Seminar Report

vvvv

Stefan Spiss
stefan.spiss@student.uibk.ac.at

27 July 2015

Supervisor: Maria Schett

Abstract

This work gives an introduction to the visual toolkit vvvv. It will give a short overview over the history of vvvv and discuss the concepts visual programming and dataflow approach as well as important properties and the environment of vvvv. Additionally the similarities according to the used concepts and the properties of other programming languages, like Max and Pure Data, will be outlined. Moreover for the understanding two different ways to generate and print the text of the song *99 bottles of beer* with vvvv will be discussed.

Contents

1 Introduction

The visual programming language vvvv is not presented on the website of *99 bottles of beer*[1]. On this website there are about 1500 different programming languages presented by the source code that generates the lyrics of the song *99 bottles of beer.* Therefore in this seminar report this language is used to create a program that generates the song text. Furthermore an overview over vvvv is given which discusses used programming paradigms, its history, its environment, its syntax and all its properties. The report also compares vvvv to other languages. At first in Section 2 and Section 3 the two programming paradigms visual programming and data-flow programming are explained, because vvvv uses both of them. After that in Section 4 the history of vvvv is discussed. Section 5 gives a overview over the syntax and the environment. Furthermore Section 6 discusses all the properties of vvvv. Then in Section 7 two programs, which both generate the 99 bottles of beer song text, are explained in detail. In Section 8 vvvv is compared with the languages Max and Pure Data, because these two are the most similar ones to vvvv. Finally Section 9 concludes the report.

2 Visual Programming Languages

Since vvvv is a visual programming language (VPL) this term is described in this section. At first the motivation for VPLs is discussed. Then the definitions for visual programming are described and in the end a short history overview is given.
The main motivation for the development of VPLs according to Marat Boshernitsan and Michael S. Downes (2004, page: 1, [2]) is highlighted by following questions: They state that "humans have long communicated with each other using images" and ask "why, then, do we persist in trying to communicate with our computers using text-based programming languages? Would we not be more productive and would the power of modern computers not be accessible to a wider range of people if we were able to instruct a computer by simply drawing for it the images we see in our mind's eye when we consider the solutions to particular problems?". So the goal of visual programming is to make the programming task more intuitive and therefore accessible to a wider range of people.
In previous papers about VPLs the term *Visual Languages* occurs and is used according to Navarro-Prieto Raquel and Cañas Jose J. (2001, page: 804, [8]), "to refer to programming languages at the highest level of abstraction and include all systems that utilize pictorial information." Furthermore in [7] these visual languages are divided into two categories: program visualizations and visual programming languages. Program visualization is the graphical presentation of aspects or runtime parameters of common text-based code/programs whereas in VPLs the graphics are used to create the program itself.
The history of VPLs is described in [2] as following. The field of visual programming developed out of the combination of computer graphics, programming languages and human-computer interaction. One of the first developments in visual programming was the *Sketchpad* system from Ivan Sutherland in 1963 designed on the TX-2 computer at

[1]99 bottles of beer website: `http://www.99-bottles-of-beer.net`

the Massachusetts Institute of Technology [11]. With that system it was possible to create 2D graphics. The most important contributions of that system to visual programming has been the graphical user interface and the possibility of user defined constraints on primitives of the 2D graphic. Another early contribution to visual programming was the development of a simple visual data-flow language (data-flow language will be discussed in the next section) also on the TX-2 from William Sutherland in 1966 [12]. The next major step in visual programming was the PhD dissertation of David Canfield Smith with the title *Pygmalion: A Creative Programming Environment* in 1975 [10]. In this work Smith used an icon-based programming approach. There the user/programmer created, modified and linked together small pictorial objects. These objects were called icons and had properties that defined the computations. Started by the work of Smith this icon-based programming approach has developed until today.
A present program using such an icon-based programming approach is for example vvvv. Other VPLs which are in use today are Max and Pure Data. These will be discussed in Section 8.

3 Data-flow Programming Languages

The language vvvv uses the data-flow programming paradigm. In this section at first this paradigm and the main motivation are explained shortly. After that an overview of the history of data-flow programming is given.
According to [3] the main motivation for the research into the field of data-flow programming was to get high parallelism. For that data-flow programming languages model the data in a data-flow graph, which is a directed graph, where the edges describe the data flow and the nodes are the operations executed as soon as on all inputs data is available. In contrast to traditional programming languages, where the program is a series of operations happening in a specific order, data-flow languages are inherently parallel. This is a enormous advantage when using distributed systems or systems with more processors.
According to Whiting and Pascoe [21] the earliest, important contributions which used the concept of data-flow were three publications. The first one is the work from Karp and Miller with the title *Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing* from 1966 [4]. Another work was the PhD thesis *A computation model with data flow sequencing* from Adamas in 1969 [1]. The PhD thesis from Rodriguez also in 1969 was the third work. It had the title *A Graph Model for Parallel Computation* [9].
Also the work from William Sutherland from 1966 (already mentioned above in the Section 2) is important, because in this work he developed one of the first visual data-flow programming frameworks [23]. As can be seen by the work of Sutherland, already very early visual and data-flow programming were combined. The reason for it is that the data-flow graph can be visualized easy. Today there is a wide range of programming languages using the data-flow approach. Examples are vvvv, but also Max and Pure Data (will be explained in Section 8) and all of them also are VPLs as already mentioned above.

4 History of vvvv

The following history is based on [5] and [6]. In 1997, the company *MESO* Digital Media System Design was founded by Stefan Ammon, Michael Höpfel, Karl Kliem, Sebastian Oschatz and Max Wolf. MESO works on projects for designing and implementing digital media systems (e.g.: media installations). One year after the establishment of MESO, Sebastian Oschatz and Max Wolf started the development of vvvv. Their goal was to build a multimedia tool to prototype and implement media installations. Sebastian Oschatz did his diploma in computer science at the TU Darmstadt in 1994 and therefore was the leading software architect of vvvv from 1998 until 2007. Max Wolf has been an associate of the vvvv group since its founding and he is working as designer. Soon Sebastian Gregor joined there group who developed many of the core algorithms of vvvv. In 2000, Joreg set up the graphical user interface for his diploma thesis and also joined the team. At the beginning vvvv was an in-house tool of MESO. In December 2002, the first public release of vvvv happened. Finally, in 2006 the vvvv group was found existing out of Joreg, Sebastian Oschatz, Sebastian Gregor and Max Wolf. All further developments of vvvv were handed over to the them. Today the development is lead by Joreg and Sebastian Gregor. Today vvvv is available for non-commercial use at `http://vvvv.org/downloads`. For commercial use licences can be bought at `http://vvvv.org/documentation/licensing`.

5 Syntax and Environment of vvvv

This section introduces the environment of vvvv and the basics of its syntax. First of all, vvvv applications are called *patches* and consist of *nodes* representing operations and *links* between nodes representing the data-flow. These links are directed so a patch actually visualizes a data-flow graph. Normally, the patches are stored as xml files on the disk.

The language vvvv has its own environment for developing programs and to run them. Therefore this environment is described in detail here according to [17] [18]. In Figure 1 an empty patch with opened main menu can be seen. To open the main menu either the middle mouse button must be pressed or SPACE + right button. The main menu consists out of four categories. Next to every command its shortcut can be seen. So in vvvv nearly everything can be done using shortcuts. On the top left side all commands regarding the window are listed, for example making snapshots, switching between different window modes (e.g.: FullScreen) or saving all. On the top right side there is the menu for the current patch. There, commands regarding the patch can be chosen. Examples are open a patch, saving a patch, lock and unlock a patch (a locked patch cannot be changed any more). On the bottom left side the menu is called "Main". There, important commands can be found to create new patches, open additional windows like the Inspektor (tool to analyse nodes) or the Finder (tool to find nodes in the current patch). Furthermore, there are different options for debugging. On the bottom right side are all commands to edit the patch like undo, redo, cut, different copy commands or different paste commands.

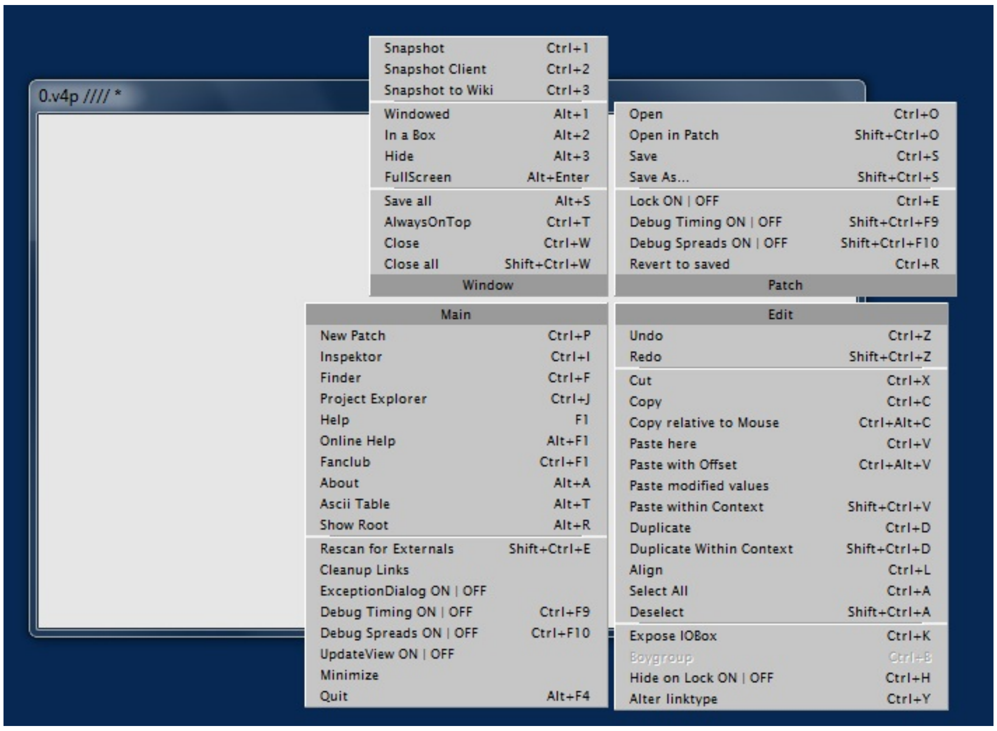


Figure 1: Empty patch with opened main menu.

The functions in vvvv are the nodes. Each node represent an operation to either generate or process output data. As can be seen in Figure 2, a node is visualized as box with its name “example” in the middle and black squares on top and bottom of it. These black squares present the input and output pins of the node. Compared with a function in a text-based programming language the input pins would be the function parameters/arguments and the output pins would be the results of the function. Every pin has a type. These types are explained in the end of this section. The most nodes have only one output pin, but there are also some which have more outputs. To add a node left double-click anywhere in the patch is enough and a list opens showing all available nodes. There the nodes are listed by their names. This names consist of several parts following the pattern: “NodeName(Category Version1 Version2 ... VersionN)”. The versions are optional, but the category is very important, because it makes it possible to have more nodes with the same name by having different categories (e.g. +(Value), +(String), +(Color)). So in vvvv polymorphism is supported by using these categories. Here are some of the categories available in vvvv. All the nodes of the category *Value* are dealing with numerical values (e.g.: mathematical operations) [13]. To compare it with other programming languages, all functions operating on integers, doubles, floats or other numerical types are handled in vvvv with nodes of only one category: *Value*.
Another category is *Strings*. In vvvv a string is the same like in other programming

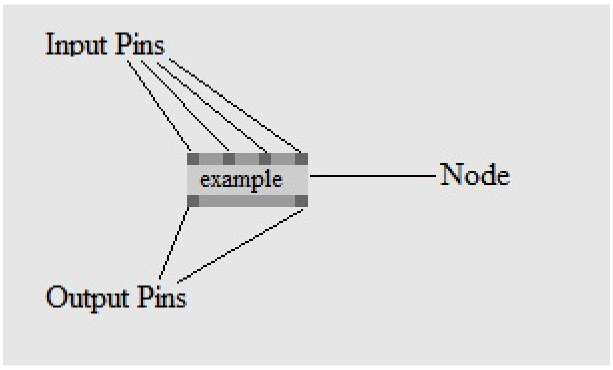


Figure 2: Example node labelled for description, after [18].

languages, just a sequence of characters. All nodes of the category Strings therefore are representing operations on strings [13]. Like in many other programming languages (e.g.: C, C++, Java) there is also a category for files. These nodes offer different operations on the file system (e.g.: read files, write files, delete files) [13].
As explained in [19] in vvvv a *spread* is a one-dimensional array of data (e.g.: Strings, Values, Colors). One element of a spread is called *slice*. As already mentioned every pin of a node has a type (e.g.: Value, String, Color). With spreads it is possible that one pin does not only hold one instance of data, but a whole spread of data. So the nodes with categories like Strings, Value, or Color can also be used with spreads of data. Furthermore there are also nodes with the *Spreads* category. These nodes are dealing with creating and operating especially such spreads [13].
The *Animation* category is for all nodes which will animate over time and because of that have an internal state (e.g.: generate motion, smooth motion, FlipFlops) [13].
Nodes of the *Devices* category can be used to control external devices and get data from them [13]. There are still more categories available in vvvv which are not explained here. The kind reader is referred to [13].

The remainder of this section follows [14]. To connect two nodes, as already mentioned above, links are used. It is only possible to connect an input pin with an output pin and only if the types of the pins match. So vvvv already takes care of type checking for the user/programmer and therefore is a strongly typed language. Furthermore an input pin can only have one link. Immediately after connecting the two nodes A and B, the data from A's output pin is available on the input pin of B.
In Figure 3 a simple +(Value) node can be seen which has two IOBox(Value) nodes as input and one IOBox(Value) as output. IOBox(Value) nodes are used to visualize input and output. So the value of the node, which is set on its input pin is shown in the node and also the output pin has that value. The +(Value) node takes the values of the two input pins, adds these two and put the result to its output pin. The result is then viewed in the IOBox(Value). In this example if one of the nodes would not have the category Value, it would not be able to connect its input or output pins with the other nodes. The

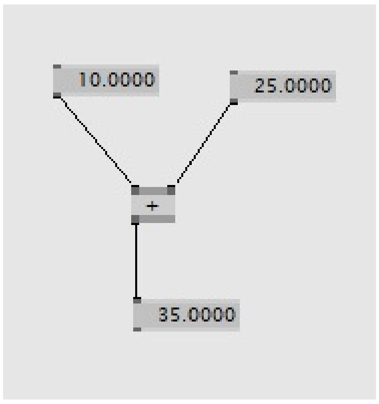


Figure 3: Demonstration of connecting different nodes with links.

reason for that are the different pin types used in vvvv.
One of the most important pin types are *numeric values*. In contrast to other programming languages vvvv does not distinguish between different types for numbers like integer, float, double or even boolean. The language vvvv just handles all this different types as numeric values.
Subtypes were introduced to edit and display the numerical values in the GUI correctly. So this subtypes are only used by the GUI and ignored for calculations. There are subtypes for all the different numerical values like floating point numbers, integers and boolean. Additionally these subtypes include flags which can be used to specify the values. Examples for that are to set the default values, the minimum and maximum values or the step sizes.
As in other programming languages (e.g.: Java, C++) vvvv also has *strings*. In vvvv these strings can have any length and supports the character sets *Universal Character Set Transformation Format 8-bit* (UTF-8) and *ANSI* which is mostly referred to as *extended ASCII*. Also for strings vvvv provides subtypes which contain information about the string. Again the information of the subtypes are used in the GUI when entering or editing strings.
Another pin type in vvvv is *color*. This is a data type that support different operations on colors and also different color models like *Hue-Saturation-Lightness* (HLS) or *Red-Green-Blue* (RGB).
There is also a pin type in vvvv for *enumerations* like in other programming languages (e.g.: Java, C++).
Furthermore to the above basic pin types there are special pins called *node pins*, which are using advanced data types. Generally it is not possible to see or manipulate the actual data of node pins. Types for these node pins are transforms, renderstates and samplerstates, textures, meshes and vertexbuffers and DirectShow audio- and videostreams. For detailed information about this advanced data types the interested reader is referred to [14].

6 Properties, Capabilities and Features of vvvv

In this section first important properties of vvvv are discussed and after that main capabilities and features according to the developers are listed.
The language vvvv is a VPL and it uses a data-flow approach as already mentioned above in Section 2 and Section 3.
The remainder of this section, if not indicated otherwise, is based on the vvvv propaganda website [16]. The language vvvv is written in Borland Delphi[2], which is a programming language originally developed by Borland and today developed by Embarcadero. Plug-ins for vvvv can be developed in the software framework .NET[3] in the programming language C#[4] which both were developed by Microsoft. Since vvvv uses DirectX[5], which is a collection of application programming interfaces (API) to handle tasks related to multimedia for Microsoft platforms, for video synthesis and processing, it is only available for Microsoft Windows.
The development environment in vvvv is hybrid visual and textual. The visual environment was already explained in Section 5. All the nodes are provided in the node libraries. Using the vvvv-sdk makes it possible to create one's own nodes. In the vvvv's addonpack, which can be seen as node library, nodes contributed from users are provided. Furthermore the visual environment is at the same time the runtime environment of vvvv. This runtime environment has only one mode, runtime. For that the environment uses frames with 120 frames per second at the most. In every frame all operations of the nodes are calculated once. All the necessary compilations happen in the background and the patches are edited/build while the program is running. Additionally to the own visual programming language the text-based languages C# and also shaders written in High-Level Shading Language (HLSL)[6], a shading language developed by Microsoft using the Direct3D API which is part of the DirectX toolkit, are compatible with the environment.
For 2D and 3D animations vvvv features three different rendering engines: DirectX9, DirectX11, which are both part of the DirectX toolkit, and a rendering engine for Scalable Vector Graphics (SVG)[7], which is a XML-based vector image format that supports animations. That makes it possible to load, animate and generate 3D models and add textures, shaders and lighting to the scenes.
Moreover vvvv also offers the open source physics simulation engines Box2D[8] and Bullet[9]. Because vvvv was designed for multi-projection installations too, it includes a vvvv specific feature called *boygrouping*. With that feature it is possible to create clusters where client PCs are controlled by a server PC. The server PC distributes content to the client

[2]for Delphi see `http://www.embarcadero.com/products/delphi`
[3]for .NET see `www.microcoft.com/net`
[4]for C# see `https://msdn.microsoft.com/en-us/vstudio/hh341490.aspx`
[5]for DirectX see `https://msdn.microsoft.com/library/windows/apps/hh452744`
[6]for HLSL see `https://msdn.microsoft.com/en-us/library/bb509635%28v=VS.85%29.aspx`
[7]for SVG see `http://www.w3.org/TR/SVG11/`
[8]for Box2D see `http://box2d.org/`
[9]for Bullet see `http://bulletphysics.org/wordpress/`

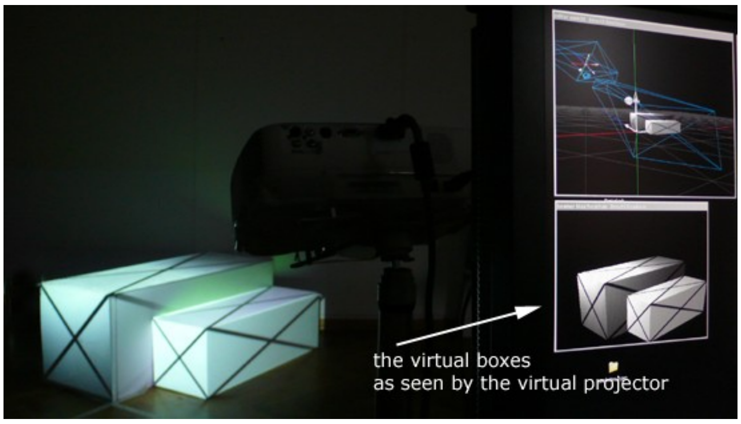


Figure 4: Example for projection of texture to a 3D-object. Here the textures are the black crosses, from [15]

PCs and they operate simultaneously to the server.
Another feature from vvvv is the projection mapping. For projections on irregular surfaces without loosing the shape of the content vvvv provides tools to load and create meshes imitating the shape of the projection surface. Having these meshes it is possible to render a virtual 3D model of the projection when using the same lens-characteristics as the real world projector has. Any flat textures used in the virtual model than can be projected on to the real world objects without looking distorted. In Figure 4 is an example of projecting textures (black crosses on white) on to two cubes where on the right side the virtual model and on the left the real world objects can be seen. The black crosses on the real world objects look undistorted and behave like flat textures independent of the spectators point of view.

Furthermore vvvv supports data visualization without wasting much time on creating the program. For that the data can be imported reading from disk, database (MySQL, PostgreSQL, SQLite, SQL, Odbc, OleDb) or the network (e.g.: HTTP, UDP). After that it is possible to parse (e.g.: xml, json, csv, regexpr), transform and finally draw (SVG, DirectX) the data as vector graphics, images, videos or lists.
Another feature of vvvv is the multitude of readers and writers for common standard protocols (e.g.: MIDI, OSC, TUIO, DMX, HTTP, TCP, UDP). This makes it easy to use physical devices (e.g.: Wii, Leap Motion, Kinect, Oculus Rift, Arduino).
The language vvvv also can be used for a wide range of sound applications. Although it cannot compete with Pure Data and MaxMSP (discussed in detail in section 8) it still supports techniques like multichannel audio playback, fast Fourier transform (FFT), which is an algorithm to compute the discrete Fourier transform (DFT) and its inverse, analysis or the computer sound card driver protocol Audio Stream Input/Output (ASIO).
As already mentioned before the easy use of physical devices combined with libraries like

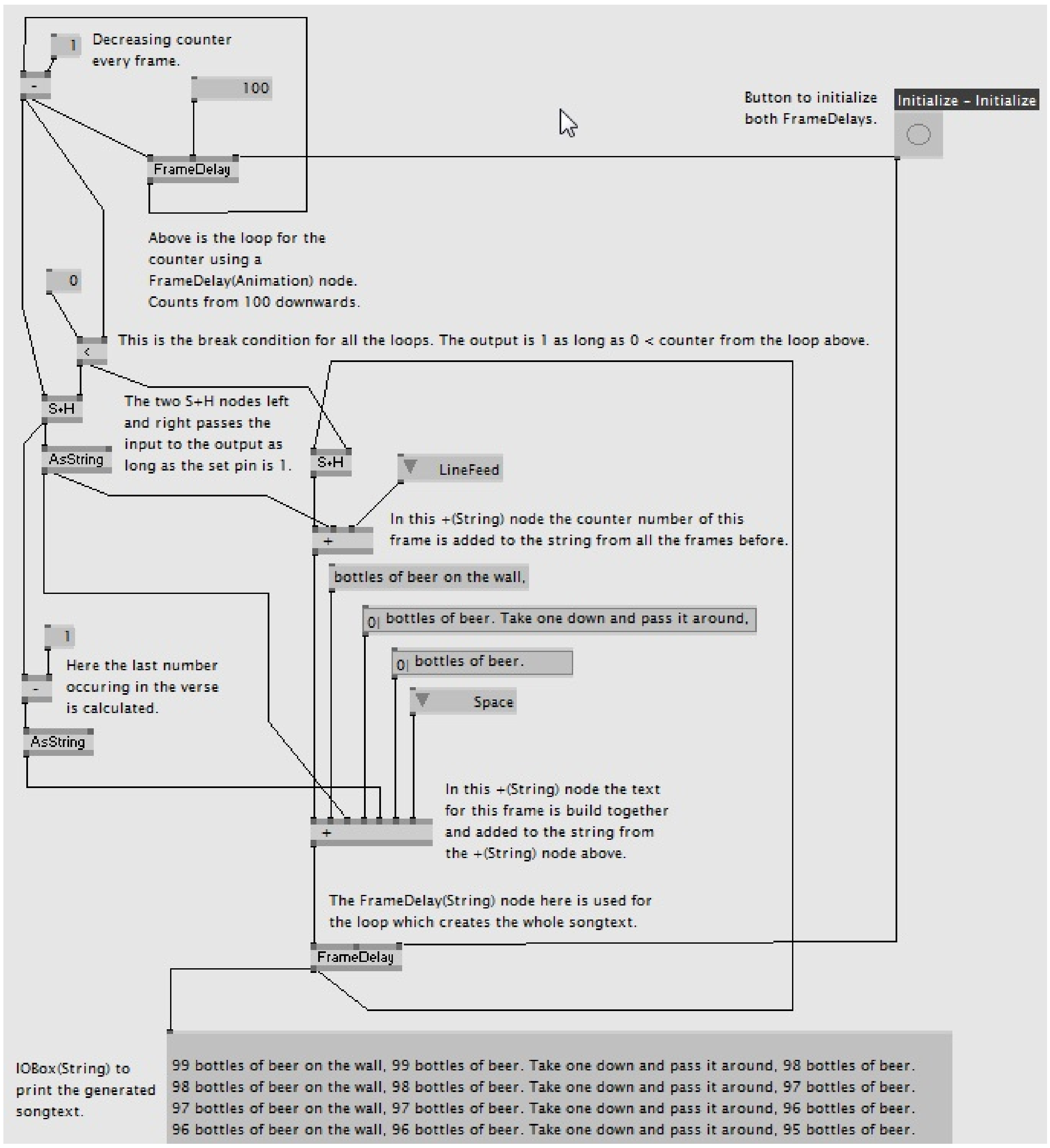
Open Source Computer Vision (OpenCV)[10], which is a library of programming functions for real-time computer vision, makes it possible to us vvvv to develop computer vision applications.

7 Implementations of 99 Bottles of Beer

To create the *99 bottles of beer* program in vvvv here two different solutions are presented. The first solution in Figure 5 is building loops, which is complicated in vvvv. The second solution in Figure 6 uses the spreads of vvvv to generate the output string in one frame. As already mentioned above in vvvv all nodes in a patch are calculated in each frame once. That fact makes the solution using loops complicated, because it is not possible to connect nodes the way that there is a node, which input is dependent on its own output, in one frame. For this reason any FrameDelay node has to be used. This nodes do not use the input value of this frame, but the one from the frame before. In Figure 5 the patch for the first solution using loops can be seen. In contrast to text-based programming languages (e.g.: Java, C++, C) where for the 99 bottles of beer program one loop is enough, vvvv needs two loops. The first loop counts from 100 downwards. The second loop always appends the new string from the current frame to the string from the frame before. Because the first loop is handling values, a FrameDelay(Value) is needed. But the second loop is working with strings, so it needs the FrameDelay(Strings) node and therefore two loops are necessary. The break condition of the loops is realized with the S+H nodes. These nodes are forwarding the input as long as the set pin is 1. With the <(Value) node it is checked if the counter from the first loop is greater than 0. As long as that is the case the output of the <(Value) node is 1. This output is the input for the set pin of the S+H nodes. So as soon as the counter is 0 or lower the S+H nodes do not let any data pass through and the output string of the whole system does not change any more. The whole output string is printed in the IOBox(String) node on bottom of the patch. All the important nodes are also described in the picture.

The solution using spreads is the way that should be used when working with vvvv. In the patch, which can be seen in Figure 6, at first the I(Spreads) node generates a spread of integers from 99 to 0. Then for each slice a verse of the song is generated and with the +(String Spectral) node the spread of strings is concatenated. This string is printed again in a IOBox(String). It has several advantages compared with the first solution using loops. At first it is much faster in computation, because everything is calculated in one frame. In contrast when using loops 99 frames are necessary which are 0.825 seconds when using the maximum frame rate of vvvv (120 fps). The solution using spreads only needs 0.008333 seconds to generate the string which is $\frac{1}{99}$ of the time needed for the loops solution. That can be calculated using the number of frames necessary to generate the string and the frame rate of vvvv. Furthermore the patch is less complicated as less nodes and links are needed.

[10]for OpenCV see `http://opencv.org/`



Decreasing counter every frame.

Button to initialize both FrameDelays.

Above is the loop for the counter using a FrameDelay(Animation) node. Counts from 100 downwards.

This is the break condition for all the loops. The output is 1 as long as 0 < counter from the loop above.

The two S+H nodes left and right passes the input to the output as long as the set pin is 1.

In this +(String) node the counter number of this frame is added to the string from all the frames before.

Here the last number occuring in the verse is calculated.

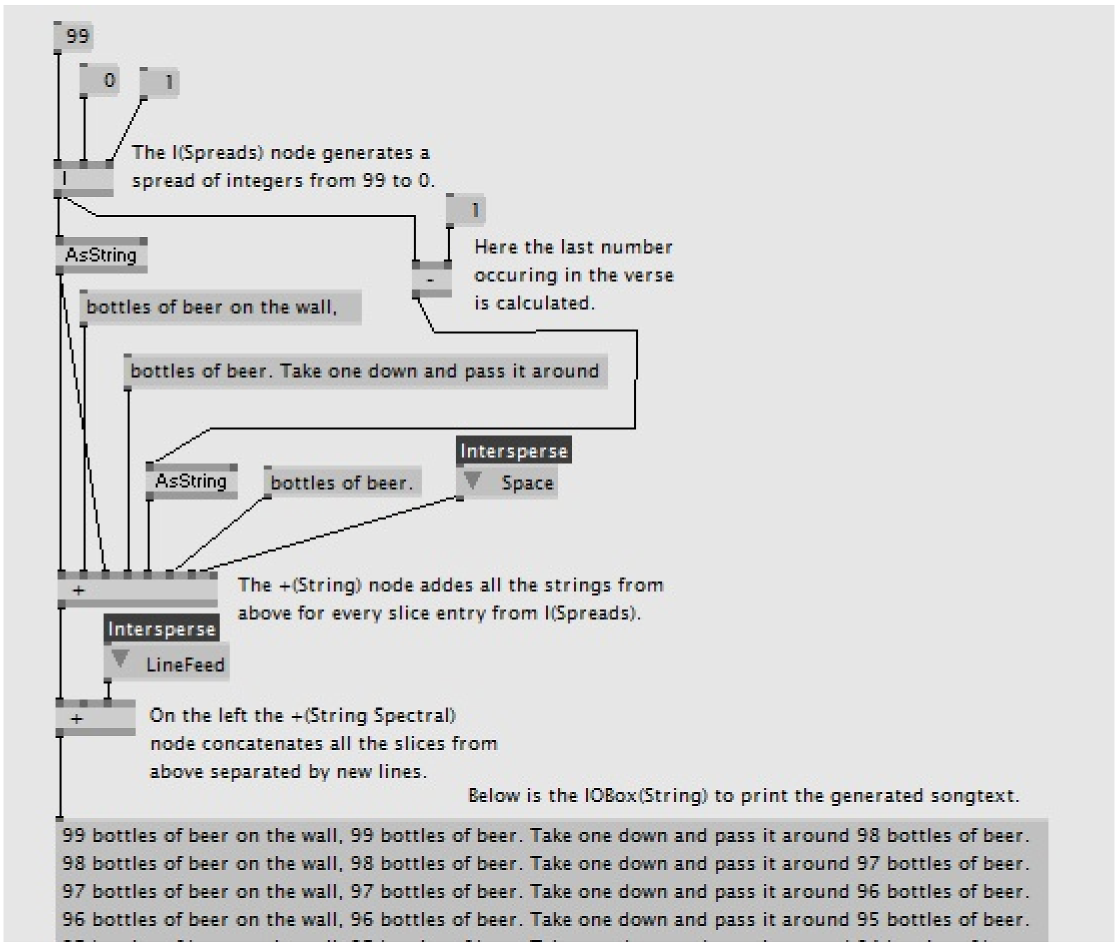
In this +(String) node the text for this frame is build together and added to the string from the +(String) node above.

The FrameDelay(String) node here is used for the loop which creates the whole songtext.

IOBox(String) to print the generated songtext.

```
99 bottles of beer on the wall, 99 bottles of beer. Take one down and pass it around, 98 bottles of beer.
98 bottles of beer on the wall, 98 bottles of beer. Take one down and pass it around, 97 bottles of beer.
97 bottles of beer on the wall, 97 bottles of beer. Take one down and pass it around, 96 bottles of beer.
96 bottles of beer on the wall, 96 bottles of beer. Take one down and pass it around, 95 bottles of beer.
```

Figure 5: A patch which generates the 99 bottles of beer song text using loops.


99
0
1
The I(Spreads) node generates a spread of integers from 99 to 0.
I
1
AsString
-
Here the last number occuring in the verse is calculated.
bottles of beer on the wall,
bottles of beer. Take one down and pass it around
AsString
bottles of beer.
Intersperse
Space
+
The +(String) node addes all the strings from above for every slice entry from I(Spreads).
Intersperse
LineFeed
+
On the left the +(String Spectral) node concatenates all the slices from above separated by new lines.
Below is the IOBox(String) to print the generated songtext.

```
99 bottles of beer on the wall, 99 bottles of beer. Take one down and pass it around 98 bottles of beer.
98 bottles of beer on the wall, 98 bottles of beer. Take one down and pass it around 97 bottles of beer.
97 bottles of beer on the wall, 97 bottles of beer. Take one down and pass it around 96 bottles of beer.
96 bottles of beer on the wall, 96 bottles of beer. Take one down and pass it around 95 bottles of beer.
```

Figure 6: A patch which generates the 99 bottles of beer songtext using spreads.

8 Comparison To Max and Pure Data

In this section vvvv is compared to other programming languages. According to [22] the two languages which are most likely similar to vvvv are *Max* and *Pure Data*. Both of these languages initially have been developed for the Musical Instrument Digital Interface (MIDI)[11] control and audio processing. MIDI is a technical standard that provides a protocol, a interface and connectors for the connection and communication of electronic music instruments and computers. Later in both languages also the creation of 3D-real-time graphics was adapted, which make them comparable to vvvv.
Coming to the similarities between vvvv, Max [24] and Pure Data [25], all three of them are VPLs. They all are using nodes with input and output pins to represent functions. These functions are combined with links. Furthermore the data-flow programming paradigm is used by them. So the data-flow in all three of them is modelled along links from node to node. Another similarity is the ability of processing music and multimedia in real-time and the possibility to develop new patches and add objects written in text-based languages is similar in all three of them.
According to [22], as Max and Pure Data do not use spreads their ability for parallel processing of objects is poor compared with vvvv. Concerning the features for sound processing Max and Pure Data are more powerful than vvvv, because both were developed for it, whereas vvvv is more powerful in video processing and graphics applications.

9 Conclusion

VPLs use pictorial information as presentation for programs. Their objective is to make the programming task more intuitive. Data-flow programming languages have been developed to get high parallelism. A program in such a language presents the data-flow. Both of these programming paradigms are employed in vvvv, which has been developed by MESO as a multimedia tool to prototype and implement media installations. Central elements of the syntax of vvvv are nodes, which are operations on data, links between nodes, which present the data-flow and patches, which can hold nodes, links, other patches and present a program. The important capabilities and features according to the official vvvv website [16] are the hybrid visual and textual programming environment, the 3 different rendering engines for 2D and 3D animations, the different physics simulation engines, the vvvv specific feature boygrouping for multi-projection installations, the support of projection mapping to project on irregular surfaces, the simplicity of data visualization, the multitude of readers and writers for standard protocols, the support for sound applications and the possibility to use vvvv for computer vision tasks. In course of this work two solutions to generate the song text for the song 99 bottles of beer have been developed: one is relying on loops, as is intuitive for text-based programming, the other is using spreads. The second solution is simpler with respect to the number of nodes and also faster with respect to the time needed for the generation. Finally vvvv had been compared to Max and Pure Data: they all are similar concerning the use of

[11]for MIDI see `http://www.midi.org/techspecs/`

the programming paradigms VPL, data-flow programming, their ability of processing music and multimedia in real-time and their possibility to write own objects in text-based languages. The differences are the higher ability of vvvv to process objects parallel and that it is more powerful in processing graphics and videos. In contrast Max and Pure Data are more powerful in processing sound.
Since vvvv was developed by MESO to create an easy to use prototyping and developing environment with focus on processing big multimedia data in my opinion they definitively reached that goal. Furthermore today the fields of application for vvvv have gotten much bigger and according to their website [20] there are many people and companies using it. It is my opinion that vvvv will grow bigger, because its approach using spreads makes it fast in processing and it is easy to use even for complicated projects like working with physical devices. Another reason that it will grow is the use of visual programming, which raises the usability also for people without any programming knowledge, as discussed in Section 2. An example for that is a friend of mine who has no knowledge about text-based programming and still uses vvvv to create visuals and video installations. Finally if someone is used to text-based programming vvvv might be a little bit confusing. The first version of the 99 bottles of beer program for example was created, because in text-based programming this is done with loops. Only after working a little bit more with it the spread-based approach became comprehensible to me.

References

[1] D. A. Adams. A computation model with data flow sequencing. 1969.

[2] M. Boshernitsan and M. S. Downes. *Visual programming languages: A survey.* Citeseer, 2004.

[3] W. M. Johnston, J. Hanna, and R. J. Millar. Advances in dataflow programming languages. *ACM Computing Surveys (CSUR)*, 36(1):1–34, 2004.

[4] R. M. Karp and R. E. Miller. Properties of a model for parallel computations: Determinacy, termination, queueing. *SIAM Journal on Applied Mathematics*, 14(6):1390–1411, 1966.

[5] meso.net. MESO about. `http://www.meso.net/about`, April 2015. Accessed: 05-05-2015.

[6] meso.net. MESO vvvv. `http://www.meso.net/vvvv`, April 2015. Accessed: 05-05-2015.

[7] B. A. Myers. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, 1(1):97–123, 1990.

[8] R. Navarro-Prieto and J. J. Cañas. Are visual programming languages better? The role of imagery in program comprehension. *International Journal of Human-Computer Studies*, 54(6):799–829, 2001.

[9] J. Rodrigues and J. E. Rodriguez Bezos. A graph model for parallel computations. 1969.

[10] D. C. Smith. Pygmalion: a creative programming environment. Technical report, DTIC Document, 1975.

[11] I. E. Sutherland. Sketch pad a man-machine graphical communication system. In *Proceedings of the SHARE design automation workshop*, pages 6–329. ACM, 1964.

[12] W. R. Sutherland. On-line graphical specification of computer procedures. Technical report, DTIC Document, 1966.

[13] vvvv.org. vvvv node categories. `http://vvvv.org/documentation/node-categories`, April 2015. Accessed: 05-05-2015.

[14] vvvv.org. vvvv pins and data types. `http://vvvv.org/documentation/pins-and-data-types`, April 2015. Accessed: 05-05-2015.

[15] vvvv.org. vvvv projection on 3d geometry. `http://vvvv.org/documentation/how-to-project-on-3d-geometry`, April 2015. Accessed: 05-05-2015.

[16] vvvv.org. vvvv propaganda website. http://vvvv.org/propaganda, April 2015. Accessed: 05-05-2015.

[17] vvvv.org. vvvv tutorial hello world. http://vvvv.org/documentation/tutorial-hello-world, April 2015. Accessed: 05-05-2015.

[18] vvvv.org. vvvv tutorial introduction. http://vvvv.org/documentation/tutorial-introduction, April 2015. Accessed: 05-05-2015.

[19] vvvv.org. vvvv tutorial spreads. http://vvvv.org/documentation/tutorial-spreads, April 2015. Accessed: 05-05-2015.

[20] vvvv.org. vvvv tutorial spreads. http://vvvv.org/users, April 2015. Accessed: 05-05-2015.

[21] P. G. Whiting and R. S. Pascoe. A history of data-flow languages. *Annals of the History of Computing, IEEE*, 16(4):38–59, 1994.

[22] Wikipedia (DE). vvvv. http://de.wikipedia.org/wiki/Vvvv, April 2015. Accessed: 05-05-2015.

[23] Wikipedia (EN). Dataflow programming. http://en.wikipedia.org/wiki/Dataflow_programming, April 2015. Accessed: 05-05-2015.

[24] Wikipedia (EN). Max (software). http://en.wikipedia.org/wiki/Max_(software), April 2015. Accessed: 05-05-2015.

[25] Wikipedia (EN). Pure Data. http://en.wikipedia.org/wiki/Pure_Data, April 2015. Accessed: 05-05-2015.