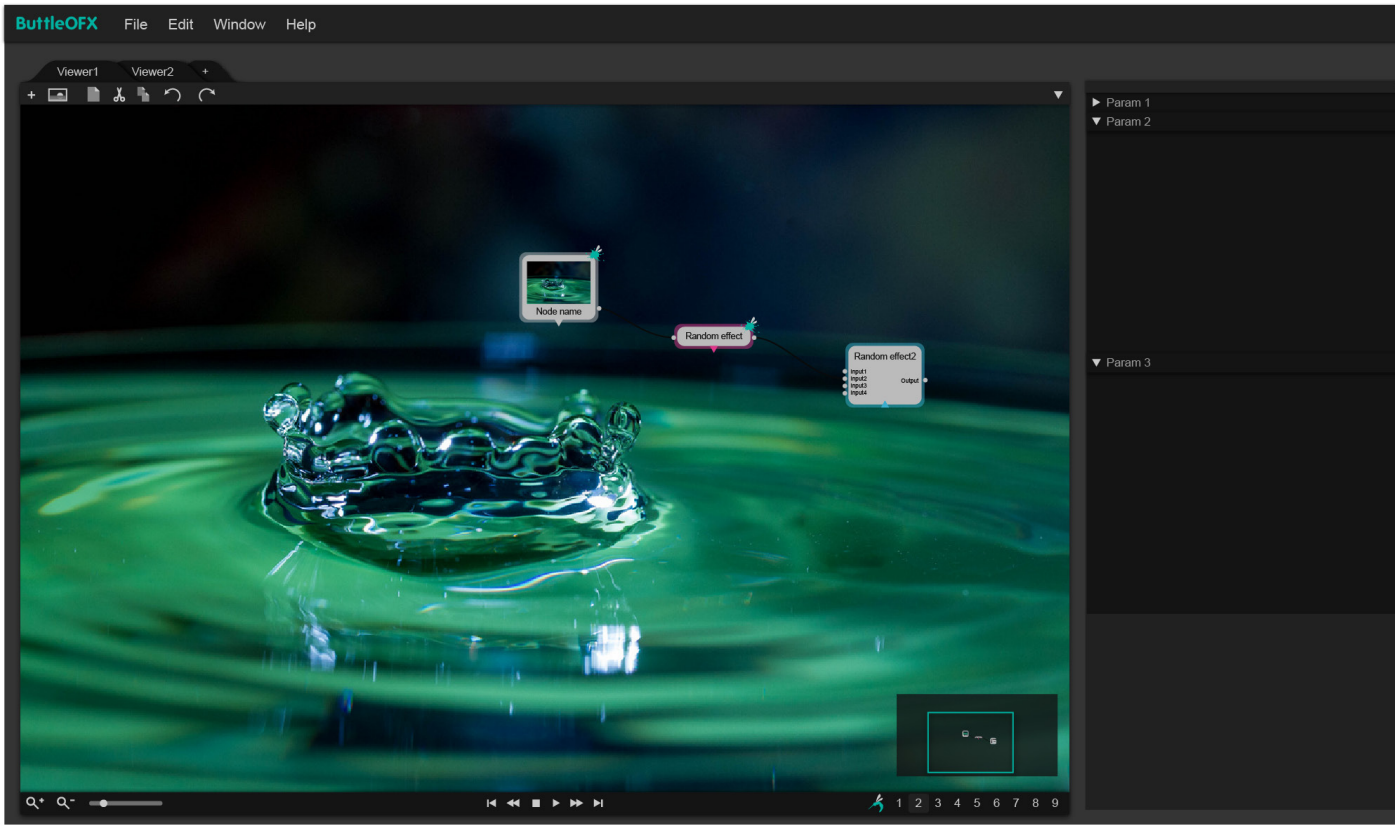


The dead insect in the movie

Finally, according to our study of the state of the art, we created a template of our application. We decided to use dark colors, and to arrange the modules in the classical way : the viewer on the top, the graph editor on the bottom and the parameters editor on the right.



How we wanted the application to look like

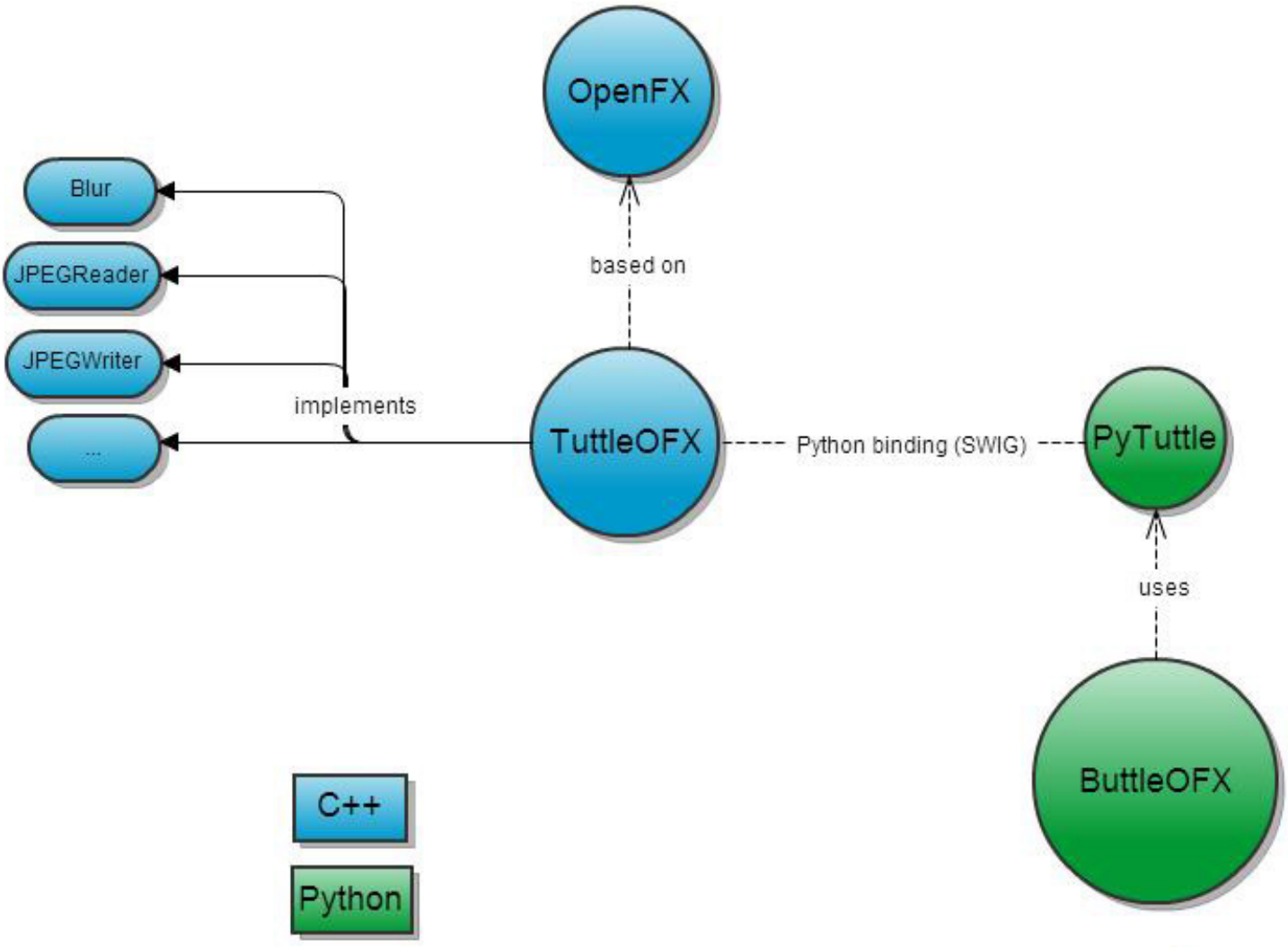


With the possibility of merging the viewer and the graph

DEVELOPMENT

Where is Buttle ? Where is Tuttle ?

As we already said before, ButtleOFX is based on the framework TuttleOFX. It contains several libraries, but the one we used for our project is the Tuttle Host Library. It enables to load OpenFX plugins and manipulate a graph of OpenFX nodes.
In order to use the functions contained in this library, we used PyTuttle, which is another library that implements the same functions as Tuttle Host Library, but in Python. It communicates with the Tuttle Host Library thanks to a binding Python : SWIG. Here is a schema illustrating the link between the different parts.



ButtleOFX's structure

3 layers : Buttle / ButtleWrappers / QML

The hardest part of the project was to set up the architecture of the application. Indeed, ButtleOFX hides many layers and it was not an easy task to link them in a proper way. The three layers are the following :

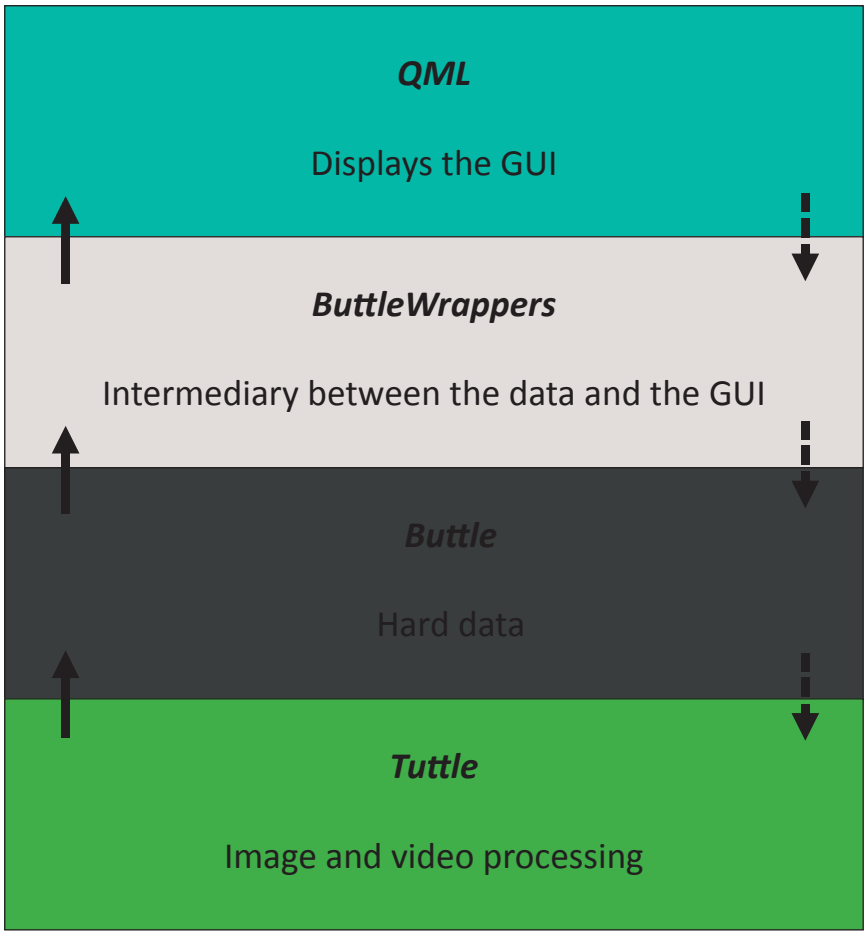
Buttle
This layer stores the hard data, it contains python objects.

QML
This layer is used to display the user interface, it contains qml objects.

ButtleWrappers
This layer is used to link the core layer and the qml layer. It creates qml objects from the python objects. This way, we can use the data of the core layer to display objects in the qml layer.

Beneath these layers, we can find the Tuttle layer which is used for image and video processing. It is linked to the Buttle Layer.

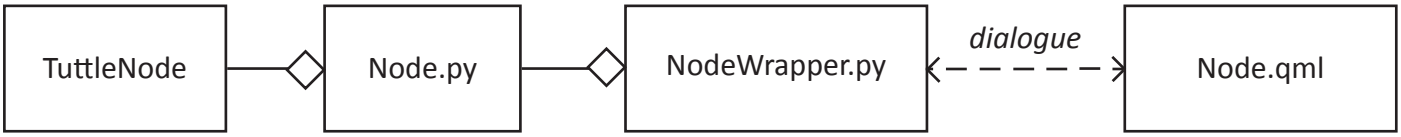
And this is how it looks like :



ButtleOFX's hidden layers

So, each object visible in the application is managed in reality with four different objects, three of them being written by us and the last one being the Tuttle object.

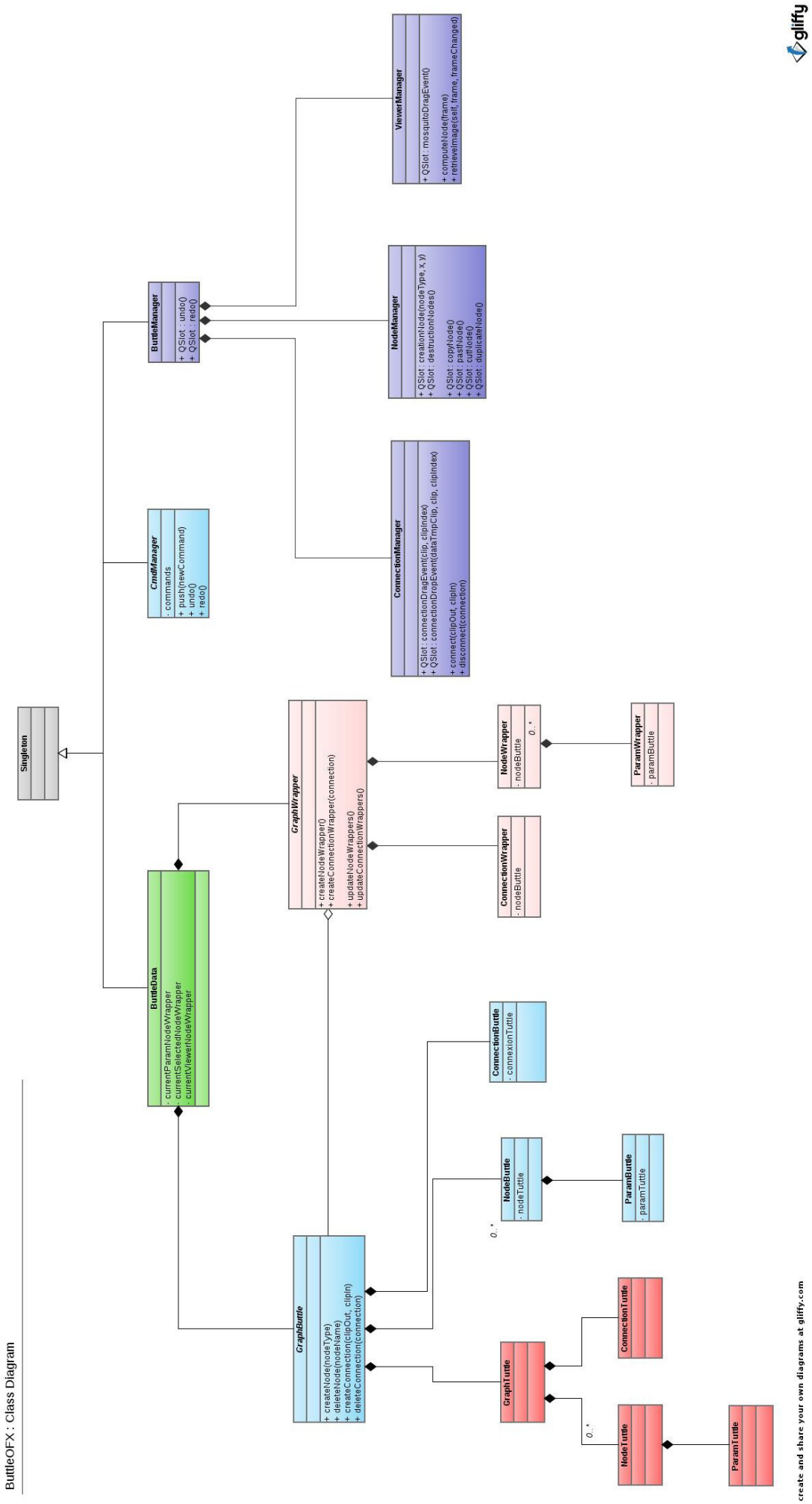
For example, a node in ButtleOFX requires the construction of all these objects, interacting mutually :



The different layers for a node

- **UML class diagram**

The architecture of our application is quite complex. That is why we needed to make a class diagram to illustrate it. Here is a simplified UML diagram.


ButtleOFX : Class Diagram
ViewerManager
+ QSlot : mosquitoDragEvent()
+ computeNode(frame)
+ retrieveImage(self, frame, frameChanged)
ButtleManager
+ QSlot : undo()
+ QSlot : redo()
NodeManager
+ QSlot : creationNode(nodeType, x, y)
+ QSlot : destructionNodes()
+ QSlot : copyNode()
+ QSlot : pastNode()
+ QSlot : cutNode()
+ QSlot : duplicateNode()
CmdManager
- commands
+ push(newCommand)
+ undo()
+ redo()
ConnectionManager
+ QSlot : connectionDragEvent(clip, clipIndex)
+ QSlot : connectionDropEvent(dataTmpClip, clip, clipIndex)
+ connect(clipOut, clipIn)
+ disconnect(connection)
Singleton
GraphWrapper
+ createNodeWrapper()
+ createConnectionWrapper(connection)
+ updateNodeWrappers()
+ updateConnectionWrappers()
NodeWrapper
- nodeButtle
0..*
ParamWrapper
- paramButtle
ConnectionWrapper
- nodeButtle
ButtleData
- currentParamNodeWrapper
- currentSelectedNodeWrapper
- currentViewerNodeWrapper
ConnectionButtle
- connexionTuttle
NodeButtle
- nodeTuttle
ParamButtle
- paramTuttle
0..*
GraphButtle
+ createNode(nodeType)
+ deleteNode(nodeName)
+ createConnection(clipOut, clipIn)
+ deleteConnection(connection)
GraphTuttle
ConnectionTuttle
NodeTuttle
ParamTuttle
0..*
gliffy
create and share your own diagrams at gliffy.com

IMPLEMENTED FEATURES

- **Viewer and time management**

  The viewer is the part of the application where the images are displayed. The display is managed with OpenGL thanks to our tutor.
  Some useful tools are present as the zoom in the image with the mouse wheel and the displacement by pressing the mouse middle button.
  We also implemented video playback. We have succeeded in obtaining a smooth video playback and a good control on it. Indeed, the video can be read without framerate drop and the user can play, pause, stop the video and also joggle in the timeline to choose a particular frame.
  It was also important to provide the possibility to compare images to see the difference between the steps of our work. So it is possible to store up to nine images in the viewer. You have to drag and drop the mosquito on the node to store its corresponding image. You can easily switch between the nine images with the keyboard if you want to compare them.

ButtleOFX's viewer

The way of functioning is that we display the image of the current node associated to the selected instance of the viewer. For that we associate in QML the node we want to display in the viewer by moving the mosquito on it. At this moment, we send the instruction to TuttleOFX to compute the effect of the node and we update the viewer.

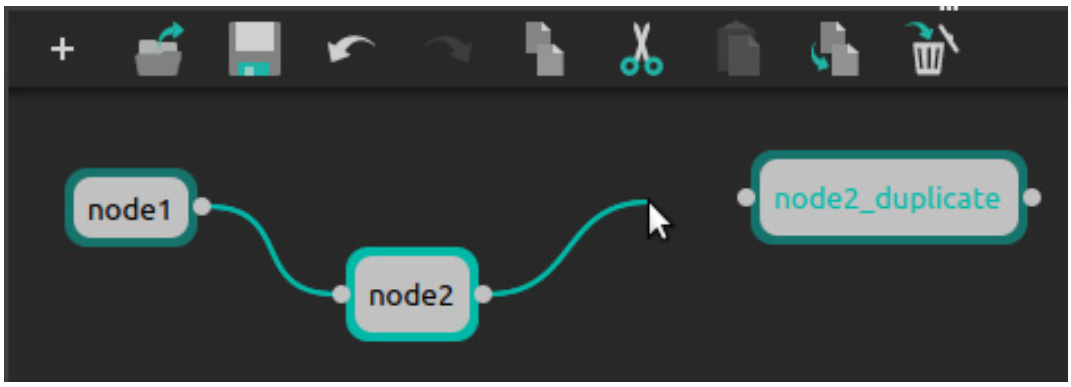
For video management, it was a little more difficult because at the beginning we were unable to read a video without frame rate problems. This drop was due to the fact that every time an image was computed, we opened and closed the file. But for a video it was a waste of ressources because we read frames one after the other, and we did not need to close the file every time the frame changed. So we used an object of TuttleOFX named a ProcessGraph which has allowed us to keep the file playing open and to acces every frame faster than before until we stop playing the video and so the file closes.

- **Graph editor**

The graph editor is the place allowing to manage the various effects applied on images. All images and effects are represented by a node in the graph editor. Every type of node can be created by clicking on the + icon of the toolbar or simply with a right click. In fact, the menu which appears is the list of plugins that are available and often correspond to an effect applicable to the image. By default, all TuttleOFX plugins are accessible, but it is possible to install any other OpenFX plugins and use them with ButtleOFX.

To apply an effect on an image or a video, it is necessary to link the corresponding nodes. A connection appears when you click on a node's output. If you drag and drop it to another node's input, the connection between the two nodes is completed. And if you display the last node of the chain in the viewer, you will see the source image with the effects linked to it applied.

For a software, it is also important to give appropriate tools to work efficiently. So for a ease of use, we implemented some controls we thought were necessary for a compositing software. A node that has been created can be moved in the graph, it is also possible to copy and paste it. More impressive, it is possible to select several nodes at once, move them, copy them, duplicate them and delete them.

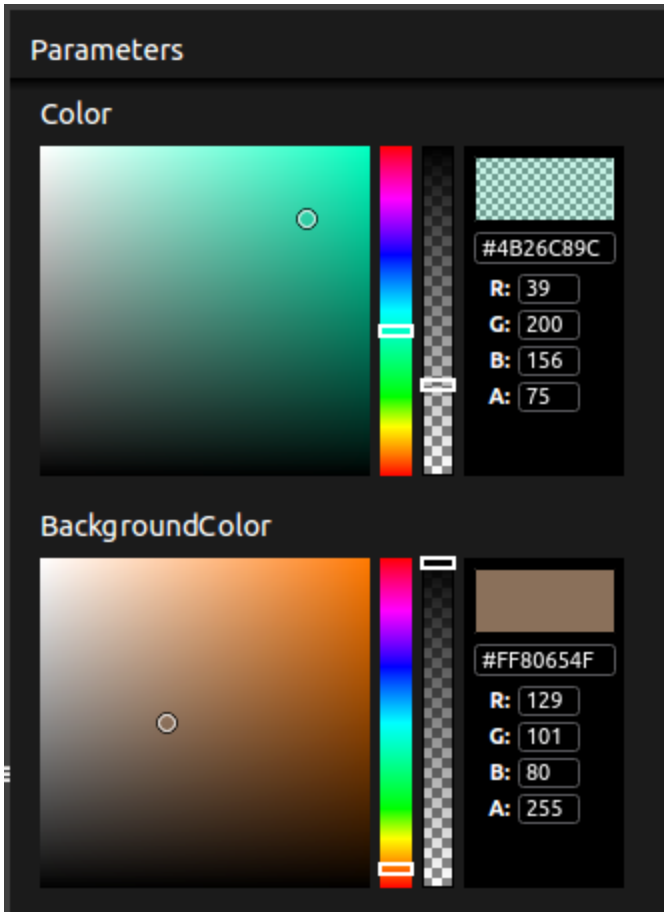


ButtleOFX's graph editor

- **Parameters editor**

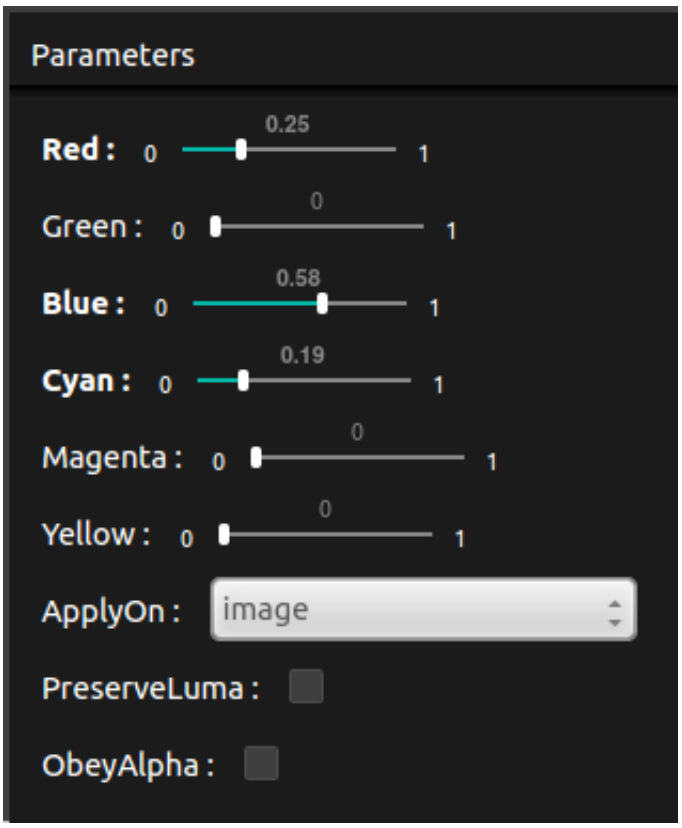
For every node present in the graph, there are adjustable parameters. These parameters are displayed in what we name the parameters editor. A double click on a node displays the corresponding parameters in the parameter editor.

There are many types of parameters : int, double, boolean, color, list of choices…
For each of them we created QML objects because every one is not displayed in the same way. We wanted to have a display corresponding to the easiest way to set a parameter's value.
For example, to display a color, we use a color picker, and for a file path, we use a textbox.



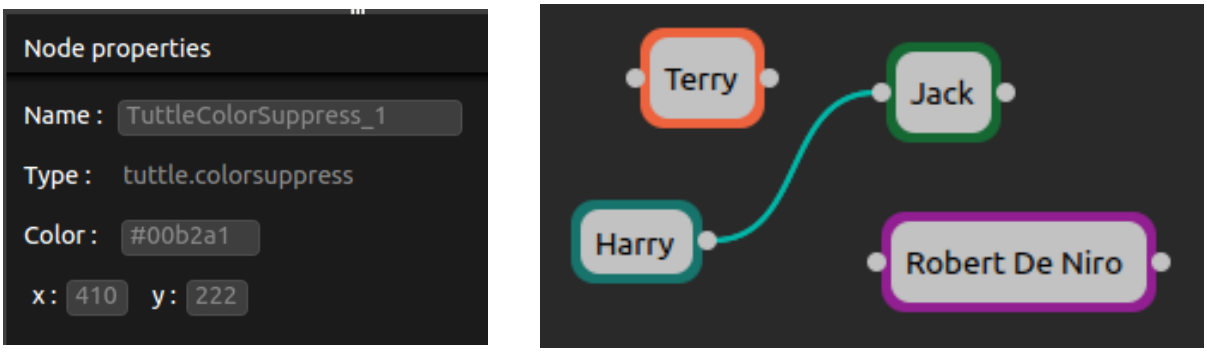
ButtleOFX's color picker

As soon as the value of a parameter affecting the image is changed, the modification can be observed in the viewer (if the image displayed depends on the node in the parameters editor).
When the value of a parameter is changed, the title of the parameter is written in bold. It is a way to easily know which one has been modified by the user. And pressing the left click on the title of a parameter resets his value to the default one.



Example of a node's parameters

The parameters editor allows also to change the visual of the node in the graph : its name, its color and its position.



Example of a customized graph

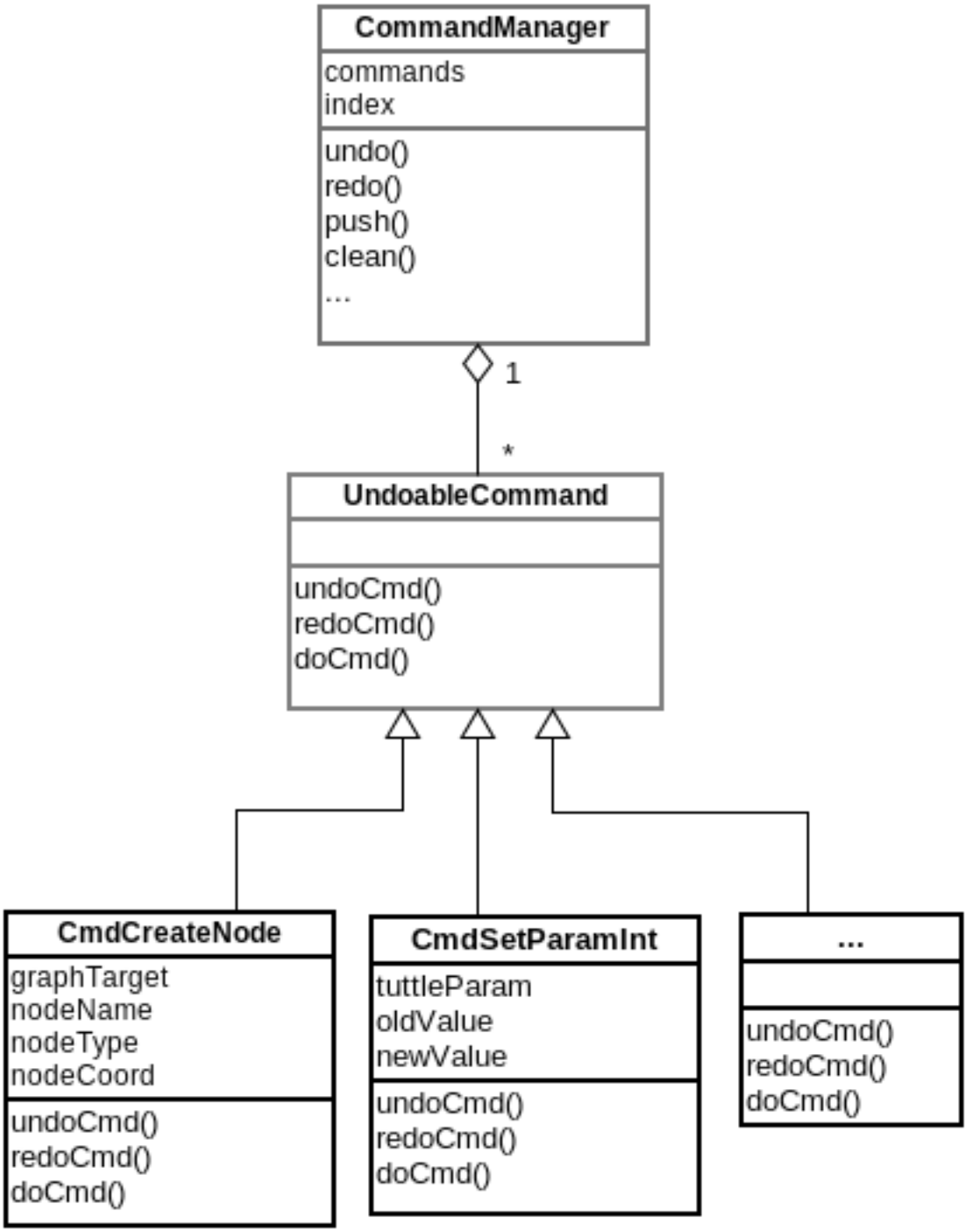
- **Undo/redo management**

  It was very important to us to provide all the necessary tools to allow the user to work in the best conditions. Therefore, we set up edition tools allowing to undo or redo the user actions. All the manipulations on the graph and on the parameters are reversible and can be undone.

  We created a CommandManager class with the Singleton pattern, which contains all the commands of the user and all the tools needed to manage them.
  All reversible commands must be represented by an object inheriting of the abstract class UndoableCommand, and thus overloading three methods : undoCmd(), redoCmd() and doCmd().

```
class UndoableCommand:
        def undoCmd(self):
        def redoCmd(self):
        def doCmd(self):
```

  When the user does an action, then an object UndoableCommand is instancied, containing all the informations needed to be able to undo the command and redo it later : the object on which the command is called, and all the parameters. This object is added in the stack of commands of the CommandManager, and the doCmd() is called.



Undo/redo management diagram

Here is an elementary example with a simple addition command :

```
class CmdAddition:
        def __init__(self, targetVariable, amount):
                self.target = targetVariable
                self.amount = amount

        def undoCmd(self):
                self.target -= self.amount
                return self.target

        def redoCmd(self):
                return self.doCmd()

        def doCmd(self):
                self.target += self.amount
                return self.target
```

Then we can use this command like this :

```
n = 3
print("n = " + n)  # n = 3
cmdAdd5 = CmdAddition(n, 7)
CommandManager.CommandManager().push(cmdAddition)
print("n = " + n)  # n = 10
CommandManager.CommandManager().undo()
print("n = " + n)  # n = 3
CommandManager.CommandManager().redo()
print("n = " + n)  # n = 10
```

We can easily undo and redo the operation ! The CommandManager also provides some methods useful to verify the validity of the requested commands, and is thus secure. If we want, we can limit the number of commands in the stack.

In order to have the most robust mechanism possible and to avoid the bugs, we tested the CommandManager with some commands on Vectors, and a lot of unit tests !

- **Save/Load graph**

  To bring ButtleOFX to something more professional and useful, you can save your work, and reload it when you want to continue your composition. Everything is saved and can be reloaded : the state of the graph, of each node (its parameters), and of the viewer (with its 9 views).

  We decided to use the standard JSON, a lightweight data-interchange format understandable by humans.

  For those functionalities, we tried to think object-oriented programming : the responsibility of saving and loading is delegated to the objects. Each element in ButtleOFX (the graph, the nodes, the parameters, the viewer...) can save and load itself : in other words, it can be serialized and deserialized. This is translated in the code in this way :

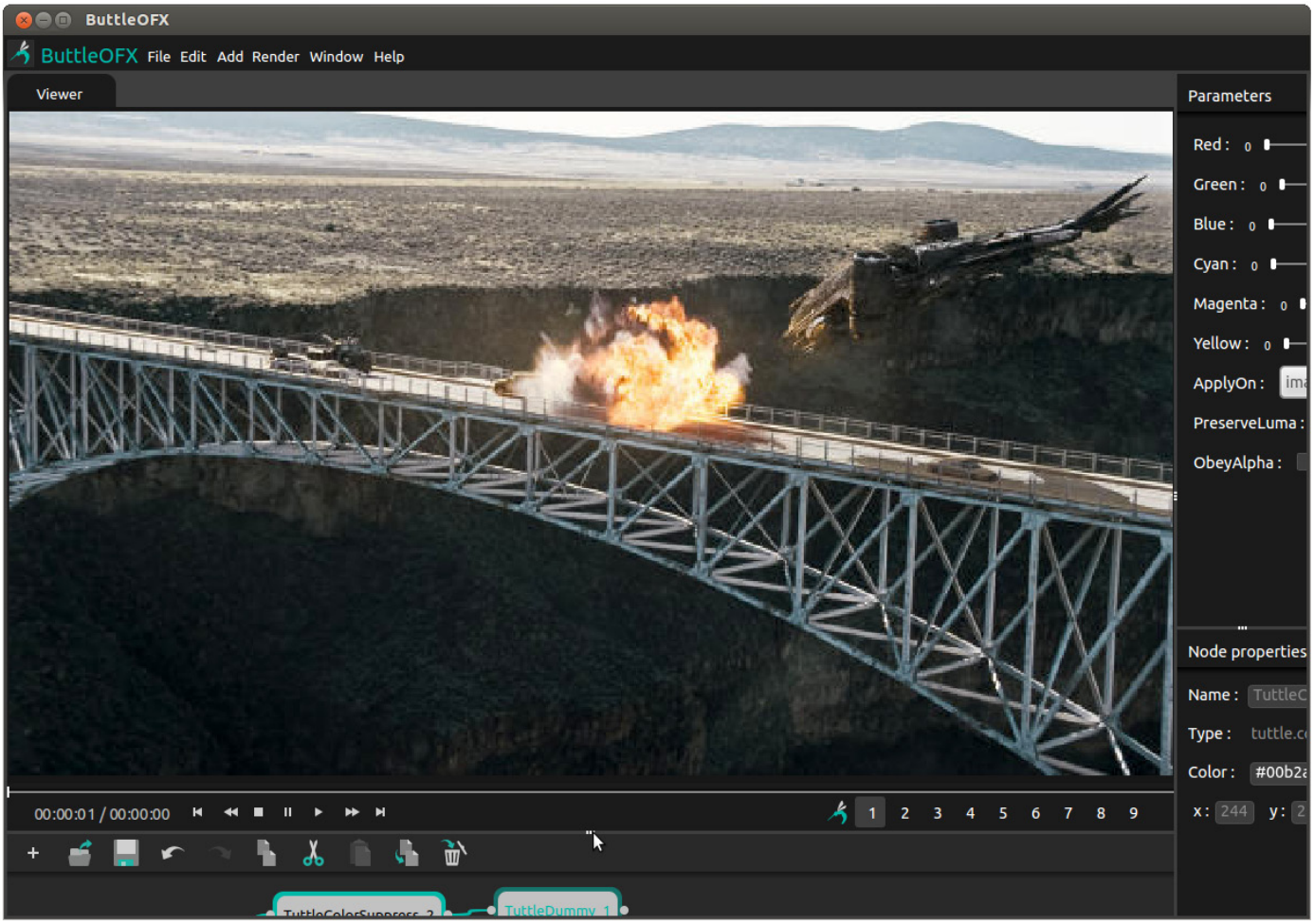
```
class Element_Buttle:
        def object_to_dict(self)  # serialization
        def dict_to_object(self)  # deserialization
```

  As you can see, we use the python build-in type dictionary (dict), very useful to represent data like this : “name : value”.

  Finally, in this section, we took care about saving the least amount of data, in order to manipulate the smallest possible JSON files. The solution was to save only the parameters with a value changed by the user, but not save the other parameters : we already know their default value !

- **Resizable application**

  You can resize the three parts of the application in the way you want. It is very convenient if you want to see a large computed image, or if you have to examine the parameters of a node which is very complex (with a lot of parameters), or also if you need to have a large view of your graph. We used the Qt components SplitterRow and SplitterColumn, which provide a way to layout items horizontally and vertically.



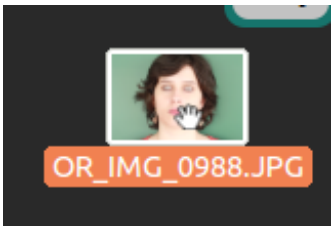
Example of customized interface

Drag and drop

For an optimal ease of use, we decided to implement the drag and drop tool for some actions in the application. Three main actions in ButtleOFX can be done by a drag & drop :

Creation of reader nodes

Dropping an image or a video from your computer to the graph of ButtleOFX will automatically create the corresponding node in the graph, with the right url parameter. The file received by the graph is analysed to check if the format can be read by TuttleOFX, and is refused if not. As we can save a graph in a JSON format, it is also possible to drop the JSON file in the application, and the graph saved in the file will be completely recreated !



Dropping an image

Connection between the nodes

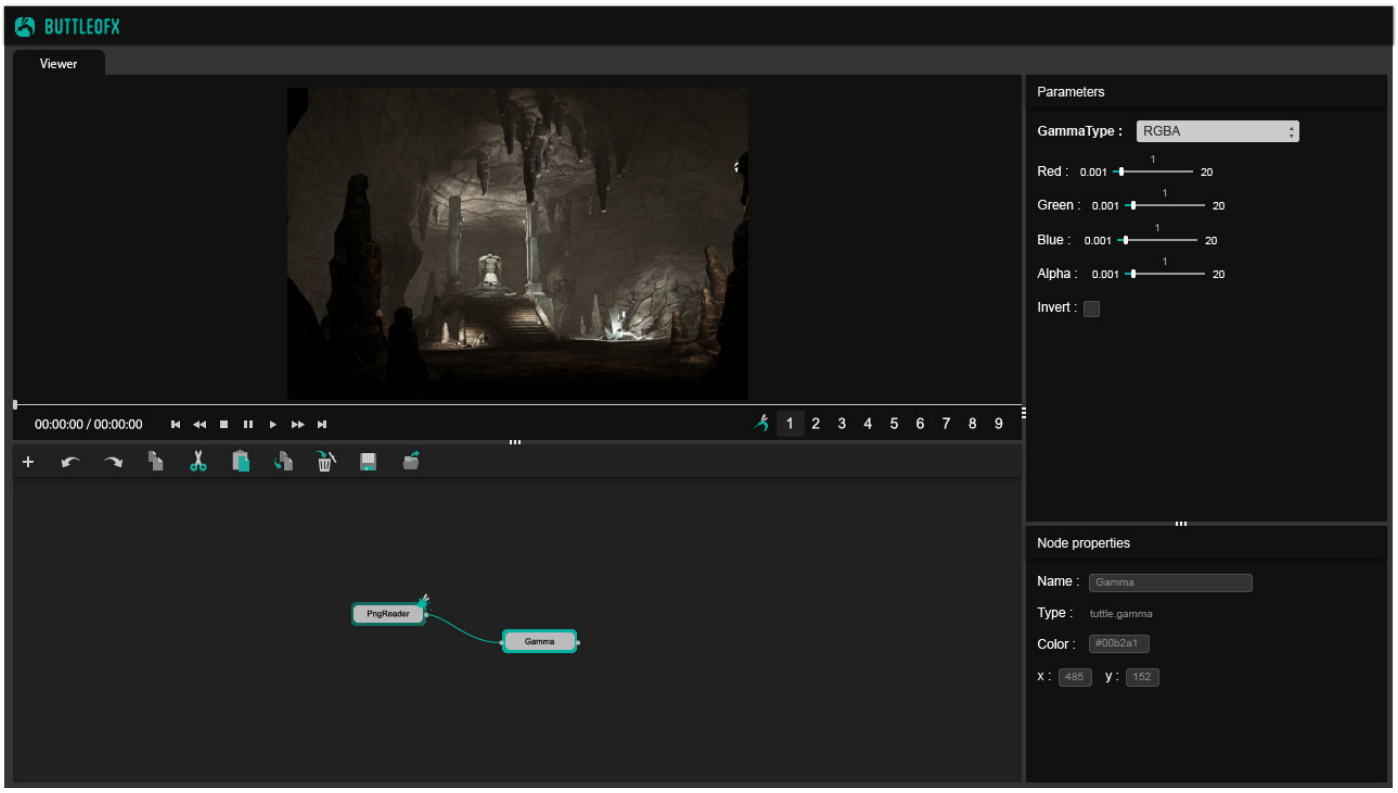
To create a connexion between two nodes, you have to click on the right clip of the first clip and drag it onto the clip of the second node. During the movement, the look of the curve is displayed under the mouse cursor. When the mouse reaches the second clip, there is a visual magnetic effect indicating that the connexion is possible.

Attribution of the viewer

To display a node in the viewer, you have to assign the mosquito on the node by dragging it and dropping it on the node.

The drag and drop mechanism we implemented uses a library called QuickMamba, written by Fabien Castan and Yann Lanthony. The tool used for the drag and drop was the DropArea, a declarative QObject written in Python. It allows to send signals between QML and Python to be able to easily catch data with QML, and send it to python to be processed.

Finally, our application looks like that :



How ButtleOFX looks like

We are proud of our work, because as you can see, the interface is very close to the template we created at the beginning of the tutored project.

3. THE DIFFICULTIES

We would be lying saying that we did not encounter difficulties during our project.

NEW LANGUAGES

The first difficulty was that the languages used were completely new to us. We did not know neither Python nor QML, the two languages we used. During the first weeks, we discovered the languages and then we were quite slow in the development. Moreover, QML is still on development, so sometimes, the solutions to our problems are not so easy. But with practice, documentation and sometimes help of our tutor, we managed to have a working application.

An issue that made us waste a lot of time is the suppression of a node. We were discovering PySide and managed to create QObjectListModels (QML lists created with PySide) to store nodes. But we encountered a considerable problem when we tried to delete objects from the QObjectListModel. It would work on some computers but cause a segmentation fault on others. We then had to try different ways to create and store nodes but all of them would fail. In the end, we discovered that the first method was the correct one, we just had to set a parent to the objects to avoid the segmentation fault.

GIT

Our tutor imposed us to use the version control system Git. We never worked with this tool and at the beginning it was quite hard to understand all its subtleties. It took us two months to start using it, and a few more months to use it without being afraid of destroying everything everytime we had to push our latest version of code.

DIFFERENT LAYERS IN BUTTLE

As our project was quite important in term of number of files compared to our other student projects, we had to divide properly the code. At the beginning, linking the different layers was not so easy to implement. We changed the architecture of the application many times to get to the final state.

The first organization we set up was divided in three directories : core, data and gui. The core directory contained the undo/redo commands. But we had core objects in the gui directory, which is not very logical.

Then, we cleaned a little to move the core files into the right directory. We also created a singleton ButtleData in which we stored the data of the current composition (the nodes in the graph, the values of the parameters).

Later, we splitted ButtleData to extract some of its information into a new singleton, ButtleManager.

Our tutor knew all that from the beginning but it was hard for us to understand the job of these objects without facing up the problem first. Even if it took us some time, we found an appropriate way of structuring our application and learned how softwares' data can be managed.

DEPENDING ON TUTTLEOFX AND BINDING

A considerable difficulty of this project was to be transplanted on an existing project, TuttleOFX. We had to discover and perfectly understand the details of its functioning. Moreover, TuttleOFX being written in C++, we had to use a Python binding to use it, and this binding did not provide all the functionalities we needed. Therefore, we regularly had to ask our tutor to make these functionalities available in Python.

All these encountered difficulties were very interesting and stimulating. We had to learn to effectively use the documentation to learn the new languages. We are now much comfortable with the use of git, which is very important for a good developer. We had to think hard to have a good architecture; it was complicated at first, but very useful once set up. Finally, it was interesting to work from an existing project, because it will probably be the same when we incorporate a company.

3. THE FUTURE OF THE APPLICATION

We were so enthusiastic when we started working on this project that we all wanted to suggest a lot of ideas. Before it could become a disorder, we decided to draw up a list of the possible features to create. From this list, we defined a scope of work in which we sorted the features per importance for each of our three modules : graph editor, viewer and parameters editor. During the project, we used to pick the bottom list items to divide up tasks. Unfortunately, we never picked up the last items so there are still a lot of features to create.

NEW FEATURES

- **Keyframes**
  Compositing is mainly used to create special effects on videos. At the moment, ButtleOFX allow users to apply effects on both images and videos. But, concerning videos, it is not possible to modify effects through the time, they are applied on the whole video. For this, we need to set up the keyframes management. Users would then be able to set different values for a single parameter during the time.

- **A better viewer**
  The current viewer is a basic viewer : it allows to display an image and zoom in and out of it. Compositing softwares offer a lot more to get information about images. ButtleOFX's viewer could, for instance, give a lot of information about each pixel of the image (RGB, transparency, light...). This feature would be very useful for a better control on the effects, in order to do a quality work on the images.

- **Preview of each node as a thumbnail**
  Because compositing can be described as a sequence of effects, it is interesting to have a clear vision of which node handles which effect. Having a little preview of the computed image in each node would be a good solution for this.

USER-FRIENDLINESS

We want ButtleOFX to be the most user-friendly possible, with attractive features for beginners. For this reason, there are a lot of unessential features that could bring a huge improvement to the application :

- **Zoom and graph miniature**
  On a classic computer screen, the graph gets quickly too small after adding a few nodes. We suggest three solutions to solve this problem :

  Enable the user to zoom in and out in the graph
  He would be able to have an overview of the graph or work on a specific part. This is already possible, but for the moment, the zoom just consists in a scale of the graphic elements, which is not very nice. This can be improved.

Display a miniature of the graph
It could be located in the bottom right corner of the panel. The user would then be able to know on which part of the graph he is working at the moment.

Allow the user to merge the graph with the viewer
This feature is inspired by the compositing software **Toxik**. The idea is to have the viewer as the background of the graph. This is a funny way to optimize the workspace.

All these options combined, the user would easily move around in the graph panel.

- **Multiple viewers**
  In the viewer it is possible to switch between nine different images. In order to increase the possibilities we can imagine allowing the user to create multiple viewers. In each viewer, the user would be able to compare nine different images.

- **Light in the connections**
  When a parameter is modified on a node, TuttleOFX computes the new image using all the previous nodes. It can take some time if it requires difficult calculations. We wanted to put a light in the connections between the nodes, corresponding to the progress of the computing.

These are some examples of the possible improvements that can be added to ButtleOFX. But there will always be new features to create and we soon noticed that : when a task was done, new ones always appeared. So we guess these lists will never end up !

CONCLUSION

We have spent a significant time, during the past 7 months, implementing as many features as possible, and we can see a relevant achievement. Thanks to our tutor, who spent a lot of his evenings giving us advices and debugging the application, we managed to develop all the essential features, plus a few extra ones, and now have a functional application.

Working on this project has brought us a lot. First, we learned new languages : Python and QML. The second one is not so used at the moment but since it is very powerful, it should be more used later and using this language could be a real plus in the future. We also discovered Git and the web-platform Github to share our code. This tool is very used in the professional world and enables to use agile methods. It helped us to work as a team. We were five students, and it is quite rare in our student projects to be so numerous. Indeed, we often work in little groups and duos. We learned to divide the project in small tasks and to schedule them in milestones. We also had to work on some code that we did not write. So everybody had to comment his code and to write it cleanly. As the project was quite big, we had to organize the code in different layers. It forced us to think about the architecture and to use some design patterns that we studied this year. Even if we had difficulties at the beginning, it forced us to learn how to resolve them using documentation.

To conclude, ButtleOFX was a fascinating project and we liked to work on it. We had a great understanding in the team that pushed us to work. That is why we planned so many features at the beginning. But we soon had to restrict our bill of specification since the basic features were not so easy to implement. But we are proud of the result and maybe another team will improve the project in the future. Because open source projects are never over !

BUTTLEOFX

OPEN SOURCE COMPOSITING SOFTWARE

Clément CHAMPETIER
Xochitl FLORIMONT
Aurélien GREFFARD
Elisa PRANA
Arthur TOURNERET

Fabien CASTAN