NTNU

Innovation and Creativity

OpenVG: Benchmarking and artistic opportunities

Brice Chevillard

Master of Science in Computer Science

Submission date: February 2007

Supervisor: Maria Letizia Jaccheri, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

OpenVG: Benchmarking and artistic opportunities

Assignment given: 01. August 2006
Supervisor: Maria Letizia Jaccheri, IDI

Abstract

OpenVG is a new open standard for 2 dimensions vector graphics for handheld devices. This project, which is a master thesis and an internship, aims to study the standard itself deeply before to study the role it can play in the future of artistic content creation.We will see that under some few conditions, OpenVG has everything to fulfil its role in the market and to attract digital artists who would like to enlarge their creation field. But the major aim of the project is to develop a benchmark for both hardware and software implementations. And to achieve this goal, a study of the theory of performance evaluation is necessary. And after developing the benchmark, it is interesting to run some few tests to illustrate some principles enounced while studying performance evaluation.

I would like to thank Letizia Jaccheri for her patience, and for helping in this project but over all for her priceless help in achieving my Norwegian scholarship.

Table of content

1 Introduction

At the beginning of this project, the aim was to explore OpenVG and to investigate about the artistic potential of this new standard and the opportunities it offers the artists. And then the goal was to study benchmarking and computer performance evaluation in order to develop a benchmark for OpenVG in order to test both hardware and software implementations.

1.1 Background

It is always amazing to consider the progression of the importance of mobile devices in our lives. The best examples are the mobile phones. But these cellular phones are not the only example. Nowadays, we all carry with us a lot of other mobile devices like PDAs, mp3 players like the very popular iPod, digital cameras, GPS receivers and so on. If we only focus on mobile phones, for example, their growing importance in our lives is now a reality. And now it seems obvious that this place in our more and more “digital” lives will be even bigger in the future. Of course after a huge boom in Europe in the late nineties, the market entered in a maturity phase. And the crazy growth of the market quickly stopped. But today the market knows a significant growth again. The last quarter of 2005 has been the best since 2001 in terms of sold mobile phone devices with 235 million of units sold and a growth of the worldwide sales about 21% for 2005 [01NET06]. The reason of this growth is that people changed their phones for much more trendy devices. Indeed, the technology progress is so fast than the devices are now as small as a credit card, they include much more than simple phoning functions. Now, most of them also include cameras, agendas, some can even play music thanks to an embedded mp3 decoders. A survey from the Finnish device maker NOKIA even announced that mobile phones may replace digital cameras and mp3 players [POC06]. And the incredibly growing bit rates of the connections coupled with the new technologies used by the 3rd generation norms for mobile communications make internet on mobile phones a reality. We could also add that it allows many more applications on mobile phones like gaming and online gaming. So we can easily understand that the part of software in these mobile devices is now more and more important.

In the same time, computers world started to open it self widely to the Open Source. An Open source software could be defined as a royalty free software, freely redistributed, with the source code available, free to be modified, platform independent, not restrictive towards other and software independent [PER06]. The most famous open source products are Linux, Open Office, Mozilla Firefox and MozillaThunderbird. One of the characteristics of the open source world is that it generates communities of interests around the open source products. These communities of interest gather people with different motivations around a same goal. These persons can be users or developers. They can be professional or hobbyists. But as they give a maximum of feed-back, or as they develop, modify, correct and improve the releases, they all work to a much better quality of the products. And thus logically the open source software tend to get a bigger and bigger place on the market. According to this NCC survey, 73% of the IT manager expect to see the open source to develop itself within their company within the next 5 years and over 50% of them adopted or are planning to adopt Open Source [NCC05].

And when we consider the weight and the success of the Open source community in the software world and the growing importance of software in the handheld devices market, it would have been quite surprising that the first ones wouldn't try to invest the software market for mobile device. Open VG, which is the main subject of this master thesis, is a solution for 2 dimensions vector graphics for handheld devices and has been validated by the Open Source Initiative (OSI). But before to present Open VG, we are going to understand the importance of graphics in mobile devices today.

1.2 Graphics in handheld devices

As we explained it sooner, the very fast progression of bit rates capabilities for the communications coupled with miniaturisation created new possibilities for mobile phones and handheld devices. If we just focus on mobile phones, in 15 years we went from the bi-bop to the third generation with all the applications we now know for them. And when we just consider these innovations , the need for graphics in handheld devices appears as obvious.

At first, mobile devices such as mp3 players, mobile phones, PDAs, or GPS were running graphically very poor operating systems. They were very often monochrome and the resolution was quite bad since it was very simple. Then appeared the coloured screens and some more and more friendly operating systems appeared. And in terms of operating systems, the more friendly it is, the more it requires graphical performances. Some phones, called Smartphones, now include both a cell phone and a PDA.

Figure 1 Screen of an old cell-phone

Figure 2 Modern mobile phone screen

The internet appeared on mobile phones for the first time with the WAP in 2000. Now other internet solutions exist like i-mod or WAP 2.0. Nowadays, internet is available on all the mobile phones. It now allows users to send emails, visit some specially designed pages. Cameras are now also very popular and common on mobile phones. 2 on 3 sold mobile phones now includes cameras. This means that the screens have to be able to display the pictures. Some mobile devices can even shoot movies in a good resolution. And thus the screen should also be able to display such movies. Some devices like the new iPods have a large screen which is especially dedicated to displaying movies or movie clips. In this case, despite the small size of the screen, the definition has to be perfect. Still considering the mobile phones, they are all now provided with games which now need some good graphical performances. And for some of them, they are also designed for supporting online gaming.

We shall not forget a recently born but exploding market which is GPS market. These devices need very good graphical performances since they need to be able to display maps in three dimensions, to zoom on them, to perform vertical rotations and to display text dynamically and thus with a good speed otherwise it's too late for the user. And when it comes to the importance of graphics, the success of portable game consoles like the Nintendo GameBoy or the Sega GameGear which were among the first released or Sony's PSP are almost exclusively based on graphic performances.

Figure 1 Sony's PSP

This trend will globally accelerate and in the very close future, starting with the real third generation of mobile phone, we will be able to hold video and audio conferences on phone. We will watch TV or movies on our cell phones. And the already proven drop of the prices of communications and services will increase the use of these offers.

Some might say that adapting the computer's graphics technology to smalls screens is easy and shouldn't make any problems. Actually, even if both kind of graphics have some common basic principles, graphics for handheld devices have also very special needs. Of course, the screens are much smaller, so you have to adapt the applications. Another “obvious” point is that the material which is in charge the graphics, whether it is managed by the CPU or by another chip, has to fit in the device. And as a consequence of the miniaturisation of the products especially the mobile phones, the space and volume of the chips have to be really reduced. If we compare the chips in charge of managing the graphics in handheld devices and in the computers, we can easily notice that it is really hard to adapt a cooling system in a handheld device. But the good point is, although graphics are important for handheld devices, it is not as determinant as for a computer. So you must not expect the same level of performance especially when you keep in mind all the constraints. And last but not least, handheld devices imply power supply autonomy. So the power consumption has to be drastically reduced. Indeed, a graphically very performant handheld device with a battery autonomy of 2 minutes is worthless. All these characteristics make that graphics for handheld devices are really apart and thus it needs its own dedicated technology.

1.3 Introducing Open VG

1.3.1 Background

Because graphics are now managed in most of the cases by chips called video card or graphic cards, the application developers needed some kind of bridge between the hardware language and their applications. That is why APIs have been developed. APIs are libraries of functions that can be directly used by the developers. And the firms which produce these graphic or video cards develop and propose the drivers which are making the link between the APIs and the hardware. These APIs are or have been the most used APIs. Many of them were designed for dimensions graphic programming.

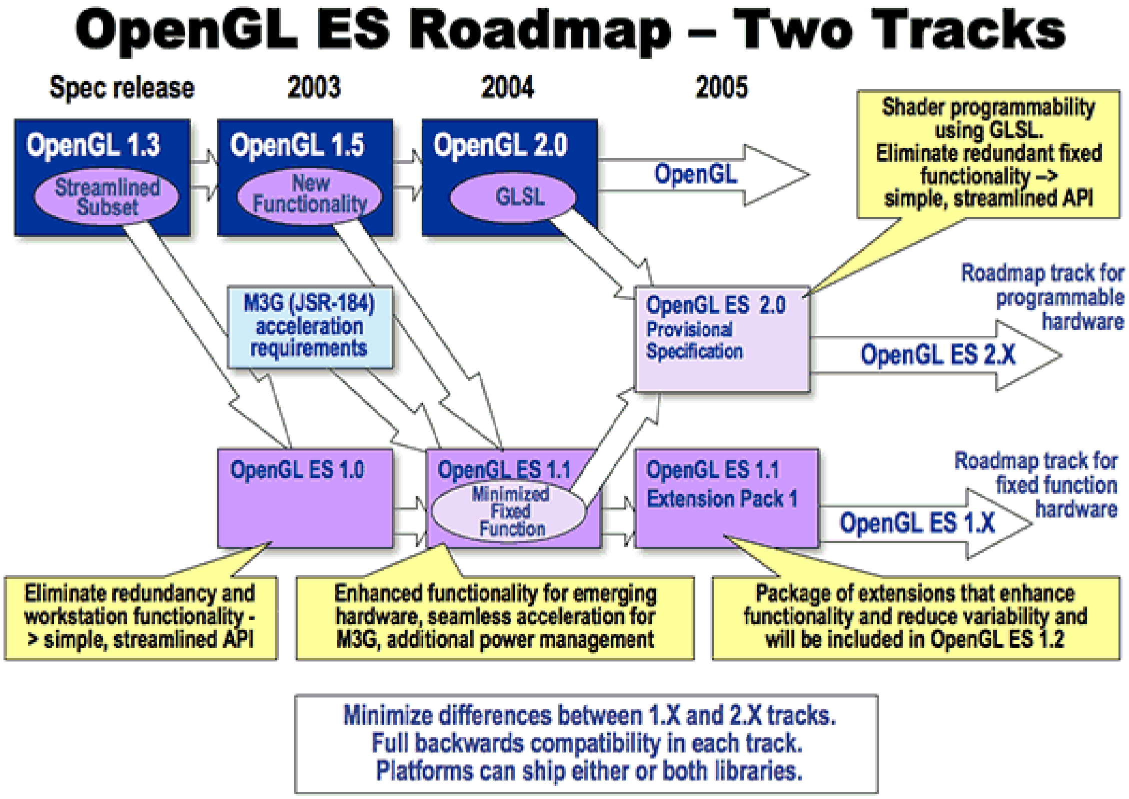
Direct X was developed by Microsoft. At first it was some kind of patch for windows. Direct X allowed developers to run their applications, in most of the case a game, in pure DOS mode. This mode, unlike the Windows mode, was offering numerous interesting and modern possibilities. To sum up, Direct X was allowing a good communication with the hardware layer. But they didn't get any success with it. The third version met a relative success with the game Need For Speed. But the industry chose to use OpenGL and that until Microsoft released Direct X 9, which is the ninth version of the API. Then it met a real success against OpenGL. [NIL06]

Glide was created in the 1990's by 3Dfx as an API that could be able to work on their famous video card Voodoo. Glide was functioning directly on top of the hardware. Glide met the success but because of the impossible renewment of Glide, 3Dfx closed. [NIL06]

SGI started to develop **OpenGL** in the beginning of the 1990's. In that moment, the aim was to develop an API for their line of video cards called Iris. And thus the API was called IrisGL. IrisGL became OpenGL as SGI wanted to publish a new and modified version of IrisGL to comfort the leader status of IrisGL on the market. From the early beginning, OpenGL met success because of its good matching with the hardware proposed in the market. It also received the support of the developers as they understood its functionment quite easily and the API works quite well. The problems with OpenGL are that it is hard to extend it and that it has a lot of useless features. In 1992, a group of actors around OpenGL created the Architectural Review Board (ARB) in order to pilot the development of OpenGL. From this cooperation, five new versions of OpenGL have been released. The latest version of OpenGL, OpenGL 2.0, proposes all the top features that puts the API at the top of the market. [NIL06]

OpenGL ES was created to target the market of handheld devices. It just kept the useful features of OpenGL. Just like OpenGL, OpenGL ES is platform independent. So when we also consider the fact that the developers who have already worked with OpenGL learn very quickly how to use the API, we can easily understand why OpenGL ES became quite fast very popular. [NIL06]

OpenGL ES also presents the good advantage of being royalty-free. It accommodates perfectly the different architectures and lowers the power consumption. It is extensible and evolving easily. That makes the API will have a long-life. Several versions have been implemented, 3 more precisely: OpenGL ES 1.X, OpenGL ES 2.X and OpenGL ES SC. This SC version is dedicated to safety critical industry like avionics or automotive.



OpenGL ES Roadmap – Two Tracks

Spec release

2003

2004

2005

OpenGL 1.3

Streamlined Subset

OpenGL 1.5

New Functionality

OpenGL 2.0

GLSL

OpenGL

Shader programmability using GLSL. Eliminate redundant fixed functionality -> simple, streamlined API

M3G (JSR-184) acceleration requirements

OpenGL ES 2.0 Provisional Specification

Roadmap track for programmable hardware

OpenGL ES 2.X

OpenGL ES 1.0

OpenGL ES 1.1

Minimized Fixed Function

OpenGL ES 1.1 Extension Pack 1

Roadmap track for fixed function hardware

OpenGL ES 1.X

Eliminate redundancy and workstation functionality -> simple, streamlined API

Enhanced functionality for emerging hardware, seamless acceleration for M3G, additional power management

Package of extensions that enhance functionality and reduce variability and will be included in OpenGL ES 1.2

Minimize differences between 1.X and 2.X tracks. Full backwards compatibility in each track. Platforms can ship either or both libraries.

Figure1 5 Versioning of OpenGL ES

As we can see it on the diagram [KHRGLES06], OpenGL ES is directly derived from OpenGL and the normal versions differ from each other by the kind of hardware they target. Indeed, OpenGL ES 2.X targets programmable hardware while OpenGL ES 1.X for fixed function hardware. To be platform independent, OpenGL ES needs to include a native platform graphics interface layer called EGL. This layer is responsible for ensuring that OpenGL ES is platform independent. So there is as many EGL versions as chips or chip manufacturers.

1.3.2 The need for OpenVG

Before defining the need for OpenVG, it might b necessary to shortly remind the reader the difference between **"normal" (or rasterized) graphics** and **vector graphics**. In **rasterized graphics**, the picture is seen and manipulated as a collection of pixels, which is quite heavy to compute and to store. And in the case of a zoom, some quality is lost. With **vector graphics**, the picture is computed and stored as a set of geometrical objects like lines, curves, points. Thus, in the case of vector graphics, it can handle more easily transformations without loosing quality [WIKVEC06]. On the following figure [ATK06] , we can observe a picture (a). When we zoom, we will observe a great loss with rasterized graphics (c) since the picture is stored as pixels and thus we zoom on the pixels. Instead of this, when we zoom on the same picture but with vector graphics (b), we get a good result since we just have to redraw the shapes from the equations we have of them. And thus the quality is preserved.

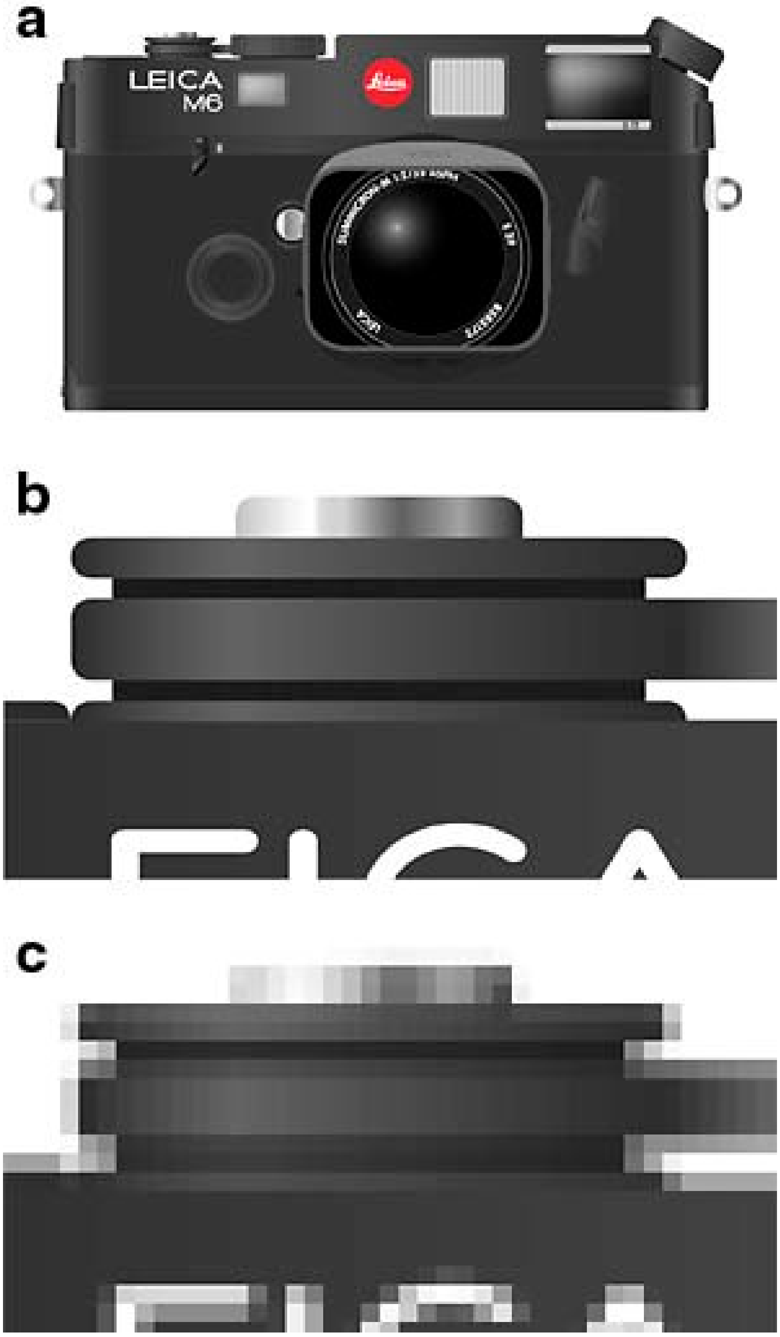


Figure 6 Difference between vector graphics and rasterized graphics

Khronos is the group of interest which created and developed OpenGL ES. After the success of OpenGL ES, they felt that there was a need from both market and developers for a vector graphics API for the small screen devices. Just like OpenGL ES, OpenVG is royalty-free and is an open standard API. OpenVG provides 2 dimensions vector graphics for handheld devices. Actually, those devices had a real need for smooth and fluidly scalable 2 dimensions graphics. It allows some great benefits like a very low power consumption (90% less than for software accelerated graphics). It makes the transition from software to hardware very soft. It provides a very smooth scalability and it has been designed to accelerate existing formats such as Flash, SVG, PDF, postscript, Vector fronts and so on [KHRVG06].

1.4 SART Project

1.4.1 Presentation

The SART is a research project. It aims to develop and exploit the interdisciplinary existing between art and software development in order to develop and enhance creativity and innovation in software development. The SART project focuses more particularly on the open source software and more especially on the communities of interest that surrounds the open source software. It relies on the open source community and on the cooperation between firms and researchers [JACC06]. The different goals of the project are as follows.

G1. Develop empirically based theories, models, and tools to support innovation in software
technology development.

G2. Develop knowledge on the interdisciplinary nature of software production in which the
software engineering process interact with artistic process and the target systems are heterogeneous embedded devices.

G3. Educate competent practitioners and researchers in the interdisciplinary field of software
innovation and transfer of knowledge to the software industry and artist communities.

G4. Publish in the leading scientific journals and conferences.

1.4.2 Actors

Maria Letizia Jaccheri, NTNU professor, PhD, will be the administrative leader of the project. Through her numerous contacts in both artistic and software development research communities, she will be the link between administration, the actors of the project and both communities. And she is already involved in Experts in Team, which is an educative program at NTNU aiming to develop interdisciplinarity. She is in especially in charge of the Art and Software village.

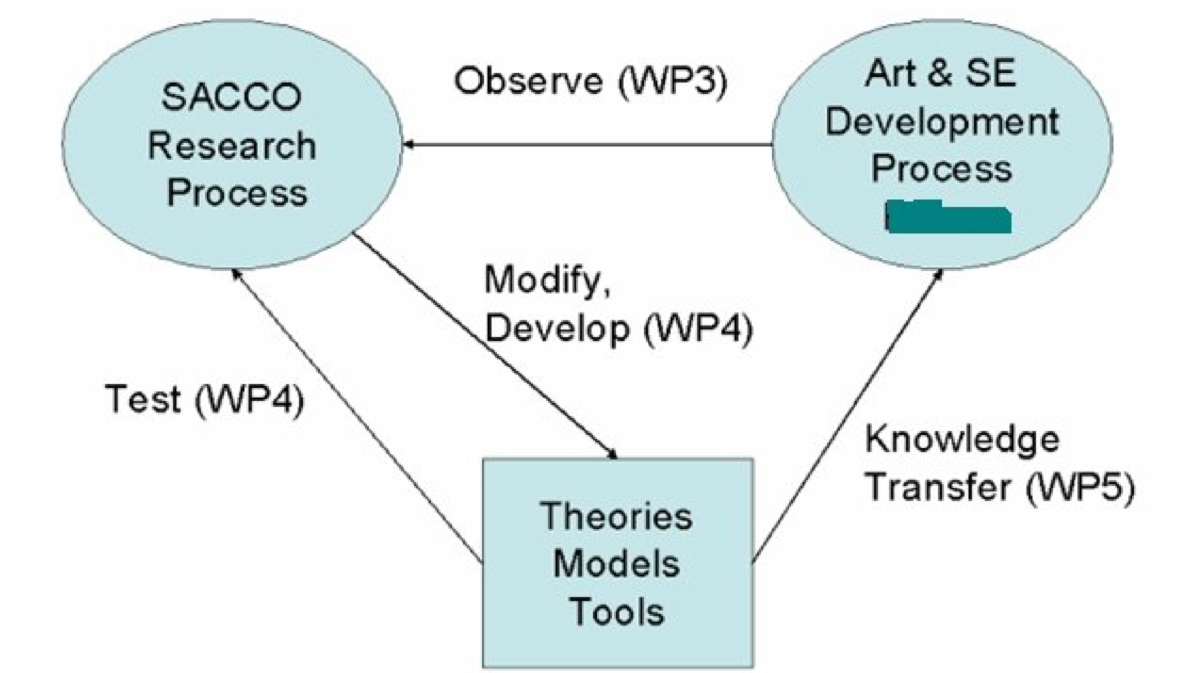
Judith Molka-Danielsen, HiM Professor, PhD, will be the research advisor and academic leader for the research group at HiM.

COMPANYX, based in TOWNT, registered business in 2001 is a firm designing graphic chips and middleware for handheld devices. COMPANYX has 20 employees in TOWNT and 5 more at its sales office in TOWNY, USA. MrZ will be the contact at COMPANYX.

The research community will also be completed by 2 PhD students and by an international pool of researchers.

1.4.3 Organization

The figure 1 above shows us how communications inside the project group work. The first step goes from the firm, COMPANYX, which submit its observations to the SART Research Process, which is actually the researchers. Then they modify and develop the theories and models they developed. That's the second step. Then these models and theories are tested and modified again following the results of the tests. That's the third step. And when the models are ready and finished, there is a transfer of knowledge to the firm. That is the last step.



1.5 Aims and challenges

This project can actually be divided in 2 parts. The first part concerns benchmarking OpenVG. Indeed, a big part of our time will be spent in studying OpenVG and benchmarking in order to develop a benchmarking software for OpenVG. It should be able to benchmark both hardware and software implementations of OpenVG. This work task is actually the subject of the internship I did at COMPANYX from the first of august 2006 to the 28th of february 2007. Although OpenVG is open source, there exists only one benchmarking software for OpenVG and it is very expensive. Thus, this benchmarking software would allow COMPANYX to save money and it would also be freely distributed on internet. So it would respect the open source character of OpenVG and would contribute to make the community richer. One big challenge is that both graphics, and thus vector graphics and OpenVG, and benchmarking are new topics for me. So it is a challenge for me and in the same time a wonderful opportunity to discover new fields. The other challenge for me is to manage both the internship and the master thesis related to this project.

The master thesis, which is the other part of the project, actually concerns the SACCO project which we detailed in the last sub-chapter. The goals for this part are numerous. At first, it would be interesting to detect and highlight innovation in OpenVG. We also thought about detecting artistic opportunities in OpenVG. And last, we would like to investigate on the community of interest gathered around OpenVG in order to « audit » artists working with or for OpenVG and artistic applications for OpenVG. And then we would like to know what can be done in order to develop the cooperation between OpenVG developers and artists.

1.6 Summary

Handheld devices are now more and more part of our lives. As our need of them increases, so do the applications. And with the applications, increase the graphics needs. Since the PC graphics are not adapted to handheld devices, some APIs have been dedicated to these applications. OpenVG is one of them. It is a 2D vector graphics API. This projects consists in implementing a benchmarking software after having studied OpenVG and also to exploit OpenVG possibilities within the SACCO project. This project is a research projet aiming at developing the cooperation between artists and software developers exploiting the communities of interest gathered around open source software. And that is the case for OpenVG. In a first time, we will propose an overview of OpenVG. Then we will explore the innovation and artistic possibilities of OpenVG. The third chapter of this project will be dedicated to benchmarking. A part in which we will discuss and detail our choices for benchmarking OpenVG will directly follow. The next part will be logicly fully dedicated to the implementation of this benchmarking software. And finally, we will try to test as mucg as possible the benchmark we have developed.

2 OpenVG

2.1 Introduction

Since graphics are now a key in the battle of handheld devices. Efforts have been done to propose some dedicated graphical solutions for small screens. Developing 2 or 3 dimensions graphical APIs for both hardware and sotfware accelerated graphics.

As we explained in the sub-chapter 1.3, OpenVG is an open source and royalty-free graphic API developed by Khronos. It is dedicated to handheld devices and allows developers to develop 2 dimensions vector graphics for small screens. The benefits are like to provide the possibility to create fine user interfaces. It also permits to display text in a very readable way on small screens especially with special characters like Chinese, Japanese, or Arab. OpenVG allows smooth transitions. And we shouldn't mention that the API has been developed in order to enable formats like SVG, Flash, PDF or Vector Fonts. There are also many other benefits. OpenVG can help developers to reduce the power consumption up to 90%. OpenVG is quite easily scalable since it allows high-quality rendering with anti-aliasing and it is platform independent. The main targets are PDAs and mobile phones. But not only, of course SVG viewers are accessible but it allows also mapping applications like GPS since vector graphics permit dynamic placement and zooming without loosing quality. It also can display all kind of writings and thus is perfect for E-book readers.

In this chapter, we want to give an overview of OpenVG. At fist it would be interesting to give some basic definitions before explaining the basic scheme of functioning. Then we would give a very short but needed presentation of EGL and its role in OpenVG. After that we will go a bit deeper and propose a clear description of the possibilities of OpenVG like the path, the paint and images. And we will finish with some more advanced features to complete the description. Maybe I should apologize in advance to the reader because of the very broad character of this chapter but it is the fruit of the research and documentation I had to do because of my ignorance in this subject as I started to work on it.

2.2 Basics

2.2.1 Notions and definitions

In this sub-chapter, we would like to develop some basic notions and eventually give some definitions. As we explained, the notions we are going to develop may be familiar for the reader but it might also be necessary to go through to clear some doubts and to explain the point of view of OpenVG on these points.

The **path** is a shape made of lines, curves and other mathematical objects. It can be closed or open. It has a starting point and an ending point. As the path is currently been built, it also has a last segment point, which is the last point of the last built segment. There can be several paths in a same picture. A path can be **filled**, **stroked** or both. **Stroked** means that the shape will be given a thickness, which has to be defined. **Filled** means that the part of the area of the picture which will be inside the path, will be painted. The developer has also the possibility to set implicit closures in the case of a filled path. The end of a stroked path is called cap. Its style can be chosen within several different **cap styles**. It is also possible to configure the line join style from a pool of choices and to decide the length. Stroked path can also be **dashed** by setting different parameters.

Basically, **colours** can be represented with several different ways. There are the colour spaces **lRGB**(linear Red Green Blue) and **sRGB** which is non linear. There also exist several greyscale colour spaces (linear and non-linear). The linear colour spaces offer the great advantage of being easy to compute but non-linear colour spaces are much better adapted to LCDs and CRTs. The linear colour space, called lRGB, is defined from a three dimensions space following the standard CIE XYZ which is:

$R = 3{,}240479\ X - 1{,}537150\ Y - 0{,}498535$

$G = -0{,}969256\ X + 1{,}875992\ Y + 0{,}041556$

$B = 0{,}055648\ X - 0{,}204043\ Y + 1{,}057311\ Z$

To convert this colour space into the non-linear colour space, called sRGB, we need the **Gamma** function. It is defined as follows [RIC05]:

If $x \leq 0{,}00304$

$\gamma(x) = 12{,}92\ x$

else

$\gamma(x) = 1{,}0556\ x^\wedge(1/2{,}4) - 0{,}00556$

And thus, we have:

$R' = \gamma(R)$
$G' = \gamma(G)$
$B' = \gamma(B)$

The linear greyscale space is defined as follows [RIC05]:

$L = 0,2126\,R + 0,7152 + 0,0722\,B$
$R = G = B = L$

And like for the RGB space, to convert the linear system in the non-linear system, we have to use the gamma function:

$L' = \gamma(L)$

For the colour spaces, there exists another space. It consists in multiplying every colour component by a float alpha which becomes the fourth component. It is called premultiplied alpha. In a concrete manner, OpenVG uses the non-linear colour space and can convert pixel formats with premultiplied alpha in the sRGB format, that is to say ($R' = \gamma(R)$, $G' = \gamma(G)$, $B' = \gamma(B)$).

2.2.2 The pipeline

OpenVG is said to work as a pipeline of 8 different stages. These stages are in the order: “path, transformation, stroke and paint”, stroked path generation, transformation, rasterization, clipping and masking, paint generation, image interpolation and blending, and antialiasing. This pipeline is actually ideal and tends to be a proposition for implementers of OpenVG. The figure 1 [RIC05] represents this pipeline.

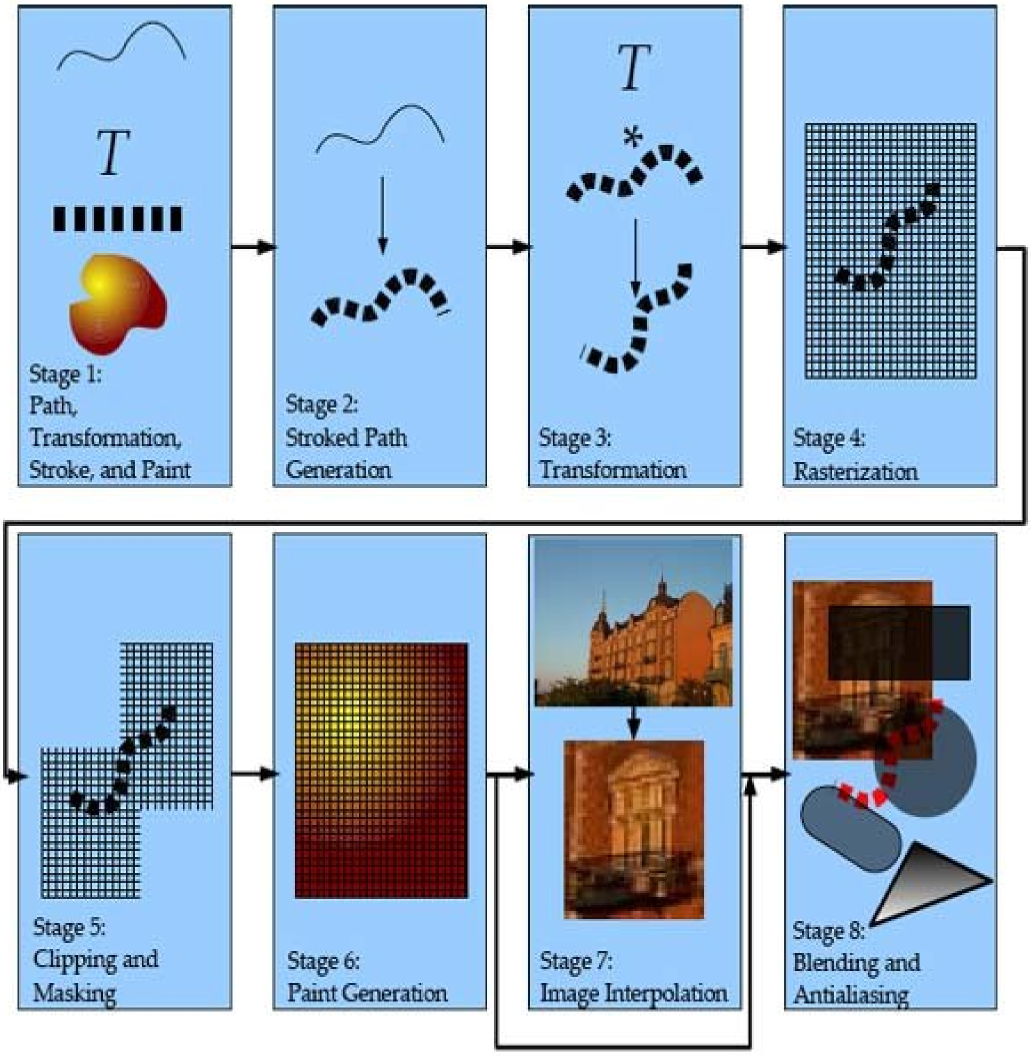


Figure 7 The OpenVG pipeline

The **first stage**, called **path, transformation, stroke and paint**, consists in setting all the parameters for the path, for any transformation, for an eventual stroke and for the paint. Then the rendering process starts with a call to **vgDrawPath**.

The **second stage,** called **stroked path generation**, consists in modifying and mapping the stroked path in the coordinate system.

The **third stage**, called **transformation**, consists in applying the transformation set to the current path and then it generates some new surface coordinates.

The **fourth stage,** called **rasterization**, consists in mapping the shapes on the pixel grid using some filtering rules. Actually the technique is to attribute a value after sampling the pixels nearby the path.

The **fifth stage**, called **clipping and masking**, consists in to give every pixel a coverage value. When the pixel isn't within the bounds of the drawing area, then it receives the value 0. The aim is to avoid computing element of the picture that will not be displayed. It allows us to save computing resource. Then for every pixel, the coverage value will be replaced by the product of the alpha mask value for this pixel and by the previous coverage value given.

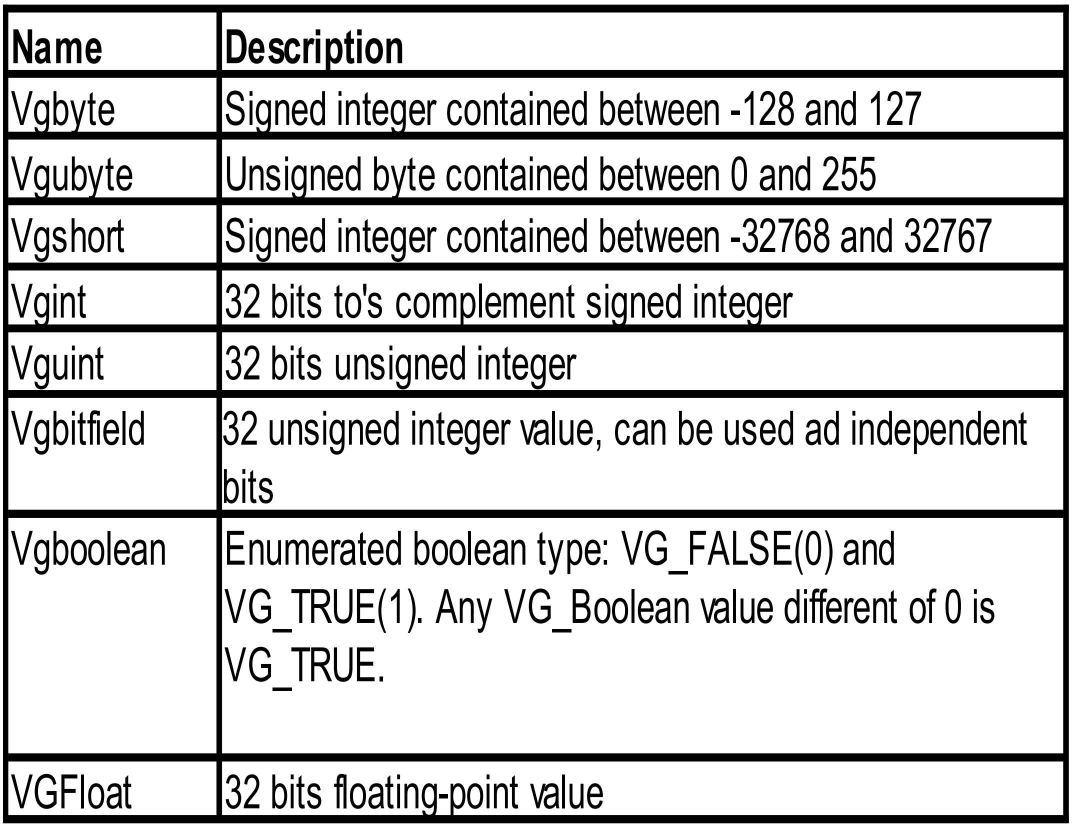
In the next stage, the **sixth** one and simply called **paint**, the aim is to generate the colours and alpha value from all the parameters from the paint.

The **seventh stage,** called **image interpolation**, consists in interpolating color values and alpha values for every pixel of the image being drawn. These values, which will be used in the next stage, are generated by using the inverted process of the one used in the previous stage.

And in the last stage, called **blending and antialiasing**, the previously computed values are used to be blended with the corresponding values using a blending rule depending on whether the image drawing mode is “stencil” or not.

Types

Here we will just explicit the types defined for OpenVG [RIC05].



Name	Description
Vgbyte	Signed integer contained between -128 and 127
Vgubyte	Unsigned byte contained between 0 and 255
Vgshort	Signed integer contained between -32768 and 32767
Vgint	32 bits to's complement signed integer
Vguint	32 bits unsigned integer
Vgbitfield	32 unsigned integer value, can be used ad independent bits
Vgboolean	Enumerated boolean type: VG_FALSE(0) and VG_TRUE(1). Any VG_Boolean value different of 0 is VG_TRUE.
VGFloat	32 bits floating-point value

Table 8 OpenVG types

2.3 EGL

This subchapter is willing to explain and develop the role of EGL, the way it works and how to use it since developers have to manipulate it when they develop with OpenVG. As we explained it earlier, EGL is a native platform graphic platform interface. It has been designed by Khronos in order to provide graphical APIs with means to manage hardware. Thus that makes the graphical APIs platform independent. The figure 1 shows the place and role of EGL in the layers stack. Still on the figure1, we can see that EGL is actually divided in two sides. OpenGL ES uses this separation but OpenVG doesn't so we won't spend that much time on this detail.

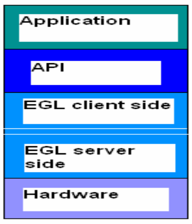


Figure 9: EGL's role in the architecture

The figure 2 shows the global functioning of EGL. EGL's aim is to provide the API with a mean to create **rendering contexts**, **drawing surfaces**. We can see on the figure 2 that the EGL layer is actually subordinated to the different APIs. Indeed, these APIs call some EGL functions to manage the transition to hardware that is to say to handle the contexts and the drawing surfaces.

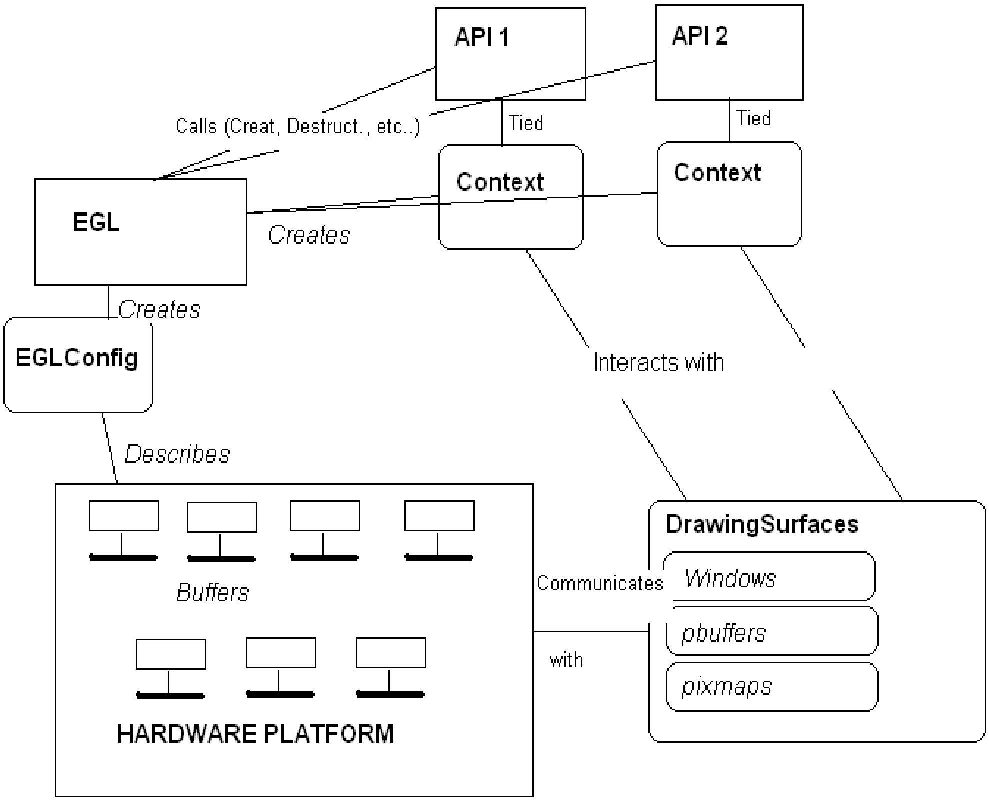


Figure 10 EGL functioning scheme

But before to expose these functions, it might be better to explain what is a rendering context and a drawing surface. A **rendering context** is some kind of state machine run by the client API [LEE05]. Its role is to maintain the API state [RIC05]. These states are listed in the table 1 [RIC05]. Each rendering thread can have several rendering APIs but only one context per rendering API but only one context per thread can be considered as **current context**.

State	Description
Drawing Surface	Surface for drawing
Matrix mode	Transformation to be manipulated
Path user-to-surface transformation	Affine transformation for filled and stroked geometry
Image user-to-surface transformation	Affine or projective transformation for images
Paint-to-user transformation	Affine transformation for paint applied to geometry
Fill rule	Rule for filling paths
Quality Settings	Image and rendering quality, pixel layout
Blend Mode	Pixel blend function
Image Mode	Image/paint combination function
Scissoring	Current scissoring rectangles and enable/disable
Stroke	Stroke parameters
Tile fill color	Color for FILL tiling mode
Clear color	Color for fast clear
Filter parameters	Image filtering parameters
Paint	Paint definitions
Mask	Alpha stencil mask and enable/disable
Error	Oldest unreported error

Table 1 State elements of a context

A **drawing surface** is, as the name says, a surface defined for some APIs to draw on. Three types of drawing surface exist: **windows, pbuffers, pixmaps**. Windows is for onscreen rendering, pbuffers are for off-screen rendering and pixmaps are used for off-screen rendering to buffers which can be accessed by native APIs. Windows and pixmaps are fully depending on the system's windows and pixmaps [LEE05].

These drawing surfaces are created following the hardware platform description which is available in **EGLConfig.** Indeed, EGLConfig offers the depth of the colour buffer and types and number of the ancillary buffers [LEE05].

EGL supports two rendering modes, that is to say 2 ways to operate the way the graphics are rendered: **back buffered** and **single buffered**. In the first one, **back buffer**, the colour buffer is allocated and owned by EGL. And when the frame is finished to be drawn, the back buffer has to be copied to the drawing surface, the native window actually. Indeed, back buffered is only supported by the window and pbuffer drawing surface. And since pbuffer surfaces don't have any windows, the buffer isn't copied. Pixmap surfaces use the second one, **single buffer.** In this rendering mode, the colour buffer with the form of a pixmap is allocated in memory when the drawing surface is created. This buffer will be used by the rendering API and at the end of the rendering process, the buffer will contain the final image [LEE05].

Now that we explained the basic notions, we can propose a set of functions which have to be called from the API. They are of course platform independent. These functions help the developer to manage drawing contexts and drawing surfaces. At first, we need to bind an API to EGL as the current API. To do so, we have to call **eglBindAPI:**

```
EGLBoolean eglBindAPI (EGLenum api);
```

Of course, *EGLenum api* is an API. So to set OpenVG as the API, the parameter is EGL_OPENVG_API. Then we are ready to create the context by calling **eglCreateContext**.

```
EGLContext  eglCreateContext(EGLDisplay dpy,
                             EGLConfig config,
                             EGLContext share_context,
                             const EGLint * attrib_list);
```

EGLDisplay is a type of EGL. It represents a physical display on which EGL can draw. In most of the cases, it is a single screen [LEE05]. EGLContext share_context is a context already existing that can be passed as an argument to share this context. In case of there is no sharing context, the value is EGL_NO_CONTEXT. Then we can create the drawing surface. If we like to create a window surface, then we will call **eglCreateWindowSurface**.

```
EGLSurface  eglCreateWindowSurface(EGLDisplay dpy,
                                   EGLConfig config,
                                   NativeWindowType win,
                                   const EGLint *attrib_list);
```

As we explained it sooner, here we can use two rendering modes: back buffer and single buffer. This has to be precised in the attrib_list with the argument EGL_SINGLE_BUFFER for single buffer model. If we like to create a pbuffer surface, then we have to call **eglCreatePbufferFromClientBuffer**. The syntax is as follows:

```
EGLSurface  eglCreatePbufferFromClientBuffer(EGLDisplay dpy,
                                             EGLenum buftype,
                                             EGLClientBuffer buffer,
                                             EGLConfig config,
                                             const EGLint *attrib_list);
```

If we like to draw on an image (VGImage), then we just have to call it with EGL_OPENVG_IMAGE as buftype and the image has to be cast as EGLClientBuffer and passed as the buffer. To change the current context, we can use **eglMakeCurrent**.

```
EGLBoolean eglmakeCurrent(EGLDisplay dpy,
                          EGLSurface draw,
                          EGLSurface read,
                         EGLContext ctx);
```

And to get the current context, we will use **eglGetCurrentContext** as follow:

```
EGLContext  eglGetCurrentContext();
```

In the same logic sense, to destroy a context, we will use **eglDestroyContext:**

```
EGLBoolean eglDestroyContext(EGLDisplay dpy, EGLContext context);
```

When rendering is in back buffered mode, there comes a time when we have to copy the back buffer into the visible one, then we have to use **eglSwapBuffers**.

```
EGLBoolean eglSwapBuffers(EGLDisplay dpy, EGLSurface surface);
```

With these few function calls in mind, the developer can start to develop with OpenVG. That is the goal of the next sub-chapter in which we will consider the path, the paint and the images and we will develop and illustrate all the possibilities of OpenVG for each of these components.

2.4 Generalities

2.4.1 Parameters

In this sub-chapter we want to propose a way to understand the general principle for handling OpenVG from the point of view of the developer. We need to explain the general principle for setting and getting the parameters. We will also spend some time to explain how transformations are handled in OpenVG.

The idea is quite simple. There is a parameter type which is an enumeration.

```
typedef enum {

/* Mode settings */
```

```
VG_MATRIX_MODE                          = 0x1100,
VG_FILL_RULE                            = 0x1101,
VG_IMAGE_QUALITY                        = 0x1102,
VG_RENDERING_QUALITY                    = 0x1103,
VG_BLEND_MODE                           = 0x1104,
VG_IMAGE_MODE                           = 0x1105,

/* Scissoring rectangles */
VG_SCISSOR_RECTS                        = 0x1106,

/* Stroke parameters */
VG_STROKE_LINE_WIDTH                    = 0x1110,
VG_STROKE_CAP_STYLE                     = 0x1111,
VG_STROKE_JOIN_STYLE                    = 0x1112,
VG_STROKE_MITER_LIMIT                   = 0x1113,
VG_STROKE_DASH_PATTERN                  = 0x1114,
VG_STROKE_DASH_PHASE                    = 0x1115,

/* Edge fill color for VG_TILE_FILL tiling mode */
VG_TILE_FILL_COLOR                      = 0x1120,

/* Color for vgClear */
VG_CLEAR_COLOR                          = 0x1121,

/* Enable/disable alpha masking and scissoring */
VG_MASKING                              = 0x1130,
VG_SCISSORING                           = 0x1131,

/* Pixel layout hint information */
VG_PIXEL_LAYOUT                         = 0x1140,

/* Source format selection for image filters */
VG_FILTER_FORMAT_LINEAR                 = 0x1150,
VG_FILTER_FORMAT_PREMULTIPLIED          = 0x1151,

/* Destination write enable mask for image filters */
VG_FILTER_CHANNEL_MASK                  = 0x1152,


/* Implementation limits (read-only) */
VG_MAX_SCISSOR_RECTS                    = 0x1160,
VG_MAX_DASH_COUNT                       = 0x1161,
VG_MAX_KERNEL_SIZE                      = 0x1162,
VG_MAX_SEPARABLE_KERNEL_SIZE            = 0x1163,
VG_MAX_COLOR_RAMP_STOPS                 = 0x1164,
VG_MAX_IMAGE_WIDTH                      = 0x1165,
VG_MAX_IMAGE_HEIGHT                     = 0x1166,
VG_MAX_IMAGE_PIXELS                     = 0x1167,
VG_MAX_IMAGE_BYTES                      = 0x1168,
VG_MAX_FLOAT                            = 0x1169
```

```
} VGParamType;
```

These context parameters are set using the 4 different versions of the vgSet function. There are versions for floats, integers, arrays of floats and arrays of integers. The two last functions need the size of the array value.

```
void vgSetf (VGParamType paramType, VGfloat value)
void vgSeti (VGParamType paramType, VGint value)

void vgSetfv(VGParamType paramType, VGint count,
                                    const VGfloat * values)
void vgSetiv(VGParamType paramType, VGint count,
                                    const VGint * values)
```

On the same principle, all the parameters can be accessed by a generic vgGet function and another vgGetVectorSize gives us the size of any vector.

```
VGfloat vgGetf (VGParamType paramType)
VGint vgGeti (VGParamType paramType)

VGint vgGetVectorSize(VGParamType paramType)

void vgGetfv(VGParamType paramType, VGint count, VGfloat * values)
void vgGetiv(VGParamType paramType, VGint count, VGint * values)
```

The same kind of generic functions also exist for object parameters. That is to say objects that need handles to be manipulated. The setting functions are vgSetParameter.

```
void vgSetParameterf (VGHandle object, VGint paramType, VGfloat value)
void vgSetParameteri (VGHandle object, VGint paramType, VGint value)
void vgSetParameterfv(VGHandle object, VGint paramType, VGint count, const VGfloat *
values)
void vgSetParameteriv(VGHandle object, VGint paramType,VGint count, const VGint *
values)
```

Of course the analogue vgGetParameter and vgGetParameterVectorSize functions exist.

```
VGfloat vgGetParameterf (VGHandle object, VGint paramType)
VGint vgGetParameteri (VGHandle object, VGint paramType)
VGint vgGetParameterVectorSize (VGHandle object, VGint paramType)
```

```
void vgGetParameterfv(VGHandle object, VGint paramType, VGint count, VGfloat * values)
void vgGetParameteriv(VGHandle object, VGint paramType, VGint count, VGint * values)
```

2.4.2 Transformations

As we explained in the sub-chapter 2.2 Basics, in the first and second stage of the pipeline, we set and then apply the transformation. Actually there are 4 different transformations used in OpenVG. They are defined in VGMatrixMode typedef enumeration.

```
typedef enum {
VG_MATRIX_PATH_USER_TO_SURFACE = 0x1400,
VG_MATRIX_IMAGE_USER_TO_SURFACE = 0x1401,
VG_MATRIX_FILL_PAINT_TO_USER = 0x1402,
VG_MATRIX_STROKE_PAINT_TO_USER = 0x1403
} VGMatrixMode;
```

The first mode is used for mapping a path from the user coordinates to the surface coordinates. The second one does the same but for an image and the two last ones concern the transformations used to map paints to fill paints and to stroked paths paints. The default value for VG_MATRIX_MODE is *VG_MATRIX_PATH_USER_TO_SURFACE* , but to change it we need to use the vgSetifunction.

These matrix modes actually use a matrix for operating the transformation. Each mode has its own matrix which can be modified. Each point is represented as a three floats array [x y 1] in which (x,y) are the coordinates of the point in the user's system coordinates. Thus, the matrix used is a 3*3 float matrix. It has the following form:

$$\begin{bmatrix} sx & shx & tx \\ shy & sy & ty \\ 0 & 0 & 1 \end{bmatrix}$$

In this matrix sx and sy represent the scaling parameters in the x and y directions, shx and shy represent the shearing parameters in the x and y directions and tx and ty are for translation. Several functions exist to modify this matrix.

vgLoadIdentity

It sets the current matrix to the identity matrix.

void vgLoadIdentity (void)

vgLoadMatrix

This function loads a matrix given as input in the current matrix.

*void vgLoadMatrix (const Vgfloat * m)*

vgGetMatrix

As the name says this function allows the user to retrieve the current matrix value.

*void vgGetMatrix (Vgfloat * m)*

vgMultMatrix

This function sets the current matrix as the product of the current matrix and the input matrix (right multiplication).

*void vgMultMatrix (const Vgfloat * m)*

vgTranslate

This function consists in appending a translation to the current transformation. It means that the current matrix is right-multiplied by the translation matrix made of the translation parameters tx and ty given as input. The multiplying matrix has the following form:

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

void vgTranslate (Vgfloat tx, Vgfloat ty)

vgScale

Based on the same principle as the previous function, this function appends a scaling transformation to the current one. The multiplying matrix has the following form:

$$\begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

void vgScale (VGfloat sx, Vgfloat sy)

vgShear

Based on the same principle as the previous function, this function appends a shearing transformation to the current one. The multiplying matrix has the following form:

$$\begin{bmatrix} 1 & shx & 0 \\ shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

void vgShear (VGfloat shx, Vgfloat shy)

vgRotate

Still on the same principle, vgRotate appends a clock-wise rotation. The angle a is given in degrees. The multiplying matrix is:

$$\begin{bmatrix} \cos(a) & -\sin(a) & 0 \\ \sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

void vgRotate(VGfloat angle)

2.5 Path

The path is one of the most essential object in OpenVG. As we explained it earlier, in a picture, there can be several paths. A path has a beginning and an end, which are both materialized by a point. It can contain moves, straight lines, arcs and all types of curves. It can be stroked, dashed or both and coloured and that with a whole set of finitions for the details. In this sub-sub-chapter, we want to present the whole set of commands, illustrate how to use them and the result that you can expect from them. For some obvious reasons we will not illustrate them all since the commands are very similar from one to the other and their use is quite intuitive and thus simple.

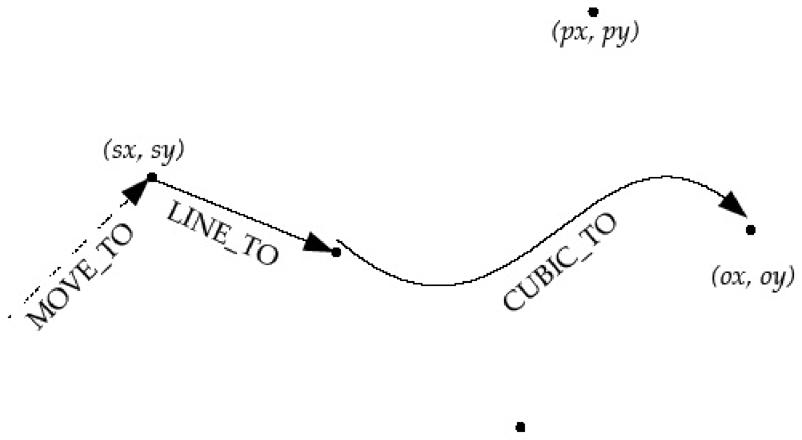


Figure 11: Segments and their reference points

At first, to draw a path, it is necessary to create it. Then we will be able to append path data to it. Indeed, from the data point of view, the path will be represented by two flows: one command flow, and one flow which is made of the parameters needed by the commands. As shown on figure 1 [RIC05], the result can be the lines, arcs , curves and so on. A path is divided in indexed segments. Each segment of a path is generated by a command and some parameters. Each time a segment is appended to the path, the command and its parameters are appended to their respective data flows. So a path can be seen as a succession of commands. These commands are passed as 8 bits integers. The bits 7 to 5 are reserved, the next 4 ones define the command and the bit 0 defines whether the parameters are relative (VG_RELATIVE = 1) or absolute (VG_ABSOLUTE = 0). So with a simple logic connector "|" it can be set. The table 1 lists all the path segment commands with their attributes and their description [RIC05]. For the needs of this table, three control points are defined. (sx,sy) represents the start of the current sub-path that is to say the first point after the last MOVE_TO segment. (ox,oy) is the last point of the previous segment and (px,py) is the last control point defined. And (xi,yi) are absolute coordinates that have to be added to (ox,oy). Ellipse parameters like (rx,ry) and rot are defined absolutely.

Type	Command	Coordinates	Value	Implicit points	Side effects
Close Path	CLOSE_PATH	None	0		(px,py)=(ox,oy)=(sx,sy) End current subpath
Move	MOVE_TO	x0,y0	1		(sx,sy)=(px,py)=(ox,oy)=(x0,y0) End current subpath
Line	LINE_TO	x0,y0	2		(px,py)=(ox,oy)=(x0,y0)
Horizontal Line	HLINE_TO	X0	3	y0=oy	(px,py)=(x0,oy) ox=x0
Vertical Line	VLINE_TO	Y0	4	x0=ox	(px,py)=(ox,y0) oy=y0
Quadratic	QUAD_TO	x0,y0,x1,y1	5		(px,py)=(x0,y0), (ox,oy)=(x1,y1)
Cubic	CUBIC_TO	x0,y0,x1,y1,x2,y2	6		(px,py)=(x1,y1), (ox,oy)=(x2,y2)
G1 Smooth Quad	SQUAD_TO	x1,y1	7	(x0,y0)=(2*ox-px,2*oy-py)	(px,py)=(2*ox-px, 2*oy-py) (ox,oy)=(x1,y1)
G1 Smooth Cubic	SCUBIC_TO	x1,y1,x2,y2	8	(x0,y0)=(2*ox-px,2*oy-py)	(px,py)=(x1,y1), (ox,oy)=(x2,y2)
Small CCW Arc	SCCWARC_TO	rx,ry,rot,x0,y0	9		(px,py)=(ox,oy)=(x0,y0)
Small CW Arc	SCWARC_TO	rx,ry,rot,x0,y0	10		(px,py)=(ox,oy)=(x0,y0)
Large CCW Arc	LCCWARC_TO	rx,ry,rot,x0,y0	11		(px,py)=(ox,oy)=(x0,y0)
Large CW Arc	LCWARC_TO	rx,ry,rot,x0,y0	12		(px,py)=(ox,oy)=(x0,y0)

Table 2 Client Side Path Segment

Now we can detail a bit these path segment commands. As the name says, CLOSE_PATH is for closing the path. MOVE_TO is for moving from one point to another without drawing. LINE_TO, HLINE_TO, VLINE_TO are very explicit too. They're used for drawing straight lines respectively horizontal or vertical. QUAD_TO is to draw Bezier curves of quadratic equations. Like follows:

$$x(t)= x0*(1-t)^2 + 2*x1*(1-t)*t + x2*t^2$$
$$x(t)= y0*(1-t)^2 + 2*y1*(1-t)*t + y2*t^2$$

The curve starts in (x0,y0) and stops in (x2,y2).

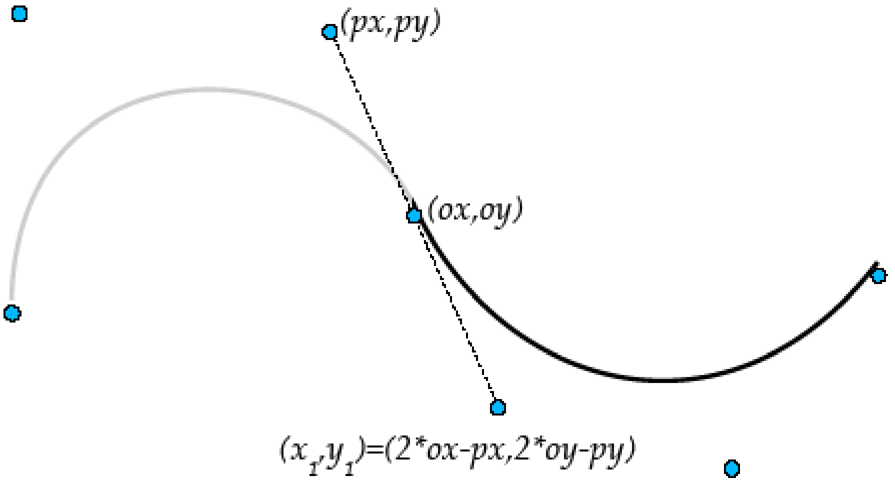


Figure 12 Smooth curve appending

CUBIC_TO is also to draw Bezier curves but with cubic equations as the one that follow:

$$x(t)= x0*(1-t)^3 + 3*x1*(1-t)^2*t + 3*x2*(1-t)*t^2 + x3*t^3$$
$$y(t)= y0*(1-t)^3 + 3*y1*(1-t)^2*t + 3*y2*(1-t)*t^2 + y3*t^3$$

The curve starts in (x0,y0) and stops in (x3,y3). (x1,y1) and (x2,y2) are control points defining tangents. SQUAD_TO and SCUBIC_TO are G1 smooth segment curves respectively quadratic and cubic curves. The particularity here is that the new segment will be append in a smooth style by defining tangents using (px,py) as shown on the figure 2 [RIC05].

The four last commands, SCCWARC_TO, SCWARC_TO, LCCWARC_TO, LCWARC_TO, are arcs defined on ellipses while considering different sections. The figure 3 [RIC05] shows which parts are considered in which cases. S stands for Small while L stands for Large, C stands for Clockwise while CC stands for Counter Clockwise as we can see it on the figure 3.

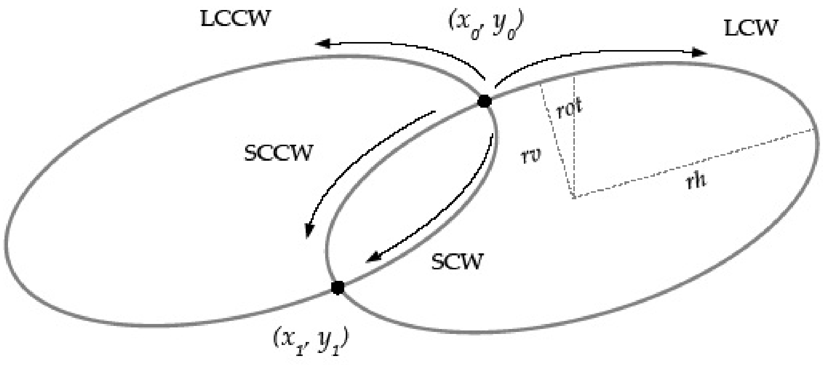


Figure 13: The different types of elliptic arcs and their parameters

Now that we can build a segment, we need to know how to create, destroy, append and manipulate a path.

vgCreatePath

This function returns a handle for the just created path. The path is created from a set of capabilities Capability actually code the functionalities which are allowed by the path. We won't list them since the are listed and available in the appendixes. The syntax of the function is:

VGPath vgCreatePath(VGint pathFormat,
VGPathDatatype datatype,
VGfloat scale, VGfloat bias,
VGint segmentCapacityHint,
VGint coordCapacityHint,

VGbitfield capabilities);

The parameter pathFormat identifies the format of the path. It is typically VG_PATH_FORMAT_STANDARD. Datatype indicates which kind of data is used for the path's data. Scale and bias define the way to interpret the incoming coordinates. For example, an incoming x will be understood as *x*scale + bias*. And segmentCapacityHint gives a hint about the number of segments in the path and the same goes for coordCapacityHint which gives a hint for the coordinates number in the parameters flow.

vgClearPath

This function removes all the segments appended to the path but the handle of the path can still be used.

void vgClearPath(VGPath path, VGbitfield capabilities)

vgDestroyPath

This function simply destroys the path including the handle which can't be used anymore.

void vgDestroyPath(VGPath path)

vgGetParameter

This function allows the developer to get any parameter of the path. But the type of the parameter needs to be precised. For example, we will use vgGetParameterf() to get a float or vgGetParameteri() to get an integer.

vgGetPathCapabilities

This function gives us the path capability.

VGbitfield vgGetPathCapabilities(VGPath path)

vgRemovePathCapabilities

This command removes the functionality associated with the capability passed in argument.

void vgRemovePathCapabilities(VGPath path, VGbitfield capabilities)

vgAppendPath

This function appends the path segments from srcPath to dstPath.

```
void vgAppendPath(VGPath dstPath, VGPath srcPath)
```

vgAppendPathData

This function appends the path data flows pathSegments characterized by the number of segments numSegments and pathData to dstPath.

```
void vgAppendPathData(VGPath dstPath,
                      VGint numSegments,
                      const VGubyte * pathSegments,
                      const void * pathData)
```

vgModifyPathCoords

This function allows us to modify some coordinates directly in the flow by using the index with the index start (startIndex) and the number of segments (numSegments) concerned from this index start and it replaces the original values by pathData.

```
void vgModifyPathCoords(VGPath dstPath,
                        int startIndex,
                        int numSegments,
                        const void * pathData)
```

vgTransformPath

This function appends a transformed copy of srcPath to dstPath. The transformation occurs only when the VG_PATH_CAPABILITY_TRANSFORM_FROM and VG_PATH_CAPABILITY_TRANSFORM_TO capabilities are enabled for srcPath and dstPath respectively.

```
void vgTransformPath(VGPath dstPath, VGPath srcPath)
```

vgInterpolatePath

This function will append to dstPath the interpolation of the path between startPath and endPath by the float amount. Of course, VG_PATH_CAPABILITY_INTERPOLATE_FROM must be enabled for both startPath and endPath and VG_PATH_CAPABILITY_INTERPOLATE_TO for dstPath.

```
VGboolean vgInterpolatePath(VGPath dstPath,
                            VGPath startPath,
                            VGPath endPath,
                                        VGfloat amount)
```

vgPathLength

This function gives us the length of a subpath from start segment and for a number of segments. To make sure that the command will work, we have to enable VG_PATH_CAPABILITY_PATH_LENGTH for path.

```
VGfloat vgPathLength(VGPath path,
                      VGint startSegment,
                      VGint numSegments);
```

vgPointAlongPath

As the name says, this function will provide us with the point (given through (x,y)) located at the distance *distance* of the subpath made of the numSegments segments starting from the startSegment segment. It gives also the tangent in this point. It is given through tangentX and tangentY. The point will be given only if VG_PATH_CAPABILITY_POINT_ALONG_PATH is enabled and the same goes for the tangent with VG_PATH_CAPABILITY_TANGENT_ALONG_PATH.

```
void vgPointAlongPath(VGPath path,
                VGint startSegment,
                VGint numSegments,
                VGfloat distance,
                VGfloat * x,
                VGfloat * y,
                VGfloat * tangentX,
                VGfloat * tangentY)
```

vgPathBounds

When VG_PATH_CAPABILITY_PATH_BOUNDS is enabled then the vgPathBounds function will give us the path bounding box through the box defined by the left low corner (minX,minY), the height *height* and the width *width.* The box returned is axis-aligned.

```
void vgPathBounds(VGPath path,
                VGfloat * minX,
                VGfloat * minY,
                VGfloat * width,
                VGfloat * height)
```

vgtransformedPathBounds

This function is the same than vgPathBounds excepted that the box returned is guaranteed to fit to the path but after the user-to-path transformation. The VG_PATH_CAPABILITY_PATH_TRANSFORMED_BOUNDS has to be enabled.

```
void vgPathTransformedBounds(VGPath path,
                              VGfloat * minX,
                             VGfloat * minY,
                            VGfloat * width,
                           VGfloat * height)
```

INTERPRETATION

Now that we know how to draw and handle a path, we can start considering some eventual options in path drawing. At first we will go through filling and then we will consider stroking, which is actually much more than an option. A path can be either filled (VG_FILL_PATH) or stroked (VG_STROKE_PATH). This choice has to be made when calling vgDrawPath. This has to be precised when we call **vgDrawPath()**. Both can also be set for the same path.

Filling

Filling consists in filling with paint the inside part of a path. In some cases, defining the inside part is easy, but in some others it becomes difficult. In OpenVG two rules have been fixed to decide which part of the surface is in and which one is not. To do so, we consider a straight line coming from the left side of the surface and each time it goes across a path segment, we add a number +1 or -1 to a counter whether the segment is oriented to up or down respectively while considering the orientation of the path. So every area of the surface receives a number. The rule Even/Odd Fill will colour only the parts of the surface with odd numbers while the other rule will colour every part of the surface with a number different of zero The figure 4 [RIC05] illustrates these 2 rules.

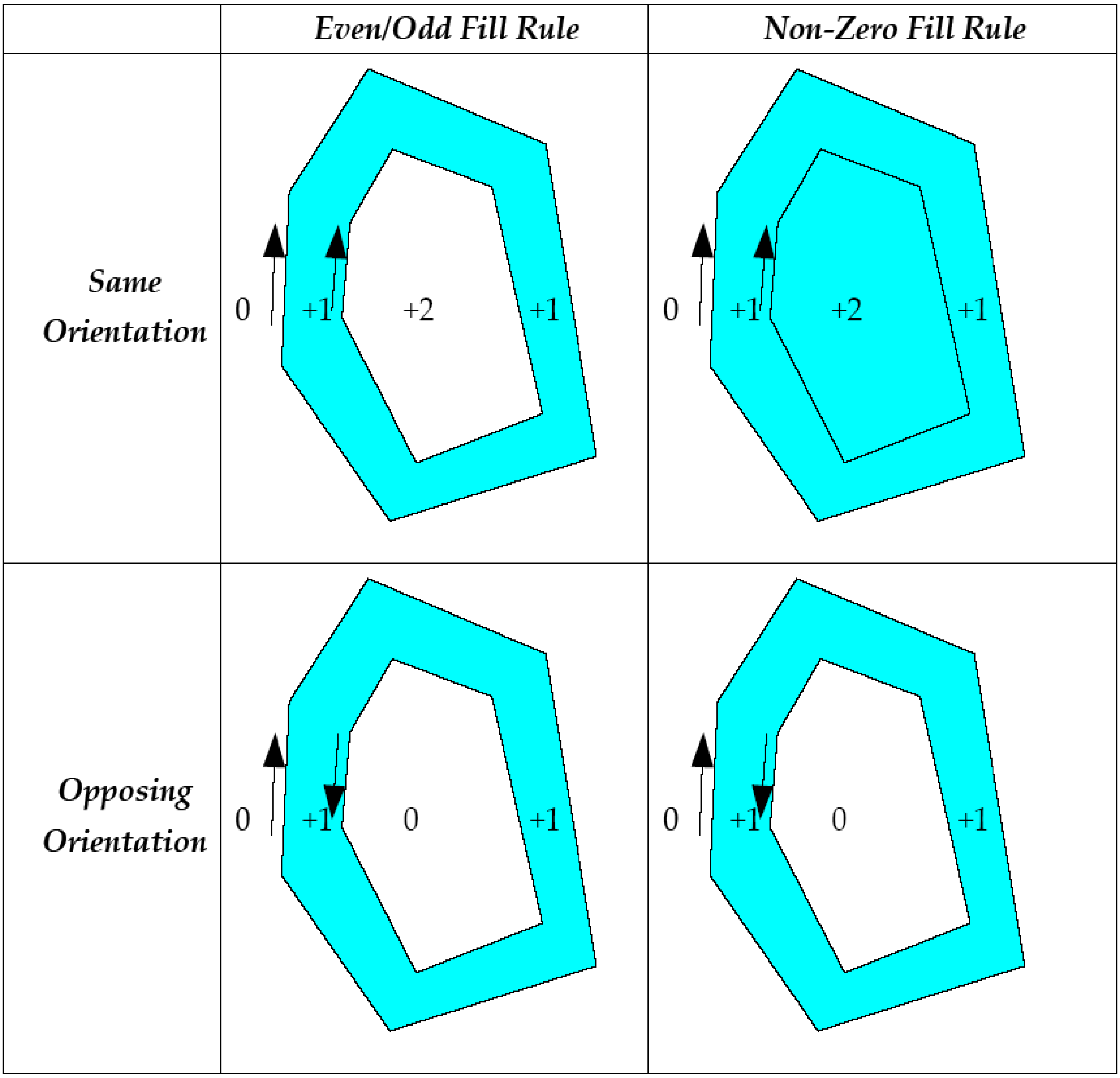


Figure 14: Filling rules examples

To do so, we just need to set the right filling rule in the VG_FILL_RULE attribute. The possibilities are VG_EVEN_ODD or VG_NON_ZERO.

```
VGFillRule fillRule;
vgSeti(VG_FILL_RULE, fillRule);
```

To finish, here is an example of a filled path and the code that generates it.

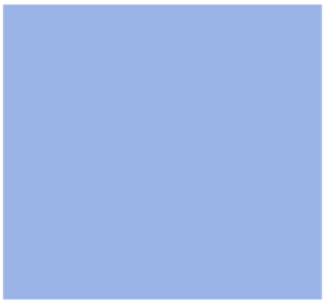


Figure15: Example of filled path

```
VGPath path =
vgCreatePath(VG_PATH_FORMAT_STANDARD,VG_PATH_DATATYPE_F, 1.0, 0, 0,
0,(unsigned int)VG_PATH_CAPABILITY_ALL);
VGubyte seg[] =
{VG_MOVE_TO_ABS,VG_LINE_TO_REL,VG_LINE_TO_REL,VG_LINE_TO_REL,VG_LIN
E_TO_REL,VG_CLOSE_PATH};
VGfloat coords[] = {100,100,200,0,0,200,-200,0,0,-200};

VGPaint fillpaint = vgCreatePaint();
VGfloat couleur[4];
couleur[0] = 0.2f;
couleur[1] = 0.4f;
couleur[2] = 0.8f;
couleur[3] = 0.5f;


vgSetParameteri(fillpaint,VG_PAINT_TYPE,VG_PAINT_TYPE_COLOR);
vgSetParameterfv(fillpaint,VG_PAINT_COLOR,4,couleur);
vgSetPaint(fillpaint,VG_FILL_PATH);

vgAppendPathData(path,6,seg,coords);
vgDrawPath(path,VG_FILL_PATH);
```

Stroking a path

Stroking a path consists in drawing a path with a certain width. It pops up many other parameters to set. We have to parameter the width of the stroked path, the end cap style of the path, the line join style, the miter limit and dashing parameters like pattern and phase. The figures 16 and 17 [RIV05] show the different line join styles and what miter limit refers to.

Dashing consists in dividing the stroked path in an alternance of “ON” and “OFF” segments. To do so, we need to define a pattern and a phase. The phase is the length starting from the starting of the first segment from which the dash pattern is enabled. But the pattern will start from the beginning of the first segment.

Here are the functions to set these parameters. At first, to set the stroke line width, we just have to call ***vgSetf(VG_STROKE_LINE_WIDTH, lineWidth).***

Then we can set the cap style by setting the integer VG_STROKE_CAP_STYLE. The options are VG_CAP_BUTT, VG_CAP_ROUND and VG_CAP_SQUARE.

```
VGCapStyle capStyle;
vgSeti(VG_STROKE_CAP_STYLE, capStyle);
```

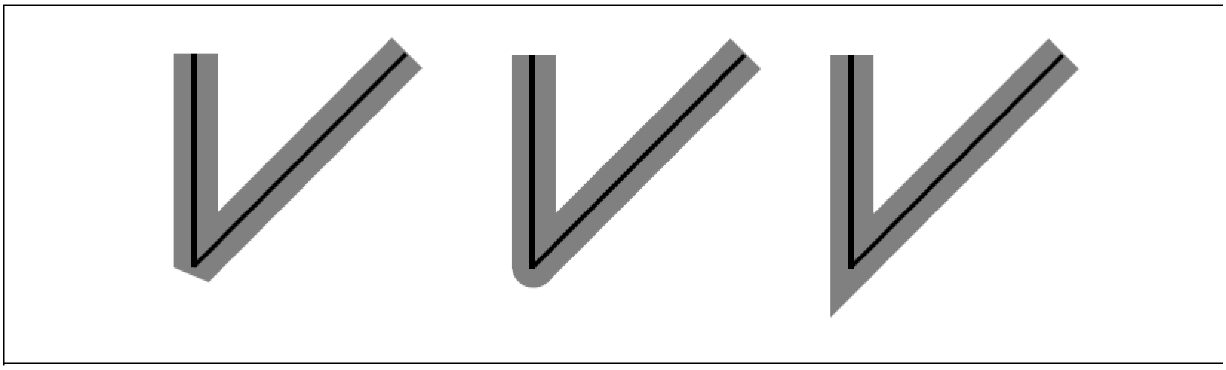


Figure 16 Line join styles

Then the same goes with VG_STROKE_JOIN_STYLE which has to be set with one of the following values: VG_JOIN_MITER, VG_JOIN_ROUND, VG_JOIN_BEVEL.

VGJoinStyle joinStyle;
vgSeti(VG_STROKE_JOIN_STYLE, joinStyle);

Then we can set the miter limit float.

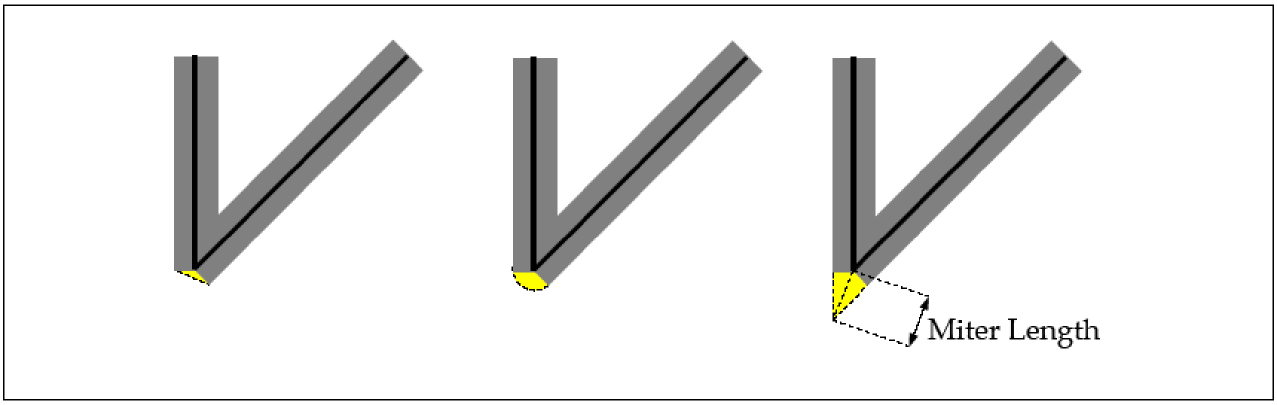


Figure 17 Miter length illustration

VGfloat miterLimit;
vgSetf(VG_STROKE_MITER_LIMIT, miterLimit);

And here we have an example of stroked path followed by the code to generate it.

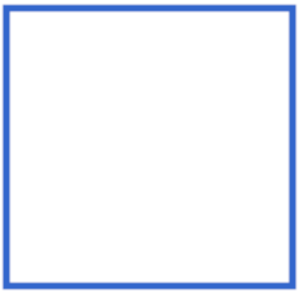


Figure18: Example of stroked path

```
VGPath path =
vgCreatePath(VG_PATH_FORMAT_STANDARD,VG_PATH_DATATYPE_F, 1.0, 0, 0,
0,(unsigned int)VG_PATH_CAPABILITY_ALL);
VGubyte seg[] =
{VG_MOVE_TO_ABS,VG_LINE_TO_REL,VG_LINE_TO_REL,VG_LINE_TO_REL,VG
_LINE_TO_REL,VG_CLOSE_PATH};
VGfloat coords[] = {100,100,200,0,0,200,-200,0,0,-200};

VGPaint strokepaint = vgCreatePaint();
VGfloat couleur[4];
couleur[0] = 0.2f;
couleur[1] = 0.4f;
couleur[2] = 0.8f;
couleur[3] = 1.0f;

vgSetf(VG_STROKE_LINE_WIDTH,5);
vgSetParameteri(strokepaint,VG_PAINT_TYPE,VG_PAINT_TYPE_COLOR);
vgSetParameterfv(strokepaint,VG_PAINT_COLOR,4,couleur);
vgSetPaint(strokepaint,VG_STROKE_PATH);

vgAppendPathData(path,6,seg,coords);
vgDrawPath(path,VG_STROKE_PATH);
```

Concerning the dashing, we need to set a pattern which is a sequence of numbers and the number of numbers in the sequence.

```
VGint maxDashCount = vgGeti(VG_MAX_DASH_COUNT);
VGfloat dashPattern[DASH_COUNT];
VGint count = DASH_COUNT;
vgSetfv(VG_STROKE_DASH_PATTERN, count, dashPattern);
```

To set these parameters with a *count of* 0 would just disable dashing. Then we just need to set the phase as follows:

```
VGfloat dashPhase;
vgSetf(VG_STROKE_DASH_PHASE, dashPhase);
```

The following figure shows an example of dashed path.

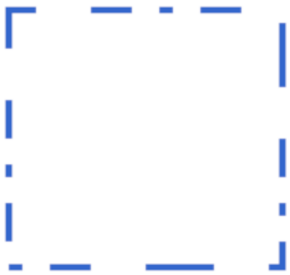


Figure18: Dashed path

And here is the code that draws the example above.

```
VGPath path =
vgCreatePath(VG_PATH_FORMAT_STANDARD,VG_PATH_DATATYPE_F, 1.0, 0, 0,
0,(unsigned int)VG_PATH_CAPABILITY_ALL);
VGubyte seg[] =
{VG_MOVE_TO_ABS,VG_LINE_TO_REL,VG_LINE_TO_REL,VG_LINE_TO_REL,VG_LINE_TO_REL,VG_CLOSE_PATH};

VGfloat coords[] = {100,100,200,0,0,200,-200,0,0,-200};
VGfloat phases[9] = {10,20,30,40,50,40,30,20,10};

VGPaint strokepaint = vgCreatePaint();
VGfloat couleur[4];
couleur[0] = 0.2f;
couleur[1] = 0.4f;
couleur[2] = 0.8f;
couleur[3] = 1.0f;

vgSetf(VG_STROKE_LINE_WIDTH,5);
vgSetParameteri(strokepaint,VG_PAINT_TYPE,VG_PAINT_TYPE_COLOR);
vgSetParameterfv(strokepaint,VG_PAINT_COLOR,4,couleur);
```

```
vgSetPaint(strokepaint,VG_STROKE_PATH);
vgSetfv(VG_STROKE_DASH_PATTERN,9,phases);

vgAppendPathData(path,6,seg,coords);
vgDrawPath(path,VG_STROKE_PATH);
```

2.6 Paint

2.6.1 Paint generalities

Generating the paint is the next step after building the path. The paint is the generic term used for the union of solid colours, gradient patterns and patterns tilling. In a more concrete manner, a paint is the union of a colour and an alpha value for every pixel concerned. A paint is computed in its own coordinates. So we need a transformation to map the paint to the user's coordinates system. That is the role of the paint_to_user transformation. We will have to consider two cases of transformation: when we consider the stroked path paint and the filling paint.

In OpenVG, the paint is an object and thus we need to create a handle to manipulate it. That is the role of the VGPaint type. This VGPaint is the type of the result obtained when we create a new paint. To do so, we use the function vgCreatePaint.

```
VGPaint vgCreatePaint (void)
```

To destroy this paint, we will use the function vgDestroyPaint as follows:

```
void vgDestroyPaint (VGPaint paint)
```

Once the paint is created, we have to set it on the current context. To do so we need to use the vgSetPaint function for which we need to define the paint mode (filling, stroking or both).

```
void vgSetPaint (VGPaint paint, Vgbitfield paintModes)
```

The vgGetPaint allows us to get then current paint object set for the paint mode given in input.

```
VGPaint vgGetPaint (VGPaintMode paintMode)
```

As we explained it in the beginning of this sub-chapter, there are 4 different types of paint. The VGPaintType enumeration defines these 4 different types.

```
typedef enum {
VG_PAINT_TYPE_COLOR = 0x1B00,
VG_PAINT_TYPE_LINEAR_GRADIENT = 0x1B01,
VG_PAINT_TYPE_RADIAL_GRADIENT = 0x1B02,
VG_PAINT_TYPE_PATTERN = 0x1B03
} VGPaintType;
```

To set one of this paint types, we have to set the corresponding value in the VG_PAINT_TYPE by using the vgSetParameter function.

2.6.2 The paint types

Now that we went through all the generalities concerning how to handle and how to manage the paint, we can start to have a look at the different paint modes.

2.6.2.1 Solid colour

As we explained it in the sub-chapter 2.2 Basics, a solid colour is an array of 4 floats between 0 and 1, defining the red, the green and the blue components plus the alpha value. So to set a solid colour, we just need to set the 4 floats then to set these values in a 4 floats array. The next step is to set the type of paint for the current paint, that is to say VG_PAINT_TYPE_COLOR. Then we just have to set the parameter VG_PAINT_COLOR with the array we defined. This way to proceed is the same whether we are considering stroke paint or filling paint.

```
VGfloat fill_red, fill_green, fill_blue, fill_alpha;
VGfloat stroke_red, stroke_green, stroke_blue, stroke_alpha;
VGPaint myFillPaint, myStrokePaint;

VGfloat * fill_RGBA = {
fill_red, fill_green, fill_blue, fill_alpha
};

VGfloat * stroke_RGBA = {
stroke_red, stroke_green, stroke_blue, stroke_alpha
};

/* Fill with color paint */
vgSetParameteri(myFillPaint, VG_PAINT_TYPE, VG_PAINT_TYPE_COLOR);
vgSetParameterfv(myFillPaint, VG_PAINT_COLOR, 4, fill_RGBA);

/* Stroke with color paint */
vgSetParameteri(myStrokePaint, VG_PAINT_TYPE, VG_PAINT_TYPE_COLOR);
```

```
vgSetParameterfv(myStrokePaint, VG_PAINT_COLOR, 4, stroke_RGBA);
```

The function vgSetColor is doing the same work but in much simpler way to use it.

```
void vgSetColor(VGPaint paint, VGuint rgba)
```

The analogue function has been defined for getting the colour of a paint, it is vgGetColor.

```
VGuint vgGetColor(VGPaint paint)
```

2.6.2.2 Gradient paints

OpenVG allows the developer to draw two kinds of gradient paints. We can draw linear gradients and radial gradients. A gradient is made of a scalar value for every point in the paint coordinates and of a colour ramp to map on it.

Linear gradients

They are defined on two points (x0,y0) and (x1,y1) of the plan. The scalar value is 0 in (x0,y0) and 1 in (x1,y1). And the progression along the line between the two points is linear, so comes the name, and the scalar value is constant along the perpendicular lines.

The linear gradient type is set by setting the parameter VG_PAINT_TYPE with the value VG_PAINT_TYPE_LINEAR_GRADIENT by using the **vgSetParameteri** function. Then we have to set the parameters for VG_PAINT_LINEAR_GRADIENT as follows:

```
VGfloat fill_x0, fill_y0, fill_x1, fill_y1;
VGfloat stroke_x0, stroke_y0, stroke_x1, stroke_y1;
VGPaint myFillPaint, myStrokePaint;

VGfloat * fill_linear_gradient = {
fill_x0, fill_y0, fill_x1, fill_y1
};

VGfloat * stroke_linear_gradient = {
stroke_x0, stroke_y0, stroke_x1, stroke_y1
};

/* Fill with linear gradient paint */
```

```
vgSetParameteri(myFillPaint, VG_PAINT_TYPE,
VG_PAINT_TYPE_LINEAR_GRADIENT);
vgSetParameterfv(myFillPaint, VG_PAINT_LINEAR_GRADIENT,
fill_linear_gradient);

/* Stroke with linear gradient paint */
vgSetParameteri(myStrokePaint, VG_PAINT_TYPE,
VG_PAINT_TYPE_LINEAR_GRADIENT);
vgSetParameterfv(myStrokePaint, VG_PAINT_LINEAR_GRADIENT,
1.  stroke_linear_gradient);
```

As we can see on this code example, linear gradients are applicable to both path filling and path stroking paints.

Radial gradients

The radial gradient involves a circle defined by its centre (cx,cy) and its radius r. The radial gradient also needs a focal point (fx,fy) which is situated within the circle. For every point (x,y) in the circle, when we consider the straight line from the focal point to this point, the scalar value is the ratio of the distance from the point to the focal point on the distance from the focal point to the circle including the point (x,y). The figure 19 [RIC05] illustrates this explanation.

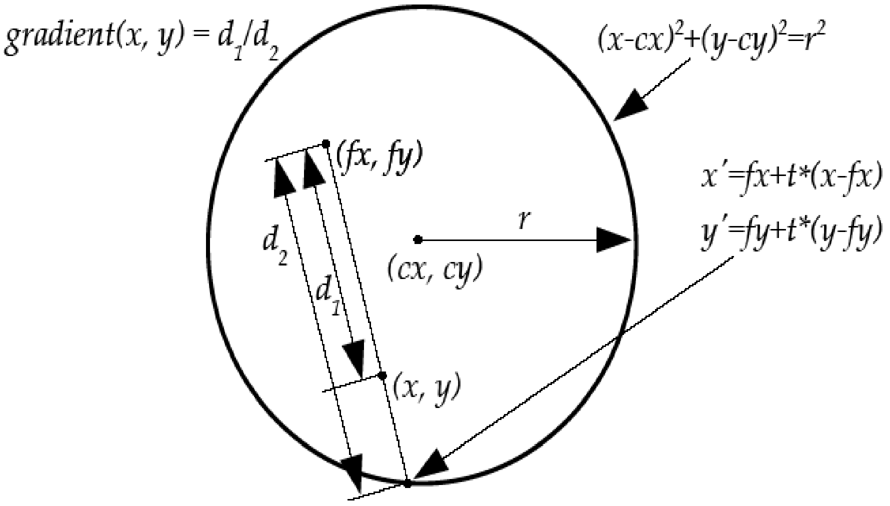


Figure 19 Computation of the radial gradient

The radial gradient type is set by setting the parameter VG_PAINT_TYPE with the value VG_PAINT_TYPE_RADIAL_GRADIENT by using the **vgSetParameteri** function. Then we have to set the parameters, a five floats array {cx, cy, fx, fy, r}, for VG_PAINT_RADIAL_GRADIENT as follows:

```
VGPaint myFillPaint, myStrokePaint;
VGfloat fill_cx, fill_cy, fill_fx, fill_fy, fill_r;
VGfloat stroke_cx, stroke_cy, stroke_fx, stroke_fy, stroke_r;

VGfloat * fill_radial_gradient = { fill_cx, fill_cy,
fill_fx, fill_fy, fill_r };
VGfloat * stroke_radial_gradient = { stroke_cx, stroke_cy,
stroke_fx, stroke_fy, stroke_r };

/* Fill */
vgSetParameteri(myFillPaint, VG_PAINT_TYPE,
VG_PAINT_TYPE_RADIAL_GRADIENT);
vgSetParameterfv(myFillPaint, VG_PAINT_RADIAL_GRADIENT,
4. fill_radial_gradient);


/* Stroke */
vgSetParameteri(myStrokePaint, VG_PAINT_TYPE,
VG_PAINT_TYPE_RADIAL_GRADIENT);
vgSetParameterfv(myStrokePaint, VG_PAINT_RADIAL_GRADIENT,
5.  stroke_radial_gradient);
```

Of course, as for the linear gradient, the radial gradient applies to both stroking path and path filling paints.

Now that we went through the two alternatives for gradients, we can have a look at the colour ramps.

Colour ramps

A colour ramp maps the scalar value computed by the gradient to a colour. More concretely, the user defines a scale from 0 to 1 in which values correspond to colours. To do so, we need to defines colour stops. A colour stop defines the starting point of a colour on the scale. It is defined by 5 floats: the value of the stop on the scale and the 4 floats that make a colour. So we will have to set a number of parameters which is a multiple of 5. But before that, we have to set the ramp spread mode we like to draw. Indeed, there exists several spread modes for the colour ramps. The first one, VG_COLOR_RAMP_SPREAD_PAD, extends the stops. The second one, VG_COLOR_RAMP_SPREAD_REPEAT repeats the stops while the third one VG_COLOR_RAMP_SPREAD_REFLECT repeats them but in reflected order. So we have to set one of these modes by setting

VG_PAINT_COLOR_RAMP_SPREAD_MODE. Then we set the stops parameters by setting a value for VG_PAINT_COLOR_RAMP_STOPS.

```
VGPaint myFillPaint, myStrokePaint;

VGColorRampSpreadMode fill_spreadMode;
VGfloat fill_stops[5*FILL_NUM_STOPS];

VGColorRampSpreadMode stroke_spreadMode;
VGfloat stroke_stops[5*STROKE_NUM_STOPS];

vgSetParameteri(myFillPaint, VG_PAINT_COLOR_RAMP_SPREAD_MODE,
fill_spreadMode);
vgSetParameterfv(myFillPaint, VG_PAINT_COLOR_RAMP_STOPS,
5*FILL_NUM_STOPS, fill_stops);

vgSetParameteri(myStrokePaint, VG_PAINT_COLOR_RAMP_SPREAD_MODE,
stroke_spreadMode);
vgSetParameterfv(myStrokePaint, VG_PAINT_COLOR_RAMP_STOPS,
5*STROKE_NUM_STOPS, stroke_stops);
```

Pattern paint

Pattern paint actually consists in interpolating the pixel values of a given image and to use them as a paint. To do so, we need at first to set the pattern paint mode. That has to be done by setting the VG_PAINT_TYPE parameter with the VG_PAINT_TYPE_PATTERN. Then we can bind the image pattern to the corresponding paint like follows above:

```
void vgPaintPattern(VGPaint paint, VGImage pattern)
```

The next and last parameter that has to be set is the tiling mode. There exist 4 different tiling modes. The first one VG_TILE_FILL consists in simply considering the pixels outside the bounds of the image pattern as using the color VG_TILE_FILL_COLOR which has to be defined then. The second one is VG_TILE_PAD. In this mode, every pixel outside the bounds of the pattern image will have the colour of the closest pixel from the pattern. The third one, VG_TILE_REPEAT, implies that the pattern image is repeated in all the directions and infinitely. The last tiling mode, VG_TILE_REFLECT, implies that the pattern image is repeated in a reflected manner infinitely. The following code sample shows how to manage all these settings.

```
VGPaint myFillPaint, myStrokePaint;
VGImage myFillPaintPatternImage, myStrokePaintPatternImage;
VGTilingMode fill_tilingMode, stroke_tilingMode;

vgSetParameteri(myFillPaint, VG_PAINT_TYPE,
VG_PAINT_TYPE_PATTERN);
vgSetParameteri(myFillPaint, VG_PAINT_PATTERN_TILING_MODE,
fill_tilingMode);

vgPaintPattern(myFillPaint, myFillPaintPatternImage);
vgSetParameteri(myStrokePaint, VG_PAINT_TYPE,
VG_PAINT_TYPE_PATTERN);

vgSetParameteri(myStrokePaint, VG_PAINT_PATTERN_TILING_MODE,
stroke_tilingMode);
vgPaintPattern(myStrokePaint, myStrokePaintPatternImage);
```

.

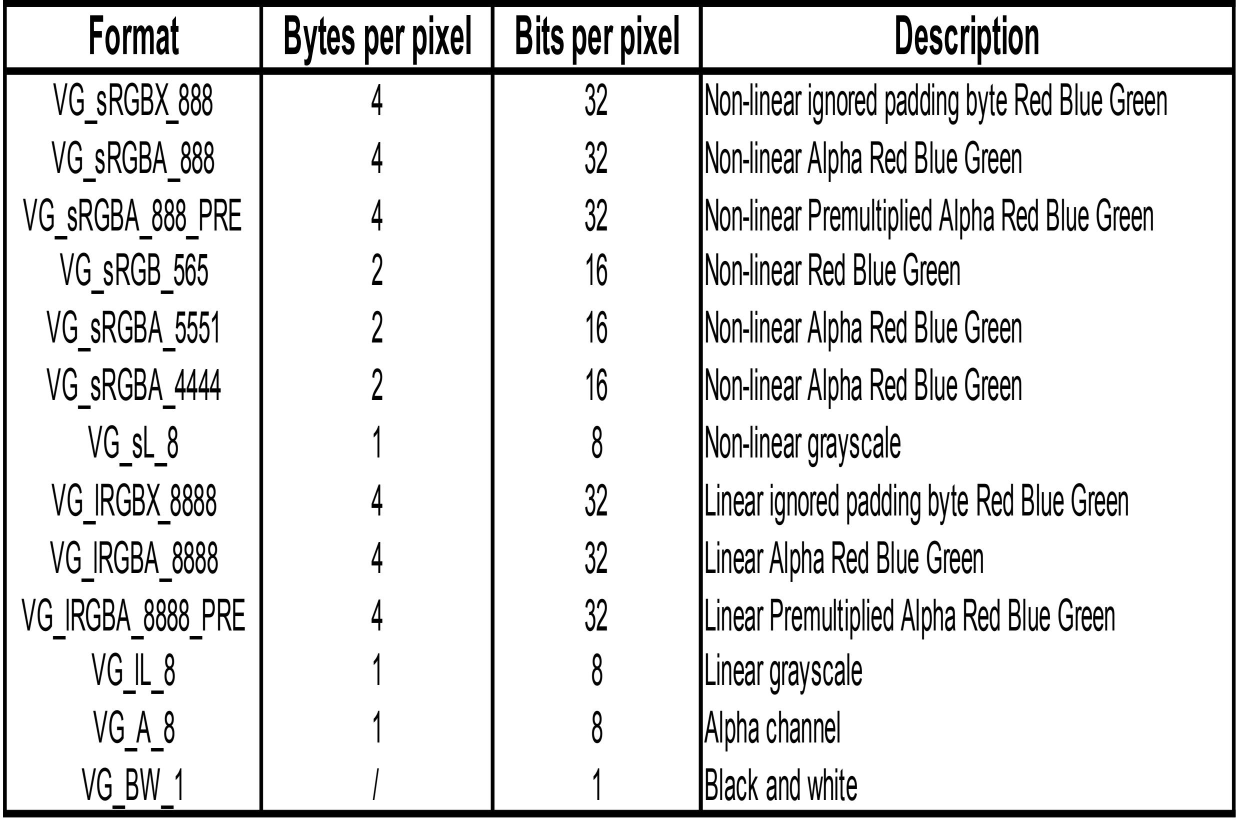
And as usual, this paint mode is available for both path filling and path stroking.

2.7 Images

In this sub-chapter we want to study the main possibilities that OpenVG offers for handling and manipulating images.

2.7.1 Generalities about images

Images can be considered as rectangular collections of pixels. Several formats exist and are accepted and handled by OpenVG. The table 1 sums ups all the formats accepted by OpenVG.



Format	Bytes per pixel	Bits per pixel	Description
VG_sRGBX_888	4	32	Non-linear ignored padding byte Red Blue Green
VG_sRGBA_888	4	32	Non-linear Alpha Red Blue Green
VG_sRGBA_888_PRE	4	32	Non-linear Premultiplied Alpha Red Blue Green
VG_sRGB_565	2	16	Non-linear Red Blue Green
VG_sRGBA_5551	2	16	Non-linear Alpha Red Blue Green
VG_sRGBA_4444	2	16	Non-linear Alpha Red Blue Green
VG_sL_8	1	8	Non-linear grayscale
VG_lRGBX_8888	4	32	Linear ignored padding byte Red Blue Green
VG_lRGBA_8888	4	32	Linear Alpha Red Blue Green
VG_lRGBA_8888_PRE	4	32	Linear Premultiplied Alpha Red Blue Green
VG_lL_8	1	8	Linear grayscale
VG_A_8	1	8	Alpha channel
VG_BW_1	/	1	Black and white

Table 3 Image formats

This image formats are defined in the enumerated type VGImageFormat. On a more formal plan, in OpenVG, images are considered as object and thus, like paints, they need an opaque handle to be manipulated. That's the aim of the VGImage type. When we create or draw images, we also need to specify a quality. This parameter defines the resampling level. Indeed, when we draw an image, in case of a zoom, there will be a mapping to operate cause for one pixel of the image, there will be several pixels on the screen. That is what VG_IMAGE_NONANTIALIASED, the first quality level, the lowest one, does. An image pixel is transposed on screen on several pixels. There is no resampling. The next quality level, VG_IMAGE_QUALITY_FASTER, allows resampling but with a medium quality that allow a good speed for drawing, because of low resource consumption. The last one, VG_IMAGE_QUALITY_BETTER, proposes the best resampling quality but is slower to render and uses more resources. These different qualities are listed in the VGImageQuality enumerated type. This quality level is set by setting the VG_IMAGE_QUALITY parameter.

OpenVG contains some inner parameters which are limiting the size, the pixel number, the memory size of an image. These parameters can be set and they will control and limit the image creation. Image creation is done by using the vgCreateImage function by specifying the image format, the dimensions of the image and the quality.

```
VGImage vgCreateImage(VGImageFormat format,
                      VGint width,
                      VGint height,
                      VGbitfield allowedQuality)
```

The dual function allowing us to destroy an image also exists.

```
void vgDestroyImage(VGImage image);
```

Of course all the parameters can be retrieved by simply using the vgGetParameter functions as follows:

```
VGImage image;
VGImageFormat imageFormat =
(VGImageFormat)vgGetParameteri(image, VG_IMAGE_FORMAT);

VGint imageWidth = vgGetParameteri(image, VG_IMAGE_WIDTH);

VGint imageHeight = vgGetParameteri(image, VG_IMAGE_HEIGHT);
```

2.7.2 Manipulating images

The functions that we are going to describe here propose some more advanced features for manipulating images including on a pixel plan.

vgClearImage

This function fills a given rectangle of the input image with the VG_CLEAR_COLOR specified.

```
void vgClearImage(VGImage image,
VGint x, VGint y, VGint width, VGint height)
```

vgImageSubData

This function reads in the memory at a given address *data* and will convert the pixels read to the specified format and store them in the input picture within the given rectangle. The function considers the datastride bytes as the separations between the scanlines.

```
void vgImageSubData(VGImage image,
const void * data, VGint dataStride,
VGImageFormat dataFormat,
VGint x, VGint y, VGint width, VGint height)
```

vgGetImageSubData

This function does the opposite task as the previous one. It reads pixels from an image within a given rectangle and stores in the memory after having converted the read pixels to the specified format.

```
void vgGetImageSubData(VGImage image,
void * data, VGint dataStride,
VGImageFormat dataFormat,
VGint x, VGint y, VGint width, VGint height)
```

Child images

A child image is an image that shares the physical memory with a portion of another image. These two images, the child image and its parent, have the property that when vgDestroyImage is called on one of them, the other still remains. To create a child image, we just have to call **vgChildImage**, specifying the target rectangle that will generate another image as the child.

```
VGImage vgChildImage(VGImage parent,
VGint x, VGint y, VGint width, VGint height)
```

We should also mention that a function which allows us to retrieve the parent image exists and it always has a result since any given image has always a parent image, at least itself.

```
VGImage vgGetParent(VGImage image)
```

vgCopyImage

This function allows us to copy pixels within a given rectangle size from one place on a source image to another given place on the destination image. A Boolean input, *dither*, specifies whether a implementation specific algorithm shall be used or not.

```
void vgCopyImage(VGImage dst, VGint dx, VGint dy,
                 VGImage src, VGint sx, VGint sy,
                 VGint width, VGint height,
                 VGboolean dither)
```

2.7.3 Images and drawing surface

There exist three different image drawing modes. They are enumerated in the VGImageMode type. They can be set by setting the VG_IMAGE_MODE. The first mode, VG_DRAW_IMAGE_NORMAL, means that the image will be drawn normally without paint generation. In the second mode, VG_DRAW_IMAGE_MULTIPLY, a paint is created and the image being drawn is multiplied, as the name says, by the paint created. The third mode, VG_DRAW_IMAGE_STENCIL, the image being drawn is used as a stencil through which the paint created is generated. That means and implies that through specific equations, every channel is being computed from specific alpha values and paint and image own channels values.

$$\alpha tmp = \alpha(\alpha image*\alpha paint, \alpha dst)$$
$$Rdst \neg c(Rpaint, Rdst, Rimage*\alpha image*\alpha paint, \alpha dst) / \alpha tmp$$
$$Gdst \neg c(Gpaint, Gdst, Gimage*\alpha image*\alpha paint, \alpha dst) / \alpha tmp$$
$$Bdst \neg c(Bpaint, Bdst, Bimage*\alpha image*\alpha paint, \alpha dst) / \alpha tmp$$
$$\alpha dst \neg \alpha tmp$$

vgSetPixels

This function copies pixels data of a given rectangle from the source image to the drawing surface where it is specified by the input.

```
void vgSetPixels(VGint dx, VGint dy,
                 VGImage src, VGint sx, VGint sy,
                 VGint width, VGint height)
```

vgWritePixels

This function consists in drawing directly on the drawing surface in the rectangle given as input, the pixels read in the memory at the *data* address specified, each scanline being separated by dataStride bytes.

```
void vgWritePixels(const void * data, VGint dataStride,
                   VGImageFormat dataFormat,
                   VGint dx, VGint dy,
                   VGint width, VGint height)
```

vgGetPixels

This function copies the pixel data from the given rectangle from the drawing surface to the image given in input.

```
void vgGetPixels(VGImage dst, VGint dx, VGint dy,
                 VGint sx, VGint sy,
                 VGint width, VGint height)
```

vgReadPixels

This function does exactly the same task as the one before but stores the pixels data after having converted it in memory at the *data* address specified. The pixel lines scanned are separated by dataStride bytes.

```
void vgReadPixels(void * data, VGint dataStride,
                  VGImageFormat dataFormat,
                  VGint sx, VGint sy,
                  VGint width, VGint height)
```

vgCopyPixels

This function consists in copying pixels from a specified rectangle to another given rectangle of the drawing surface.

```
void vgCopyPixels(VGint dx, VGint dy,
VGint sx, VGint sy,
VGint width, VGint height)
```

2.8 Advanced features

In the previous sub-chapters, we discovered all the basic features offered by OpenVG. In this sub-chapter, we go through more advanced features. That means it is not absolutely necessary to know how to use these features but they provide us with a higher level of rendering performance.

2.8.1 Image filters

Image filtering consists in modifying an image by applying a sequence of imaging operations. When we consider a pixel in the RGBA colour space, its data can be seen as a 4 floats vector [R G B α]. An imaging operation will be represented by a 4*4 float matrix. And during this imaging operation, every pixel's data will be computed by calculating the product of the operation matrix and the pixel vector. Some other operations can also consist in just adding a 4 floats vector to the pixel. This is done by the function vgColorMatrix which operates the imaging operation on the given source image to produce the destination image. The input matrix will be 4*5, given like follows: {m00, m10, m20, m30, m01, m11, m21, m31, m02, m12, m22, m32, m03, m13, m23, m33, m43, m04, m14, m24, m34}. And the function proceeds as follows:

$$\begin{bmatrix} R_{dst} \\ G_{dst} \\ B_{dst} \\ \alpha_{dst} \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \cdot \begin{bmatrix} R_{src} \\ G_{src} \\ B_{src} \\ \alpha_{src} \end{bmatrix} + \begin{bmatrix} m_{04} \\ m_{14} \\ m_{24} \\ m_{34} \end{bmatrix}$$

*void vgColorMatrix(VGImage dst, VGImage src, const VGfloat * matrix)*

vgConvolve

This function realizes a convolution of a given kernel and an image according to the given tiling mode. The result is stored in a destination image. The result is multiplied by the scale while the bias is added o every component. The shift integers define the translation. The formula is the following:

$$s\left(\sum_{0 \le i < w} \sum_{0 \le j < h} k_{(w-i-1),(h-j-1)}\, p(x+i-shiftX\,,\, y+j-shiftY)\right)+b$$

void vgConvolve(VGImage dst, VGImage src,
VGint kernelWidth, VGint kernelHeight,
VGint shiftX, VGint shiftY,

```
                    const VGshort * kernel,
                    VGfloat scale,
                    VGfloat bias,
                    VGTilingMode tilingMode)
```

vgSeparableConvolve

This function, which is quite similar to the previous one, realizes a separable convolution of a source image and the two kernels given (one per axis). The following formula sums up this kind of convolution,

$$s\left(\sum_{0\leq i<w}\sum_{0\leq j<h} kx_{(w-i-1)}ky_{(h-j-i)}p(x+i-shiftX\ ,y+j-shiftY)\right)+b$$

```
void vgSeparableConvolve(VGImage dst, VGImage src,
                         VGint kernelWidth, VGint kernelHeight,
                         VGint shiftX, VGint shiftY,
                         const VGshort * kernelX,
                         const VGshort * kernelY,
                         VGfloat scale,
                         VGfloat bias,
                         VGTilingMode tilingMode)
```

vgGaussianBlur

This function performs the convolution of a source image and two kernels which are actually the Gaussian function in a standard axis deviation. The formula is the following:

$$k(x\ ,y)=G(x\ ,s_x)*G(y\ ,s_y)=\frac{1}{2\pi\, s_x s_y}e^{-\left(\frac{x^2}{2s_x^{\ 2}}+\frac{y^2}{2s_y^{\ 2}}\right)}$$

```
void vgGaussianBlur(VGImage dst, VGImage src,
                    VGfloat stdDeviationX,
                    VGfloat stdDeviationY,
                    VGTilingMode tilingMode)
```

vgLookup

This function passes every channel of every pixel of the source image in its appropriate lookup table and stores the result in an image dst. The outputs are converted in the

RGBA color space and the given input parameters will define whether the colour space is linear or not and premultiplied or not.

```
void vgLookup(VGImage dst, VGImage src,
              const VGubyte * redLUT,
              const VGubyte * greenLUT,
              const VGubyte * blueLUT,
              const VGubyte * alphaLUT,
              VGboolean outputLinear,
              VGboolean outputPremultiplied)
```

vgLookupSingle

This function executes exactly the same task except that only one channel is passed through a single lookup table. The channel which has to be modified is precised in sourceChannel. The alternatives are: VG_RED, VG_GREEN, VG_BLUE, VG_ALPHA.

```
void vgLookupSingle(VGImage dst, VGImage src,
                    const VGuint * lookupTable,
                    VGImageChannel sourceChannel,
                    VGboolean outputLinear,
                    VGboolean outputPremultiplied)
```

2.8.2 Scissoring and masking

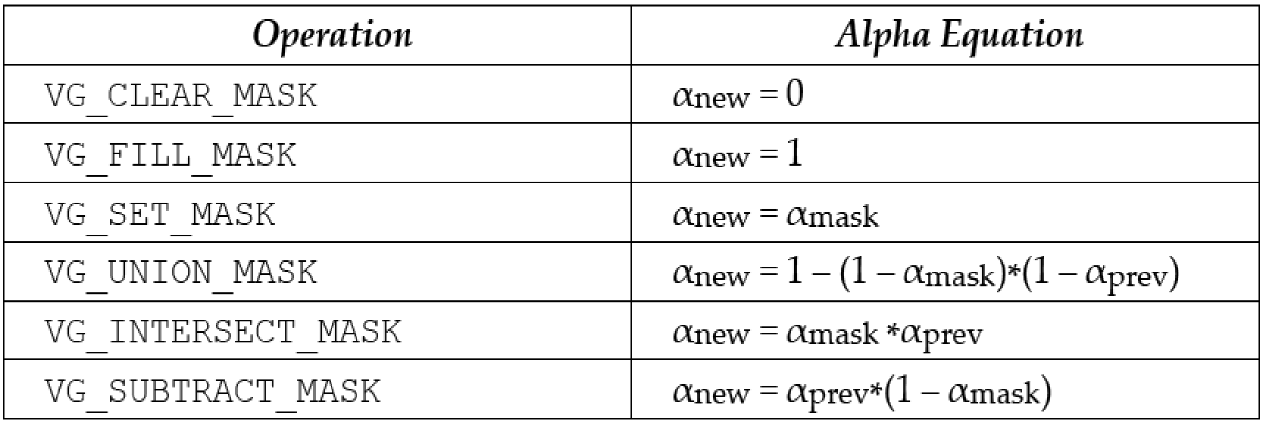
Scissoring

To set the scissoring rectangles, we have to set the VG_SCISSOR_RECTS parameter by using the vgSeti function. The input will be an array of integers with a multiple of 4 entries in it.

```
#define NUM_RECTS 2
/* { Min X, Min Y, Width, Height } 4-Tuples */
VGint coords[4*NUM_RECTS] = { 20, 30, 100, 200,
50, 70, 80, 80 };
vgSetiv(VG_SCISSOR_RECTS, 4*NUM_RECTS, coords)
```

Masking

In addition to the alpha masking, some other masking operations are available. They are combined to the initial alpha values. To use these operations, we have to specify a rectangle which will be the operation area. The mask operations are listed and detailed in the table 1 [RIC05].



Operation	*Alpha Equation*
VG_CLEAR_MASK	$\alpha_{new} = 0$
VG_FILL_MASK	$\alpha_{new} = 1$
VG_SET_MASK	$\alpha_{new} = \alpha_{mask}$
VG_UNION_MASK	$\alpha_{new} = 1 - (1 - \alpha_{mask}) * (1 - \alpha_{prev})$
VG_INTERSECT_MASK	$\alpha_{new} = \alpha_{mask} * \alpha_{prev}$
VG_SUBTRACT_MASK	$\alpha_{new} = \alpha_{prev} * (1 - \alpha_{mask})$

Table 4 Additional masking operations

These additional masking operations have to be set by using the vgMask function fro which you have to specify the concerned rectangle as an input.

void vgMask(VGImage mask, VGMaskOperation operation,
VGint x, VGint y, VGint width, VGint height)

2.8.3 Blending

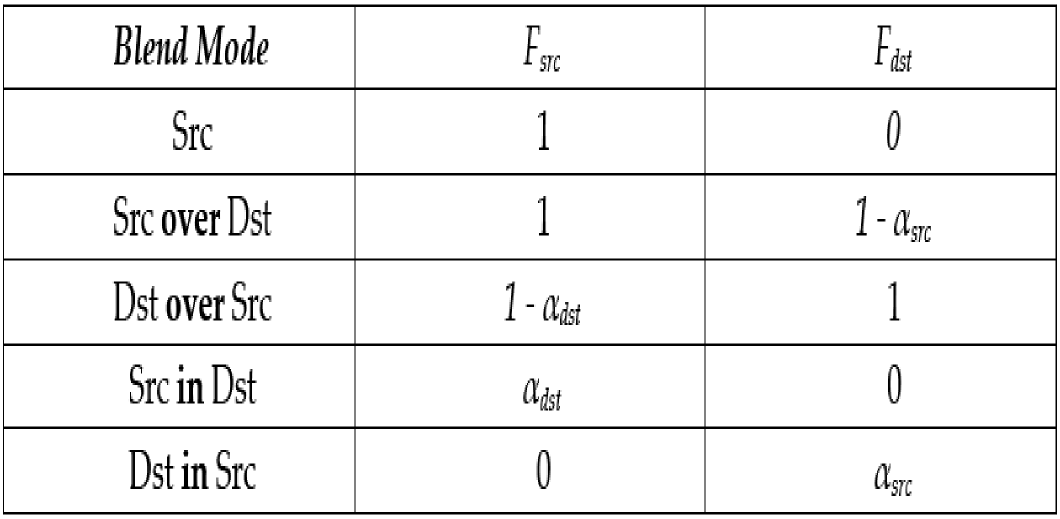
We have already explained what blending is and we already gave some examples of blending modes. In this sub-section, we will just propose and explain more blending modes. But before to do so, we can remind that blending transforms each channel value by using blending equations and replacing the destination pixel by the blended tuple:

$(c(Rsrc, Rdst, \alpha src, \alpha dst), c(Gsrc, Gdst, \alpha src, \alpha dst), c(Bsrc, Bdst, \alpha src, \alpha dst), \alpha(\alpha src, \alpha dst))$

Actually some blending rules, called Porter-Duff rules, are included. These rules take the form of blending equations

$$\alpha(\alpha src, \alpha dst) = \alpha src * Fsrc + \alpha dst * Fdst$$
$$c'(c'src, c'dst, \alpha src, \alpha dst) = c'src * Fsrc + c'dst * Fdst$$

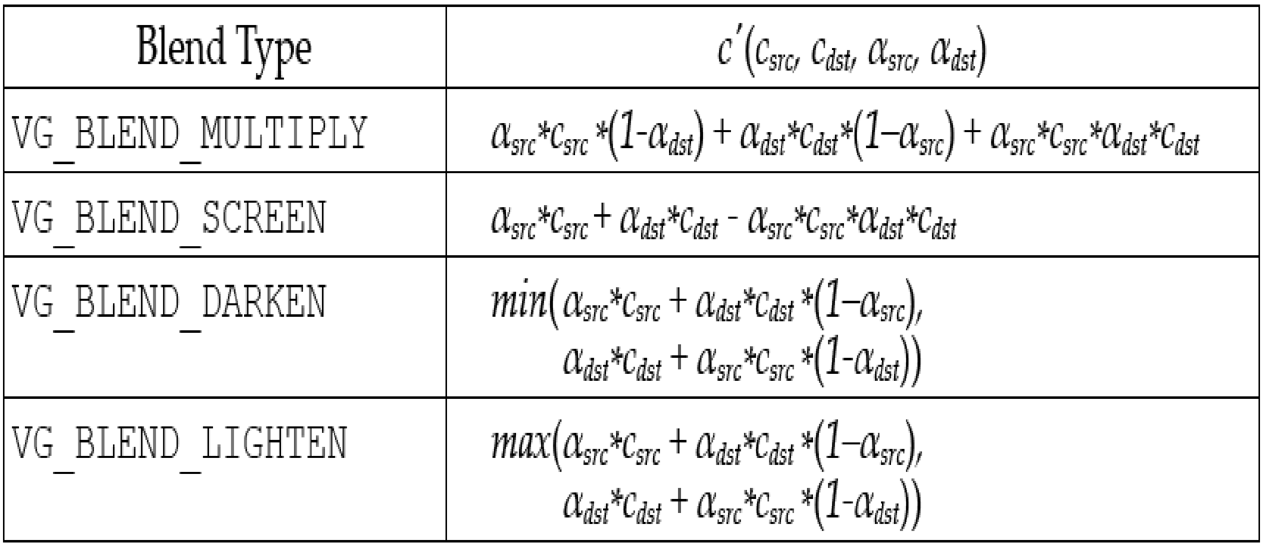
where $c' = \alpha * c$ and Fsrc and Fdst are defined by the blending modes.



Blend Mode	F_{src}	F_{dst}
Src	1	0
Src **over** Dst	1	$1 - \alpha_{src}$
Dst **over** Src	$1 - \alpha_{dst}$	1
Src **in** Dst	α_{dst}	0
Dst **in** Src	0	α_{src}

Table 5 Porter-Duff Blending modes [RIC05]

Besides Porter-Duff, some other modes exist. They are defined in the table 3 [RIC05].



Blend Type	$c'(c_{src}, c_{dst}, \alpha_{src}, \alpha_{dst})$
VG_BLEND_MULTIPLY	$\alpha_{src}*c_{src}*(1-\alpha_{dst}) + \alpha_{dst}*c_{dst}*(1-\alpha_{src}) + \alpha_{src}*c_{src}*\alpha_{dst}*c_{dst}$
VG_BLEND_SCREEN	$\alpha_{src}*c_{src} + \alpha_{dst}*c_{dst} - \alpha_{src}*c_{src}*\alpha_{dst}*c_{dst}$
VG_BLEND_DARKEN	$min(\alpha_{src}*c_{src} + \alpha_{dst}*c_{dst}*(1-\alpha_{src}),$ $\alpha_{dst}*c_{dst} + \alpha_{src}*c_{src}*(1-\alpha_{dst}))$
VG_BLEND_LIGHTEN	$max(\alpha_{src}*c_{src} + \alpha_{dst}*c_{dst}*(1-\alpha_{src}),$ $\alpha_{dst}*c_{dst} + \alpha_{src}*c_{src}*(1-\alpha_{dst}))$

Table 6 Additional Blending Mode

A last blending mode exists. It is called additive blending. The blending equations are:

$\alpha(\alpha src,\alpha dst) = min(\alpha src + \alpha dst, 1)$

$$c(csrc, cdst) = (\alpha src*csrc + \alpha dst*cdst) / \min(\alpha src + \alpha dst, 1)$$

All these modes can be set or changed by simply using the vgseti function.

```
typedef enum {
VG_BLEND_SRC = 0x2000,
VG_BLEND_SRC_OVER = 0x2001,
VG_BLEND_DST_OVER = 0x2002,
VG_BLEND_SRC_IN = 0x2003,
VG_BLEND_DST_IN = 0x2004,
VG_BLEND_MULTIPLY = 0x2005,
VG_BLEND_SCREEN = 0x2006,
VG_BLEND_DARKEN = 0x2007,
VG_BLEND_LIGHTEN = 0x2008,
VG_BLEND_ADDITIVE = 0x2009
} VGBlendMode;

vgSeti(VG_BLEND_MODE, mode);
```

2.8.4 The VGU library

This optional utility library proposes some high-level primitives. The functions of this library return error codes. These error codes are enumerated in the defined type VGUErrorCode.

```
typedef enum {
VGU_NO_ERROR = 0,
VGU_BAD_HANDLE_ERROR = 0xF000,
VGU_ILLEGAL_ARGUMENT_ERROR = 0xF001,
VGU_OUT_OF_MEMORY_ERROR = 0xF002,
VGU_PATH_CAPABILITY_ERROR = 0xF003,
VGU_BAD_WARP_ERROR = 0xF004
} VGUErrorCode;
```

Geometric primitives

vguLine

This function appends a line segment to a path. (x0, y0) is the starting point of the segment and (x1, y1) is the ending point.

```
VGUErrorCode vguLine(VGPath path,
                     VGfloat x0, VGfloat y0,
                     VGfloat x1, VGfloat y1)
```

vguPolygon

This function appends a polyline or polygon to the given path. The coordinates of the points are given as input and the point number as well.

```
VGUErrorCode vguPolygon(VGPath path,
                        const VGfloat * points, VGint count,
                        VGboolean closed)
```

vguRect

This function appends an axis-aligned to the given path. Inputs are the starting point and the dimensions.

```
VGUErrorCode vguRect(VGPath path,
                        VGfloat x, VGfloat y,
                        VGfloat width, VGfloat height)
```

vguRoundRect

This function appends a round corner rectangle to the given path. The point which has to be specified is the left-low corner of the bounding rectangle. We also need to precise the size of the ellipses that model the corner.

```
VGUErrorCode vguRoundRect(VGPath path,
                              VGfloat x, VGfloat y,
                              VGfloat width, VGfloat height,
                              VGfloat arcWidth, VGfloat arcHeight)
```

vguEllipse

This function appends an ellipse to the given path. We just need to give the centre and the dimensions.

```
VGUErrorCode vguEllipse(VGPath path,
                        VGfloat cx, VGfloat cy,
                        VGfloat width, VGfloat height)
```

vguArc

This function allows us to append arcs to a path. Three types or arcs are defined. They are enumerated in the VGUArcType. VGU_ARC_OPEN is an arc segment only, while VGU_ARC_CHORD is the arc plus the straight line that binds the two endpoints of the arc and VGU_ARC_PIE draws the arc plus the lines from the centre to the endpoints of the arc. To draw an arc, whatever its type is, we need to set the centre, the width, the height, the start angle and the angle extent.

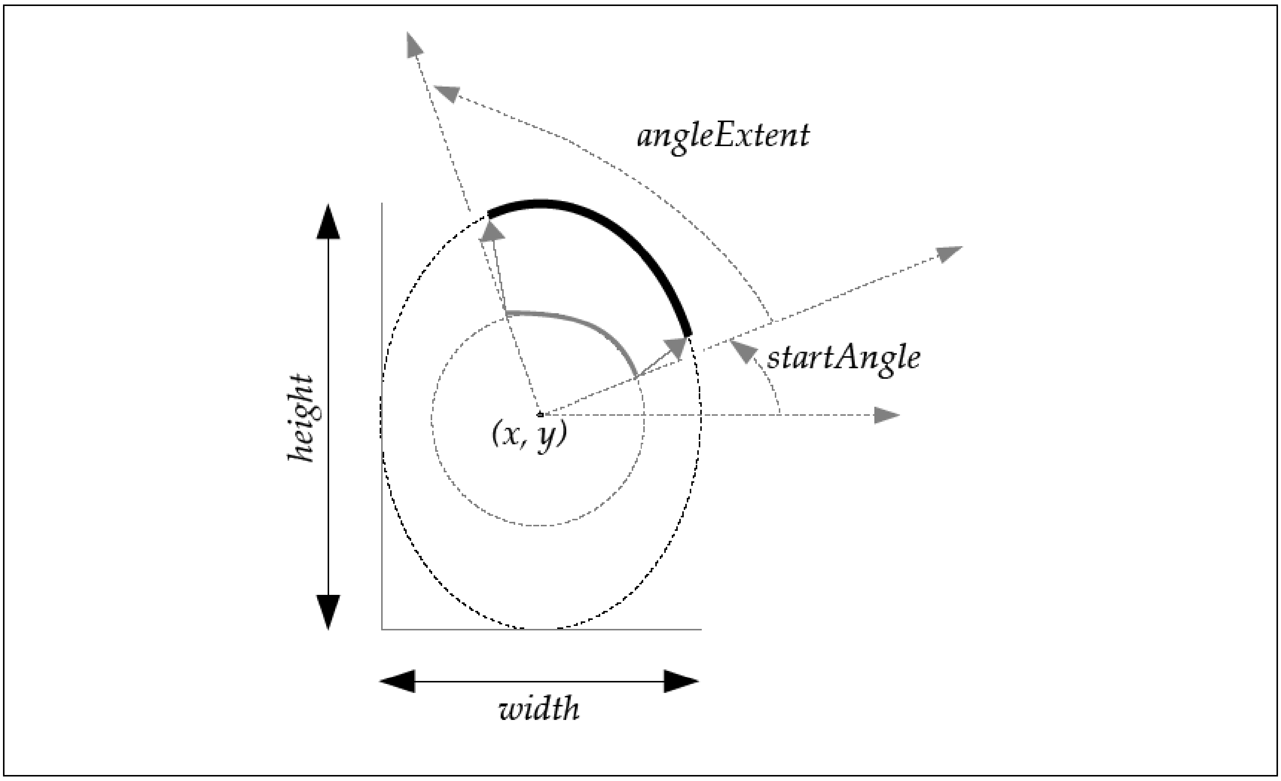


Figure 20 Arc parameters illustration

```
VGUErrorCode vguArc(VGPath path,
                    VGfloat x, VGfloat y,
```

VGfloat width, VGfloat height,
VGfloat startAngle, VGfloat angleExtent,
VGUArcType arcType)

Image warping

VGU also provides some more advanced tools for computing transformations.

vguComputeWarpQuadToSquare

This function computes the transformation matrix for the transformation which transforms (sx0,sy0) in (0,0), (sx1,sy1) in (1,0), (sx2,sy2) in (0,1), (sx3,sy3) in (1,1).

*VGUErrorCode vguComputeWarpQuadToSquare(VGfloat sx0, VGfloat sy0,VGfloat sx1, VGfloat sy1,VGfloat sx2, VGfloat sy2,VGfloat sx3, VGfloat sy3,VGfloat * matrix)*

vguComputeWarpSquareToQuad

This function does the same like the previous one but the transformation has the opposite effects. That is to say that (0,0) is mapped to (dx0,dy0), (1,0) to (dx1,dy1), (0,1) to (dx2,dy2), (1,1) to (dx3,dy3).

*VGUErrorCode vguComputeWarpSquareToQuad(VGfloat dx0, VGfloat dy0,VGfloat dx1, VGfloat dy1,VGfloat dx2, VGfloat dy2,VGfloat dx3, VGfloat dy3,VGfloat * matrix)*

vguComputeWarpQuadToQuad

This function still does the same as the previous ones but from one given quad to another.
That is to say (sx0,sy0) to (dx0,dy0), (sx1,sy1) to (dx1,dy1), (sx2,sy2) to (dx2,dy2) and (sx3,sy3) to (dx3,dy3).

*VGUErrorCode vguComputeWarpQuadToQuad(VGfloat dx0, VGfloat dy0, VGfloat dx1, VGfloat dy1,VGfloat dx2, VGfloat dy2, VGfloat dx3, VGfloat dy3,VGfloat sx0, VGfloat sy0,VGfloat sx1, VGfloat sy1,VGfloat sx2, VGfloat sy2,VGfloat sx3, VGfloat sy3,VGfloat * matrix)*

2.9 Summary

This chapter was the opportunity to discover all the possibilities offered by OpenVG and how to develop with OpenVG. But at first we had to understand the way it works. So we examined the pipeline process and studied the 8 stages. And then we explained some very basic principles for OpenVG and more generally for graphics like the colour spaces. Then we had to explore and explain EGL which is a native platform interface. This was also the opportunity for us to detail the notions of drawing context, drawing surface and EGLconfig. Then we fully entered in the subject by explaining some generalities about OpenVG like how to set parameters, the generic functions *vgsetParameter* and *vggetParameter*. We also explained the mechanisms of transformations. Then we were ready to develop the three big issues of OpenVG: the path, the paint and the image. For all of them, we tried to give a complete overview of the possibilities offered. And we finished by developing some more advanced features like the VGU library, the masking or the blending.

The first feeling we have when we consider what we now know about OpenVG, is that it looks like very interesting and that the possibilities seem to be infinite. We also wanted to discover what advantages artists would have to use this new standard, what they think about it, their reflections after using it. And we also wanted to study how innovative OpenVG is and why. That is precisely what we want to develop in the next part of this project.

3 Innovation and artistic opportunities in OpenVG

3.1 Goals

As we explained it earlier, although the main and overall subject is OpenVG, this project is actually divided in a master thesis and an internship. These two parts have distinct goals. While the internship side of this project concentrates purely on the realization of the benchmarking software, the master thesis is rather oriented towards the SART project. In the first and introductory chapter, we had explained the goals and principles of this project and how we wish to participate to this project by adding this project like a small stone in the building process. That is the subject of this chapter.

Indeed, since we are studying OpenVG, and since it is an open source project, we thought that it could be interesting to study the artistic opportunities offered by OpenVG, to survey what artists think about OpenVG ad to study what could help them to feel involved in OpenVG. We would also be interested in the part of innovation in OpenVG. After having considered numerous eccentric or hardly manageable solutions, we figured out that the best and most simple solution to achieve our goals was to interview artists concerned by OpenVG. In this context, art will be reduced to demo graphics.

To achieve these goals and to perform a good and useful interview we had to prepare this interview. That is why we will first detail the way we prepared this interview. Then we will expose which topics we want to explore through the interview and thus prepare the questions we want to ask. Then we will propose a summary of this interview. And then we will analyze the content of the interview.

3.2 Research method

In this sub-chapter, we want to explain how we decided to realize this interview and the methodology we want to use for both preparing and conducting this interview.

Process

As we explained it in the previous sub-chapter, in the context of the SART project, we want to investigate on the potential of OpenVG for artistic applications and the part of innovation it has. To do so, we thought that it could be interesting to collect information on that subject. That is how we arrived to the idea of realizing some kind of survey. The idea is to test the potential and attraction of OpenVG on artists and the relation between the standard and the artists. And we shouldn't forget to mention that we are also interested in spotting the part of innovation OpenVG carries. Then we had to consider what kind of information we want to collect. Indeed, the method for the investigation will vary depending on the type of the information we want to collect. In our case, we need to collect qualitative information. And our source of information is artists themselves. So according to the observations Siw Elisabeth Hove and Bente Anda made during their experiences [HOV05], in our case, the interview is the best way to collect the information we want to get. That is the reason why we are going to interview two artists who are working at COMPANYX. And in this context art will be reduced to demo graphics.

Methodology

Several types of interview exist. Since we have a precise idea of what we want to talk about, but we wish to get as much information as we can, we can say that the interview we want to realize is a *semi-structured* interview [HOV05]. Concerning the decision of realizing two interviews or one of both of them in the same time, we decided to make a “group” interview. Indeed, we thought that it would be more efficient to interview the two persons in the same time. The reason is that these two artists work together so they know each other quite well. Thus by interviewing them together, we would keep this friendly and known environment around them. So we think that, this way, they would feel more comfortable and thus be more encline to talk.

Concerning the interviewer, although this project is a single man's work, it would have been very easy to ask for some help to someone else if it was necessary. But actually, and still according to Hove [HOV05], if there are two interviewers, then there is a risk that it appears like a confrontation or a test. And then the risk to loose information due to a bad climax is important. The other point concerning the interviewer, that is to say me, it is really important that this person knows a bit the subject, so that he can understand what is said when goes a bit technical [HOV05]. But on the opposite, if the interviewer is a specialist of the subject, the interviewee may feel that he is tested by the interviewer who already knows all the answers to the questions he is asking. And then the problem would be the same as before. The interviewees would feel like tested and their probable reactions would be to feel uncomfortable and then to talk as less as possible.

We will detail the questions in the next sub-chapter but we can explain some tips for preparing questions now. There are many different types of questions but there are some that should be avoided like “why” questions and questions that can be answered by “yes” or “no” [HOV05]. It is also advised to start with informal questions and then to enter progressively in the subject. But overall there are some questions that should be handled really carefully. These questions are generally related to economical topics, opinions about colleagues or customers, analysis of why things failed and questions about the interviewee's own competences and mistakes. For this kind of very sensible questions, a big work shall be done by the interviewer in order to ensure the subject that the interviewer is on the same side as him. To do so, there are some techniques like employing “we”, “us”, “they” or “them”. These sensitive questions have to be asked late in the interview. And all the questions will be sent before to the interviewees before the interview so that they can prepare the interview as well. To finish, we want to audio record the conversations. But at first, we have to ask the interviewees if they agree with. Audio taping has the big interest that we won't loose any information. Thus we won't stress on that point and we will be more concentrated on the discussion. And that leads to a better quality of information.

3.4 Conversations with Gabor Papp

The first meeting with Gabor Papp, a Hungarian generative artist officing in Sweden, happened in Trondheim in October 2006 during the Maskin festival. He was exhibiting his work and since he was absolutely not involved in OpenVG, it appeared to be interesting for this research work to ask him about OpenVG and which perspectives he could see in this new standard in relation with his work. Because it was clearly hard to meet during this festival, we discussed online by using an instant messaging software.

Like many other artists, he started as a demo graphics artists with some friends in the early nineties working in assembler and C. Working later as computer science engineer, he dropped his job to start again some studies and to study art and he was more interested in generative art. Although it is rather hard to give a short definition, it could be summed up by art which is generated by some relatively autonomous systems, very often computing systems. He is now working in a research institute and besides creating some pieces of art he also helps others in their work. In his work, he has a strong affinity with mathematic objects such as fractals, rules based systems and neural networks. He likes to explore technology. He is working in most of the cases on the Linux platform and programming in C/C++ but he doesn't really use APIs. As he is working on a new open standard called Freeframe and which is real-time video plug-in, he is also getting interested in hardware acceleration and mobile phones as a platform. Therefore, after explaining what OpenVG is and which benefits it could have for his work, he became very interested in.

Indeed, because he is interested in working on mobile platforms and because vector graphics are perfect for objects like fractals, which Gabor Papp particularly likes, since he can zoom on their infinite patterns without loosing quality, OpenVG appears to match perfectly its technological needs. But after explanations from me and deeper documentation on his side, and although his interest grew bigger, he expressed his disappointment about the fact that a cross platform implementation doesn't exist yet. Indeed, there only exist an implementation for the Windows platform. All the other implementations are included in chips so for the developers who would be interested in OpenVG and who don't work on Windows, it can appear as a bad point. That was his only regret.

The questions of the discussions can be found in the annexe 3.

3.5 Synthesis

To understand the content of this research work, we have to get some distance with the different experiences. And then we make the observation that through the experiences of Mr E and Mr T, Gabor Papp's one and mine, we have a broad range of experiences. And this range of experiences allows us to have an objective but passionate opinion, and to drive a complete analysis of the artistic opportunities offered by the standard and the standard itself from an artistic point of view.

Through Gabor Papp's interview, we have the point of view of someone who is an artist and present in the scene for a long time but who has absolutely no experience and who even never heard about OpenVG. So his experience is not interested to know what can be done with OpenVG and how far you can push it but his reaction reflects the potential of interest of OpenVG for artists. It comes from the dissemination that mobile platforms represent for them. It might also come from the fact that in general, these artists are interested in their art but also in technology. And like Gabor Papp, their artistic research also involves a strong affinity for technology and the wish to explore new technologies. He could also point out the problems and deficiencies of the standard up to now when it comes to be attractive and accessible.

On the opposite, Mr E and Mr T, with their very strong experiences and important commitments to OpenVG, didn't reveal anything concerning the attraction of the standard for artists. But they showed us, through the combination of their big experiences and their knowledge of the standard, what can be done with the standard. They also revealed ways for the development for artistic contents for OpenVG. And in the same time they revealed what was missing in the standard and tracks for future development of the standard like shading or multi-API management.

My own experience appears as an in between position. These 7 months of everyday work with OpenVG gave me a short but intensive experience and this in different topics around OpenVG like the standard and the way to use it but also the position in the market and the attraction for artists. My first reaction is that OpenVG comes to fill in a lack for a 2 dimensions vector graphics in the mobile devices sector. This position in the market, added to the proximity with OpenGL and its success plus the huge dissemination network which will support it, led to the attraction we have been the witness through the interviews. These interviews also gave me the complement I was missing besides my development experience. Indeed, my lack of experience in computer graphics made that it was hard for me to get deeper in this technical world but made of trends.

4 Performance evaluation

4.1 Introduction

The progression in hardware during the last 40 years has been exponential. And if it was quite easy to measure performance in the early years of computing, nowadays performance evaluation in computing can become a very complex problem which became a whole discipline. Indeed, several factors make it hard. At first, the range of technologies is now very broad and two systems can be really different in the technology used for their conception. The other problem is that it is very often the case that one system has a different behaviour and thus level of performance depending on the application. The other problem is that computers and computing system became such a big market involving huge quantities of money. So any performance evaluation tool can have huge financial effects. That is the reason why it is very important to communicate around the method used by the performance evaluation tool. That means to say what is evaluated, how and in which context, for which application type and why these choices have been made. The aim is to make sure that there is no doubt about the methodology and the technique used. Because otherwise a doubt can generate mistakes in the analysis of the results and then have important consequences. Of course, the independence of the evaluation tool from the manufacturing firms is very important. Performance evaluation has several aims. Of course, one of the most known aim is to be able to compare systems in order to make purchases. Indeed, the firm which is about to make a big purchase will want to know which system fits the best its needs. But it can also be for checking the capability of a system to run an application. And performance evaluation also plays a role in the conception of the hardware. In effect it is a very good tool for engineers for getting feedback and to control the quality of what they achieved in order to improve the systems they work on.

The aim of this part is to propose an initiation and overview of the theory dealing with performance evaluation. To do so we will consider at first the problem of the metrics. We will propose a definition and explain the goals and the reason to be of the metric. Then we will detail the characteristics of a good metric and we will show some examples of metrics. We will go further by bending over the problems coming around the measurement. We will talk about errors and mathematic tools for the analysis of the raw values. And the last sub-chapter will be dedicated to benchmarks which are a method among others for assessing performance. We will more precisely give a definition, detail all the kinds of benchmarks and all the strategies possible in benchmarking.

4.2 Metrics

4.2.1 Definitions and goals

When we speak of performance evaluation, we are talking about evaluation. It means that we need to measure the performance. So we have to know what is interesting to measure. And from these measurements, we will derivate a data that will be called a performance metric. Typically, these measurements are of three kinds. We can measure a count of events, the duration of a time period and the size of a parameter. We are often interested in the number of occurrences of an event per time unit. This kind of performance metric is called throughput or rate metric. The experience showed that choosing a metric won't be that easy and in all the cases, this choice will depend on the context. [LIL00] And we should not forget that some metrics that are used for some other technical purposes may not be appropriate for performance analysis and the opposite is also true.

4.2.2 Characteristics

As we started to show it just above, choosing the good metric can be a difficult task. All the problem with the metrics is that we want them to minimize the error risk, in particular interpretation errors. Indeed, there is already enough potential sources of errors to add another one and as we explained in the previous sub-chapter, errors in performance evaluation can have big consequences. And although there is no proven theory to do so, the empirical experience in performance evaluation gives us some characteristics for a good performance metric. These characteristics have to be interpreted as necessary but not sufficient. They are the following:

Linearity

It means that there must be a proportional relation between the performance and the metric. This characteristic comes from the fact that human beings, and over all their brains, are used to think in a linear way. The aim is by having a metric that respects the way the human brain tends to work, we limit the risk of error while interpreting the results.

Reliability

A reliable metric that when a system A has a higher value for the metric than system B, then A will always outperform B as long as it is for the same conditions of use.

Repeatability

A metric is said to be repeatable when the same value is measured every time it is measured in the same conditions.

Easiness of measurement

This could appear as superficial for a metric, but it is necessary. Indeed, a metric which is not easy to use won't be used or even worst, it will generate some interpreting or measuring errors.

Consistency

A consistent performance metric is a metric for which the unit has the same definition for every system. The consequence of an inconsistent system is quite obvious: it is impossible to do any comparison between systems. And thus either the metric is not used either it leads to wrong conclusions, which is even worst.

Independence

This characteristic appears rather as obvious when we know the consequences on the market that a performance evaluation tool can have. So the metric has to be manufacturer independent otherwise in won't be used.

4.2.3 Some metrics

Now that we discussed what a good metric is and its necessary characteristics, we can detail some few performance metrics.

Clock rate

The clock rate, which defines how many cycles are executed per second, is very often the performance characteristic which is announced for a computer while it is a bad metric.

Indeed, it doesn't say how much computation is executed per machine cycle. So the metric is not reliable. And it doesn't take in account the fact that processor may not be the bottleneck of the system or the processor could be the bottleneck but have some input/output problems. So because of all these shortcuts, this metric is clearly not linear.

MIPS

MIPS stands for Millions of Instructions Per Second. And as it says it is the number of instructions executed by the processor per second in millions. The problem with this metric is that it depends on the processors and their instructions sets. Indeed, a single instruction will be interpreted differently depending on the processor, translating it in differently sized computation loads. So the metric is not reliable and not consistent. And like for the clock rate, since the processor may not be the bottleneck, the MIPS metric is not linear.

MFLOPS

MFLOPS stands for Millions of Floating-point Operations Per Second. As the name says and by analogy with MIPS, it counts the number of float operations executed per second. Although this metric corrects one problem of the MIPS metric, it also introduces a new problem. Indeed, a system which doesn't perform any float operation can be very powerful but will get the value 0 for the MFLOPS metric. Thus it is not reliable.

SPEC

SPEC, which stands for System Performance Evaluation Cooperative, is a manufacturer association. They developed their own metric. It is computed by running a set of benchmark programs and measuring their execution times. These times are normalized by the standard times and are summarized by a geometric mean. Although this method is still in use and appears as a reference [APP06], the metric is not perfect. Indeed, it is obviously not linear and it appeared to be not reliable. [LIL00]

QUIPS

QUIPS stands for QUality Improvements Per Second. QUIPS measures the quality of a solution by applying a mathematic formula. The metric is thus independent, consistent, repeatable, and linear [LIL00] but the problem is that it is not necessary reliable [LIL00].

Execution time

The execution time can appear as the most simple and unsophisticated metric developed but it remains a very good alternative. The principle is quite simple. A benchmark program is run and we measure the execution time. And the most performant system is the one with the shortest execution time. This metric is linear, consistent, reliable, easy and independent. But because of operating system issues like cache misses, input/output, the execution time is not repeatable. But it can be easily corrected by calculating a mean.

4.2.4 Discussion

Speedup and relative change

Speedup and relative change are some simple mathematic tools for comparison. Speedup is the ratio of the two metric values of the two systems compared. And thus when the metric is a throughput, then the speed up becomes the inverted ratio of the execution times.

$S = R2 / R1 = D/T2 / D/T1 = T1/T2$ where R is the metric and in the case of a throughput with a fixed amount of computation D, $R=D/T$.

The relative change is defined as follows:

$\Delta = (R2 - R1) / R1 = (D/T2 - D/T1) / D/T1 = (T1 - T2) / T2$

Means and end metrics

A particular attention should be paid to the choice of the metric according to the context. Metrics can be divided in 2 categories: mean and end metrics. The first ones measure a progress towards a goal while the second ones measure exactly what has been done. For example, a throughput will consider all the workload executed whether it is useful or not while an execution time will measure the exact time that has been necessary for executing the workload and it considers only what has been done and useful. This difference can lead to huge errors if we haven't chosen the appropriate kind of metric [LIL00].

4.3 Measurement and analysis

In the previous sub-chapter, we have been studying the metrics and all the quality issues around them. In this sub-part, we want to study the measurement and the treatment of the data for a good analysis by the performance analysts. In a more concrete way, it means that we will study the different risks and sources of error while measuring, including the measuring device. Then we will detail some mathematic tools to sum up raw data. That is to say the different means and the variability.

4.3.1 Errors

Although we want a performance evaluation to be as close and precise as possible, it remains to be an experimental process. And as all the experimental process, it includes the very mighty risk of errors in the measurement. And that is the reason of the following discussion: to point out and explain the sources of errors in the measurement. To do so, it is necessary and interesting to explain the characteristics of a measuring device.

Measuring device

As we just explained, the measuring device can be a source of error also called noise. Indeed, it has its inner characteristics that have an influence on the quality of the measurement. These characteristics can be hardly avoided but limited by just knowing their existence.

The first one of these characteristics is the **accuracy**. Accuracy is defined as the absolute difference between the real value and the mean of the values measured by the device. This difference shall increase a bit with the age of the device.

The second characteristic is called **precision.** It deals with the repeatability of the measurements. That is to say, the precision is relative to the probability to get a similar value for 2 measurements. The following figure [LIL00] illustrates these two characteristics on a repartition graph.

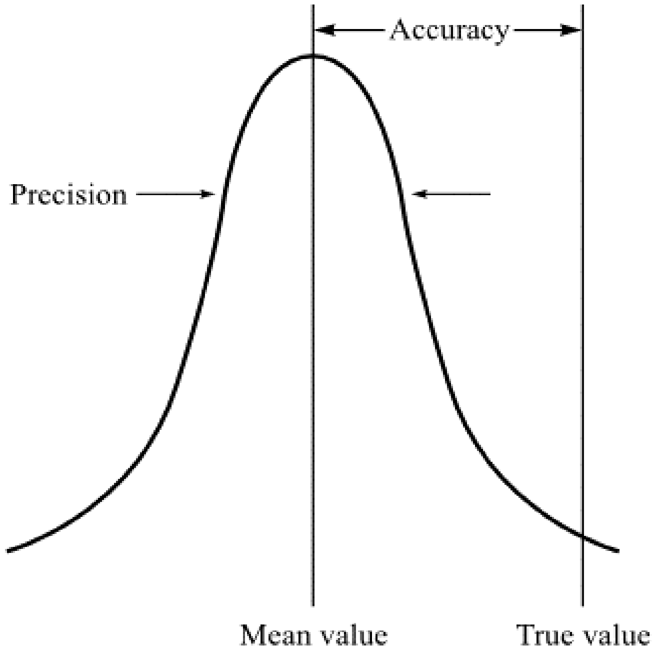


Figure 21: Illustration of the inner characteristics of the measuring device

The last characteristic of the measuring device is the **resolution**. It is the smallest difference that the device can measure between two distinct measurements.

Other sources of errors

The measuring device is not the only source of perturbations. Many perturbations can come from the system itself. Indeed, the program tested can have its behaviour modified by events such as user interaction, system interruptions but also more non-deterministic events like cache misses, input/output, exceptions. More globally, the errors and their sources can be classified in two groups: **systematic** errors and **random** errors. Systematic errors are mainly due to mistakes while measuring. It can be errors in the process, wrong conditions, or any other kind of errors that typically can be avoided. Random errors are, unlike systematic errors, completely unpredictable. They are very hard to avoid.

4.3.2 Mathematic tools

In the previous sub-chapter dealing with metrics, we saw that metrics like execution times are not repeatable but that it can be fixed by simply considering the mean instead of the raw data. For this reason and also because means are quite useful to summarize raw data, it is interesting for us to have a short overview of the existing theory for mathematic tools such means and variability.

Means are useful for numerous reasons like making a metric repeatable or summarizing data since in general people tend to prefer one number for comparing performance. But means are useful only if we choose the good mean otherwise the result could be false and lead to wrong conclusions with the effects we know. In the rest of the sub-chapter we will focus a bit more on the execution times and execution rates. According to David J Lilja [LIL00], a good mean for summarizing execution times shall be proportional to the sum of these times. And a good mean for summarizing the execution rates shall be inversely proportional to the sum of the execution times. Now that we know these theory points, we can propose an overview of the different means.

Arithmetic mean

This mean is from far the most known mean. It is defined as the sum of the values divided by the number of samples like follows:

$$\bar{x}=\frac{1}{n}\sum_{i=1}^{n} xi$$

So we can very easily see that the arithmetic mean is appropriate for execution times. Now if we consider the execution rates defined by x=F/T, then we get the following result:

$$\bar{x}=\frac{1}{n}\sum_{i=1}^{n} F/Ti=\frac{F}{n}\sum_{i=1}^{n} 1/Ti$$

And thus it is not appropriate for execution rates since it is not inversely proportional to the sum of the execution times.

Harmonic mean

This mean is defined as the ratio of the number of sample and the sum of the opposites of the values like follows. So it is definitely not appropriate to execution times.

$$\bar{x}=\frac{n}{\sum_{i=1}^{n}\left(1/xi\right)}$$

And if we consider the case of the execution rates, we get the following results:

$$\bar{x}=\frac{n}{\sum_{i=1}^{n}\left(Ti/F\right)}=\frac{Fn}{\sum_{i=1}^{n} Ti}$$

So the harmonic mean is definitely appropriate for summarizing the execution rates.

Geometric mean

This mean is defined as the n root square of the product of the samples. That is to say like follows:

$$\bar{x} = \sqrt[n]{\prod_{i=1}^{n} xi}$$

And although the SPEC benchmarks use this mean for summarizing the normalized execution times, we can see very easily that it is appropriate neither for the execution times nor the execution rates.

Variability

Variability can't replace the means. It is rather a complementary tool. The aim is to give an idea of how the repartition around the mean of the measurement values is. The absolute difference between the mean and the most extreme value. But then it is too much sensitive to the extremes. So it is necessary to use the following formula:

$$s = \sqrt[2]{\sum_{i=1}^{n} (xi - \bar{x})^2}$$

4.4 Benchmarking

As the title says, this sub-chapter is more especially dedicated to benchmarking. We want to propose a definition of benchmarking in computing, then we want to propose an overview of the different types of benchmarks. And we will finish by explaining all the different benchmarking strategies.

4.4.1 Definition

Benchmarking is a single kind to evaluate performance of computing systems. The aim is to assess the performance of a system, device or object by running tests and trials against it [WIKBE06] in order to simulate its behaviour. Benchmarking applies to hardware and to software as well. But very often, benchmarking consists in evaluating the performance of a given hardware system in a specific use in order to compare systems or to check the viability of a system for the specific use. The principle of benchmarking comes from the observation that the best way to evaluate a performance was to measure it on a working system. Usually, benchmarks run a high number of tests. [LIL00] [WIKBE06]

4.4.2 Types of benchmarks

These are the different benchmark programs that have been developed by performance analysts.

Single instruction time

At first, analysts thought that, since almost all the instructions of a system have a similar execution time, to know the execution time for a single was enough to evaluate the performance. To do so, they chose the addition instruction to compare machines.

Instruction-execution times

This method, proposed by Jack C. Gibson to face the evolution of the systems differencing the execution schemes for instructions, consists in classifying instructions by the number of clock cycles they need to be executed. Then a CPI metric is calculated as a weighted mean of the relative effectives of each class and the number of clock cycles they need to execute. Then it is easy to calculate the execution time for a program:

$$\text{Tprogram} = \text{N} * \text{CPI} * \text{Tclock}$$

where N is the number of instructions of the program and Tclock the duration of a clock cycle. The problem with this method is that the CPI depends directly and only on the set of instructions executed. And this number of instructions depends on the system. It also doesn't consider the input/output and delays.

Synthetic benchmark programs

Synthetic benchmark programs is exploiting the idea of Gibson's instruction mixes by executing a well chosen program with a representative set of instructions of the application instead of the application itself. So it can appear as the complement of instruction mixes. But this method skips the problems caused by special instruction orderings and details of programs that are specially resource consumers and determinants of the performance.

Microbenchmarks

This type of benchmarking aims to evaluate the performance of a specific component of a system by considering the element benchmarked as the performance bottleneck of the system.

Program kernels

As the name says, program kernels are benchmarks that focus especially on the most resource consuming loops of the application. The small size of these kernel benchmarks makes them easier to port. But their problem is that they tend to forget some major aspects of the system like the operating system or the memory organization and the problems that come with.

Application benchmark programs

These benchmarks are actually real « useful » applications or collections of « useful » programs. Indeed they produce a result unlike kernel and synthetic benchmarks. So they don't skip the determinant elements of real applications but because they are smaller, they are much easier portable. Using a collection of programs allows us to focus each time on a specific part of the system.

4.4.3 Benchmark strategies

There are globally three different strategies in benchmarking: fixed amount of computation, fixed period of time and a mixed mode where both time and computation are not limited. We want to detail a bit deeper these three strategies.

Fixed computation

As the name says, this strategy implies that the benchmark runs a defined and fixed amount of computation and we measure the execution time. Although that the execution time makes the difference between the systems, the metric which is actually calculated is the execution rate. But the benchmark will actually not be the same amount of computation depending on the system on which it is ported. So we have to make an approximation which is that the amount of computation is the same on all the system tested.

Fixed-time benchmarks

This strategy is the opposite of the one described just above. It consists in measuring the amount of computation executed in a fixed amount of time. This strategy especially fits the case of applications that are used to run in a defined amount of time and looking for executing the biggest load of computation. And as a consequence, it is perfect for comparing systems within a very wide range of performance levels since the execution time stays the same but only the result changes. But the problem is that this kind of benchmarks just propose a propose a problem but it doesn't take in account the quality and complexity of the solution. Indeed, in that case, a system solving the problem with a cleverer algorithm will be equal in performance as the one with a less clever algorithm.

Variable computation and variable execution-time

In this type of strategy, neither the computation nor the execution time are limited and measured. But instead of, a solution quality is mathematically defined and taken as the metric as well as the execution rate is.

4.5 Summary

As a necessity, in this chapter, we discovered the theory existing around performance evaluation and more especially benchmarking for computing systems. We developed the notions of metric which is the basis when we deal with the notion of performance. And we discovered that choosing a metric was of course not the fruit of chance but the result of the application and checking of precise characteristics that make a good metric. We also developed and explained mathematical tools for analyzing raw measured values to end up with a qualitative discussion about measurement errors. Then we started to study more precisely benchmarking by giving a definition and explaining the particularity of this performance evaluation method, detailing the different kinds of benchmarks developed and their utility to finish with an overview of the different strategies of benchmarking. All long this chapter, we never precise what kind of system we were talking of and what kind of application we were dealing with. This chapter was actually an insight in performance evaluation theory. Starting from now, we are going to conceive benchmarks for OpenVG, which implies a very precise kind of system tested, which is embedded devices, and a very single and particular application type, which is graphics. So we are going to conceive our benchmarks for this particular case by applying the very broad theory we just studied.

5 First benchmark iteration

This part is dedicated to the first iteration of the OpenVG benchmark. That means that this first realization of our program is definitely not complete compared to the final version and probably content imperfections or defaults. This first iteration is not that easy to realize in the sense that although we have some guidelines and that it is not supposed to be perfect, this first step is always a bit hazardous. And that is the reason why we will implement this benchmark in several iterations. Indeed, the aim is to get as much feedback as possible after finishing an iteration. This feedback is necessary to improve and will be included in the next version. The final aim is to get each step a bit closer to the final version and hopefully to the expected result.

5.1 Goals

For this first version of our program, we have to limit ourselves to a set of features to test. Indeed, since we are not sure about the way to proceed, it is better to focus on a defined set and to try to work on this small set of features. So for this first iteration we chose to focus on curves, which is not the smallest category of features anyway. So we will test all the types of curves and with all the lengths. It is also important to test both the absolute and relative coordinates' curves commands.

The aim is to test how fast the implementation draws these different types of curves. It means that we have to test both filled path mode and stroked path mode although that we can already say that the stroked path mode shall probably be a bit longer to compute and thus to draw. Here again, in the case of the stroked paths, we want to test a wide range of widths.

Concerning the paints, since we will focus on the path, we need to use only the simplest type of paint available, that is to say solid colours. We will try to vary also the colours.

Another important point to stress is the retesselation. That is to say how easily the platform tested can handle the transformations and to redraw the paths in the case of curves, since that is what we want to test. Thus we will try to apply all the different types of transformations to the paths we draw.

For this first iteration and after having studied computer performance evaluation and benchmarking, we understood that the execution time for a fixed amount of computation is one of the best metrics as long as the number of measurements is high enough to make it repeatable.

5.2 Implementation

The tests

The idea of this first iteration of the benchmark program we want to develop is to use randomness to test a broad range of lengths, colours, and shapes of curves. So the principle is to draw all the types of segments between fixed points of the screen by simply associating them with randomly generated numbers passed as parameters. To do so and to maintain repeatability and portability, it is necessary to include a random number generator in our code. We decided to use a uniform and Mersenne type number generator published by Agner Fog [FOG05] under GNU general public licence.

The program starts by drawing in filling mode a path of four quadratic curves between the points (100,100), (300,100), (300,300) and (100,300). Two segments are in relative coordinates and two are in absolute coordinates. All the other parameters such as colours, alpha channel, and coordinates are randomly generated. Then we apply a random translation and we redraw it. We do the same with a random scaling transformation then with a random shearing transformation and with a random rotation. Then this sequence of 5 paths drawn is repeated 50 times. That makes a total of 1000 segments drawn for the quadratic curves. The figure 1 shows an example of what is drawn.

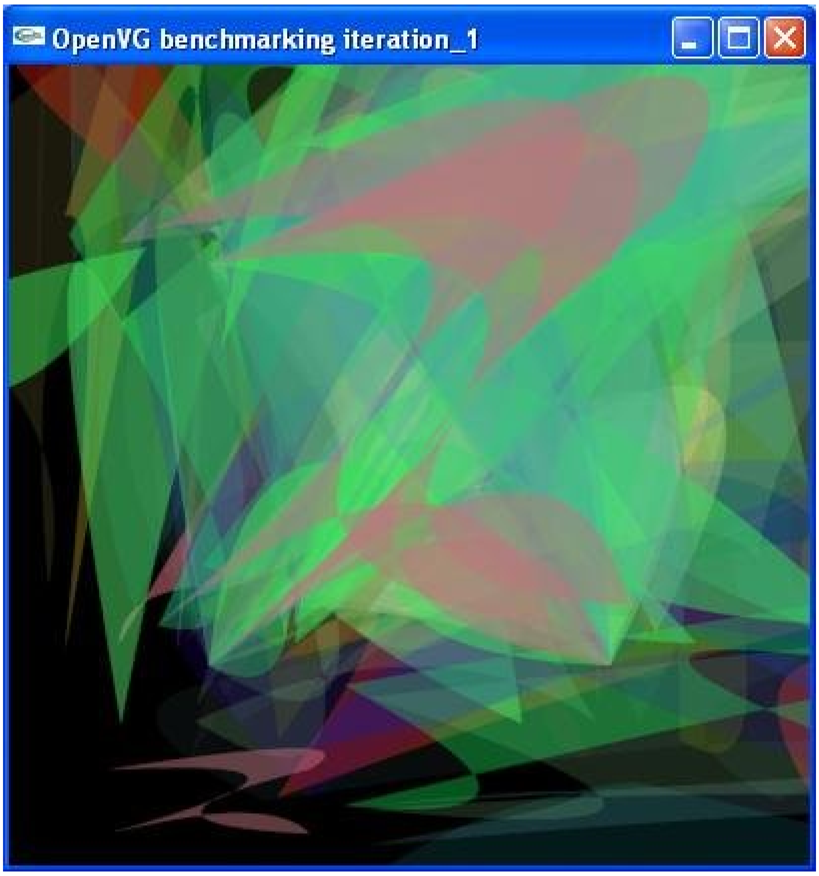


Figure 22: Quadratic curves benchmark for filled mode

Then we restart this previous sequence with cubic curves. The basic path and its four contact points remain unchanged. So a total of 1000 segments drawn for cubic curves. Then we repeat this with the elliptic arcs. Both clock wise and counter clock wise senses are tested. We also test both large and short segments and of course we use both absolute and relative coordinates. So the basic path becomes an octagon that fits the points (100,100), (200,50), (300,100), (350,200), (300,300), (200,350), (100,300) and (50,200). All the other parameters remain unchanged. And we draw this path and its transformed versions 50 times. That makes a total of 2000 segments drawn for the elliptic arcs. We also want to test the G1 Smooth connections, so we repeat the previous step by drawing a new basic path of 5 segments that is to say a basic quadratic curve to start and a sequence of G1 smoothly connected segments alternating cubic and quadratic curves, absolute and relative coordinates. And we draw this path and its 4 transformed versions 50 times. That makes a total of 1500 segments drawn for the G1 smoothly connected curves and a total of 5500 curves.

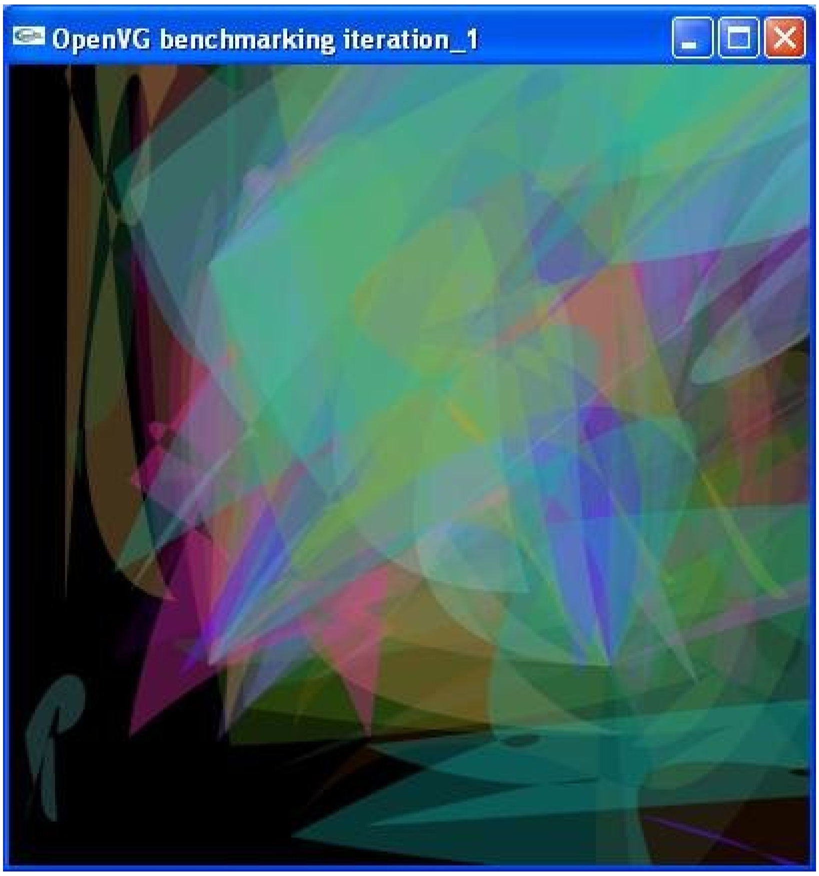


Figure 23: Cubic curves benchmark for filled mode

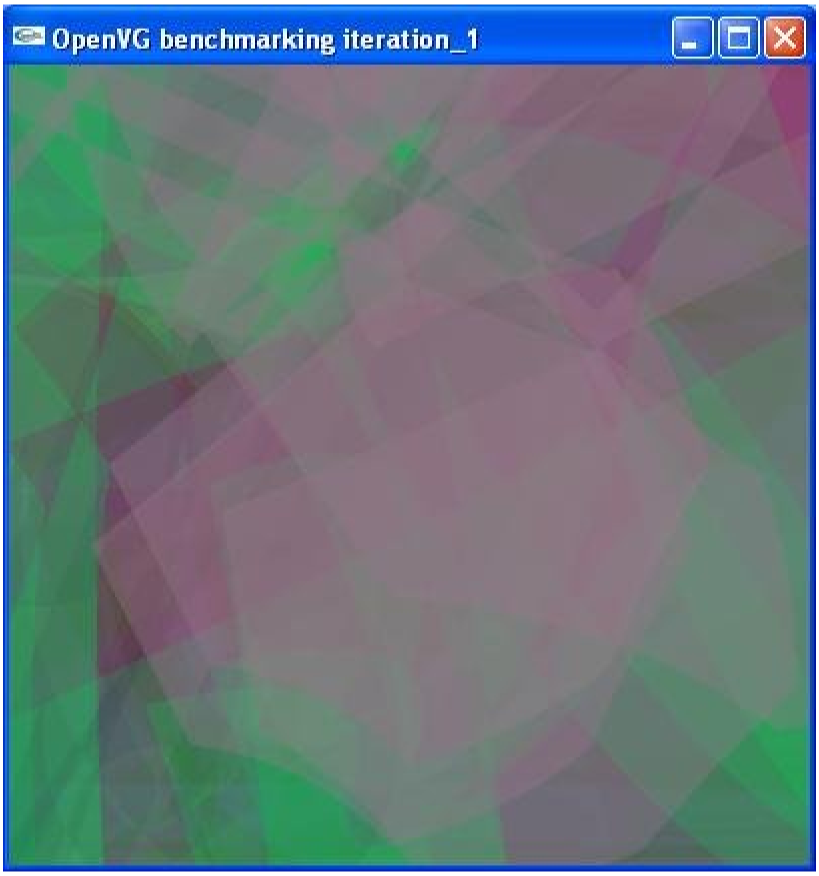


Figure 24: Elliptic arcs benchmark for filled mode

Figure 25: G1 smoothly connected curves benchmark for filled mode

To complete the tests, we have to repeat all these steps but with a stroke path mode. A randomly generate number defines the width of each path. That makes a total of 11000 segments drawn.

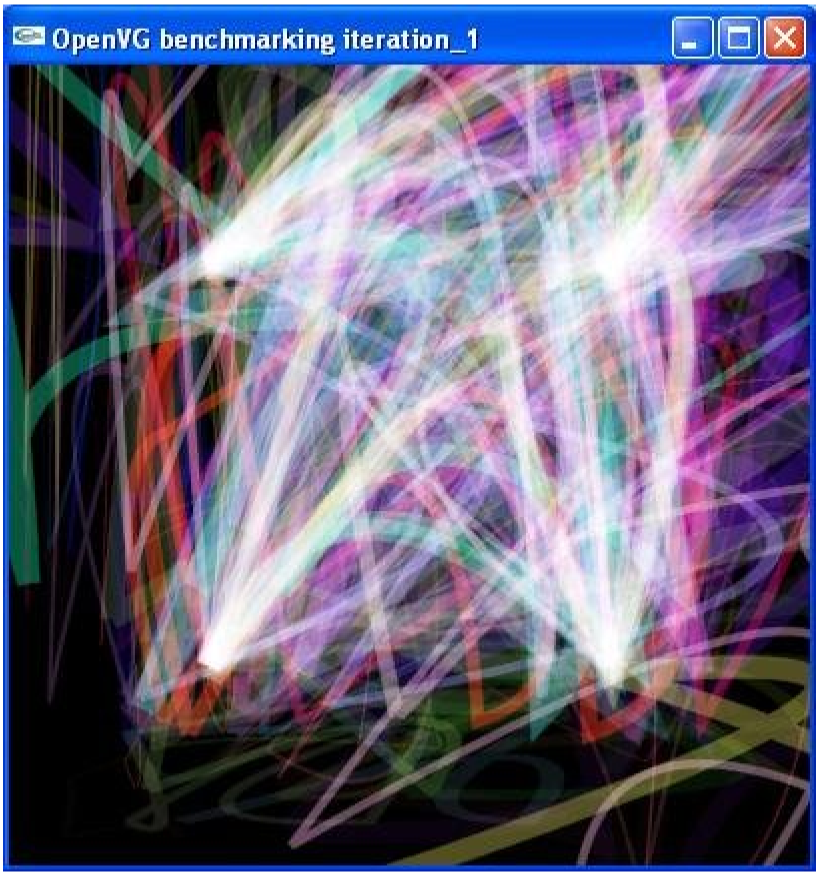


Figure 26: Quadratic curves benchmark for stroked mode

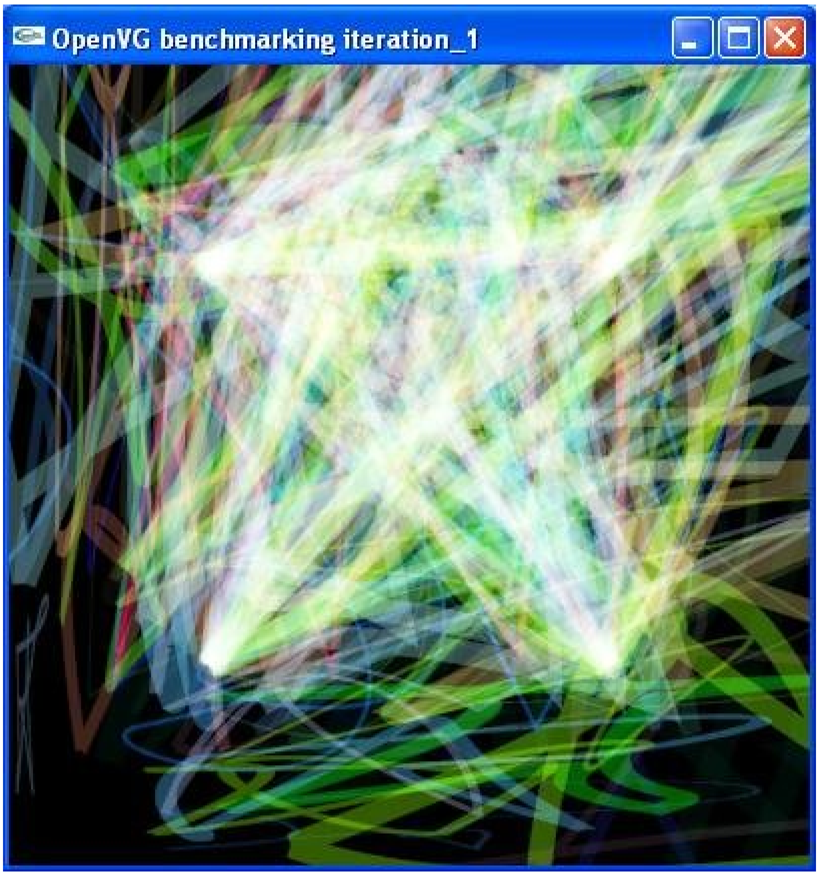


Figure 27: Cubic curves benchmark for stroked mode

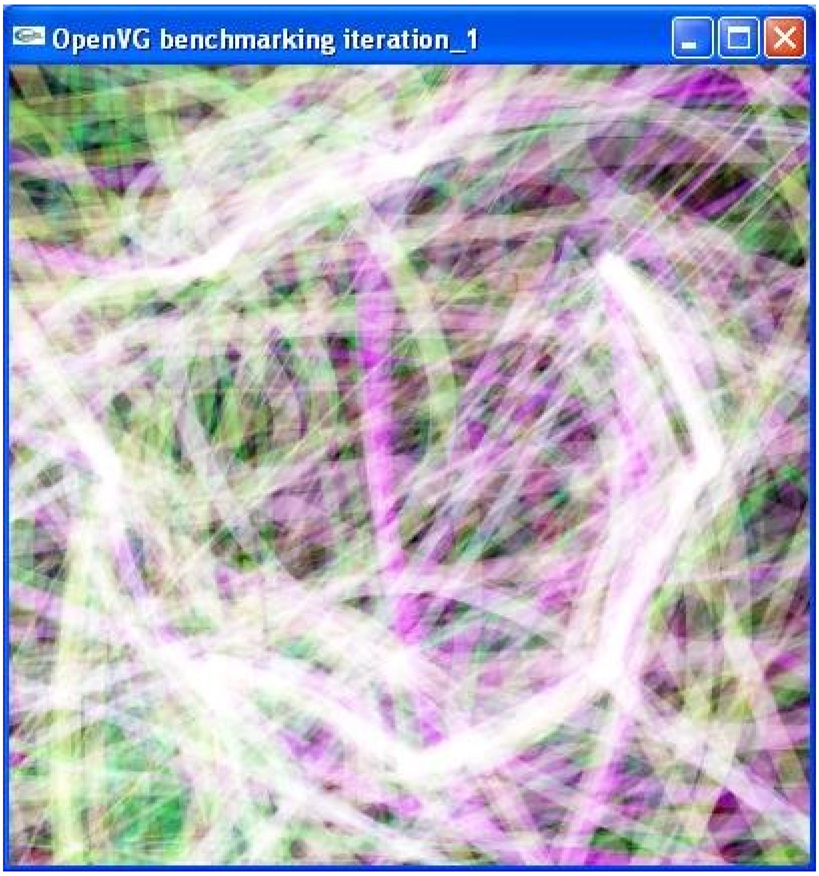


Figure 28: Elliptic arcs benchmark for stroked mode

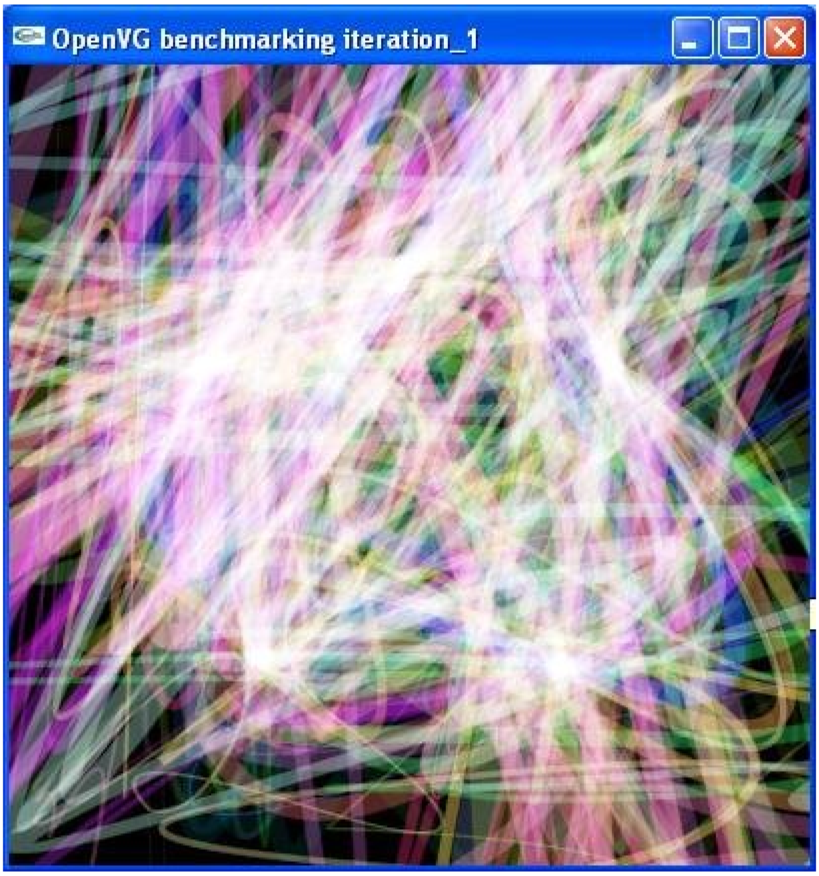


Figure 29: G1 smoothly connected curves benchmark for stroked mode

We measure the execution time by calculating the difference between two getTickCount commands. The end tick count value is measured after the swap of the buffers. We measure execution times for every type of curve and for every paint mode.

The files

iteration.c

This file, which contains the main, mainly holds all the glut and egl commands to set all the display and windows parameters and it calls the different functions that draw the different type of curved path.

Itertypepaint draws by calling the appropriate curve types drawing functions with the appropriate paint mode and swapping the buffers before to compute the execution times. One call to this function with one paint mode will draw all the paths of all the types for this paint mode.

`Mainloop` just calls itertypepaint for both paint modes and displays the execution times.

random_tools.c

This file holds the call to the random generator number. And it can be done by simply calling the getfloat function defined in this file.

quad_1.c, CUBIC.c, ellipses_1.c, G1Smooth.c

These 4 files draw the basic path defined sooner and its 4 transformed versions for respectively quadratic, cubic, elliptic and G1 smoothly connected curves. The functions that have to be called with a paint mode in argument by itertypepaint in iteration.c are respectively quad_1, cubic_1, ell_1, and g1.

5.3 Problems encountered

The first problem we encountered was to draw the paths one by one. Indeed, at first, they were only drawn every 50 iterations of the basic steps. To fix this, it was necessary to draw only the single step in the functions to be called by the itertypepaint function and to call these basic step functions 50 times. The big difference is that after the call to the basic functions, quad_1 for example, we swap the buffers to display it on the screen.

The random number generator also caused big troubles. Indeed, to be platform independent, our benchmark can’t use the random functions of the C library because it is directly depending on the platform. So it can influence the results and it can also make the results re not repeatable. That is the reason why we need to include a generator in the code. This caused some problems of initialization and other problems of utilization which have been solved by a deeper documentation.

When we introduced retesselation, at first we just simply wanted to change the transformation matrix and then to call for the appropriate basic function but then we wouldn’t test retesselation since we wouldn’t retessel the last path drawn but another one. So we had to include the transformations in the basic functions by just modifying the matrix redrawing and then to modify again the matrix before to prepare the next transformation.

The last and not least problem we had to face was the introduction of the execution time measurements. To do so, we decided to use the function getTickCount before and after each section of the tests and to compute the difference. The main problem comes from the point in code where to put the stop GetTickCount function. After a very carefully documentation, it appeared that the ending value had to include the return of the buffers’ swap. We also wanted to exclude the random generation of the numbers of the execution times but we still haven’t found a nice way to achieve it. That leads us to the next sub-chapter.

5.4 Feedback and observations

The first reactions after the demonstration of the first iteration were globally good but some points have to be improved in the next iteration. That is the topic of this sub-chapter.

The first big point which has to be improved is the metrics. Although the execution time is a good metric, it would be better to compare the results with a reference execution time. It would also be interesting to define a confidence interval and when a result falls outside this confidence interval, it shall not be considered and the value for the concerned test has to be measured again. Some statistics about the measurements shall also be introduced. But another metric shall also be considered. Indeed, it is also interesting to use the number of segments drawn per second. And finally, the benchmark shall compute a mark based on these criteria.

Another point that can be improved is the exclusion of the random generation of the measurement of the execution times. It could be done by simply filling in an array of the values needed and to exclude this step of the measurement.

Some modifications should also be brought to the settings of the display like the possibility to change very quickly the type of screen by just modifying two constants. It shall support the WVGA, VGA, QVGA, QCIF and CIF formats. It has also been suggested that the benchmark shall propose several levels of verbose, typically 3 levels.

The next iteration shall also include the possibility to dump frame buffers in order to keep a track of tests. Indeed, this would allow us to save an image of the screen. This feature would allow the developers who are using the benchmark to compare the resulting image with what it is supposed to draw. Then they can test the implementation they are developing.

To finish, after a concerted discussion with the persons following the project at COMPANYX, it has been agreed that the next iteration will extend the tests to all path segments and will also provide tests for image manipulations and image filters.

6 Second iteration

Initially, there was supposed to be three iterations in the development process. But due to a probable lack of time, this plan has been changed. We preferred to release a very complete second and last iteration. This decision lead to a much longer iteration concerning the second one and there have been several modifications after receiving some simple and low level feedback. We will mention but not detail all the modifications which have been done all long this second iteration.

6.1 Goals

As we explained it, this second iteration is based on two different goals. The first one is to correct the bad points and problems which have been pointed at in the first release. The second goal is to develop a much more complete set of features tests.

Among the details that had to be corrected from the previous version, there were the following points. It was important to exclude the random generation of the parameters from the measurement of the execution times. We also want to include the possibility to change easily the format of the destination screen or window. It has also been said that it would be nice if we could manage to dump buffer and to store them into files. The aim is to be able to compare the results obtained and to keep some tracks of the execution. And over all it appeared essential to improve considerably the metrics, the measurements and the panel of the results proposed. We talked about to propose execution rates and normalized metrics. It has also been proposed to set a trust interval for measurement. And it could be interesting if we could generate a global mark for the platform tested

In the previous iteration, we only tested curved paths. It has been decided in agreement with COMPANYX to focus on the paths with a complete coverage and to test images and image filters. That implies that we have to include lines in the paths benchmark. And another benchmark has to be done for images and image filters. To do so, we want to break with the style of the first iteration and to propose, for the images and image filters tests, a completely different style of test. And if it can be funny or represent something then it would be perfect. That means that we will not use any longer random generation in this part. This could bring more simplicity. Like for the paths it is also important to stress retesselation. It also appears to be important to test images in combination with paths. That is to say to draw images and paths in the same time and in both kind of superposition. For the filters, we will of course test all the types of filters: colour matrix, kernel convolution, separable kernel convolution, Gaussian blurs, lookup tables, and single lookup table filters. This means that we have to make some choices because it is important that the total execution time of the benchmark is not too long. In this context, to test retesselation on image filters appears as not essential and very long especially since there are 6 different kinds of filters, so if we do it for one, we have to do it for all of them and it quickly becomes awfully long because filters are quite slow to render in general.

To finish, a particular effort had to be done on the portability. It consists in sorting the code depending on whether it is platform independent or not.

6.2 Implementation

Lines

Since the feedback concerning the first iteration was pretty good, it appeared better not to redo everything but just to include what has been already done and to proceed to modifications for the points which shall be changed. And that is what has been done. So we kept what has been done in the first iteration and we included at first the iteration testing the line segments that is to say to test the VG_LINE_TO, VG_VLINE_TO, VG_HLINE_TO segments and for both of them we tested both absolute and relative coordinates. The basic pattern remains the same: a 8 segments path made of a succession of VG_LINE_TO, VG_HLINE_TO and VG_VLINE_TO alternating the two different types of coordinates. And this basic pattern is drawn with random parameters. And then it is drawn again after applying each type of transformation with random parameters. Like in the previous version of the benchmark, this basic iteration is repeated several times. This is now fixed by the REDUNDANCY constant. It is first run in filled mode and later in stroked mode.

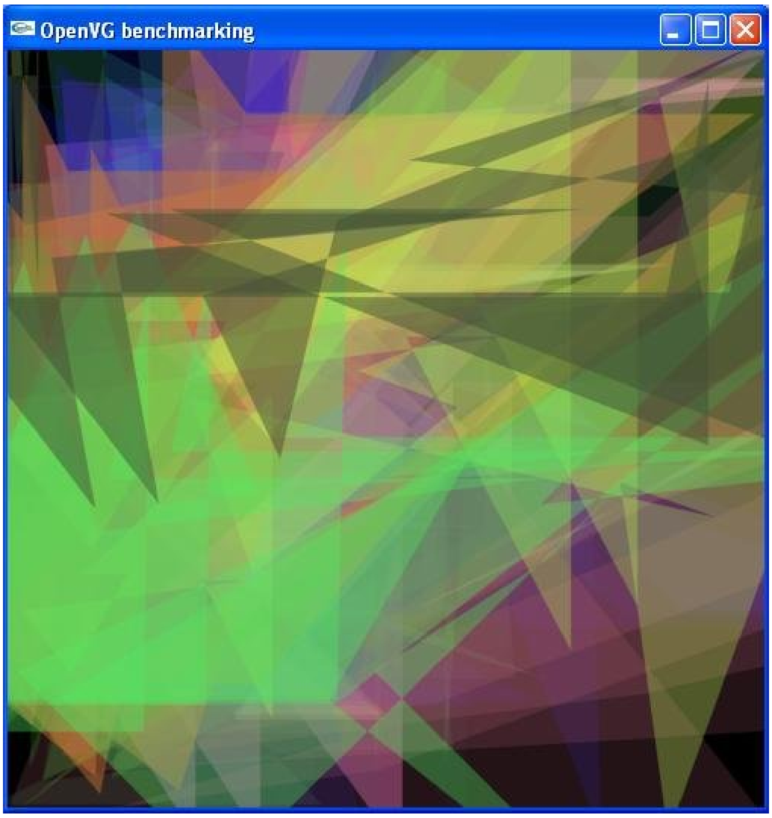


Figure 30: Lines test for the filled paint mode

The figure 1 above, which is a screen capture, shows what the lines test for the filled paint mode. The picture 2 represents the same test but for the stroked paint mode.

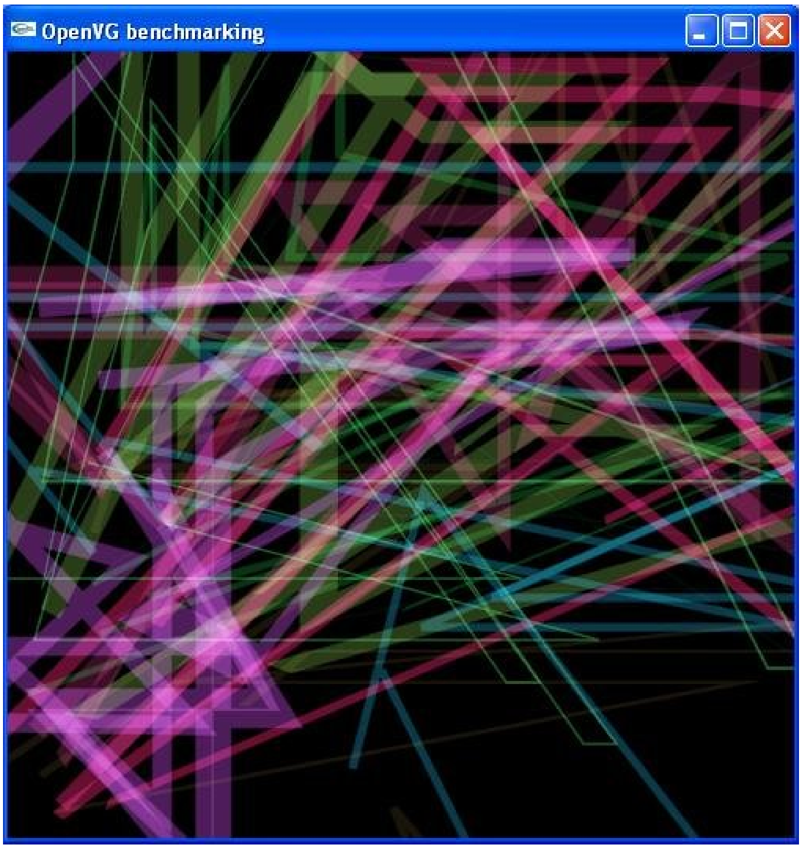


Figure 31: Lines test for stroked paint mode

Dashing

This lines test completes the path test concerning the segments to be tested. But there was a characteristic we forgot to test which is the dashing for stroked paths. So it has been decided to introduce it like an option which can be validated by simply changing the value of a constant like a Boolean. All the parameters settings for this mode are set in the dedicated files: lines.cpp, quad_2.cpp, CUBIC2.cpp, ellipses_2.cpp and G1Smooth_2.cpp. The parameters are also randomly generated. And we activate the option, we obtain the following results.

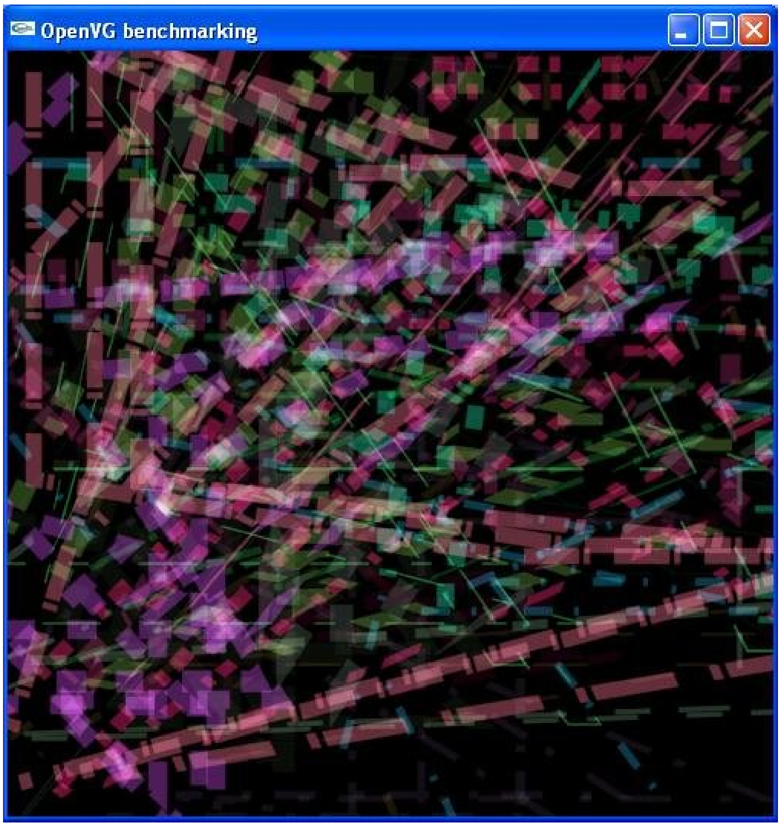


Figure 32: Lines test for dashed stroked paint mode

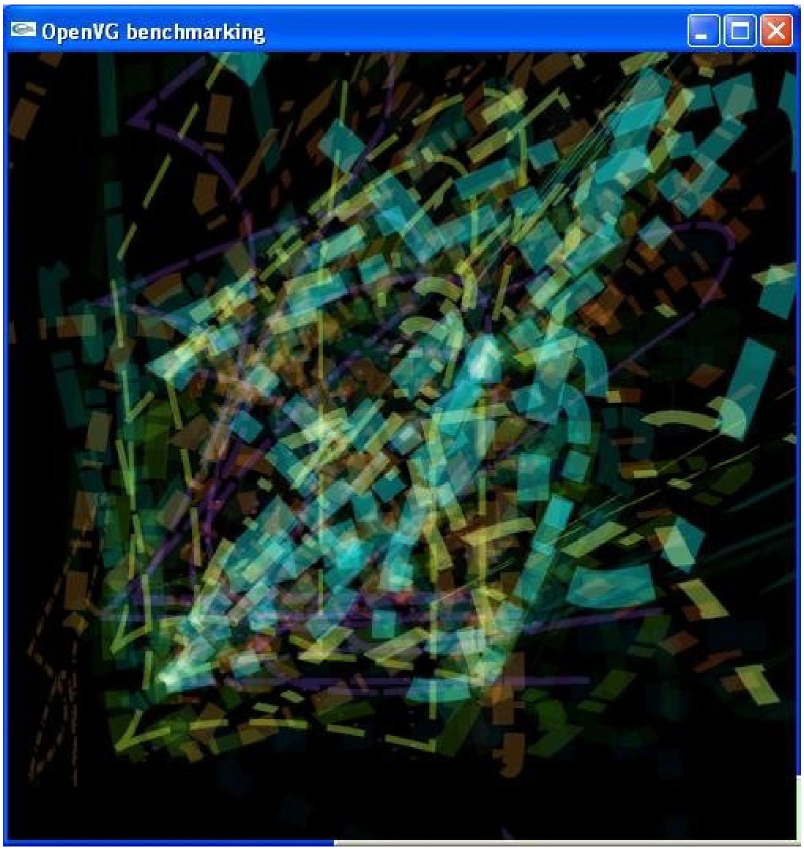


Figure 33: Quadratic curves test for dashed stroked paint mode

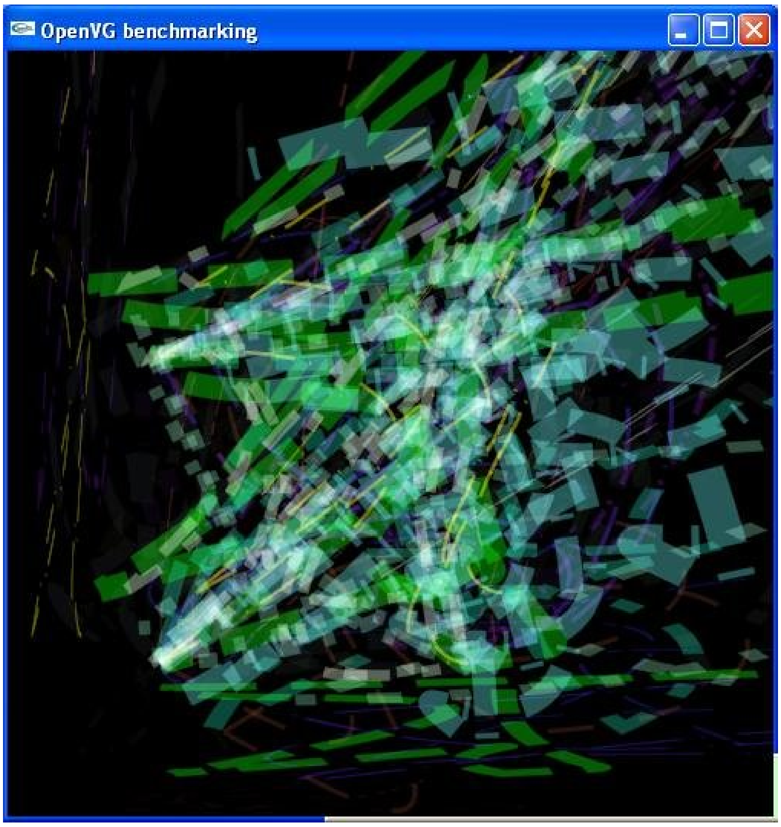


Figure 34: Cubic curves test for dashed stroked paint mode

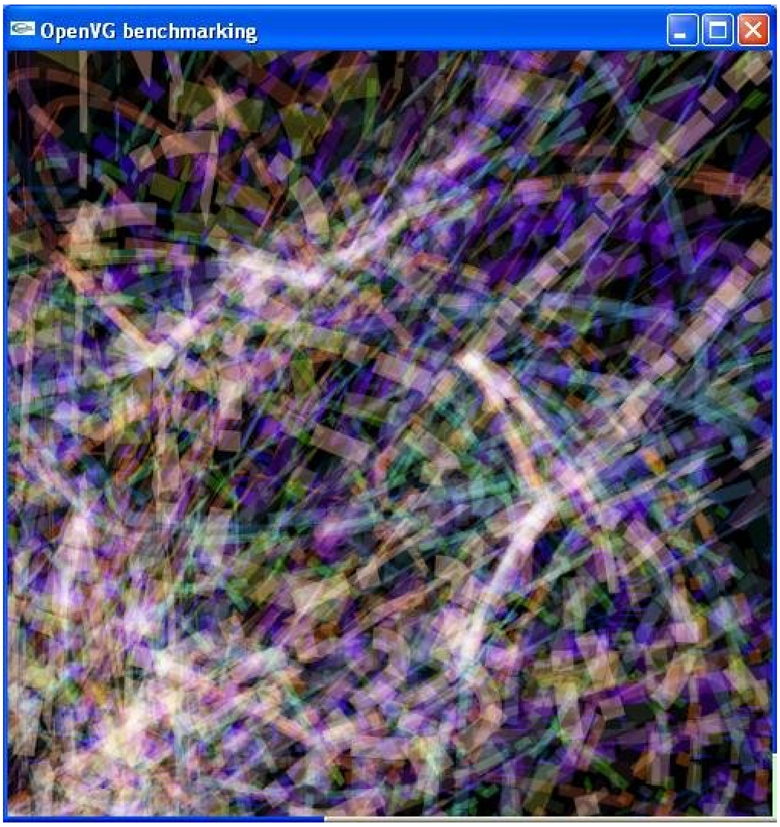


Figure 35: Ellipses test for dashed stroked paint mode

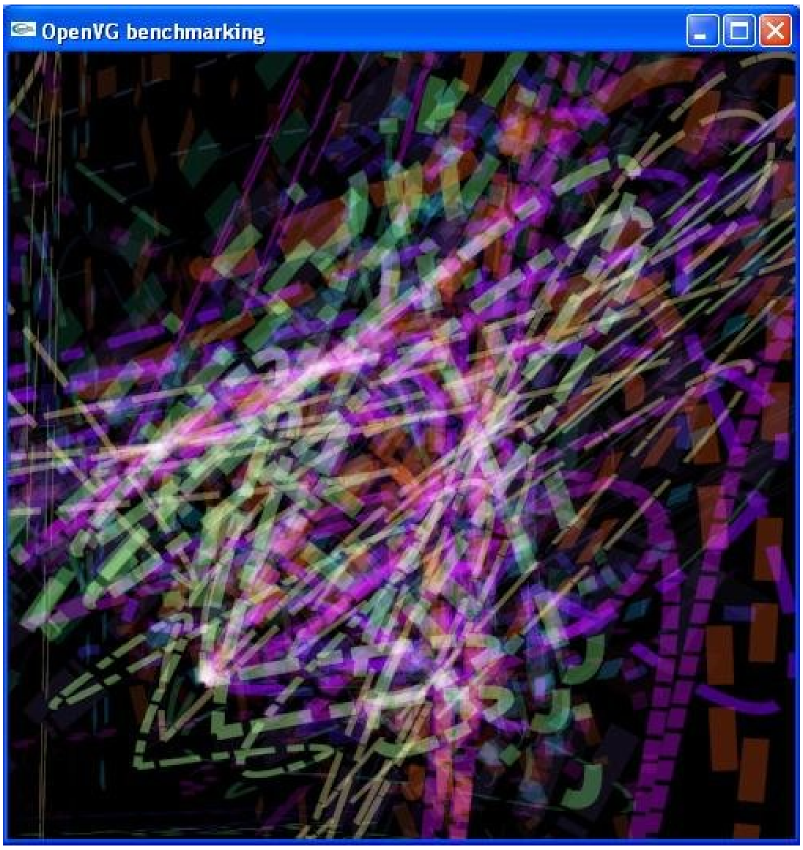


Figure 36: G1 connected curves test for dashed stroked paint mode

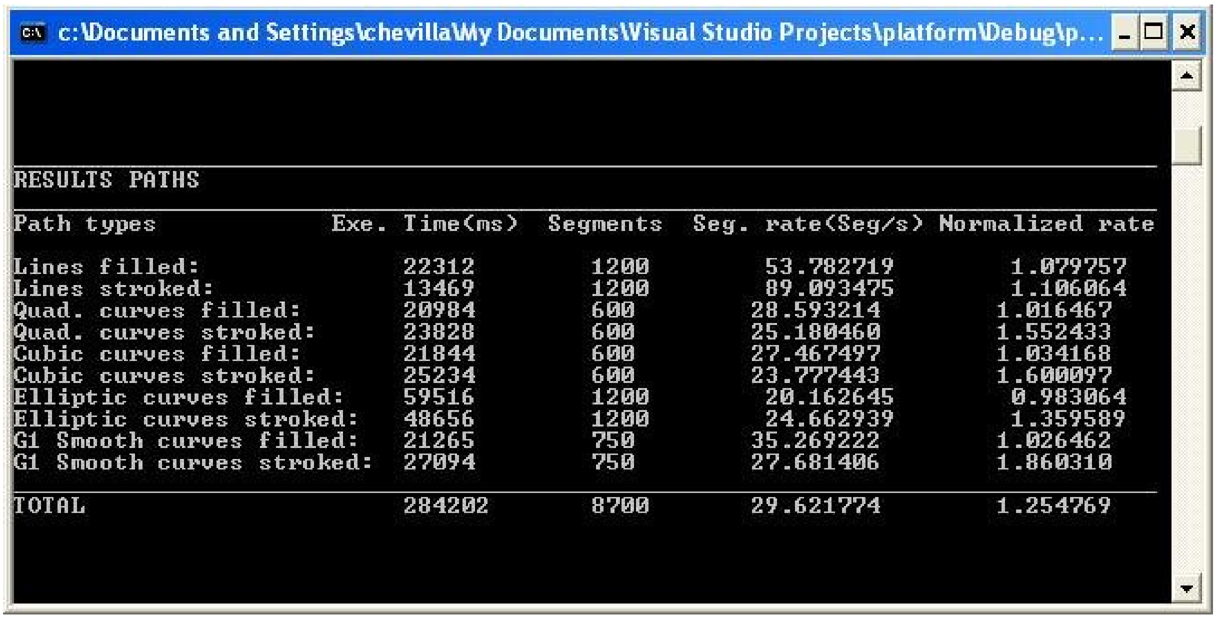
These new features which have just been presented totally complete the paths test. To finish with this part of the benchmark, as we explained it earlier, we proceeded to several important modifications.

The modifications

To start with these modifications, as asked by COMPANYX, the size of the screen and/or window can be changed. Indeed, by simply changing the values of the constants H and W defined in main.cpp, all the windows, parameters and calculus which are based on the size of the screen are modified consequently.

Another modification made on the previous version and to be integrated in this new version concerns the exclusion of the random number generations from the execution times measured. It is made by simply generating the table of the parameters needed by the basic iteration and then to start the timer and to execute this basic iteration by passing the parameters table in the arguments when calling the functions.

And we also modified the metrics. We introduced the number of segments drawn. And over all we calculate an execution rate for all the sections of the benchmark in segments per second. And we also proposed the same execution rate but normalized. The rates used for normalization have been measured for non dashed stroked paths, screen dimensions of 500*500 and a REDUNDANCY of 100 to have a good approximation of the rates by multiplying the number of measurements. These rates are defined as constants in the output_tools.cpp. Then we summarize the execution rates and the normalized rates with an harmonic mean, since this mean fits the best the execution rates like we saw it in the chapter dealing with performance evaluation.


c:\Documents and Settings\chevilla\My Documents\Visual Studio Projects\platform\Debug\p...

```
RESULTS PATHS

Path types            Exe. Time(ms)  Segments   Seg. rate(Seg/s) Normalized rate

Lines filled:              22312       1200        53.782719       1.079757
Lines stroked:             13469       1200        89.093475       1.106064
Quad. curves filled:       20984       600        28.593214       1.016467
Quad. curves stroked:      23828       600        25.180460       1.552433
Cubic curves filled:       21844       600        27.467497       1.034168
Cubic curves stroked:      25234       600        23.777443       1.600097
Elliptic curves filled:    59516       1200        20.162645       0.983064
Elliptic curves stroked:   48656       1200        24.662939       1.359589
G1 Smooth curves filled:   21265       750        35.269222       1.026462
G1 Smooth curves stroked:  27094       750        27.681406       1.860310

TOTAL                      284202      8700       29.621774       1.254769
```

Figure 37: Displaying results for the paths benchmark

Images

As we explained it above, the benchmark for the images has been conceived and driven with a totally different concept. Indeed, since we have to manipulate images, then it would be better and more interesting to manipulate images which have a sense. The idea was also to propose something funny and particular for this benchmark. And since we also need to stress retesselation by using the transformations, the idea of a funny animation appeared as more and more obvious. And the idea of sketching the introduction animation from the James Bond movie appeared but instead of letting the figure of James Bond appear in the centre of a gun, the logo of NTNU was supposed to appear. Then after shaking up and down, the blood comes on top and in transparency.

At first, it draws the logo of NTNU in bright blue on a white screen. It is made of one path to draw the letters and filled with a solid colour. It is rendered on the drawing surface and then the pixels of the window are captured to be stored in an image. Then the idea is to use a horizontal translation to carry the NTNU logo from outside the window on the right towards the centre of the gun passing under it. The gun is made of two circles of same centre painted with a solid colour in black. And it is striped in grey from the centre to outside. These stripes are the repetition by rotation of a path made of two quadratic curved segments filled with a grey solid colour. The constant STEP defines the step of the animation.

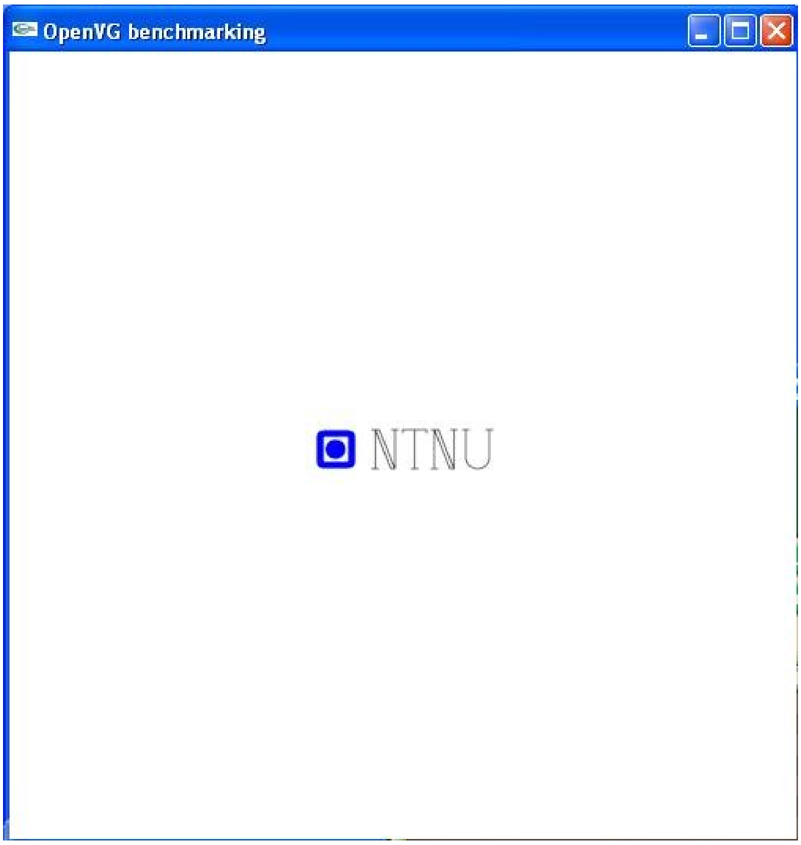


Figure 38: Drawing and capturing the image of the logo

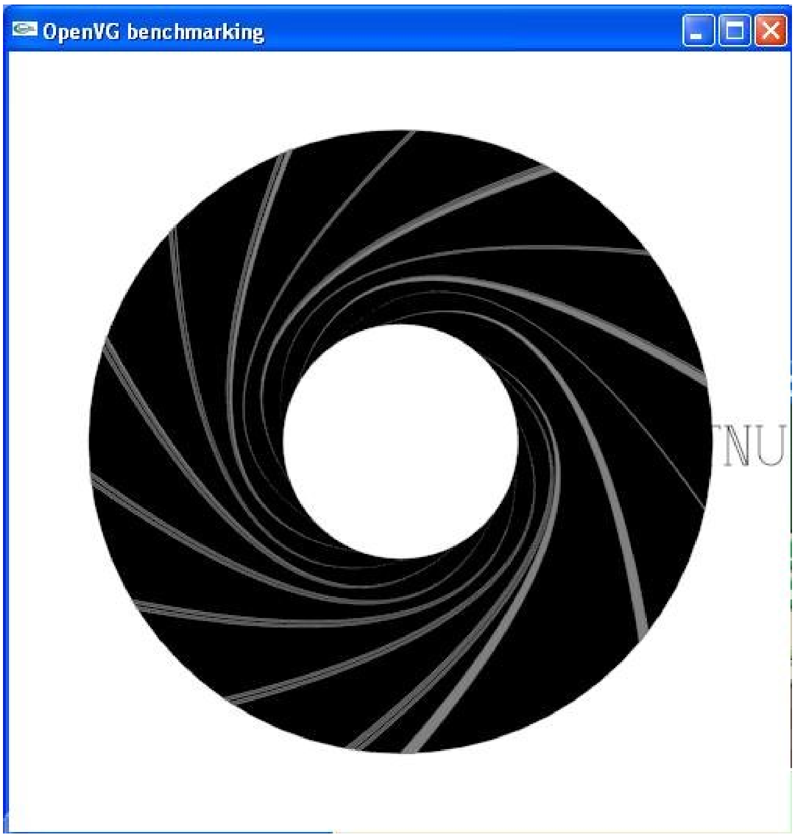


Figure 39: Image of the logo in translation behind the drawn path

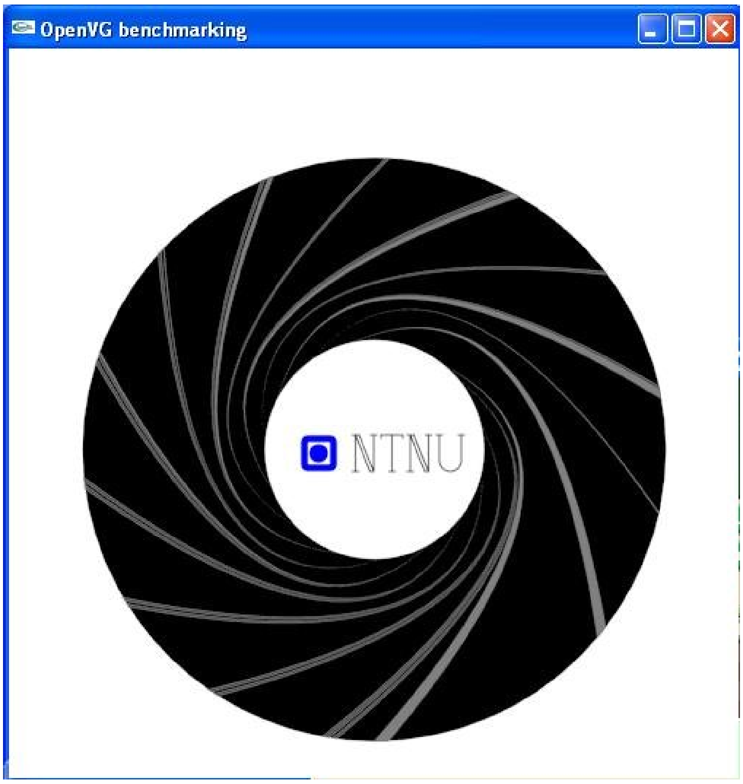


Figure 40: End of the animation: the logo is placed in the center of the gun

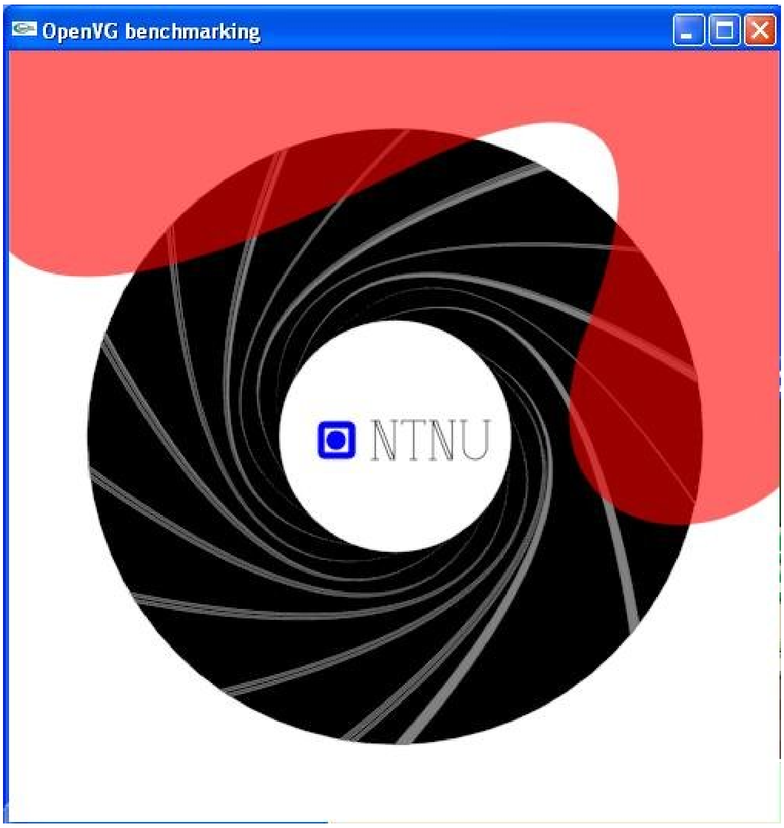


Figure 41: The blood path comes over the captured image and in transparency

For this benchmark, we measure the execution time for the whole animation considering that the number of times the images are drawn is sufficient to measure a repeatable rate. The new metrics developed for the paths benchmark are kept. That is to say a normalized execution rate is also computed.

Image filters

The filters benchmark is based on the images benchmark. Indeed, the idea is to use the ending scene image of the animation and to apply every filter on it with random parameters and to repeat it a REDUNDANCY number of times. We also measure the execution time for every filter and propose a normalized execution rate. The random number generations are excluded from the execution time measurements. Initially, the parameters were generated by random generation using as a maximum value the maximal value for the type necessary. But because the result on the screen was not interesting, in the sense that the whole screen was black of uniform, these values have been manually and empirically reduced to obtain a more interesting result on the screen. The following figures illustrate the different sections corresponding to the different filters tested.

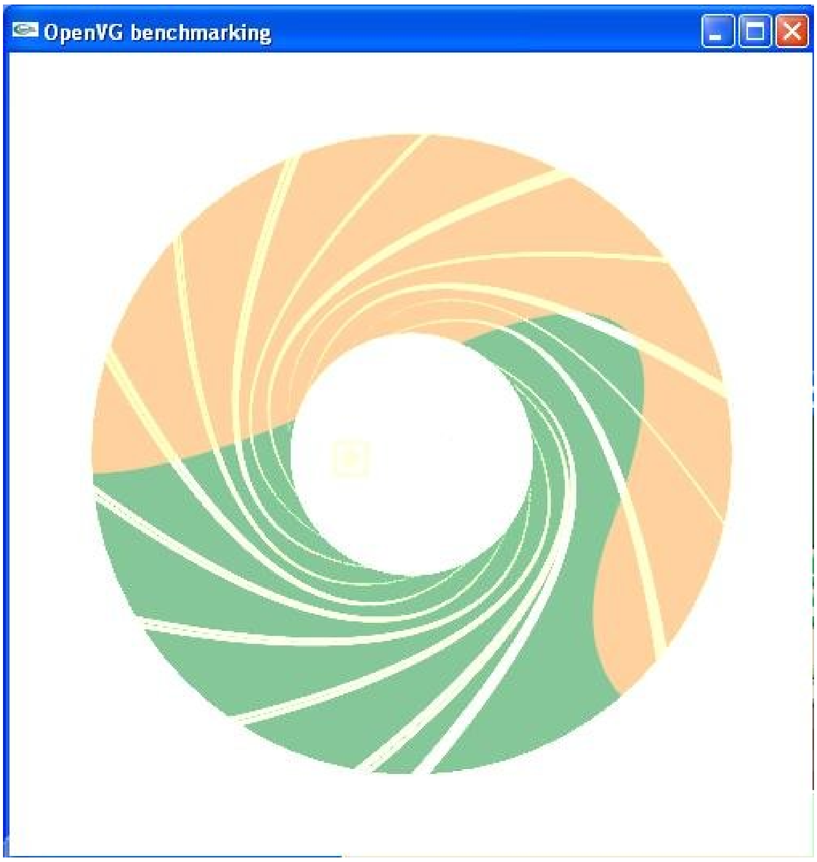


Figure 42: Test for color matrix filter

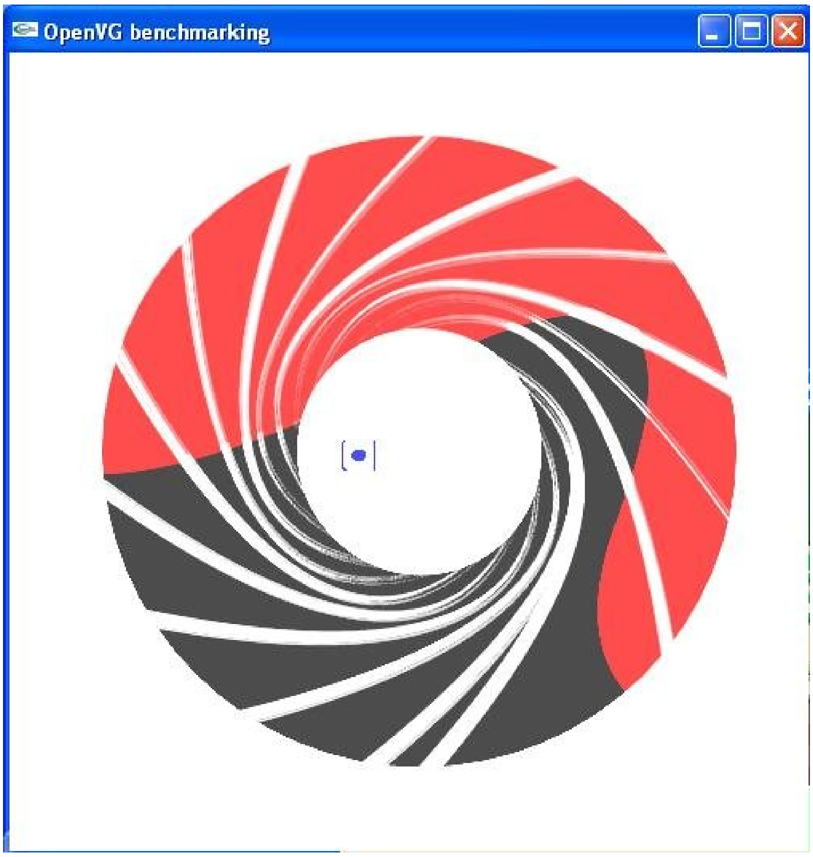


Figure 43: Test for kernel convolution

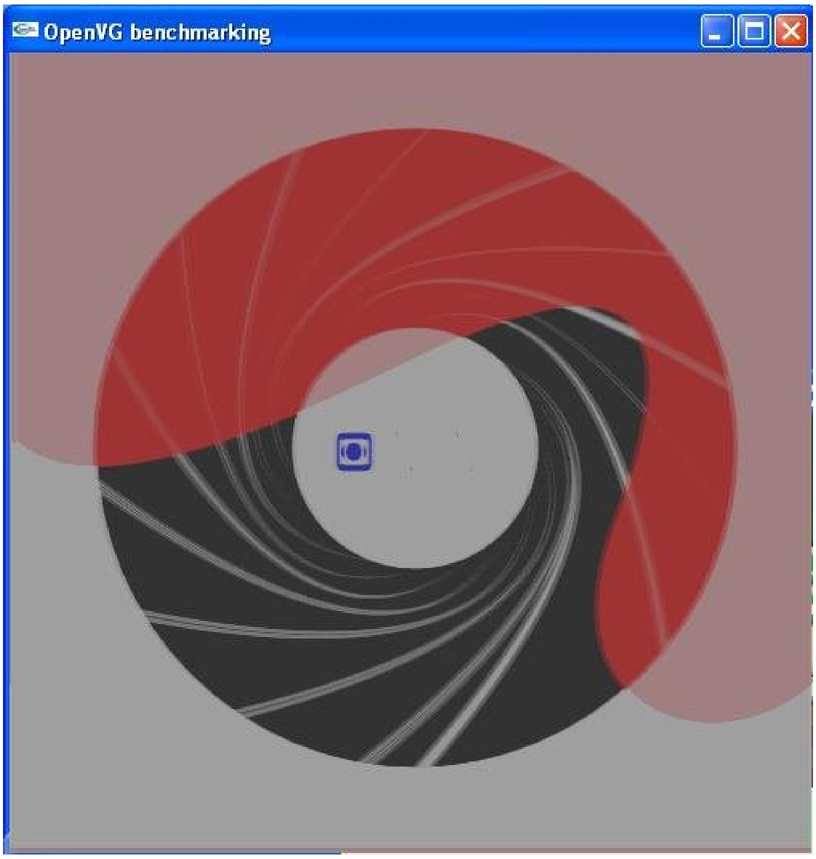


Figure 44: Test for separable kernel convolution filter

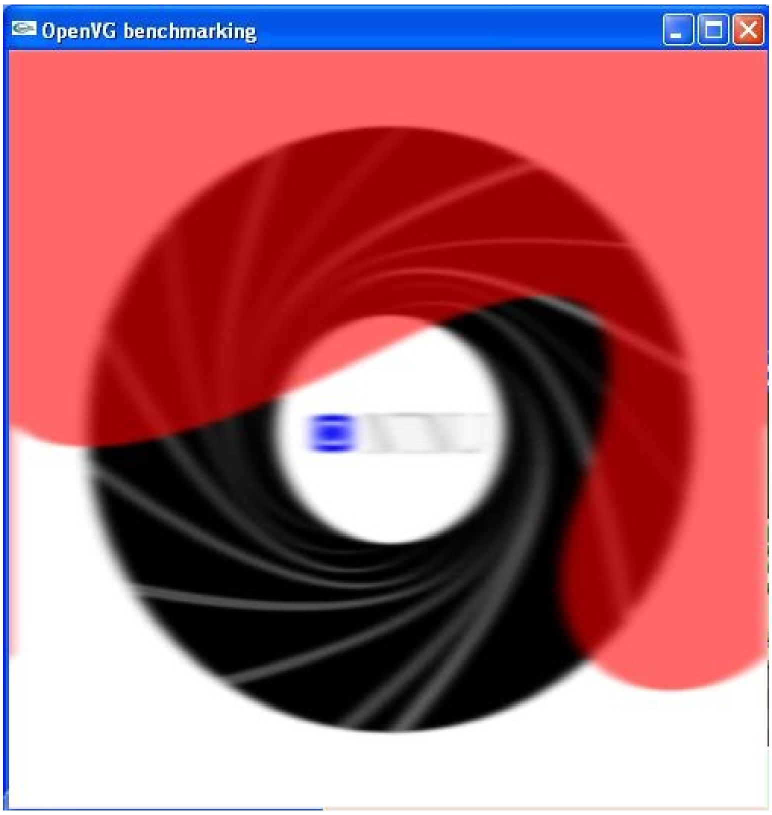


Figure 45: Test for Gaussian blur filter

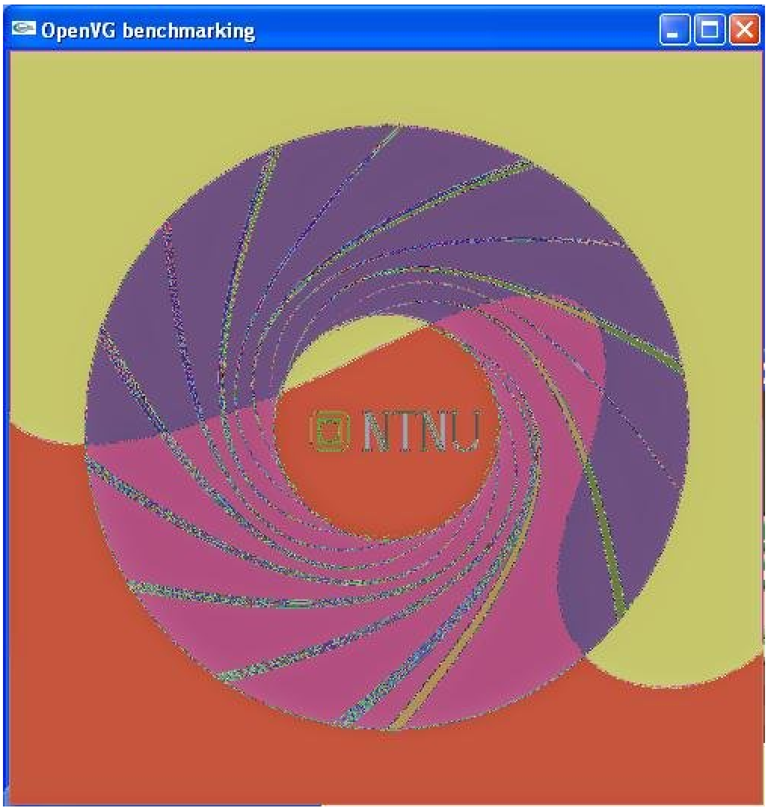


Figure 46: Test for lookup table filter

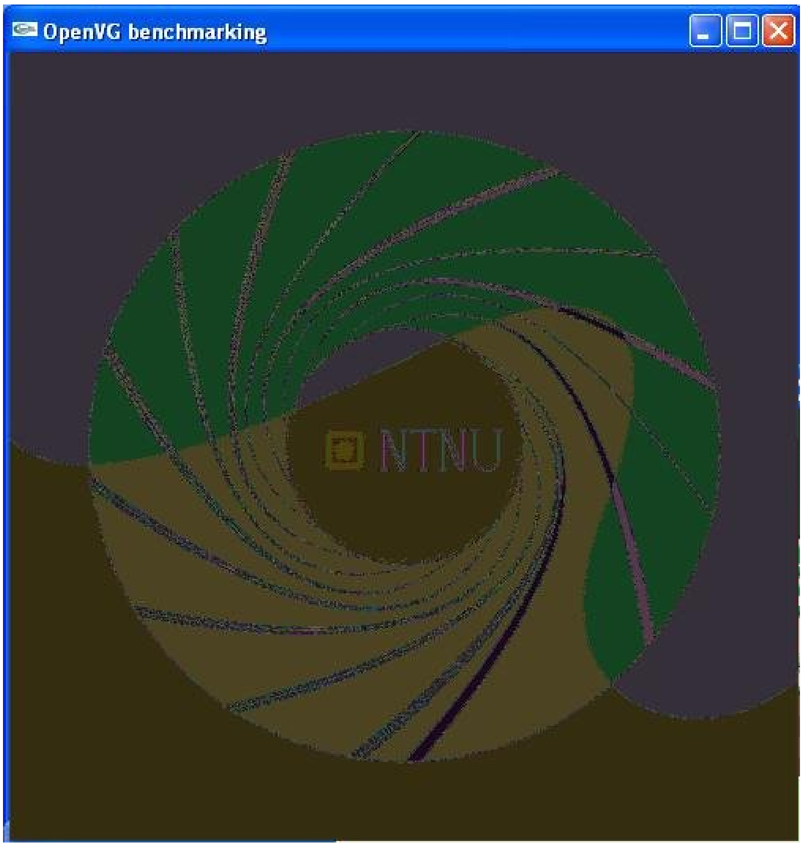
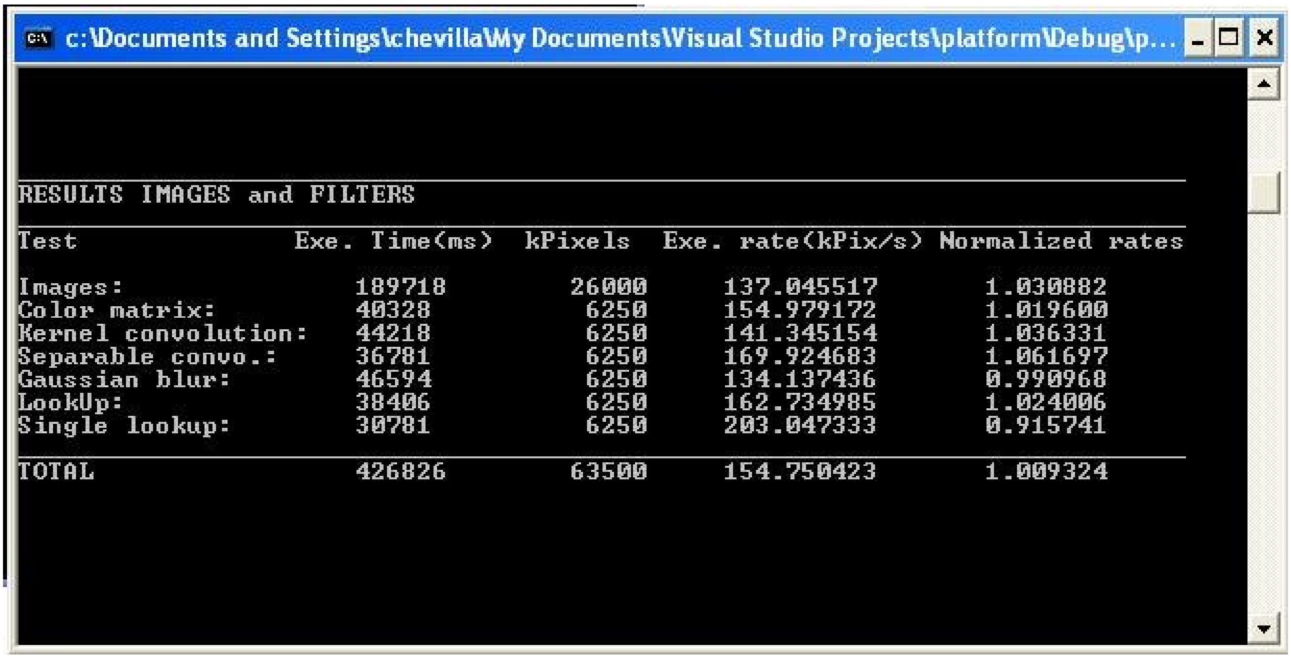


Figure 47: Test for single lookup table filter

And at the end of the tests, the program displays the following statistics and measurements.


c:\Documents and Settings\chevilla\My Documents\Visual Studio Projects\platform\Debug\p...

```
RESULTS IMAGES and FILTERS

Test               Exe. Time(ms)  kPixels   Exe. rate(kPix/s) Normalized rates

Images:                189718       26000      137.045517        1.030882
Color matrix:          40328         6250      154.979172        1.019600
Kernel convolution:    44218         6250      141.345154        1.036331
Separable convo.:      36781         6250      169.924683        1.061697
Gaussian blur:         46594         6250      134.137436        0.990968
LookUp:                38406         6250      162.734985        1.024006
Single lookup:         30781         6250      203.047333        0.915741

TOTAL                  426826       63500      154.750423        1.009324
```

Figure 48: Display of the results of the images and filters tests

Dumping buffers

The last featuring of the benchmark that has to be underlined is the possibility offered to dump the buffer into memory and then into a specially created file. The aim is to provide users a way to compare the results of the benchmark on different platforms and over all to compare the way it is rendered on both by keeping a track of the execution. The principle is quite simple: all long the execution of the benchmark, the images (in most of the cases the final scenes) are stored in memory before to be stored in the .dat files which are especially created. This is presented as an option and has to be validated by changing the constant and giving it a non null value. The pixels data are stored in memory according to the given mode then. Later these data are simply written into the files.

The files

This part consists in one of the most important modification which has been made after the end of the development of the software. This was mainly done to make sure that the software is easily portable. This consists in isolating in different files platform dependant code, typically glut code for the windows platform, and platform independent code, typically OpenVG code and EGL code.

The file main.cpp is organized in the following way:

The main function calls all the functions in the right order.

```
int main()
{
      window = platform_initialize(H,W,window);
      egl_init();
      vg_init();
      disp(vg_draw);
      vg_finish();
      platform_close();
}
```

The **platform_initialize** function calls the native code to create the window. It is located in the platform.cpp file which calls the platform dependent code file. Then the **egl_init** function creates the drawing surface, the context and binds the API to the context and the drawing surface. This function is included in main.cpp. Then **vg_init** will create all the images and allow all the memory necessary. The **disp** function calls a platform dependent code that runs the **vg_draw** function which makes all the calls of the benchmark. The **vg_finish** function destroys all the image objects created and free all the allocated memory. And the function **platform_close** calls a platform independent code function which eventually closes the window which has been created.

As we said, **vg_draw** is responsible for the part of the code which is purely relevant of the benchmark. After clearing the screen in black, the function calls the **itertypepaint** function with the filled path paint mode in parameter to start the complete tests set for the paths in the filled path paint mode. Then **itertypepaint** is called again but with the stroked path paint mode and the constant DASHED to start the complete tests set for the stroked path paint mode and in dashed mode or not as the constant DASHED says. When returning from these **itertypepaint** calls, the screen is cleared in white. Then the images tests can start by calling the **image_test** functions included in main.cpp. Then the **filters_test** function is called to run the filters tests. This last function is also included in the main.cpp file. Then we generate the buffer files if the option is validated. To do so we call a **generate_file** function from the output_tools.cpp file. This function creates the file and stores all the data inside. And then the functions responsible of displaying the results are called. These two functions, **printdata_paths** and **printdata_images,** are included in the output_tools.cpp file.

The **itertypepaint** function is very similar to the one which was in the first release. The main difference is that the lines test has been included in and we also introduced the modifications concerning the exclusion of the random generation of the parameters from the execution time measurements. The lines test repeats a basic pattern by calling the **lines_2** function which is included in the lines.cpp file.

6.3 Problems encountered and challenges

As we can guess, this iteration was a bit more difficult to realize. The following points brought some difficulties in particular.

At first, the isolation of the platform dependent code, such as the glut library code, for portability reasons, was not that easy. It led to a complete reorganization of the code. The main problem was that it was difficult to isolate while it is actually strongly linked with the OpenVG code. And actually it allowed to perform a cleaning of the code by eliminating all the unused and commented code.

The exclusion of the random generations of the parameters for the paths and the filters benchmarks forced a complete reorganization of the code by isolating in files the basic patterns for every type of path segment. And the rest is executed from the main.cpp file in the **itertypepaint** function. The parameters are passed through a pre-generated table.

Then comes the difficulties encountered in the images test section. Indeed, it was from being easy to design the NTNU logo, the gun with the stripes and the blood. Indeed, the paths have to be mathematically designed. The hardest was probably the NTNU logo. It was hard to perfectly imitate the style of the logo.

Still in the same kind of problems, as we said earlier, we had to make some modifications on the parameters passed to the random generators for the filters tests. Indeed, the look on the screen was not very good or totally monochrome. So it has been decided to change the values in order to get something more interesting on the screen. It was the case for the convolutions and the lookup filters in particular for the single lookup table filter test for which the screen was remaining black.

The total execution time was also a challenge since the execution of the whole software shall not exceed 5 or 6 minutes. Therefore, some choices had to be made. In during the conception the time limitations had to be kept in mind. But in the same time, the execution times can be multiplied by 5 or 6 when the benchmark is running on a platform with an hardware acceleration of the standard.

The last feature which was a source of problems is the buffer dumping. Indeed it was not obvious to capture the pixels and to store the data in memory and then to redirect these data in files.

6.4 Feedback and observations

The global on this last version has been good. Among the problems that had to be corrected was the logo of the firm in the images test. It had to be changed by the logo of the NTNU. There were also some problems when porting the benchmark on small screens. Indeed, in some cases the “action” was outside of the frame. But this has been corrected. Among the regrets, we could mention the fact that we didn’t test capstyles and joinstyles for stroked paths. It would also have been interesting to test clipping and scissoring. But the execution time factor set very strict limits.

7 Testing the benchmark

7.1 Aims

Now that the benchmark is finished, it is tempting to test it and to observe the behaviour of OpenVG in different contexts. Of course, the aim is get some more information about the standard itself but it could also allow to compare some different platforms. And it is also a way to go on testing the software itself by using it in a casual and normal use and to detect eventual bugs undetected until now. We will analyze the influence of different parameters such as dashed paths, the size of the screen and the redundancy factor.

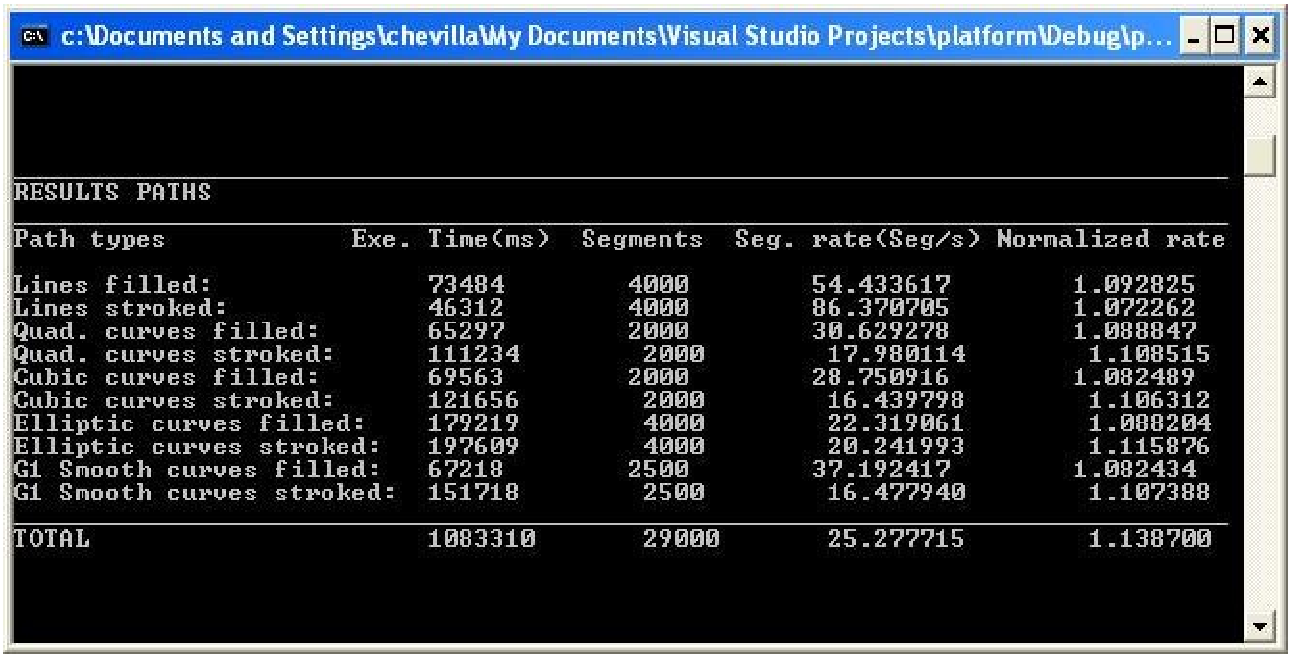
7.2. Tests

7.2.1. Description of the platform

The development platform on which we are going to run a battery of tests is a PC running a Pentium IV processor cadenced at 2,4GHz with 512M of memory. The machine is also using a graphic card. It is a RADEON 9200 Series with 128MB of dedicated memory. And the benchmark is running the OpenVG Reference implementation.

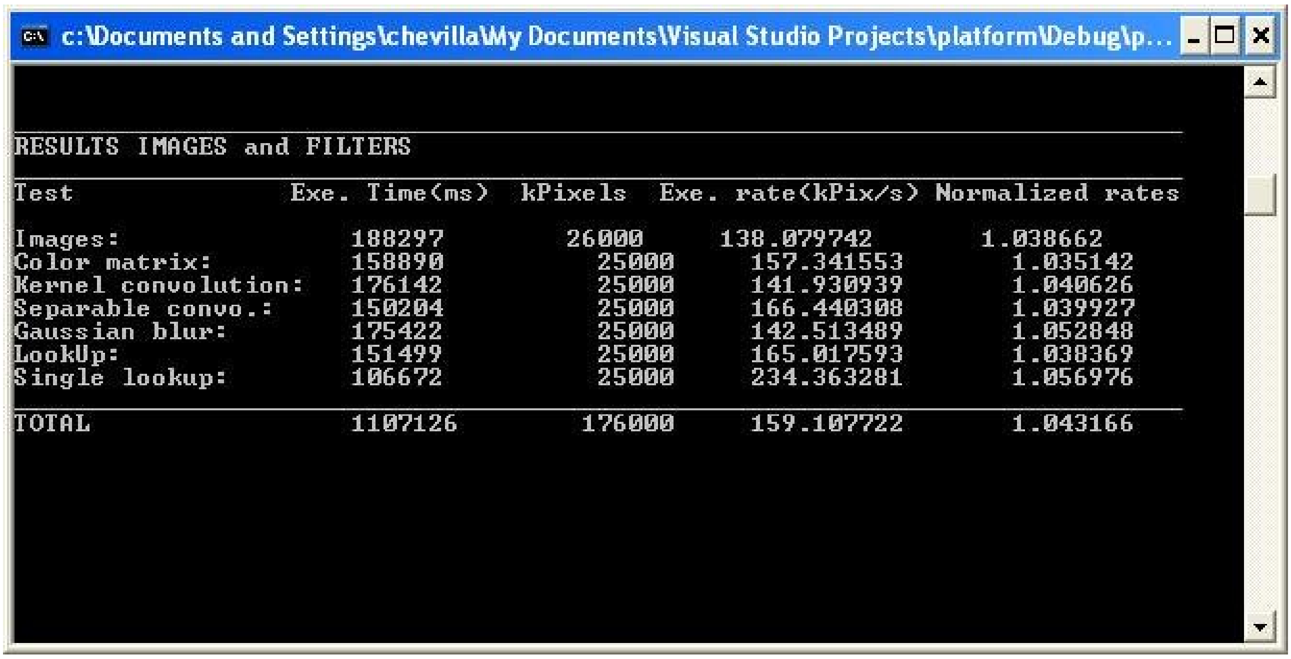
7.2.2 Reference test

At first, it would be interesting to run the benchmark with the settings that led to reference rates. That is to say a 500*500 screen, stroked paths not dashed and a REDUNDANCY of 100. In these conditions, in theory all the normalized rates should have the value 1. Here are the results we got after the test.


c:\Documents and Settings\chevilla\My Documents\Visual Studio Projects\platform\Debug\p...

```
RESULTS PATHS

Path types            Exe. Time(ms)  Segments  Seg. rate(Seg/s) Normalized rate

Lines filled:              73484        4000        54.433617        1.092825
Lines stroked:             46312        4000        86.370705        1.072262
Quad. curves filled:       65297        2000        30.629278        1.088847
Quad. curves stroked:      111234        2000        17.980114        1.108515
Cubic curves filled:       69563        2000        28.750916        1.082489
Cubic curves stroked:      121656        2000        16.439798        1.106312
Elliptic curves filled:    179219        4000        22.319061        1.088204
Elliptic curves stroked:   197609        4000        20.241993        1.115876
G1 Smooth curves filled:   67218        2500        37.192417        1.082434
G1 Smooth curves stroked:  151718        2500        16.477940        1.107388

TOTAL                      1083310       29000       25.277715        1.138700
```

Figure 49: Results for path for the reference test



c:\Documents and Settings\chevilla\My Documents\Visual Studio Projects\platform\Debug\p...

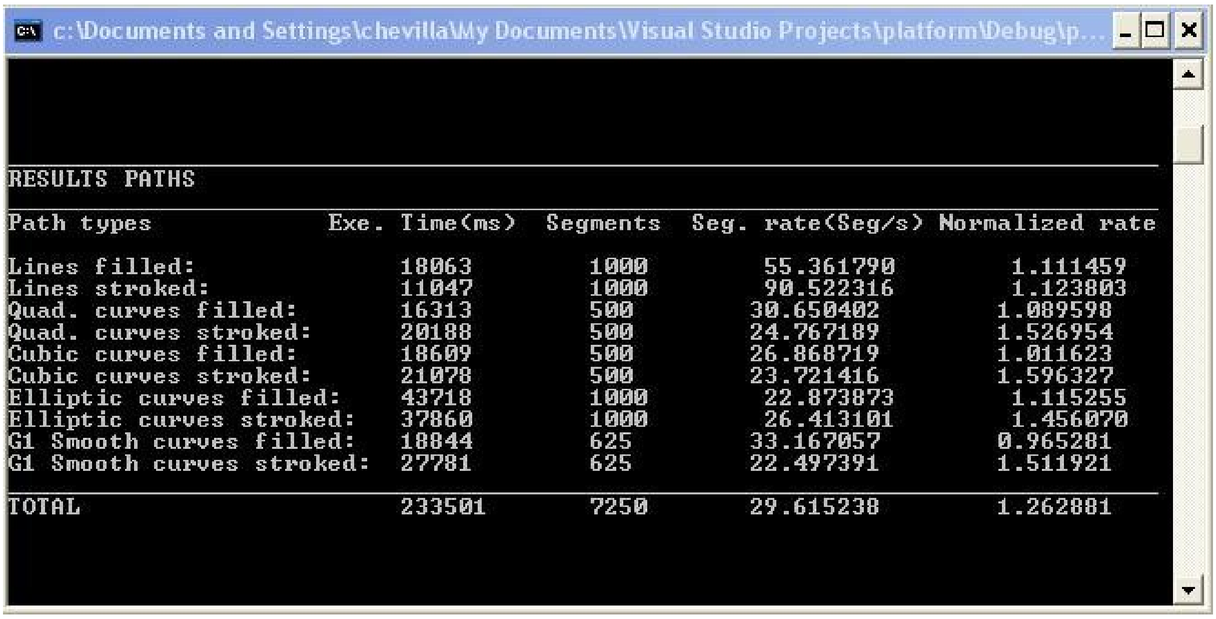
```
RESULTS IMAGES and FILTERS
------------------------------------------------------------------------------
Test              Exe. Time(ms)  kPixels  Exe. rate(kPix/s) Normalized rates

Images:               188297       26000     138.079742        1.038662
Color matrix:         158890         25000     157.341553        1.035142
Kernel convolution:   176142         25000     141.930939        1.040626
Separable convo.:     150204         25000     166.440308        1.039927
Gaussian blur:        175422         25000     142.513489        1.052848
LookUp:               151499         25000     165.017593        1.038369
Single lookup:        106672         25000     234.363281        1.056976
------------------------------------------------------------------------------
TOTAL                 1107126       176000     159.107722        1.043166
```

Figure 50: Results for the images and filters for the refenrence test

What we can observe in the results is that we can have up to almost 10% of difference on the rates compared to the reference rates. That shows that it is subject to internal variations that can not be controlled although they can be predicted in the sense that we already knew before that unavoidable events modify the performance of the system which is benchmarked. Actually, this test is useful for the interpretation of the following tests. And it shows that perturbations up to 10% can be due to internal events which generate a performance perturbation. But over 10%, the reasons are much different.

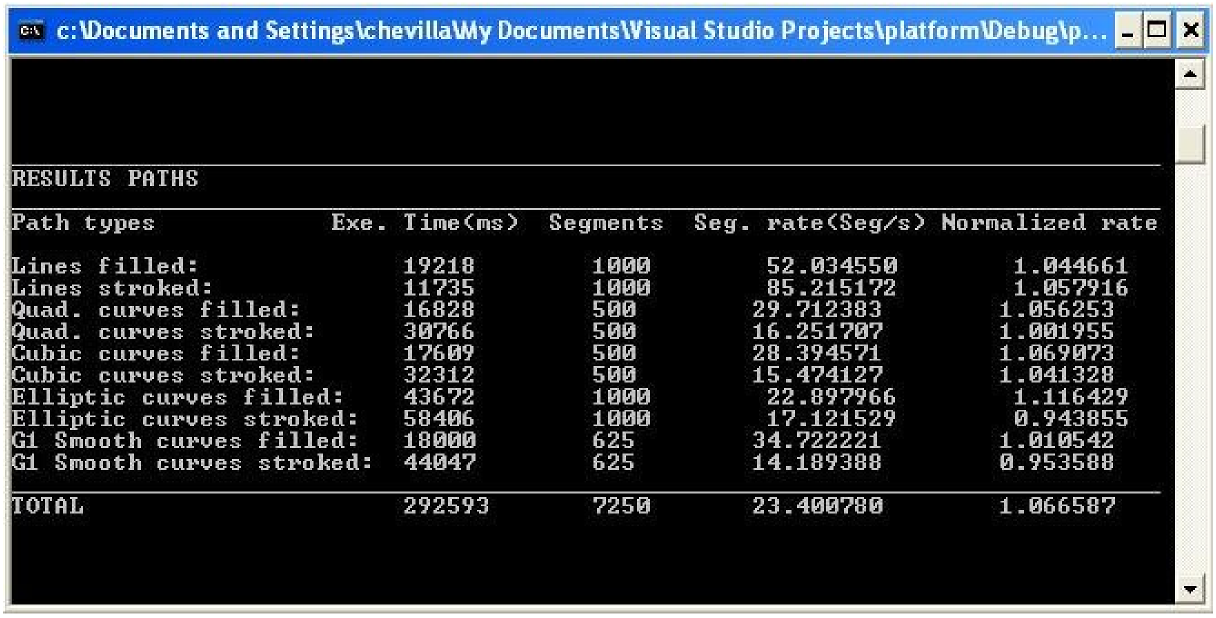
7.2.3 Dashing test

In this test, we will observe the influence on the rendering speed of the dashing of the stroked paths. To do so, we will choose a REDUNDANCY of 25 and a window size of 500*500. And we get the following results.


c:\Documents and Settings\chevilla\My Documents\Visual Studio Projects\platform\Debug\p...

```
RESULTS PATHS

Path types            Exe. Time(ms)   Segments   Seg. rate(Seg/s) Normalized rate

Lines filled:              18063        1000        55.361790         1.111459
Lines stroked:             11047        1000        90.522316         1.123803
Quad. curves filled:       16313        500        30.650402        1.089598
Quad. curves stroked:      20188        500        24.767189        1.526954
Cubic curves filled:       18609        500        26.868719        1.011623
Cubic curves stroked:      21078        500        23.721416        1.596327
Elliptic curves filled:    43718        1000        22.873873         1.115255
Elliptic curves stroked:   37860        1000        26.413101         1.456070
G1 Smooth curves filled:   18844        625        33.167057        0.965281
G1 Smooth curves stroked:  27781        625        22.497391        1.511921

TOTAL                      233501       7250       29.615238        1.262881
```

Figure 51: Results of the dashing test for dashed stroked paths


c:\Documents and Settings\chevilla\My Documents\Visual Studio Projects\platform\Debug\p...

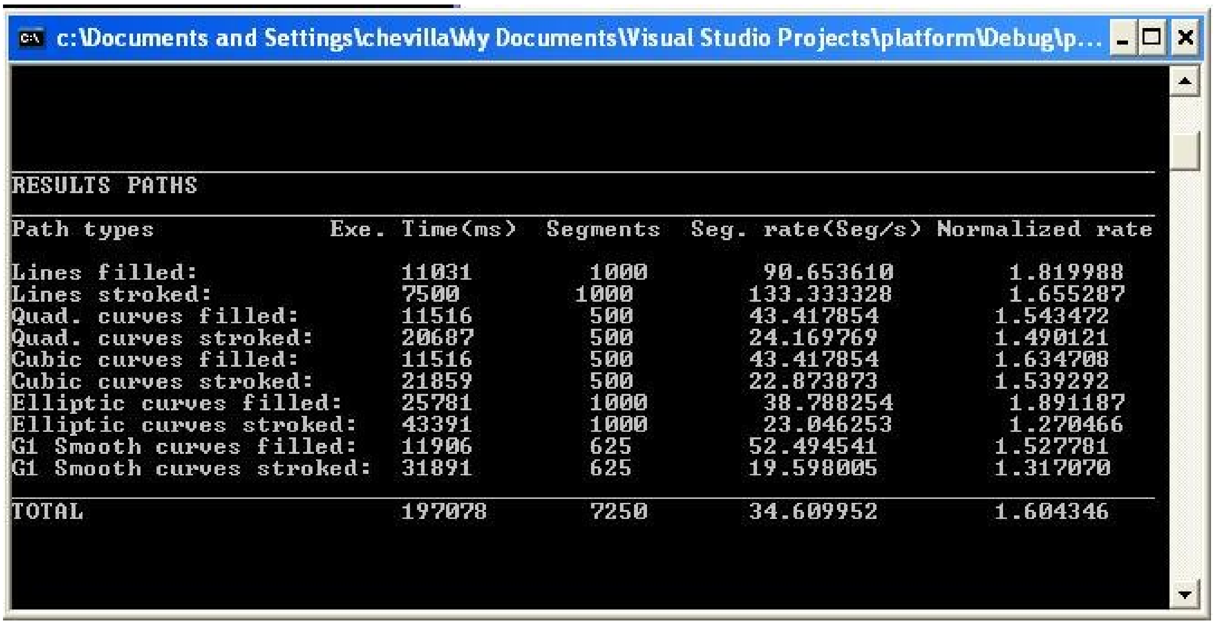
```
RESULTS PATHS

Path types            Exe. Time(ms)   Segments   Seg. rate(Seg/s) Normalized rate

Lines filled:              19218        1000        52.034550         1.044661
Lines stroked:             11735        1000        85.215172         1.057916
Quad. curves filled:       16828        500        29.712383        1.056253
Quad. curves stroked:      30766        500        16.251707        1.001955
Cubic curves filled:       17609        500        28.394571        1.069073
Cubic curves stroked:      32312        500        15.474127        1.041328
Elliptic curves filled:    43672        1000        22.897966         1.116429
Elliptic curves stroked:   58406        1000        17.121529         0.943855
G1 Smooth curves filled:   18000        625        34.722221        1.010542
G1 Smooth curves stroked:  44047        625        14.189388        0.953588

TOTAL                      292593       7250       23.400780        1.066587
```

Figure 52: Results of the dashing test for non dashed paths

The results show that dashing has a strong influence on the rendering of the stroked paths. Indeed, the dashed paths are faster rendered than the non dashed stroked paths. And the global normalized rate for the test with dashed stroked paths is 26% higher than for the normal paths. And of course, it includes the filled paths. So globally, when we only consider the stroked paths, except for the lines paths, the global speed raise is around 50%. This speed up is very important and raises some questions about the implementation. Indeed, we could expect that the result was the opposite since the patterns of the paths are much more complex than normal stroked ones.

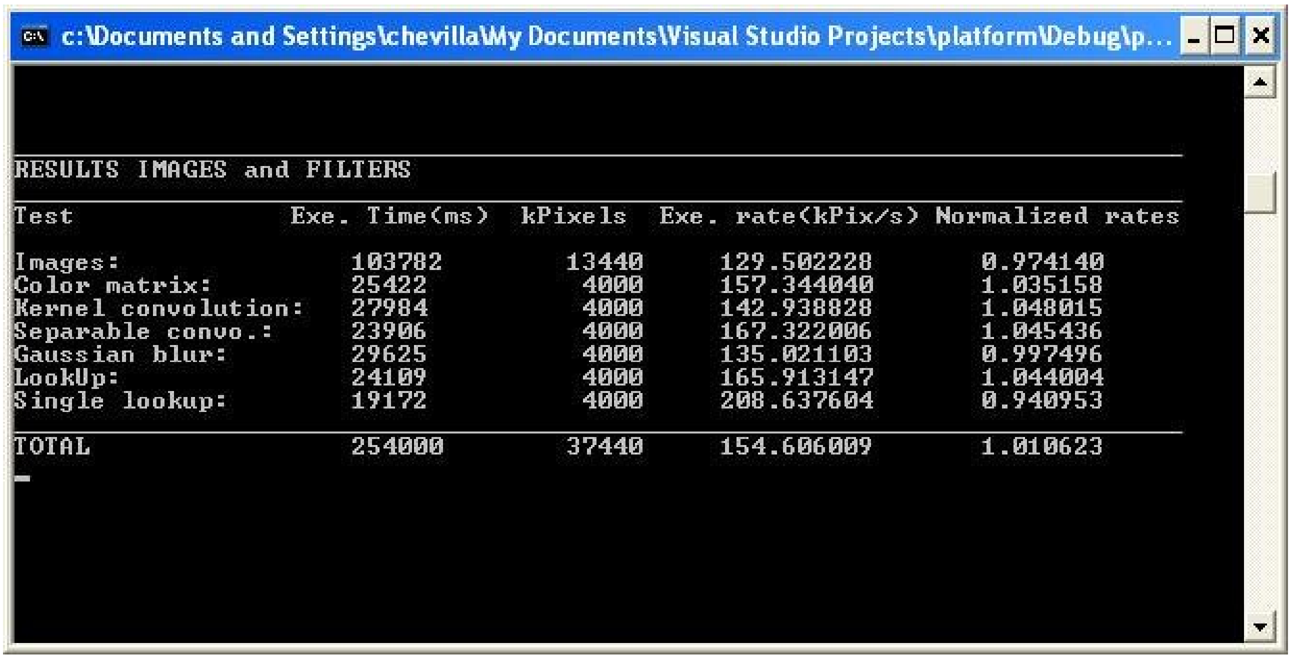
7.2.4. Influence of the size of the screen

In this test, we want to measure the impact of the size of the screen on the rendering. To do so, we will run the benchmark for 3 different screen or window sizes. They will be 400*400, 600*600 and 800*800. The REDUNDANCY value remains at 25 and the stroked paths are not dashed. Here are the results.


c:\Documents and Settings\chevilla\My Documents\Visual Studio Projects\platform\Debug\p...

```
RESULTS PATHS

Path types             Exe. Time(ms)  Segments   Seg. rate(Seg/s) Normalized rate

Lines filled:               11031        1000        90.653610        1.819988
Lines stroked:              7500        1000        133.333328        1.655287
Quad. curves filled:        11516        500        43.417854        1.543472
Quad. curves stroked:       20687        500        24.169769        1.490121
Cubic curves filled:        11516        500        43.417854        1.634708
Cubic curves stroked:       21859        500        22.873873        1.539292
Elliptic curves filled:     25781        1000        38.788254        1.891187
Elliptic curves stroked:    43391        1000        23.046253        1.270466
G1 Smooth curves filled:    11906        625        52.494541        1.527781
G1 Smooth curves stroked:   31891        625        19.598005        1.317070

TOTAL                       197078       7250       34.609952        1.604346
```

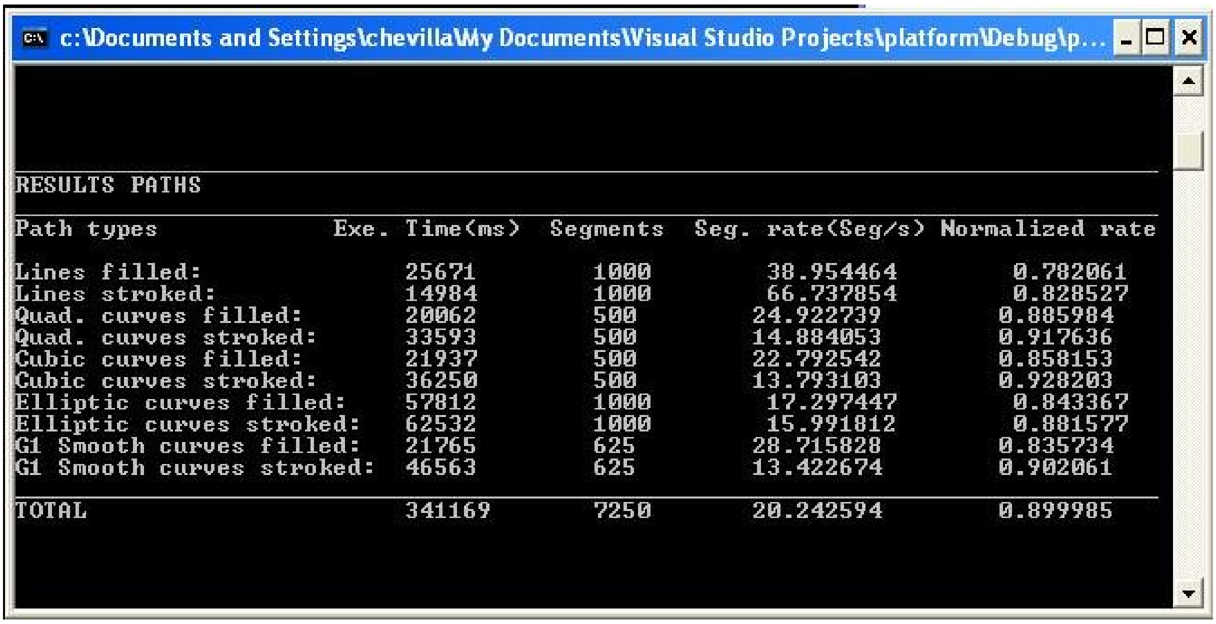
Figure 53: Results of the paths benchmark for a 400*400 screen


c:\Documents and Settings\chevilla\My Documents\Visual Studio Projects\platform\Debug\p...

```
RESULTS IMAGES and FILTERS

Test                 Exe. Time(ms)  kPixels   Exe. rate(kPix/s) Normalized rates

Images:              103782        13440     129.502228        0.974140
Color matrix:        25422          4000     157.344040        1.035158
Kernel convolution:  27984          4000     142.938828        1.048015
Separable convo.:    23906          4000     167.322006        1.045436
Gaussian blur:       29625          4000     135.021103        0.997496
LookUp:              24109          4000     165.913147        1.044004
Single lookup:       19172          4000     208.637604        0.940953

TOTAL                254000        37440     154.606009        1.010623
_
```

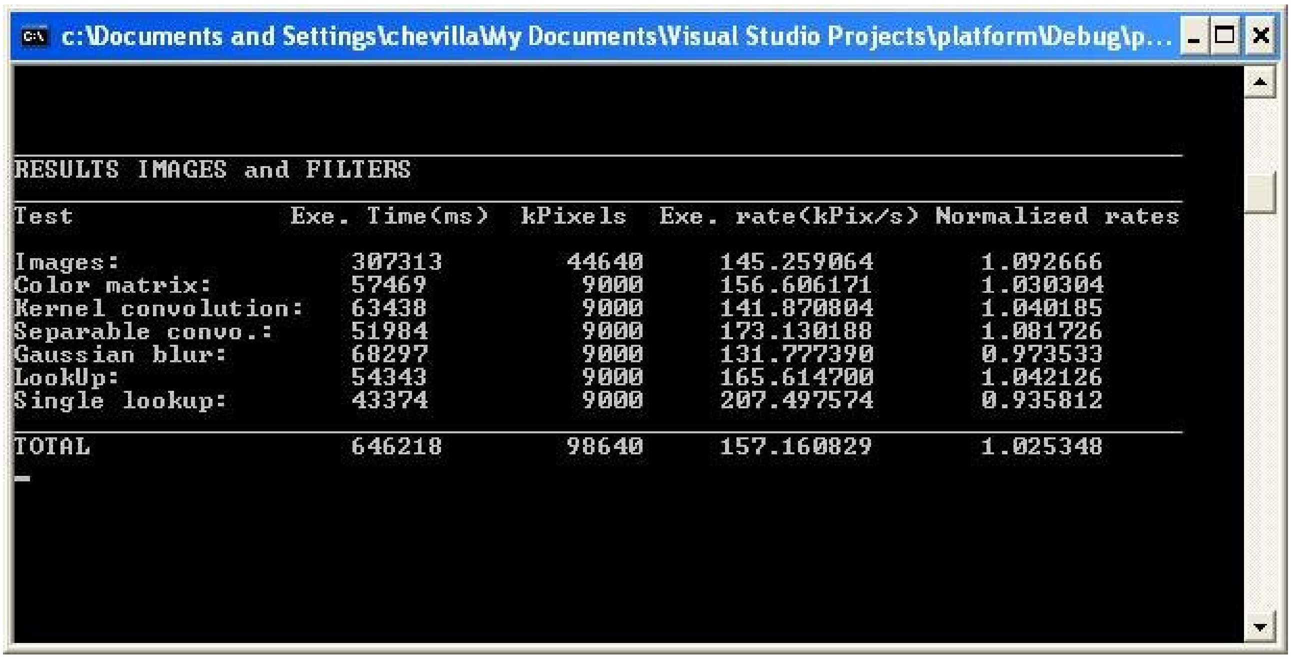
Figure 54: Results of the images and filters benchmark for a 400*400 screen

We can observe a raise of 60% of the execution rate for the paths benchmark for a 400*400 pixels screen. But there is no serious incidence on the images and filters rates.


c:\Documents and Settings\chevilla\My Documents\Visual Studio Projects\platform\Debug\p...

```
RESULTS PATHS

Path types              Exe. Time(ms)  Segments  Seg. rate(Seg/s) Normalized rate

Lines filled:                25671        1000      38.954464         0.782061
Lines stroked:               14984        1000      66.737854         0.828527
Quad. curves filled:         20062        500      24.922739        0.885984
Quad. curves stroked:        33593        500      14.884053        0.917636
Cubic curves filled:         21937        500      22.792542        0.858153
Cubic curves stroked:        36250        500      13.793103        0.928203
Elliptic curves filled:      57812        1000      17.297447         0.843367
Elliptic curves stroked:     62532        1000      15.991812         0.881577
G1 Smooth curves filled:     21765        625      28.715828        0.835734
G1 Smooth curves stroked:    46563        625      13.422674        0.902061

TOTAL                        341169       7250      20.242594        0.899985
```

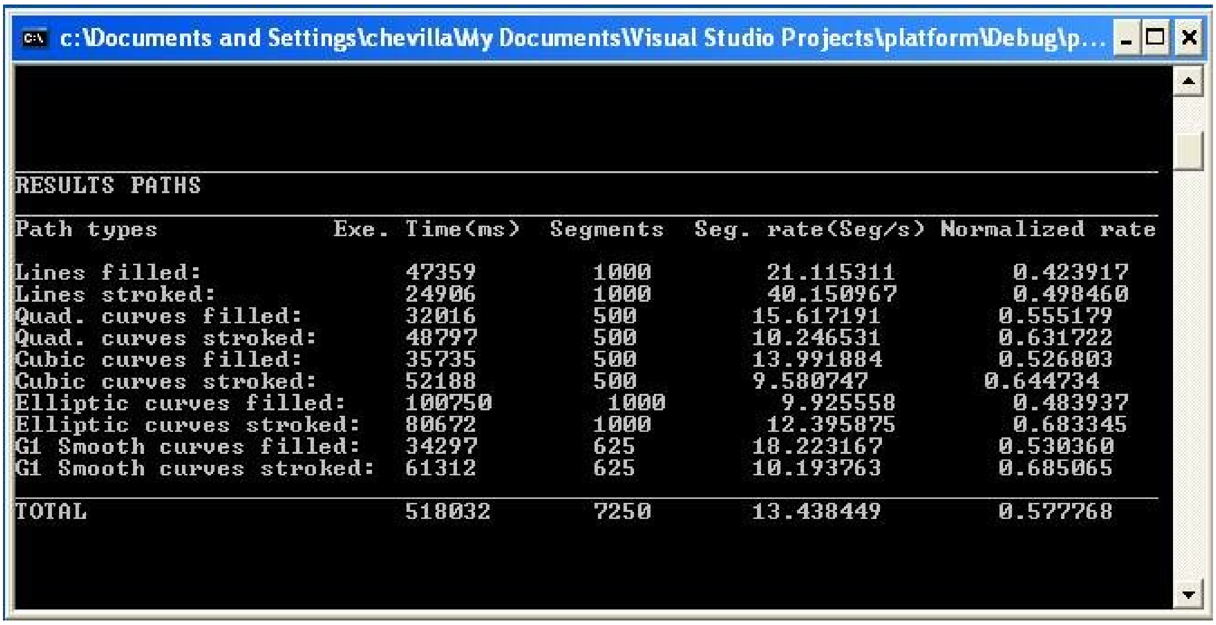
Figure 55: Results of the paths benchmark for a 600*600 screen


c:\Documents and Settings\chevilla\My Documents\Visual Studio Projects\platform\Debug\p...

```
RESULTS IMAGES and FILTERS

Test                Exe. Time(ms)  kPixels  Exe. rate(kPix/s)  Normalized rates

Images:                 307313       44640      145.259064         1.092666
Color matrix:           57469         9000      156.606171         1.030304
Kernel convolution:     63438         9000      141.870804         1.040185
Separable convo.:       51984         9000      173.130188         1.081726
Gaussian blur:          68297         9000      131.777390         0.973533
LookUp:                 54343         9000      165.614700         1.042126
Single lookup:          43374         9000      207.497574         0.935812

TOTAL                   646218       98640      157.160829         1.025348
_
```

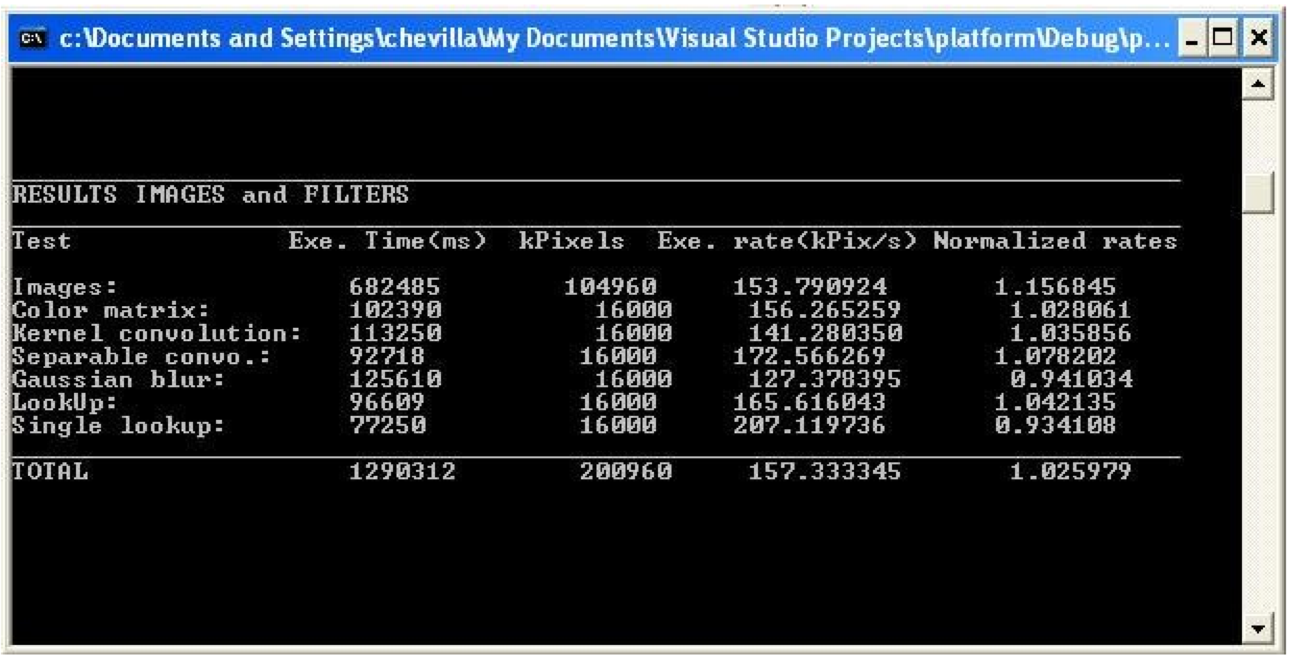
Figure 56: Results of the images and filters benchmark for a 600*600 screen

Like for the first size of the screen tested, we can observe that it has only an influence on the execution rates of the paths benchmark. Indeed, for a screen size of 600*600, we can observe a decrease of 11% on the execution rates for the paths benchmark and relatively unchanged rates for the images and filters benchmark. If we run the same test for a 800*800 screen size, we will probably observe the same tendency.


c:\Documents and Settings\chevilla\My Documents\Visual Studio Projects\platform\Debug\p...

```
RESULTS PATHS
---------------------------------------------------------------------------------
Path types             Exe. Time(ms)   Segments   Seg. rate(Seg/s) Normalized rate

Lines filled:           47359            1000        21.115311         0.423917
Lines stroked:          24906            1000        40.150967         0.498460
Quad. curves filled:    32016            500        15.617191        0.555179
Quad. curves stroked:   48797            500        10.246531        0.631722
Cubic curves filled:    35735            500        13.991884        0.526803
Cubic curves stroked:   52188            500        9.580747        0.644734
Elliptic curves filled: 100750            1000         9.925558         0.483937
Elliptic curves stroked: 80672           1000        12.395875         0.683345
G1 Smooth curves filled: 34297           625        18.223167        0.530360
G1 Smooth curves stroked: 61312          625        10.193763        0.685065
---------------------------------------------------------------------------------
TOTAL                   518032           7250       13.438449        0.577768
```

Figure 57: Results of the paths benchmark for a 800*800 screen


c:\Documents and Settings\chevilla\My Documents\Visual Studio Projects\platform\Debug\p...

```
RESULTS IMAGES and FILTERS
---------------------------------------------------------------------------------
Test               Exe. Time(ms)  kPixels   Exe. rate(kPix/s) Normalized rates

Images:             682485        104960     153.790924        1.156845
Color matrix:       102390          16000      156.265259        1.028061
Kernel convolution: 113250          16000      141.280350        1.035856
Separable convo.:   92718          16000     172.566269       1.078202
Gaussian blur:      125610          16000      127.378395        0.941034
LookUp:             96609          16000     165.616043       1.042135
Single lookup:      77250          16000     207.119736       0.934108
---------------------------------------------------------------------------------
TOTAL               1290312         200960      157.333345        1.025979
```

Figure 58: Results of the images and filters benchmark for a 800*800 screen

And of course, we can observe that the tendency is confirmed. Indeed, we can observe a global decrease of 43% on the normalized rates and almost no incidence on the normalized rates of the images and filters benchmark results. Actually, if we analyze the definition of the metrics used for the rates, we understand a bit better the results. Indeed, the metric used for the execution rates of the images and filters benchmark is the number of pixels drawn per second. And of course, the number of pixels drawn for the tests raised proportionally with the size of the screen. And if we look closer the execution times, we observe that the execution times gradually raised as we increased the size of the screen. But of course, for the paths benchmark, the number of segments drawn didn't change so mathematically the execution rates decreased as the execution times raised as a consequence of the raise of the size of the screen.

7.3 Summary

These three simple tests allowed us at first to play with the benchmark to observe its behaviour. It also permitted to highlight a behaviour for the implementation. It was important to show up what the random error margin can be in order to be able to make distinctions between random disturbances and other phenomenons. That is why we ran the reference test in the same conditions but in the limits of what can be controlled. That is to say that we can not avoid OS internal events which modify the performance at a given moment. That is how we figured out that it was faster to draw dashed stroked paths than non dashed stroked paths. But it was also the opportunity for us to understand the influence of a metric on the interpretations. That was the case when we tried to understand the influence of the size of the screen. It was initially planed to run tests for other implementations especially hardware platforms such as the one developed by COMPANYX.

8 Conclusion

It is definitely not easy to sum up a 7 months internship and master thesis work. The time period is long. The work achieved and the quantity of information and knowledge acquired are very important especially when you are making your first steps in a completely new discipline like computer graphics for me. Indeed this project had a much newer character than just discovering a new standard.

And logically, the first step was to study OpenVG. It starts with understanding the global functionment of the standard. After understanding the types and the concepts of the transformations related to the images, paths and paints, the next step was to assemble everything in the pipeline that represents the rendering process of OpenVG. It was also important to bend over EGL, the native crossplatform layer on top of which OpenVG runs. Then we were ready to develop the concepts of paths, paints and images which are very basic and elementary objects in OpenVG. And we finished with more advanced features such as filters, blending modes and VGU an additive and advanced graphic library.

This small knowledge led us to start working on the investigation about artistic opportunities. This part of the work, in the frame of the SART Project, consisted in running two interviews of artists. After studying how to conduct such interviews and how to prepare them, we started to work on these meetings. The first interview was with Mr E and Mr T both developers and demo graphics at COMPANYX. Their huge experience with OpenVG allowed a very intensive and interesting conversation around possibilities offered by OpenVG. Then we decided to interview Gabor Papp who is very experienced in computer graphics art but who never heard about OpenVG before. His reaction was very interesting and revealed even more the attraction and potential of OpenVG for the artists.

The fourth part was dedicated to study the theory of the computer performance evaluation. It starts with knowing what a good metric and the importance of choosing a good one. This theory chapter included also an analysis about the different errors that can be introduced in the measurements. And after detailing some mathematic and statistic tools, we considered more precisely the benchmarks. We detailed the different types of benchmarks and the different strategies for benchmarking.

Then we were ready to develop our benchmark. The development was driven following an iterative process. Indeed, after developing a simple and primitive first version, we developed a final version based on the feedback received for the first version. In this first version we developed exclusively tests for curved paths. The second iteration was completely covering the paths, the images and the filters. The development process was supposed to be in 3 iterations but has been reduced to 2 iterations due to a lack of time. The benchmarking part of this rich project finishes by a testing part on the development platform. The aim was to test the random error margin and the influence of different parameters such as the size of the screen and the dashing.

This project was finally a very good experience in the computer graphics world and in the computer performance evaluation area, which were both completely unknown for me. OpenVG appears today as a still young and discrete standard. But the study about the possibilities offered to artists showed that it has its role to play in the future of the graphics for handheld devices. And because the standard is still young, it lacks of implementations and content for it. It might also suffer at the moment from the success of OpenGL ES, which is the 3 dimensions API for handheld devices. But OpenVG has of course a different role and thus a place on the market. The attraction for artists and content developers could be also increased by developing a creation tool with a user friendly interface. This could be a track for further development around OpenVG in the frame of the SACCO Project.

Annexe 3: Interview with Gabor Papp

In which direction, OpenVG could be interesting for you in your work? And why? Is it only for the huge potential of the dissemination network? Or is it because of the huge improvement to have a hardware accelerated standard for 2D vector graphics for portable devices?

Since you have a strong experience in the demo graphics scene, how do you think the demo graphics scene will welcome this new standard?

You told me that you were mainly interested in art and more especially generative art. But could you also imagine to develop some commercial content with OpenVG, based on your experience and savoir-faire? And if yes, what kind of content? Do you believe that generative art could be adapted for a commercial use? And if yes, how?

What could be improved in OpenVG to better match the expectations from the artists or to seduce even more artists?

BIBLIOGRAPHY

[01NET06] Telephones mobiles: un marche fleurissant, http://www.01net.com/article/306577.html, web article from 01net.com, 2006

[POC06] Nokia survey finds mobile phones may replace cameras and MP3 players http://www.pocket-lint.co.uk/news.php?newsId=3545, *Amber Maitland,* 2006

[PER06] Open source definition, http://www.opensource.org/docs/definition.php, Bruce Perens, 2006

[NCC05] Over two thirds of senior IT Professionals expect businesses to develop Open Source strategies,http://www.ncc.co.uk/aboutncc/press_rel/open_source_survey_results.cfm, National Computing Center, 2005

[NIL06] Audio decompression software for handheld devices, Andreas Danner Nilsen, Matser Thesis, NTNU, 2006

[KHRGLES06] OpenGL ES specifications, http://www.khronos.org/opengles/, Khronos, 2006

[KHRVG06] OpenVG specification, http://www.khronos.org/openvg/, Khronos, 2006

[ATK06] Anthony Atkielski (Agateller), 2006

[WIKVEC06] Vector graphics definition, http://en.wikipedia.org/wiki/Vector_graphics, Wikipedia, 2006

[JACC06] Project Description for SACCO = Software art creativity community openness, Maria Letizia Jaccheri and Judith Molka-Danielsen, 2006

[RIC05] OpenVG Specification Version 1.0, Daniel Rice, Sun Microsystems Inc, 2005

[LEE05] Khronos Native Platform Graphics Interface (EGL Version 1.2), Jon Leech, 2005

[WIKMP06] History of mobile phones, http://en.wikipedia.org/wiki/History_of_mobile_phones, Wikipedia, 2006

[LIN04] Mobile Phones - Technology - Third Generation Phone http://wiki.media-culture.org.au/index.php/Mobile_Phone_Technology_-_Third_Generation_Phone, Sunny Lin, 2004

[HOV05] Experiences from Conducting Semi-Structured Interviews in Empirical Software Engineering Research, Siw Elisabeth Hove, Bente Anda, 2005.

[LIL00] Measuring Computer Performance: A practitioner's Guide, David J. Lillja. Port Chester,NY, USA: Cambridge University Press, 2000.

[WIKBE06] Benchmark, http://en.wikipedia.org/wiki/Benchmark_(computing), Wikipedia, 2006.