Pixels vs. Vectors: Presentation of Large Images on Mobile Devices

René Rosenbaum
Institute for Computer Graphics
University of Rostock
rrosen@informatik.uni-rostock.de

Christian Tominski
Institute for Computer Graphics
University of Rostock
ct@informatik.uni-rostock.de

Abstract

Mobile devices have become popular in recent years, and are more and more used to handle multimedia data such as digital images. Due to size and application, these devices suffer from limited resources. Especially, if large images must be displayed, the needed system resources often exceed the capabilities of a mobile device.

In this paper, we want to affiliate in which circumstances the use of raster or vector images leads to advantages during the display of large images on mobile devices. Therefore, we point out limitations of current devices and derive problems that might arise from these restrictions. Taking this into account, different approaches, how to display raster and vector data on mobile devices are discussed, evaluated and substantiated by concrete tests. As we will show in an example, conclusions, when to use a certain approach or image class, can be derived easily.

1. Introduction

The enthusiasm for mobile computing is still unbroken as a growth of 23% in sales of handheld devices and smart phones during the third quarter of 2002 has shown [1]. Nowadays, the opportunity to handle multimedia data opens new horizons in ubiquitous computing, and services like MMS (Multimedia-Messaging-Service) show that there is a demand for such offers. Nevertheless, handling multimedia data is still expensive regarding resources of the mobile device. Especially, if large images must be processed the system's limitations are quickly reached.

Many publications describe the processing of images in mobile environments [2,3], but limitations of mobile devices and their effect on the handling are in general only briefly described or simply mentioned. Furthermore, there is mostly no additional distinction between raster or vector data. Due to the fact, that raster and vector data have a completely different way to describe image content, they must be differently processed, and have, therefore, other needs regarding system resources.

To show in which circumstances the use of raster or vector data leads to better results during the presentation of large images on mobile devices is the aim of this publication. To affiliate valid statements, limitations of current mobile devices are shown and explained, and statements for future trends are given in Section 2. There, we focus on the display process only and neglect limitations regarding image transmission. Based on this, typical problems, which might occur during the image display are derived and demands are formulated (Section 3). To realize these demands, properties of raster and vector images together with approaches for an eligible handling are described (Section 4) and verified (Section 5). We use these approaches to give a concrete example aligned to different demands. Since there are huge differences regarding the handling of raster and vector data, we are closing our contribution by giving guidelines for the use of raster and vector images in mobile environments.

2. Limitations of mobile devices regarding a display of large images

Due to fast progression in this research field, properties of mobile devices change quickly. Thus, current limitations regarding the handling of large images are not the same as a few years ago (cp. [4]). We found, that some constrains still exist (screen size/resolution, processing power), whereas some have decreased (storage space) and some can only be found any longer in current's low cost devices (lack of color). To give the reader an impression of current hardware, we compiled a list of important properties for different devices and types (Table 1).

Screen size: The relatively small screen size is one of the major drawbacks of mobile devices if used to present raster or vector images. The display size of current devices varies dependent on the intended use and spreads from 2 to 10 inches. Thus, the available space for presenting images is very limited. One option for enlarging the screen size by keeping the size of the mobile device could be to project it onto a larger surface or to transmit it to a secondary output device [5].

Screen resolution: Related to screen size, restrictions in screen resolution are based on current display technology and limitations of the human visual system. Current screen resolution of mobile devices is low compared to stationary gadgets, and thus, affects the image handling and presentation quality. A raster image takes up a larger proportion of the screen on a low resolution device

than it would on a high resolution device and less information can be displayed. Due to common TFT-technology as visual output for PDAs, vector data must be converted into pixels. Thus, a better quality of the output signal can be achieved at high resolution, due to better approximation of the original vector data.

The screen resolution of mobile devices will undoubtedly increase, just as screen resolution in other systems, e.g. PCs, has increased.

Processing power: The handling of images, regardless if raster or vector data, is hardly affected if there is a lack of processing power. Due to the size of mobile devices, it is not possible to include hardware with similar performance of stationary devices. Among other things, this is due to the energy supply and basic architecture. Most of the current mobile devices use processors based on RISC-technology. This allows a smaller design and energy efficiency. Only Tablet-PCs are based on CISC-design (cp. Table 1).

Apart from Tablet-PCs there are currently no other mobile devices that use a GPU. Thus, the performance regarding the processing of graphical data is innately much slower even with the development of faster mobile CPUs.

Storage space: Mobile devices are also limited in temporal and permanent storage space. The need for permanent storage space differs for raster and vector data, due to the different approach to describe image content. The needed temporal memory depends mainly on the processing of image data. Interestingly, most mobile devices combine both memories into one and share the resources. Thus, current devices allow a temporal memory up to 64MB. Due to their size and strong alignment to PCs, current Tablet-PCs offer even more than 256MB. The limitation in temporal storage of data is also related to lack of processing power, since low memory slows the whole system down. Of course, with further development in this field, mobile devices will have more temporal storage, but there will still be a huge difference compared to stationary devices.

Other limitations: Beside limitations already mentioned, there are a number of additional points, which can not be explained in full detail here or which are out of scope for this paper. These are the lack of font sizes, and limited user interaction. All problems, which might occur by using data transmission, are also omitted in our considerations, since it does not affect the presentation of the image we are focusing on.

Table 1: Specification of different mobile devices (3/2003).

Device	Display	Resolution	Colors	Processing power	Storage
Mobiles:					
Samsung SGH-T100	1.8"	128x160	12bit	-	-
Nokia 7650	-	176x208	12bit	ARM 104 MHz	3.6MB perm.
Sony Ericsson T610	1.75"	128x160	16bit	-	60KB + 2MB perm.
Sony Ericsson T800	2.9"	208x320	12bit	ARM 9	16MB + memstick
Palmsize:					
Palm Zire	-	160x160	4bit	Dragonball EZ 16MHz	2MB
Palm M515	-	160x160	16bit	Dragonball VZ 33MHz	16MB, SD-Card
Palm Tungsten-T	-	320x320	16bit	TI-OMAP1510 144MHz	16MB RAM, SD-Card
Handheld:					
Sony Clie PEG-NZ90	-	320x480	16bit	Intel PXA250 200MHz	16MB RAM
Compaq iPAQ H3970	3.78"	240x320	16bit	Intel PXA250 400MHz	64MB RAM, SD-Card
Compaq iPAQ H5450	3.78"	240x320	16bit	Intel PXA 400MHz	64MB RAM, SD-Card
Fujitsu Siemens LOOX 600	-	240x320	16bit	Intel PXA250 400MHz	64MB RAM, SD-Card
Sharp Zaurus SL-5500G	3.5"	240x320	16bit	StrongARM SA-1110	64MB RAM, SD-Card
Toshiba Pocket PC e740	3.5"	240x320	16bit	Intel PXA250 400MHz	64MB RAM
Tablet-PC:					
Fujitsu Siemens ST4110	10.4"	1024x768	24bit	Ultra Low Voltage P3-M 800 MHz	512MB RAM
Compaq Tablet PC	10.4"	1024x768	32bit	Transmeta Crusoe 1000 MHz	256MB RAM
Stationary PC:					
	21"	2048x1536	32bit	Intel PIV 2800 MHz	1 GB RAM

3. Presenting image data on screen

The presentation of large images on mobile devices causes a lot of problems due to limited capabilities of such devices. In this section, these problems are affiliated and possible solutions are given.

An image, regardless if vector or raster graphic, can be characterized by the following points:

- *image size*: dimensions in image representation,
- *file size*: size of the file that contains the image,
- *file format*: description of the image content,
- *precision*: details and colors.

Here, the first point refers to the dimension of the pixel grid for raster graphics, or the vector bounding box for vector graphics.

Furthermore, main properties characterizing a mobile device regarding image handling and display are:

- *screen resolution*: the visible screen area,
- *screen properties*: the possible precision,
- *computation*: processing power and memory.

Based on these statements, critical situations, which may occur, if there are major differences between image- and devices-related characteristics can be derived:

1. The pictures *image size* is bigger than the *screen resolution* of the mobile device.
2. The image cannot be loaded into temporal memory due to low *computation* capability of the mobile device and/or *file size* and *format.*
3. The image presentation on a mobile device, due to image *precision,* does not express the original image content.

It is obvious, that even if the image utilization adds value to a mobile user, the limitations derived from the use of mobile devices impose a series of constraints, which must be considered.

Problems which might occur while displaying large images on a small display can be reduced if the image is displayed using an appropriate presentation technique. Common approaches are:

- Panning: to explore different parts of the image,
- Zooming: to get more or less detailed views of the image, and
- Hybrid techniques: a combination of both.

For obvious reasons, zoom and pan has been implemented in actually every viewer, where the needed screen space exceeds the screen boundaries. Nevertheless, to explore large images, these techniques have to be performed several times and even more often if detailed insight into the image content combined with an overview of the image is needed to understand the whole information. This is where hybrid techniques, which combine both techniques within a single view can be used. Regardless, which technique is used to reduce the problem of a small *screen resolution* during image display, the following points must be fulfilled to enable the mobile user to work in an appropriate way:

1. High presentation quality
2. Short presentation and update time
3. Eligible use of system resources

The actual realization of these demands varies depended if raster or vector graphics are used. Therefore, these points are highly relevant due to their strong dependency on device capabilities and limitations.

While hitches with *precision* are almost removed if using current hardware, problems, which occur while loading a large image file still exist. They strongly depend on the *computation* abilities of the device mostly paired with *file size* and *format*. While *file size* is mostly influenced by available temporal memory, the usefulness of a *file format* depends in general on the *computation* property of the device. Here, there are strong differences between raster and vector graphics and even within these classes. They can be exploited to enable devices with feeble hardware to load large images anyhow.

In the next section, we describe different ways to present large images on screen. They will be evaluated based on the demands derived in this section.

4. Presenting raster and vector graphics

Before display, the image content has to be rendered onto a discrete screen raster. This is in general a device context, which provides an interface to draw information to screen. To display raster as well as vector graphics, this screen device context must be used. Conceptually, it can be seen as a simple bitmap with a certain size no larger than the current *screen resolution*.

Raster graphics, like Bitmap-, GIF- or JPEG-images, describe image content by single image points, which are arranged on a regular 2D-grid of certain *image size* and *precision*. They are used in areas where a content description by geometrical objects is difficult or even impossible, e.g. digital photography. Every picture element is independent from each other regarding its color. Since the screen device context uses the same principle, no conversion is necessary to map the image bitmap to display. Nevertheless, a high presentation quality can only be achieved if the image content is not further modified. This affects especially zooming operations, where pixels must be removed or inserted before mapping. Here it is a fact, that fast algorithms lead a to worse presentation quality and vice versa, and an alignment depended on external demands is necessary.

Additionally, while processing, pixel values must be stored in memory. Thus, a space of at least

$$image\_size_X \times image\_size_Y \times \frac{precision}{8}$$

bytes is needed to hold a copy of the image in memory. Here, *precision* represents the available number of image colors and is specified in bit per pixel (bpp). Accordingly, this can be quite expensive in terms of memory space if using large images.

Furthermore, this large amount of data must be read, loaded and sometimes even decoded, which also increases the need for processing power. This depends basically on the used *file format,* whereby formats producing a small *file size* need generally more processing power for decoding, but are faster to read and load.

Vector graphics use graphical primitives and their attributes to describe image content. Such primitives are points, lines, rectangles, circles, with attributes like fill-, and stroke-color, or stroke-width. Due to this principle, it is necessary that the image content can be described with such primitives. A typical application for vector graphics are technical drawings. Macromedia Flash [6] and SVG (Scalable Vector Graphics) [7] are common *file formats* to store such graphics in mobile environments.

Due to their analogue nature, vector graphics have the property of inherent levels of detail at any scale. To achieve different zoomed as well as panned image views, a transformation matrix is applied to primitives. Thus, only one manipulation of the matrix is needed to pan and zoom the image without any decrease in quality. Of course, this approach achieves very good visual results. Nevertheless, this high quality can not be carried to the display since it decreases slightly during the needed raster-conversion and mapping of the vectors to the screen device context. Furthermore, the conversion is costly in terms of processing power, especially if using a large number of vectors, but can be even worse if areas must be filled or refreshed. Thus, the needed raster-conversion is one of the major drawbacks of vector graphics regarding display quality and processing power. To reduce processing power, in certain circumstance an internal bitmap representation of the vector data can be useful. We refer to this approach as *indirect drawing*, while *direct drawing* renders vector data immediately to the screen device context. Indirect drawing makes sense especially if mainly panning is used, since than the internal bitmap representation does not need to be altered and all the advantages of raster graphics regarding a fast mapping of a bitmap to the screen device context can be exploited. Anyhow, if using the indirect drawing, an internal bitmap representation also needs additional memory.

Concerning the general requirement for the use of memory, less temporal memory is needed for vector than for raster data. This is due to the reason, that the description of a primitive is in general smaller than its pixel representation. Thus, *file size* compared to raster graphics is also smaller, and the demands on processing power while loading are less. This might not always be the case, and depends strongly on *image size* for raster and the number and kind of primitives for vector graphics. Here, the used *file format* plays also an important role, but statements given for raster graphics can applied accordingly.

In the next section, the given statements will be verified by concrete tests using current mobile hardware. Furthermore, the derived results are used to implement an interactive presentation technique for large images under consideration of the stated limitations and demands.

5. Pixels vs. Vectors: Experimental results

In this section, we present some results derived from our experiments with large raster and vector images on mobile devices. The results are taken by using a Toshiba e740 running with Pocket PC 2002 and a Sharp Zaurus SL550G running with Linux (cp. Table 1). The programming environment we used to develop our testbed was Personal Java, Specification 1.2a. For running the testbed, we used the Jeode-Runtime environment.

Presentation quality, presentation- and update time, the influence of *file size* and *format*, and the use of system resources are the key figures we are focusing on. They are used for the following tests, which will compare raster and vector images regarding the display of large images on mobile devices.

5.1. Presentation quality

Here, we want to compare presentation quality for differently zoomed and panned images.

Regarding a panning operation raster and vector images deliver the same results. This is due to the nature of panning, which is simply changing the part of the image content that is currently displayed. Since such an operation does not alter image content in any manner, quality is not decreased in both classes.

With regard to zooming, the results are rather different. Here, the image content must be scaled. We got the worst results in presentation quality if build-in system functions are used to scale bitmaps. Especially, the downscaling procedure is rather simple and omits pixel rows and/or columns without regard of their color value. Hence, the results are poor. We got better results by using a more complex approach (i.e. filtered scaling), but in certain cases there is still some important image content removed. For vector images, we achieved the best visual results. By using
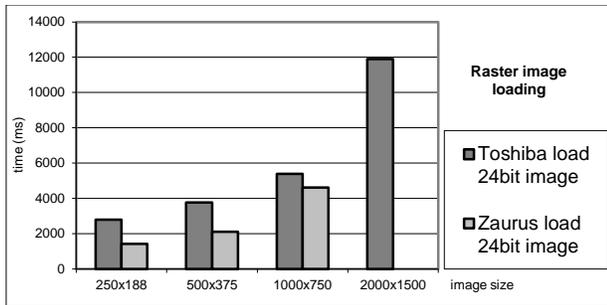
Figure 1: Raster image loading using BMP-images.

vector transformation, the image content can be scaled without any loss of information. However, before drawing the transformed primitives to display a raster conversion is necessary. Since display resolution is generally low on mobile devices this results in a decrease of quality unless anti-aliasing methods are used. Nevertheless, the presentation quality of vector data is in general quite good compared to raster data.

5.2. Presentation- and update time

Results regarding the response rate of a mobile system can be split into two parts. The presentation time, which spread from loading and creating a memory representation of the image until the first presentation, and the update time, which is the time required to render a new presentation of the image after interaction.

With regard to loading raster graphics, examples of different *image size* have been selected. As shown in Figure 1, loading time increases linearly with *image size*. Due to the reason, that used *image format* plays also an important role, we measured properties of BMP- and JPEG-Images regarding loading time. JPEG-encoded images are much smaller than BMP-encoded images, and, as Figure 2 shows, can be read more than 10 times faster. While *file size* of BMP-encoded images depends only on *image size*, it may vary for JPEG-images. As shown, if image content changes, compression ratio and file size are also different, which affects time needed to read the image. To state loading time, processing power to decode image content must be also considered. Since image content in BMP-files is stored uncompressed, no additional efforts are necessary. Not surprisingly, JPEG-images need significantly more time for decompression than for simply reading the file from permanent memory.

Additionally, we found that image loading is generally faster on the Zaurus, but due to memory limitations, it was not possible to load the largest image (2000x1500x24bit) on Sharp's device.

For vector images, it is more difficult to derive statements regarding loading time. Here, loading time depends even stronger on the used *file format* than it does
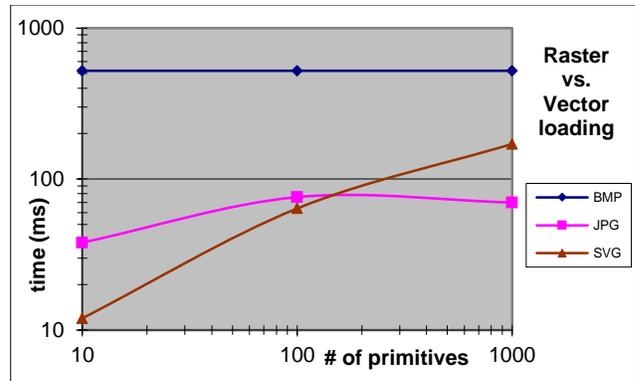

Figure 2: Time needed to read same image content using different raster and vector formats.

for raster images. While for raster images syntax and semantic is inherent in binary pixel representation, vector image syntax has to be parsed and analysed, before semantic can be added in order to create a valid internal representation. If using Macromedia Flash, which is based on an optimized and even compressed binary description, files are fast to read. Contrary, the vector format SVG is based on an XML grammar. Here, the *file size* is in general bigger, and the processing and interpretation of the file content costs much effort. Thus, loading SVG files is innately slower than loading Flash. Due to the absence of a freely available SDK for assessing Flash-Files, we constricted our measures to the loading of SVG-files (cp. Figure 2). As assumed, *file size* and therefore loading time correlates with the number of primitives used to describe the image content.

As seen in Figure 2, time to load raster graphics varies only slightly if image content is changed. Contrary, vector graphics depend strongly on the number of primitives used to describe the actual image content. Thus, if only a view primitives are needed, vector graphics are very fast to load. The break-even in our example is reached by using slightly more than 100 primitives. The concrete value depends on the complexity of the used primitives. If more primitives are needed better results are achieved by using raster graphics.

If an image has been loaded, i.e. is available in temporal memory, it can be drawn on screen. Raster images can be rendered directly to display. Thus, display pixels must only be set to a certain color and update time mainly depends on *screen resolution*. As shown in Figure 3, if *image size* exceeds *screen resolution* update time does not change anymore. If *image size* is lower than *screen resolution*, obviously, an even faster update time can be achieved.

For vector images, we decided to measure the display time of several kinds of graphical primitives. In doing so, the measures are independent of image content and more general statements can be given. For a different number of
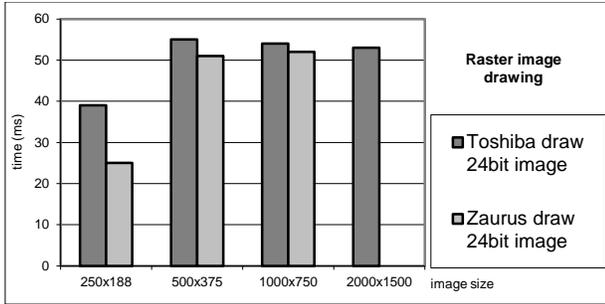
Figure 3: Raster image drawing.

primitives update times for drawing and filling using *direct* and *indirect* (see Figure 4) drawing have been measured. As figures show, *direct* drawing is fast if only a few primitives must be drawn. If the number of primitives increases, the update time for *direct* drawing increases dramatically. Here, *indirect* drawing, using a background bitmap, is up to 3 times faster, but needs more time if only a few primitives must be drawn.

Interestingly, drawing time strongly depends on the used system. In our case, we found that the Zaurus machine is significantly faster than Toshiba's e740. Thus, as complexity of the vector primitive is increased (lines, rectangles, triangles), the break-even point is reached earlier on the Zaurus.

Regarding an interactive image manipulation using panning and zooming, we examined how panning and different scaling operations affect update times. Since panning simply draws a different image part, there is no

difference to ordinary update time for raster images. For zooming we compared low quality and high quality built-in system functions. The performance of the approaches in respect of response rates is highly related to the quality of the presentation. The results we got in our experiments show the eligibility of this assumption. We can conclude that approaches, which offer high quality, are slow regarding their response rates, while simple operations lead to fast execution time (see Figures 5 and 6).

Panning and zooming for vector images is realized by using a transformation matrix. In order to compare transformation times, we measured how long such matrix multiplications take (see Figure 7). This depends especially on the number of primitives, respectively points. We found that with our tested mobile devices vector transformations can be computed rather fast. As expected, calculations in double precision are slower than in integer precision. Here again, the concrete results depend on the used system. In our case, transformations calculated with Toshiba's e740 are 2 times faster than on the Zaurus.

An interesting fact discovered during our studies is that integer transformations are even faster than on a stationary desktop computer (Toshiba e740 vs. 1,7 GHz Pentium4).

Overall, drawing of vector data needs in general more computing power than for raster data. Nevertheless, if the number of primitives is very small, the efforts to draw them can be less than to draw a whole bitmap image.
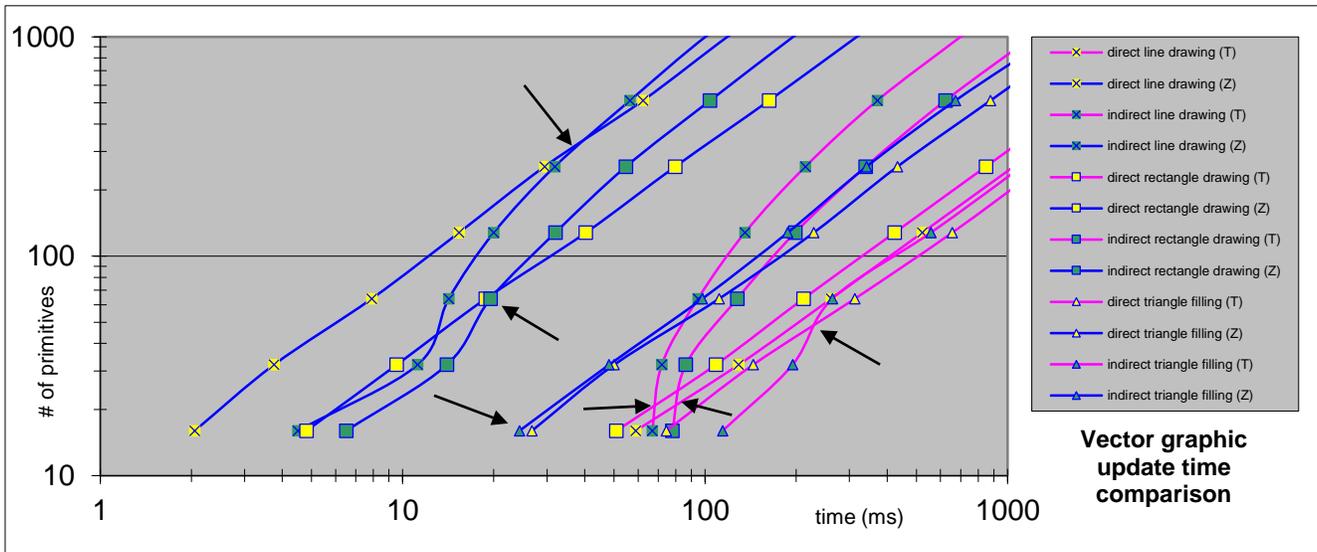

Figure 4: Comparison of update times for *direct* and *indirect* drawing of different vector primitives on different mobile devices (Toshiba e740 (T) and Sharp Zaurus SL550G (Z)). Arrows depict break-event points of *direct* and *indirect* drawing.
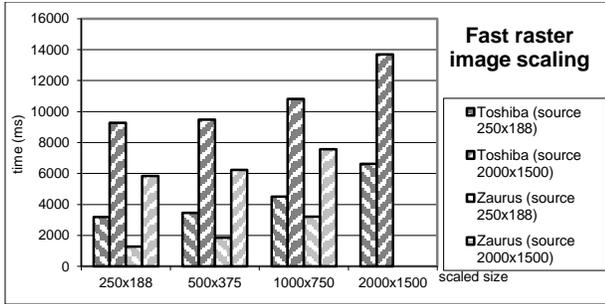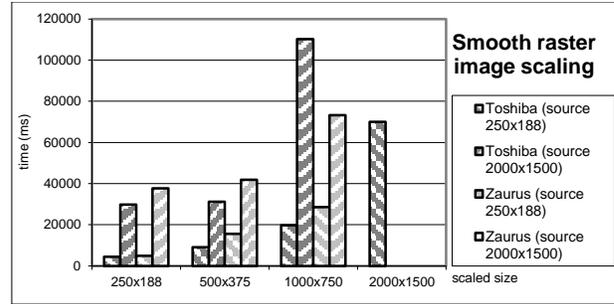
Figure 5: Fast raster image scaling.

Figure 6: Smooth raster image scaling.

5.3. Use of system resources

In this paragraph, we focus on the use of system resources, in particular the used memory. Nowhere, the difference in handling of raster and vector images is as large as by considering this point. Whereas, for raster images a bitmap copy of the original image has to be stored in memory, pure vector based approaches need to store primitives as representation of the image content. Thus, there are completely different needs on memory requirements.[1]

Both classes need a screen device context for drawing onto screen. This context has to be created or is provided by the system and needs at least a memory of

$$screen\_resolution_x \times screen\_resolution_y \times screen\_properties$$

bit. In our test environment this size is around 150kB.

To keep a copy of the image content in temporal memory, for raster graphics every pixel must be stored. For an image of 1024×1024 pixels with a precision of 16 bpp, the memory is around 2MB.

For vector images an internal representation of the primitives must be stored. There are a large number of different approaches to create such internal representations (e.g. a DOM[2] tree for SVG) that depend heavily on the used vector format and its properties, like precision and the number of objects. Due to the variety of possible approaches and implementations, no general statements can be given. One of our example images consists of around 10000 polylines, whereby every line is stored together with its properties in a 20 byte memory structure. Thus, the space required to keep an internal representation of the complete image in memory is around 195kB.

We can conclude, that regardless if using raster or vector data, for almost every current mobile device the provided temporal memory is enough to display even large images. Only by using ultra-mobile devices, like mobiles and low-cost PDAs (cp. Table 1), or if parts of the memory are shared with the operating system (e.g. for a RAM disk

like on Zaurus) the problem may occur. All other devices still have some reserves to serve even bigger images. Hence, we did not consider more efficient techniques, which keep only parts rather than the whole image in memory.

5.4. Giving measures a meaning

Apart from simply zooming and panning, images can be also represented by using more sophisticated display-techniques. Nevertheless, such approaches can be mostly seen as a combination of panning and zooming applied to different parts of the image and screen.

Here, we want to show how statements of the previous subsections can be used to realize an suitable mobile implementation of the rectangular FishEye-View [8]. Thus, we mainly consider the critical points: presentation quality and response rate.

The FishEye-View consists of an undistorted or even zoomed focus area embedded into a distorted context (see Figure 8). The degree of context distortion/scaling is based on size, position and zoom factor of the focus.

Since raster image drawing is fast, even on mobile devices (around 50 ms for a display filling image), our first approach was a FishEye-View based on raster data. Nevertheless, to create the context a number of scaling operations must be performed. As stated above, scaling of
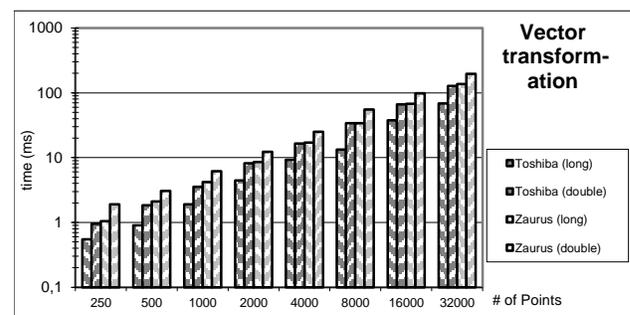

Figure 7: Vector transformation.

---

[1] Here, we neglect the memory needed to store and manage the program code during execution.

[2] DOM = Document Object Model.

raster data is computational expensive if a high presentation quality must be achieved. Using our testbed, high quality downscaling of a 2000x1500 pixel image takes about 40 seconds. Low quality scaling is faster. Thus, we use low quality scaling during interaction, and consider the response rate as more important than quality. Later on, if the system is idle, the context is refined using a high quality scaling. Nevertheless, scaling operations are the bottleneck. Therefore, strategies for reducing the number of scaling operation are needed. Since there should be sufficient memory, using several context bitmaps at different pre-calculated resolutions is a possible solution. Hence, during interaction, a scaling of the image is not necessary and the context is created by executing fast panning operations on the context bitmaps. By using successive refinement and context bitmaps an interactive high quality FishEye-View for raster images can be realized.

Nevertheless, even by using high quality scaling, valuable image content is removed. If visual information is available as vector data, the previous approach can be even enhanced. For this purpose, the vector data is used to create high quality focus and context bitmaps, which will be used instead of the lower-quality raster-scaled versions. Thus, a fast scaling coupled with a high quality image presentation by an increased memory consumption can be achieved.

To show the interested reader that properties of raster and vector graphics must be selected depended on current demands, we present a second and completely different implementation of the FishEye-View. Here, another advantage of vector graphics, fast vector transformations, is used to create a less memory consuming version of this display technique. Different transformation matrices are used to transform the focus and context regions on the fly. Up to 32000 points can be transformed in less than 150 milliseconds using double precision matrix multiplications. While this works very well for a small number of primitives (<500), it is impossible to render more than these at interactive response rates. In such circumstances, we use *indirect* drawing, which is generally faster than *direct* drawing.

This version of the FishEye-View generates image presentations at highest quality, whereas update time heavily depends on number and kind of vector primitives within the image.

As shown for the rectangular FishEye-View, the stated advantages and disadvantages of vector and raster graphics can be used to decide when to use each graphic type, and how to combine them in interactive presentation techniques for large images depended on current demands and constrains.

Figure 8: Rectangular FishEye-View.

6. Conclusions

Summarizing the paper, we reviewed the frequently quoted restrictions of mobile devices regarding their needs for presentation of large images. We found that some of the limitations no longer exist (lack of color) or will be obsolete in the near future (memory), while others still affect image presentation (e.g. display size). Furthermore, we described how these limitations affect the presentation of large images. Different guidelines to overcome these problems are given dependent if using raster or vector data. These statements are verified by concrete examinations and can be summarized as:

1. Drawing *raster* data is *fast* on mobile devices.
2. A simple *scaling* of raster data is *fast* but leads to *low quality* presentations.
3. Filtered *scaling* of raster data is *slower* but leads to *more appealing* presentations.
4. Matrix multiplications for *vector transformations* are *fast*.
5. Rendering *vector primitives directly* to display is *generally slow*.
6. Rendering *vector primitives indirectly* (first to a background bitmap and than to screen) is *faster*.
7. Loading time depends strongly on the used file format.
8. Loading of *vector data* is *fast* if a certain number of primitives is not exceeded.

Our claim was also to answer whether raster or vector images are more suitable for presentation of large images on small mobile devices. We found that both graphic classes have their eligibility depended on image content and external demands. Nevertheless, raster images offer in general the best trade-off to reduce constrains caused by limitations of current hardware. Our measures show that raster operations are fast. Drawing a display filling raster image takes about 50 milliseconds. During this time just around 100 vector primitives can be drawn. Additionally, raster images are easier to handle since their representation

is generally not as complex as for vector images. On the other hand, vector graphics offer better quality, especially for scaling operations, and are still the first choice for applications focusing on this demand. Furthermore, if image content can be described with only a few primitives, vector data might be even faster processed than raster data. As computational and graphical power of mobile devices increases in the future, the application area of vector graphics will also.

References

[1] CONTEXT – IT Information services, "Newsletter", London, October, 2002.

[2] I. Chisalita and N. Shahmehri: "Issues in Image Utilization within Mobile E-Services", In: *Proceedings of 10th International Workshops on Enabling Technologies*, June, Massachusetts, 2001.

[3] T. Rist: "Customizing Graphics for Tiny Displays of Mobile Devices", In: *Proceedings of IPNMD01*, Italy, 2001.

[4] I. Cooper and R. Shufflebotham: "PDAWeb Browsers: Implementation Issues", *Technical Report 11-95*, University of Kent*, UK, November, 1995.

[5] R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar, and John Light: "The Personal Server: Changing the Way We Think about Ubiquitous Computing", In: *Proceedings of UBICOM02*, Springer, LNCS 2498, Sweden, 2002.

[6] S. Wolter and S. Ünlü: "Flash MX - Grundlagen und Praxiswissen", Galileo Press, Bonn, 2002.

[7] World Wide Web Consortium W3C: "Scalable Vector Graphics (SVG) 1.0 Specification", www.w3c.org/TR/SVG/, 2001.

[8] U. Rauschenbach, S. Jeschke, and H. Schumann: "General Rectangular FishEye Views for 2D Graphics", In: *Proceedings of IMC2000*, Rostock, 2000.