Dynamic windows scaling in Unix-like systems interface

Vladimir Diomin, Dmitriy Kostiuk, Alexander Nikoniuk
Brest State Technical University, 224017, Moskovskaya str., 267, Brest, Belarus,
dmitriykostiuk@bstu.by

Abstract: *The models of a scaled down window and of a window with variable scale are presented, as far as their practical implementation for Unix-like systems as the Compiz window manager extension modules. An implementation specific is provided for the X Window System environment. Changes in the original window are dynamically mapped to the compressed image due to hardware-accelerated OpenGL backend. The concept is especially useful for portable devices with reduced screen resolution as it makes them effectively running software originally designed for desktop computers.*

Keywords: windows scaling, OpenGL, GUI, window manager.

1. INTRODUCTION

Due to limited hardware resources large areas can't be widely used for the information output in contemporary computers and computer-based devices. GUIs, or graphic user interfaces, use special approaches and auxiliary navigation controls to overcome this limitation, making it possible to see whole working space with less details or in schematic manner [1]. The main problem at developing such approaches and controls is the difficulty to combine the backward compatibility with previously developed applications and the strict object-oriented paradigm [2] (which makes software objects to be intuitively analogous to real world objects). Until recent years such auxiliary GUI elements were sketchy and constrained to stay compatible. But currently they are becoming more detailed and visual because of the possibility to use more resources, including higher pixel density of screens and hardware-accelerated graphics.

Many of these approaches are based on using downscaled images to preview and to manipulate real objects. Downscaled images are known as thumbnails and can be often seen in popular applications. Being placed on the periphery of the workspace, thumbnails are copying the natural model of human vision [3] and are among the best ways to improve the intuitivity of an interface.

Efforts to use downscaled images in window management were less successful. The metaphor was several times implemented in experimental graphical shells by placing miniaturized image of a hidden window instead of its icon in a taskbar. These miniatures are called mini-windows and are very useful with dynamic mapping of window content, allowing user to track down substantial changes in hidden applications. Unfortunately main experiments with mini-windows were carried out in times of relatively weak processors and window managers with non-accelerated graphics, when real-time mapping of miniaturized windows was impractical or even impossible to carry out.

Last five years brought hardware acceleration to window management in all widely spread desktop operating systems (OS). There are auxiliary elements of graphics shells that use scaled window images, dynamically mapped with use of graphics processor of a video-adapter. But while the technical implementation is ready, such window miniatures are used in a very limited way and for a short period of time only – as pop-up thumbnails, in window selector or in workspace (virtual desktop) toggle screen. Spoken above GUIs were designed in the face of necessity to hide lack of dynamical mapping and/or its effect on processor load, and therefore window scaling is still used in a small extent there.

Several years after the appearance of hardware-accelerated window managers, netbooks (and some tablet PCs) have appeared in common usage as a new class of portable computers with smaller screen size and resolution, but capable of running standard PC OS and applications. From the other side, mobile hardware and OS designed for smartphones are narrowing the gap with portable PCs concerning the computation power.

In both cases developers have a problem with placing classical application in reduced resolution workspace. Modifying application interface is not always possible and purposeful. Outlined situations make strong demand for window managers providing scaled down windows for constant operation, if done in proper parts of an interface, without harm to its overall readability.

In this article we present two models based on window scaling and their experimental implementation for Unix-like systems. The first model is based on discrete window scaling and is close to mini-windows metaphor. The second one uses nonlinear window scaling and is especially designed for reduced resolutions.

2. DISCRETE WINDOW SCALING MODEL

This model in its general form supposes dividing workspace into several levels, each one with its own scale factor (Fig.1). All parts of a window have same scale, and the scale factor depends on the area, upper left corner of a window (the x_0; y_0 point) belongs to.

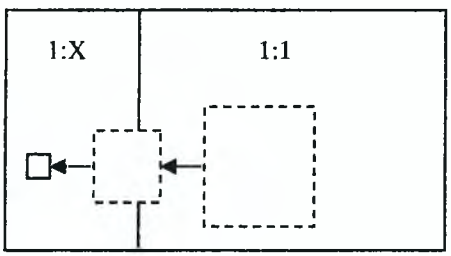

Fig. 1 – Discrete levels of window scaling

So transformed i^{th} pixel coordinates are calculated as

$$\begin{aligned}\tilde{x}_i &= x_i \cdot a_{\pm}(x_0)\cdot[\theta(x_i)-\theta(x_i-x_0)],\\ \tilde{y}_i &= y_i \cdot a_{\pm}(x_0)\cdot[\theta(y_i)-\theta(y_i-y_0)]\end{aligned}, \quad (1)$$

where $\alpha(x)$ is a scale change function, and θ is a Heaviside step function [3, 4]. Within the mini-window approach $\alpha=f[\theta(y_p-y_0)]$, because of only two possible stages: unscaled window outside the dock panel, or window in the panel area downscaled with a ratio depending on the panel lesser dimension (stated as y_p global screen coordinate for horizontal panel).

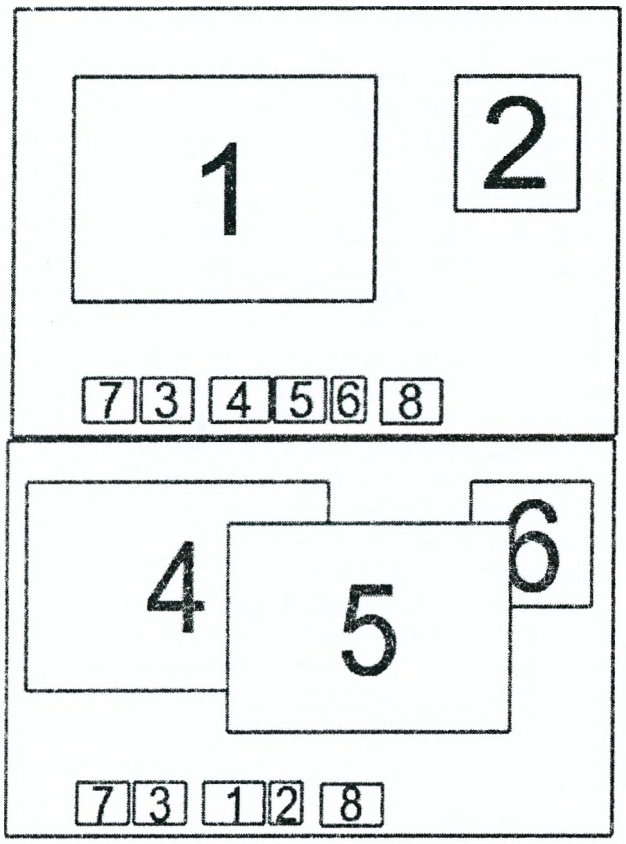

Fig. 2 – Grouped mini-windows

As user decides what windows should be miniaturized into the dock panel, the auxiliary mechanism to remember several schemes of windows placement and to quickly switch between them should substantially improve the usability. To provide such a possibility we have modified the model. The modification is referred as model of grouped mini-windows [4].

Window miniatures are placed along with chosen screen border as in classical mini-windows variant. User can reorder mini-windows by dragging, and same action can be used for grouping, as placing one miniature over another groups them, arranging side-by-side close to each other (Fig. 2).

Grouping mini-windows allows unfolding whole group with one click. E.g. clicking the central group shown in the upper part of a Fig. 2 would return windows No. 4, 5 and 6 to their proper places on the screen (as in lower part of Fig. 2).

A coverflow effect («turning pages» of objects like album pages) is proposed to visualize going through mini-windows groups with Super-Tab keyboard shortcut [4].

Moreover, same approach allows grouped launching of applications. Panel of mini-windows can hold special docklet – a pseudo-group with icon, representing the history of windowed interface operation. Being selected in mini-windows list it would shade screen and paint previously used windows groups over it. At small amount of groups scaled virtual desktops can be painted as in very large pager, but when their amount is to big to place all groups with suitable scale factor, grouped miniatures can be used instead. By choosing one of these groups on the shaded screen, user starts applications of a group and with placing their windows at remembered screen positions. Group applications launching should be especially effective at systems equipped with solid-state drives for their negligible seek times.

3. NONLINEAR WINDOW SCALING MODEL

This model supposes continuous change of a scale within one window. Its ratio smoothly increases towards periphery of the window, while central part stays unscaled.

Coordinates are translated along with following expression, generalized to $k=\overline{1;n}$ dimensions:

$$\begin{aligned}\tilde{P}_i^{(k)} &= E_i^{(k)} + F_i^{(k)} + G_i^{(k)},\\ E_i^{(k)} &= \alpha\left(\frac{P_i^{(k)}}{P_{src}^{(k)}}\right)\cdot\delta\left[\theta(P_i^{(k)})-\theta(P_i^{(k)}-P_{src}^{(k)}-\delta)\right]\\ F_i^{(k)} &= (P_i^{(k)}-P_{src}^{(k)})\cdot\left[-\theta(P_i^{(k)}-P_{src}^{(k)}-\delta)-\right.\\ &\left.-\theta(P_i^{(k)}-P_W^{(k)}+\delta-P_{src}^{(k)})\right],\\ G_i^{(k)} &= \left[P_W^{(k)}-\alpha\left(1-\frac{P_i^{(k)}-P_W^{(k)}-P_{src}^{(k)}+\delta}{P_{Wsrc}^{(k)}-P_W^{(k)}-P_{src}^{(k)}+\delta}\right)\cdot\delta\right]\times\\ &\times\left[\theta(P_i^{(k)}-P_W^{(k)}+\delta-P_{src}^{(k)})-\theta(P_i^{(k)}-P_{Wsrc}^{(k)})\right]\end{aligned} \quad (3)$$

Here peripheral areas are named as E and G, while F is central unscaled area, and δ is the width of scaled area.

Following values are used at transforming window image. Source image has $i=\overline{0;Wsrc-1}$, and converted image has $i=\overline{0;w-1}$ (values and dimensions of source image marked with *src* index). $P_{src}^{(k)}$ – is the shift of a top left corner (point with zero local coordinate) of the destination image relative to the corresponding point of a source image. In two-dimensional coordinates $P_i^{(1)}=x_i, P_i^{(2)}=y_i, k=\overline{1;2}$, and being considered as $\tilde{P}_i^{(k)}=f(P_i^{(k)},\delta,P_W^{(k)},P_{src}^{(k)},P_{Wsrc}^{(k)})$, (3) transforms to (3'):

$$\begin{aligned}\tilde{P}_i &= f\left(P_i^{(k)}=\begin{bmatrix}x_i\\ y_i\end{bmatrix},\delta,P_W^{(k)}=\begin{bmatrix}w\\ h\end{bmatrix},\right.\\ &\left.P_{src}^{(k)}=\begin{bmatrix}x_{src}\\ y_{src}\end{bmatrix},P_{Wsrc}^{(k)}=\begin{bmatrix}w_{src}\\ h_{src}\end{bmatrix}\right),\end{aligned} \quad (3')$$

where w_{src} and h_{src} are width and height of source window, while w and h are same values of a resulting one.

Different scale functions can be considered within the model. Particularly, curve like $\alpha(x) = x^{1/K}$ allow elements near the unscaled are to be distinguished better.

4. IMPLEMENTATION OF THE MODELS

We have developed implementations of both models for GUI of Unix-like systems. The choice was driven by the possibility to use source code of all components, as far as strong modularity and extensibility of graphic environments, based on the X Window System and standard for contemporary Unix and Unix-like OS. An experimental object-oriented branch of the Compiz window manager was used as the basis for the implementation. Compiz in its turn uses OpenGL library for hardware acceleration. One of the features of OpenGL, the framebuffer objects, provides window manager with full access to the inactive window image [5]. Application treats framebuffer objects as ordinary windows, while window manager sees them as textures viable to be used in common multi-texture paint routines (Fig. 3). Compiz window manager itself is a combination of relatively independent modules interacting through API, provided by several core classes. Therefore this architecture is a good ground for building experimental graphic environments [6].

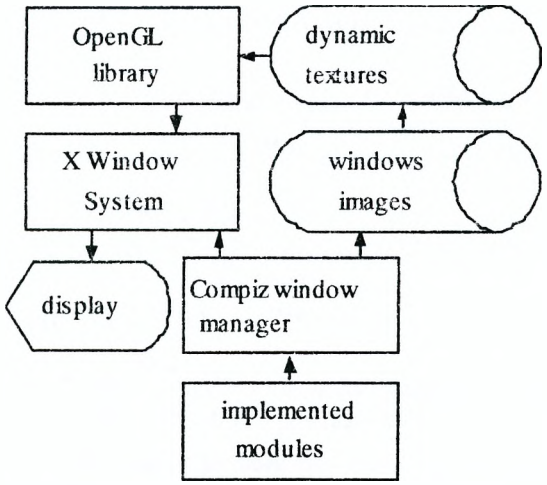

Fig. 3 – Architecture of the implementation

Developed extension modules use three functional parts: initialization code, event handlers and calculating functions.

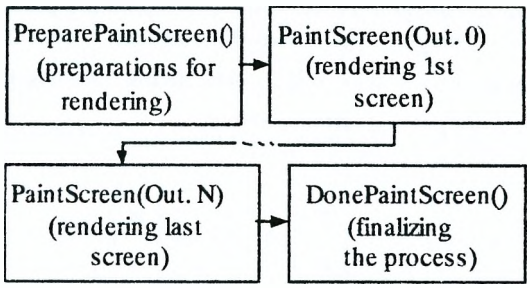

Fig. 4 – Routines of the rendering chain

Initialization code is called at module load. It provides system with pointers to constructors and destructors of display context, all screens of each display and all windows of each screen. Access to display context allows to hook into event handling routine, and screen object makes it possible to participate in the chain of the screen image refreshing routines, which itself includes preparation stage, screen rendering and finalizing stage (Fig. 4).

Preparation stage is used to make different preliminary calculations, e.g. to evaluate new reference points coordinates. Rendering method calls its parent class implementation to refresh screen image, and then it can perform wide range of actions with this image, accessible through the OpenGL library. It has access to the output device, physical coordinates, and all windows of a desktop are presented as correspondent texture images. Finalizing stage may force screen repaint to provide real-time animations.

One of technical problems arising while implementing models as the extensions to a window manager, is caused by specific treatment of the X Window System for windows, been not on the current screen (especially minimized ones). When minimization event occurs, the image pixmap is immediately destroyed, as far as correspondent texture, to be recreated at restoring window from the minimized stage. This architectural feature of the X-server allows to save resources at the cost of windows which are not currently in use, and the effect is clearly seen in all compositing window managers with scaled previews in window selector or pop-up thumbnails, as minimized windows are never previewed there. Therefore we use "fake minimization", where window is replaced by its miniature but not minimized in terms of X Window System. However the difference is not distinguishable from the user's point of view.

When minimization event occurs, old coordinates of the window are stored and updated with new values in boundaries of the dock panel area, and a scale factor is calculated and stored into the window data structure. This value used to identify mini-windows by a screen paint routine.

At screen initialization module hooks its handlers to necessary events:

- PreparePaintScreen, as explained above;
- PaintWindow, that is invoked each time at window repaint to check scale factor and change window transformation matrix if necessary;
- DamageWindowRect, to reduce resource consumption by bounding the fragment to be repainted.

Service routine also tracks mouse click on mini-window to restore it to nontransformed stage.

Window scaling affects the image only. The application itself has no changes in the window size, as backward compatibility would be broken in opposite case. But the same is true for the X-server. Therefore from the system point of view all controls inside the window keep their old coordinates, and mouse pointer should be moved to their previous unscaled position to interact with them. That is unacceptable, as mouse actions out of window bounds would activate its content.

One of possible ways to solve the problem is to use X-server modification with input redirection. There is an experimental branch with this feature, proposed in 2007. Window manager should provide X-server with a mesh of triangular elements, every even triangle describing a

triangle on a transformed window and every odd triangle describing a triangle on the actual window. There were few stability problems with this implementation, and, unfortunately, only one extension module could set an input mesh at a time [7].

Therefore we use simpler approach, in which all mouse events are blocked for miniaturized window content. Instead mouse pointer is used to unminiaturise window, to highlight it and to drag to another position in a dock. The implementation uses standard XShape extension, initially designed for applications that use transparency channel to show non-rectangular windows. It blocks mouse events on external transparent parts of the window. The approach is known as "input shaping" and is used in several standard Compiz modules. While being inaccessible to mouse, mini-windows still can be controlled by keyboard, and if necessary that allows interaction with them without return to the original scale.

Module of nonlinear window scaling has close architecture. It is activated when window is dragged towards any of the screen boundaries (or, more properly, towards the current workspace boundary). If dragging doesn't stops after reaching the boundary, compression of outer part of the window begins, so user can kineticaly control the size of window scaled part and is able to keep balance between minimizing window area and having enough of its readability.

To make off-screen part of a window visible, the whole window is transformed with scale factor close to 1:1. After that additional reference vertices are added to the window substrate figure, and those belonging to an area with variable scale are shifted along with chosen scale function (Fig. 4). The transformation can be carried out along with one window side or two bordering sides.

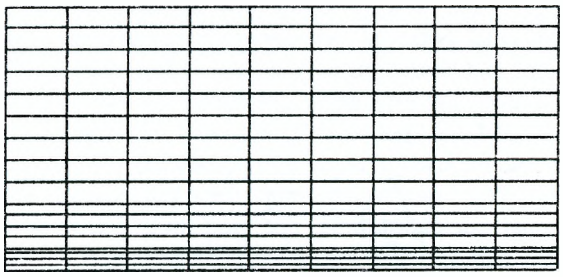

Fig. 4 – Transformed mesh of a window

As in case of mini-windows, input is shaped, but only partially, and main part of the window is still accessible with mouse. That is enough for many typical applications, which have most of mouse-controlled elements, like system menu and button panels, grouped in one part of a window (Fig. 5 shows partially compressed window with menu, activated by a mouse click).

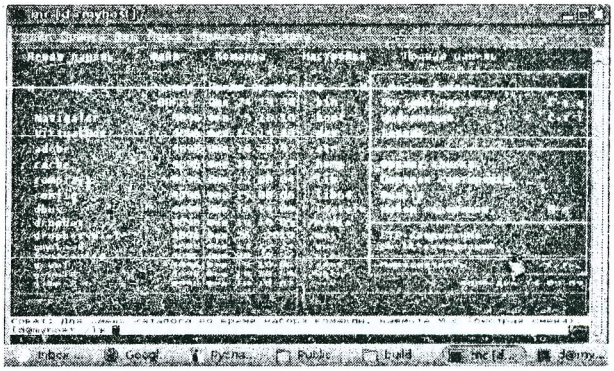

Fig. 5 – Window with variable scale factor

5. CONCLUSION

Presented models of discrete and continuous scaling of the windows and their implementation for Unix-like OS allow to achieve more effective and saving usage of the screen area, especially appreciable for portable devices running unmodified computer software. Due to hardware-accelerated graphics windows images are transparently mapped in real time without high processor loads, and this allows user to observe the changes in large amount of opened windows and to work with several windows at once on relatively small screen.

6. REFERENCES

[1] J. Raskin. *The Humane interface: new directions for designing interactive systems.* ACM Press Series, 2000. 233 p.

[2] D.A. Kostiuk, V.V. Diomin. Model of mini-windows dynamically mapped in hardware-accelerated graphic interface, *Vestnik BrGTU* 5 (2009). P. 71-74.

[3] H.V. Gomanova, D.A. Kostiuk, K.L. Kostiuk. Applying peripheral vision model to hardware-accelerated graphical user interface, *Vestnik BrGTU* 5 (2007). P. 33-35.

[4] V. Diomin, D. Kostiuk. Grouped windows focus switching with variable scale factor. *Proceedings of the IV International Academic Conference of Young Scientists "Computer Science & Engineering 2010 (CSE-2010)"*, Lviv, Ukraine 25-27 November 2010, P. 32-33.

[5] O. Chapuis, N. Roussel. Metisse is not a 3D desktop. *Proceedings of the 18th annual ACM symposium on user interface software and technology.* NY, USA, P. 13-22.

[6] E.V. Gomanova, I.N. Borushko, D.A. Kostiuk. Peripheral vision model for graphical user interface. *Contemporary information computer technologies: book of scientific articles [in Russian]*, Grodno, 2006. P. 22-27.

[7] S. Spilsbury, *Input Redirection Update.* http://smspillaz.wordpress.com/2008/10/21/input-redirection-update/