Design of user interfaces for selective editing of digital photos on touchscreen devices

Thomas Binder, Meikel Steiding, Manuel Wille and Nils Kokemohr

Google (formerly Nik Software)

ABSTRACT

When editing images it is often desirable to apply a filter with a spatially varying strength. With the usual selection tools like gradient, lasso, brush, or quick selection tools, creating masks containing such spatially varying strength values is time-consuming and cumbersome. We present an interactive filtering approach which allows to process photos selectively without the intermediate step of creating a mask containing strength values. In using this approach, the user only needs to place reference points (called control points) on the image and to adjust the spatial influence and filter strength for each control point. The filter is then applied selectively to the image, with strength values interpolated for each pixel between control points. The interpolation is based on a mixture of distances in space, luminance, and color; it is therefore a low-level operation.

Since the main goal of the approach is to make selective image editing intuitive, easy, and playful, emphasis is put on the user interface: We describe the process of developing an existing mouse-driven user interface into a touch-driven one. Many question needed to be answered anew, such as how to present a slider widget on a touchscreen. Several variants are discussed and compared.

Keywords: Selective image editing; reference point; user interface design; touchscreen device; mobile phone; slider widget; computational photography

1. INTRODUCTION

In interactive image editing it is a frequent requirement to apply a filter to an image and, at the same time, to spatially vary the application strength of that filter. For example, in an image of a landscape the user may want to change the appearance of the sky while leaving the vegetation untouched. In other words, every pixel in the image should have its own filter application strength value based on its context. These strength values can be stored in a mask. While applications like Adobe Photoshop® are able to vary certain parameters based on a mask, the creation of the mask itself may be difficult and time-consuming, or even impossible (depending on the quality requirements). Standard mask creation tools are limited to rectangular or elliptical selections with hard or feathered edges, as well as selections created via gradient, lasso, and brush tools. Masks are quickly created by the first tools; at the same time their use is limited. The converse is true for the lasso and brush tools. Photoshop's quick selection tool also takes into account color and textures, however extra work is required to feather the boundaries of the selection.

The present work introduces an interactive filtering approach which allows to process photos selectively without the intermediate step of creating a mask. After adjusting the spatial influence and filter strength for each control point, the filter is applied selectively to the image, with strength values interpolated for each pixel between control points. Note that the filter does not need to be specified; control points are compatible

Corresponding author: Thomas Binder `tbinder@google.com`.

Cite as: T. Binder, M. Steiding, M. Wille, N. Kokemohr, "Design of User Interfaces for Selective Editing of Digital Photos on Touchscreen Devices," Proceedings SPIE Vol. 8667 (Multimedia Content and Mobile Devices), 86671N (2013). DOI: `http://dx.doi.org/10.1117/12.2003147`

with arbitrary filters. A pixel that is both close in space and similar in brightness and color receives a high application strength, while a pixel that is far in space or different in brightness or color is assigned a low one, or is not affected at all. It is important to realize that no segmentation of objects is performed nor is any kind of high-level knowledge of image contents required. The power of the approach lies in its simplicity.

In Section 2 we describe control points, their parameters, and how the interpolation works. Since the main goal is to make selective image editing intuitive and easy, the user interface (UI) is a crucial aspect. UI issues are discussed in Section 3. A conventional, mouse-driven UI approach for control points is sketched. With the advent of mobile devices we were faced with the question of how to present that UI on a touchscreen. The problem that needed to be solved was how to present and operate a slider on a touchscreen. For reasons of continuity, we did not want to emulate brushing the application strength mask on the touchscreen, but rather to carry over and adapt the principles of the mouse-driven UI. The reasoning behind the development process is described. For example, the variables in Fitts's Law[1] must be reinterpreted for touchscreens. Several touch-driven UI variants of control point based selective image editing are discussed and compared. One of the approaches is implemented in Nik Software's Snapseed for iOS App.

The Snapseed App has been released for iOS and Android. Both platforms have individual design guidelines[2,3] making sure all apps behave consistently on that platform. The present work does not deal with such platform differences, since it was one of the development goals to achieve near identical behavior of the iOS and the Android App and establish a "Snapseed brand". Contrasting other work[4,5] user studies were not carried out in the scope of the present work.

In the following we discuss further related work. Note that, due to the extremely fast development of mobile hardware, many of the available texts appear outdated. The book by Shneiderman and Plaisant[6] is a current standard reference on human-computer interaction. Fitts's law[1] has been studied in the context of touchpads.[7] Wasserman[8] and Nilsson[5] discuss aspects of software engineering for mobile devices and user interface design patterns, respectively. The concept of control points as outlined in this paper has been branded as U Point® Technology, which is protected by patents[9,10] in major markets. Special topics like saving battery power by avoiding certain colors,[11] using tactile feedback[12] which is available in most mobile phones, or including gyroscope and accelerometer feedback for a better user experience are outside the scope of this paper.

2. THE BACKEND

2.1 How control points work

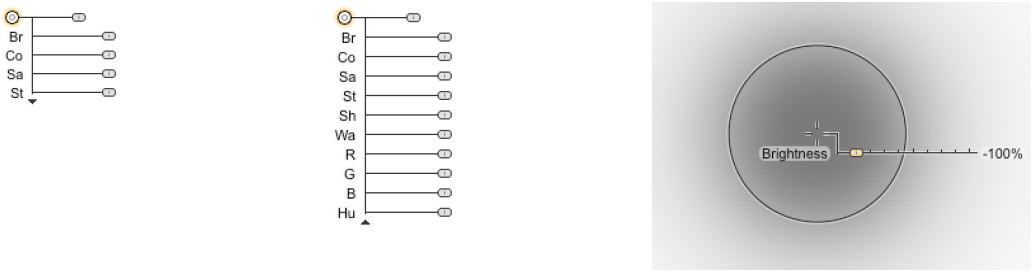
A standard type of control point, present in Nik's Viveza 2 plugin, is called Brightness-Contrast-Saturation control point (see Figure 1). Such control points represent a superposition of many filters. For illustrative purposes it suffices to deal with a single filter, say saturation. Note that all filters in question are pixelwise, i.e. no neighborhoods are required in order to compute the result of a single pixel. We describe three ways for implementing control points. They follow the same basic idea but differ in details. The alternatives illustrate design decisions that have to be made in order to adapt the idea to a specific situation.

Let $I : \mathbb{R}^2 \to \mathbb{R}^3$ be a digital color image which should be filtered. Then a pixel p of I at location (x, y) may be viewed as a 5-tuple $p = (x, y, l, a, b)$, where l, a and b are YCC values at position (x, y) in I. For a size value $r > 0$ the distance function $d_r : \mathbb{R}^5 \times \mathbb{R}^5 \to \mathbb{R}^+$ computes a distance between pixels using this interpretation; it is defined by

$$d_r(p_1, p_2) = \frac{A}{r}\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} + B|l_1 - l_2| + C(|a_1 - a_2| + |b_1 - b_2|).$$

Here A, B, C are positive constants defining the relative influence of size, brightness and color, respectively. The size value r governs the spatial extent of the influence of a control point. Note that, according to our experience, the actual choice of transformation from RGB to YCC, and also questions of transforming inputs to an absolute color space do not matter in practice. A computationally simple transformation is therefore a good choice. Since our computations are based on influence strengths or weights rather than distances, distance values must be

Figure 1. Brightness-Contrast-Saturation control points of Viveza 2. Left: Collapsed view of a control point, showing only the size slider (top, no label) and the sliders for Brightness (Br), Contrast (Co), Saturation (Sa), and Structure (St) filters. Middle: Expanded view of the same control point, obtained by clicking the small triangle on the bottom. The expanded view reveals further sliders which are also affecting hue. Right: While any of the control sliders is dragged, the control point changes appearance as shown here: All but the dragged slider disappear transiently, the slider name is spelled out, the slider value is shown as percentage, and the circular outline of the selective effect is displayed. The spatial extent of the selective effect and the radius of the circular outline changes while dragging the size slider.



inverted in order to obtain weights: the larger the distance the lower the weight. For this purpose we use a function $f : [0, \infty) \to [0, \infty)$ which is monotonously decreasing and quickly approaching zero; an exponential function $f(d) = \exp(-\mu d)$ where $\mu > 0$ is a suggesting choice.

We are now ready to describe the first control point variant for our saturation filter. The user selects a global saturation parameter $\gamma_0 \in [-1, 1]$ to be applied to I. For example, $\gamma_0 = 0$ stands for no change, and $\gamma_0 > 0$, $\gamma_0 < 0$ increases or decreases saturation, respectively; $\gamma_0 = -1$ yields a grayscale image. The global change is modified selectively by placing n control points on the image, located at pixels $u_1, \ldots, u_n$, associated with size values $r_1, \ldots, r_n$ and local saturation parameters $\gamma_1, \ldots, \gamma_n \in [-1, 1]$. Processing the image selectively amounts to computing an individual saturation parameter γ for each pixel p by

$$\gamma = \frac{f(d_0)\gamma_0 + \sum_{i=1}^{n} f(d_{r_i}(p, u_i))\gamma_i}{f(d_0) + \sum_{i=1}^{n} f(d_{r_i}(p, u_i))}, \tag{1}$$

where the value $d_0 \in \mathbb{R}$ is the distance to an additional imaginary control point, located at a large distance, representing the global saturation change. Note that in (1) the local parameters γ_i have the same impact and scale as the global parameter γ_0: in a neighborhood of a control point, its local parameter will roughly replace the global one. However, for reasons discussed later, we found that sometimes this approach is not optimal. We prefer a second variant in which the local parameters act as a modification of the global value. This can be achieved by

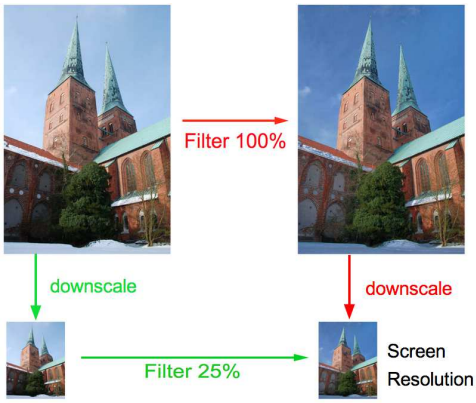
$$\gamma = \gamma_0 + \frac{\sum_{i=1}^{n} f(d_{r_i}(p, u_i))\gamma_i}{f(d_0) + \sum_{i=1}^{n} f(d_{r_i}(p, u_i))}. \tag{2}$$

Control points according to (1) and (2) are called absolute and relative control points, respectively. All Nik products currently in the market, with the exception of Color Efex Pro 4, as well as the older Dfine 2 and Sharpener 3, follow the relative approach.

The third way does not require the availability of a filter parameter and is thus more generic. So suppose all we have is the input image I and the filtered image $\tilde{I}$ and we still want to be able to apply the filter selectively. In this case selective filtering can be done using opacity blending between I and $\tilde{I}$ based on a spatially varying mask. The corresponding control points are called selection control points. Suppose $\tilde{p}$ is a pixel of $\tilde{I}$, the filtered result of pixel p in I at the same position. In the presence of absolute control points an opacity σ is computed as in (1), namely

$$\sigma = \frac{f(d_0)\sigma_0 + \sum_{i=1}^{n} f(d_{r_i}(p, u_i))\sigma_i}{f(d_0) + \sum_{i=1}^{n} f(d_{r_i}(p, u_i))}, \tag{3}$$

Figure 2. Possibilities for accelerating image processing. Left: Lazy filtering — just the displayed tiles (inside the main window) are computed. Right: Filtering on the unscaled original image (red/dark arrows) is safe but slow, while filtering in screen resolution (green/light arrows) is faster and safe as long as the diagram commutes.



where $\sigma_0 \in [0,1]$ is the global opacity and $\sigma_i \in [-1,1]$, $i = 1,\ldots,n$ are the local modifications. Finally, the actual output pixel is given by $q = \sigma p + (1-\sigma)\tilde{p}$. Control points in Nik's Dfine 2, Sharpener 3, and Color Efex Pro 4 are implemented according to (3).

Sometimes it is desirable to judge the combined strength of control points at a particular location. Viveza 2 offers the possibility to do so on a user-selected subset of all control points. Since estimating the influence from the actual filtering effect can be hard, showing a strength mask is more appropriate. Such a strength mask can be displayed upon user request or while dragging the size slider of a control point. It is given by

$$m = \frac{\sum_{i=1}^{n} f(d_{r_i}(p,u_i))s_i}{f(d_0) + \sum_{i=1}^{n} f(d_{r_i}(p,u_i))} \in [0,1], \tag{4}$$

where $s_i = 1$ if the strength mask should include control point $\#i$ and $s_i = 0$ otherwise, and m is the value of the strength mask at p, ranging from black ($m = 0$) to white ($m = 1$). The relationship of (4) with Eqs. (1)–(3) is immediately clear.

While the above formulas illustrate the general principle, immediate results obtained this way will not be satisfactory from a photographic point of view. A number of additional steps and tweaking is required for overall high quality. For example, if the input image I is noisy, its noise is propagated to the values obtained via Eqs. (1)–(3). This may cause heavy and abrupt changes of the filtered image preview while changing the position of a control point. Such an effect is undesirable since it counteracts the predictability of the application's behavior. A straightforward solution is to replace I by a denoised or smoothed version of I as input to Eqs. (1)–(3). The input to the actual filtering would still be the original image I.

2.2 Performance considerations

As outlined control points can be added on top of any filter. On the upside, there is no performance cost when the user decides that the global change is sufficient and does not use control points ($n = 0$). The overhead for n control points is roughly n times the overhead of a single control point. Control points tend to become expensive when there are many of them — and current control point duplication and grouping features encourage the user to have many. Since control points occur in all of Nik's products, it is worthwhile to consider performance topics.

Figure 2 shows two well-known general concepts for speeding up image processing applications. First, splitting the image memory into tiles is a good idea. On the one hand, it is a technical requirement, since the memory address space is often heavily fragmented. So it is not reasonable to expect that a large amount of memory can

Figure 3. The desktop UI by example of Viveza 2. The main window is shown in the middle. Top right: Controls for adding and grouping of control points. Middle right: If no control point is present, or no control point is selected, the main sliders are the global sliders. Whenever a control point is selected, the main sliders represent the selective sliders of that control point (the latter case is shown). Lower right: List view of all control points and groups of control points, with checkboxes to enable/disable, to display the strength mask. Lower left: Expanded view of control point (enlarged section from main image).

be allocated in a single piece. As a rule of thumb we found that allocating more than 20MB in a single piece is likely to cause problems on 32bit operating systems. On the other hand, dealing with smaller memory chunks reduces caching costs: the smaller the tile size, the closer together the memory addresses of neighboring pixels in the same column. A common tile size is 512 by 512 pixels. Second, filtering in the requested resolution rather than at full resolution is essential and often possible.

With the prevalence of multi-core CPUs and modern graphics hardware, parallelization of computations has become a key issue. For an implementation on a multi-core CPU, tiling once more is advantageous since the tile computations can run in parallel on the cores. On the other hand, implementing control points in any form is straightforward, non-recursive, and does not require spatial neighborhoods. It is therefore well suited for a GPU implementation.

One can come up with more sophisticated ideas for cutting back on the computational complexity of control points. Depending on the choice of the function f the effect of control points is attenuated quickly with increasing spatial distance. Therefore, when computing a certain tile, all control points farther away from that tile than a certain threshold could be ignored. Another example of an optimization is based on the assumption that the user will not change all control points within a single action, but will touch control points one after the other, adjust position and sliders successively for each control point. Therefore just one summand $i = i_0$ in the sum in (1), (2), (3) changes. An implementation could take this into account by reusing the value $\sum_{i=1, i\neq i_0}^{n} f(d_{r_i}(p, u_i))$.

3. THE USER INTERFACE

3.1 The desktop UI

The purpose of this section is to describe and discuss a desktop UI for control points, based on the UI of Viveza 2. The same approach is present in other Nik Software products, with only slight modifications. Viveza 2's control point UI is shown in Figure 3.

Figure 4. The mobile UI by example of the Selective Adjust module of Snapseed iOS. The main display is shown in the middle. On the bottom of the main display there are controls to add control points, display the value of the currently selected slider, hide all control points, undo the last operation, and final application and exit (from left to right). Top right: Transient view of control point during a strength change. Bottom right: Transient view of control point during a size change; the effect strength is visualized by a red overlay. Bottom left: Transient view of a filter change (enlarged section from main image).

The control point desktop UI is "backward compatible" with ordinary UIs in the sense that it lists the global, non-selective sliders on the right-hand side. The user can dial in all global effects via these sliders just as in any other photo editing application — knowledge and understanding of control points is not required. The main and obvious difference is the Add Control Point button (cf. top right of Figure 3). This button is the key to selective adjustments: Once pressed it is released with a click on the main image preview, in the location where the selective adjustment should be made, which adds a control point at that location. Since control points are relative in most products (cf. (2)), the added control point initially will not change the filtering effect; all of its sliders are still in the neutral position. The user selectively adjusts the effect in the vicinity of the control point by adjusting its sliders. He/she proceeds to successively place control points, and to adjust their sliders and position until satisfied. The selective editing workflow is finalized by pressing the OK button in the lower right corner of the application window, which initiates the final filtering at full resolution.

The desktop UI features displaying the strength mask (4) instead of the main image preview. In order to show the strength mask, the user has to switch on the strength mask checkbox of at least one control point (right of percentage values in the control point list at the bottom right of Figure 3). The strength mask is displayed for the checked control points. The applications returns into the regular mode once the last checkbox is unchecked. The strength mask shows how control points influence each other, and the characteristics of this influence is an element of the workflow: suppose the effect of a control point "bleeds" into an adjacent object of brightness and color close enough to the one below the control point. In many cases such bleeding is undesirable. It is then recommended to place a neutral control point on this object which blocks the selection of the first control point. Put differently, the user is advised to add control points until satisfied. This effect of control points influencing each other can be seen from the strength mask by switching on the strength mask checkboxes of the control points in question.

In Viveza 2 it is possible to select a control point by clicking on it, to copy it to a clipboard, and to insert a new control point with identical slider settings, at a position chosen by an additional mouse click, via the usual Control-C and Control-V keyboard shortcuts. A single control point can be duplicated using Control-D, with automatic positioning next to the original control point. It is also possible to group control points, in which case any operation made on a single control point is automatically forwarded to all other control points in the group. Copy and paste actions refer to single control points only, deletions work on groups as well. Using the duplicate action it is simple to obtain large control point counts and achieve fine control over the selective application of the filter in question.

A further feature which has not been mentioned so far is the possibility to choose a target color at a particular pixel. This feature comes in handy when a similar look should be achieved across several images, or when applying control points prior to panorama stitching. It works as described in the following. When clicking on the color patch below the main sliders (not shown in Figure 3), a color picker pops up. Selecting a color here starts a search algorithm which sets the control point slider values such that the selected color coincides with the color under the control point *after* filtering. There is also a pipette tool to pick that color from the main image preview. Clearly, this makes only sense when all three dimensions of color can be modified via control points. This is the case for the Viveza 2 variant in Figure 1 where the color space is spanned by either brightness, hue, saturation sliders or red, green, blue sliders. Again, the search algorithm determines slider settings that transform the pixel color under the control point to the target color. Since there usually will be more than one set of slider settings that achieves this, the one with the lowest sum of slider elongations is chosen.

3.2 The mobile UI

Figure 5 shows three variants of mobile UIs which are compared in Figure 6. Variant 1 is a straightforward mobile analogy of the desktop UI. The mobile UI of the Selective Adjust module of the Snapseed App is shown in Figure 4 and is equivalent to Variant 3 shown in Figure 5 (c). Although global sliders are not available in Variant 3, it would be possible to implement them in a way similar to the desktop UI: When no control point is selected, vertical and horizontal swipes are available to control the choice of filter and its global strength, respectively. Variant 2 is between Variants 1 and 3: its disadvantage is that the global value is detached from the local value, and one cannot see the current state of everything easily.

One of the main questions that had to be answered is how to present a slider on a touchscreen device. Most image processing applications running on mobile hardware follow the conventional, desktop-like approach: the control looks more or less the same as on the desktop; the user's finger takes over the function of a mouse pointer. Although immediately clear for people transitioning from desktop to mobile, there are some problems associated with this approach. Most of all, the slider eats up valuable screen estate. It needs to be large enough to tolerate most fat-fingering. But any finger covers and hides the actual slider position, in contrast to the desktop where the mouse pointer is designed to never cover anything. Subtle changes of such a slider may still be difficult or impossible; it needs to be long enough such that it offers sufficient fine-grained control. Finally, the auto-rotation feature present in most devices makes this even harder: In essence two UIs have to be developed, one for landscape, and one for portrait orientation of the tablet or phone.

While moving towards an object (for example, with a mouse pointer on a display, or with a finger operating a touchscreen) there is a trade-off between speed and accuracy. Fitts[1] obtained a quantitative expression for this fact. This is known as Fitts's law, which states that the time required to move towards the object increases with the quotient of distance to the object and its width. It is straightforward to re-interpret this in the context of a mouse-driven desktop: Widgets far away from the current position of the mouse pointer must be large in order to minimize the time it takes to move there. This has obvious implications on UI design: frequently used widgets should be large or in the center of the display. Things are slightly different for a touchscreen. Here the distance to the object is the distance between the finger and the touchscreen surface. As there are no statistics about where the finger is located above the touchscreen and at which distance, it makes only sense to optimize the navigation time via the size of the widgets. This argument, together with the ones raised in the previous paragraph lead to the following solution: a vertical single-fingered swipe anywhere on the touchscreen operates controls selecting the filter, and the same horizontal swipe selects the local strength of that filter, both times

Figure 5. Three variants of mobile touch-driven user interfaces. Note that selected control points are circled. (a) UI Variant 1 with UI controls similar to the original desktop UI. The size slider is contained in the combobox on the bottom as well. (b) UI Variant 2 with global control on the bottom of the screen. (c) UI Variant 3 reflecting the UI of the Snapseed App.

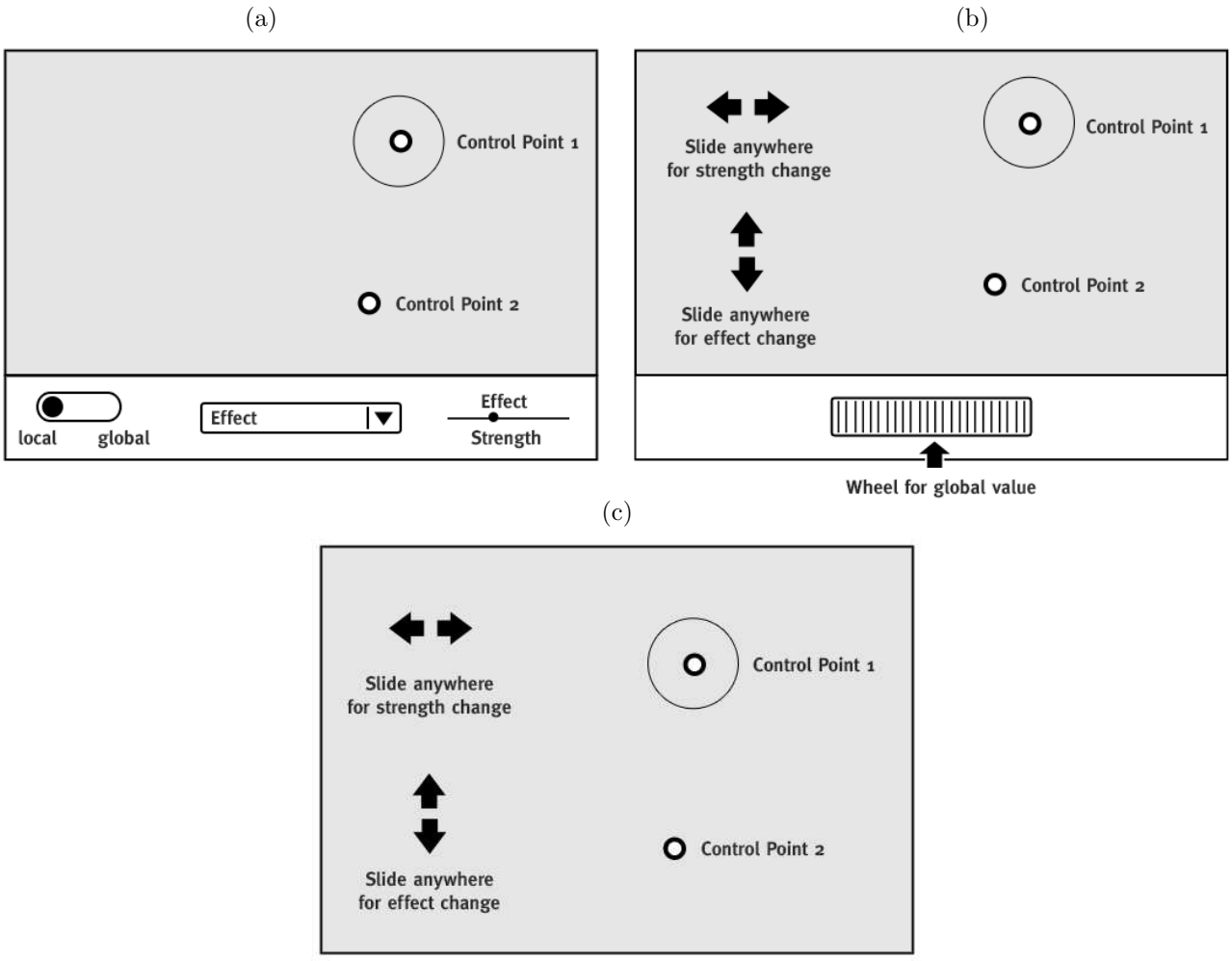


Figure 6. Comparison of the three UI variants shown in Figure 5.

	Variant 1	Variant 2	Variant 3
Control points as dots on image	✓	✓	✓
Select single control point by tapping on dot	✓	✓	✓
Global sliders available	✓	✓	
Vertical slide for parameter change		✓	✓
Horizontal slide for value change		✓	✓
Strength mask shown transiently during size change	✓	✓	✓
Two-finger gesture for size change		✓	✓
Local and global sliders follow consistent UI	✓		

with respect to the currently selected control point. A similar approach of large swipes in orthogonal directions can be found (for example) in the iOS video time line control and in the Photoshop Express App (first released in September 2011).

Since the mobile UI will mostly be operated on-the-go, simplicity, ease of use, and a low number of controls are important. For this reason a number of features from the desktop UI were not carried over to the mobile UI. Such features include multiple selection and grouping of control points, and the color picker feature. Note also that the mobile UI only shows the strength mask of the control point whose size is changed; here it is not possible to show the combined strength mask for two or more control points.

While the idea and concept of selective filtering using control points is powerful, care has to be taken to embed the idea in a user interface that is simple, intuitive, and meets the user's needs and expectations. Several candidate UIs meeting these criteria have been presented and compared. The mobile sector is currently undergoing heavy development; only time will tell which approach will become the commonly accepted gold standard for image processing on touch-driven devices.

ACKNOWLEDGMENTS

We would like to thank our colleagues Björn Beuthien, Daniel Fenner, and Florian Kriener for helpful comments and Jens Denger for illustration design.

REFERENCES

[1] Fitts, P. M., "The information capacity of the human motor system in controlling the amplitude of movement," *Journal of Experimental Psychology* **47**, 381–391 (1954).

[2] Apple Inc., "iOS human interface guidelines," (2012). Version of 19 September 2012.

[3] "Android design guide." http://developer.android.com/design/index.html (2012). Version of 7 November 2012.

[4] Balagtas-Fernandez, F., Forrai, J., and Hussmann, H., "Evaluation of user interface design and input methods for applications on mobile touch screen devices," in [*Human-Computer Interaction – INTERACT 2009*], *Lecture Notes in Computer Science* **5726**, 243–246 (2009).

[5] Nilsson, E. G., "Design patterns for user interface for mobile applications," *Advances in Engineering Software* **40**(12), 1318–1328 (2009).

[6] Shneiderman, B. and Plaisant, C., [*Designing the user interface: Strategies for effective human-computer interaction*], Addison-Wesley, 5th ed. (2009).

[7] Oehl, M., Sutter, C., and Ziefle, M., "Considerations on efficient touch interfaces — how display size influences the performance in an applied pointing task," in [*Human Interface and the Management of Information*], Smith, M. J. and G., S., eds., *Lecture Notes in Computer Science* **4557**, 136–143 (2007).

[8] Wasserman, A. I., "Software engineering issues for mobile application development," in [*Proceedings of the FSE/SDP workshop on future of software engineering research (FoSER)*], 397–400 (2010).

[9] Kokemohr, N., "User definable image reference points," US Patent No. 6,728,421 B2 (Apr. 2004).

[10] Kokemohr, N., "User definable image reference points," European Patent EP1449152 A1 (Aug. 2004).

[11] Vallerio, K. S., Zhong, L., and Jha, N. K., "Energy-efficient graphical user interface design," *IEEE Transactions on Mobile Computing* **5**(7), 846–859 (2006).

[12] Poupyrev, I., Maruyama, S., and Rekimoto, J., "Ambient touch: designing tactile interfaces for handheld devices," in [*Proceedings of the 15th annual ACM symposium on user interface software and technology*], *UIST '02*, 51–60 (2002).