Importance-Driven Compositing Window Management

Manuela Waldner[1], **Markus Steinberger**[1], **Raphael Grasset**[2], **and Dieter Schmalstieg**[1]

[1]Institute for Computer Graphics and Vision
Graz University of Technology
Graz, Austria
{ waldner | steinberger | schmalstieg }@icg.tugraz.at

[2]HIT Lab NZ
University of Canterbury
Christchurch, New Zealand
raphael.grasset@canterbury.ac.nz

ABSTRACT
In this paper we present importance-driven compositing window management, which considers windows not only as basic rectangular shapes but also integrates the importance of the windows' content using a bottom-up visual attention model. Based on this information, importance-driven compositing optimizes the spatial window layout for maximum visibility and interactivity of occluded content in combination with see-through windows. We employ this technique for emerging window manager functions to minimize information overlap caused by popping up windows or floating toolbars and to improve the access to occluded window content. An initial user study indicates that our technique provides a more effective and satisfactory access to occluded information than the well-adopted Alt+Tab window switching technique and see-through windows without optimized spatial layout.

Author Keywords
compositing window management, visual saliency, space management, transparency

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: User Interfaces—*Windowing systems*

General Terms
Design, Human Factors.

INTRODUCTION
Although more than 25 years have passed since the emergence of the WIMP metaphor, the window concept still prevails as the unique interface to control multiple applications on the screen. Today, the need of information workers for multiple applications to successfully complete a single task and the common usage of floating menus or multiple windows per application, lead to a standard desktop being cluttered with a large number of open application windows. Even with increasing display space, users just tend to keep more application windows open [13].

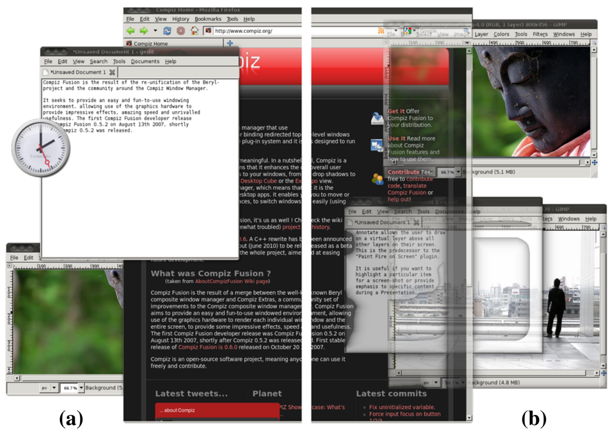


Figure 1. (a) Conventional overlapping windows occlude important information in obscured windows. To reveal this information, the user has to change the window layout manually. (b) Importance-driven compositing spatially arranges occluded windows and applies see-through compositing to maximize content visibility and interactivity.

Most current window managers employ multiple overlapping windows and leave the spatial arrangement of the individual windows merely to the user. As a result, important information may be hidden in occluded windows (Figure 1a) and users spend a significant amount of time switching between windows to reveal obscured content [13].

Different window management techniques to increase the amount of simultaneously visible information have been explored. Automatic window layout techniques (*e.g.*, [18, 3]) reduce the amount of window overlaps and, as a result, the empty screen space. A limitation of these systems is the lack of knowledge about the application content. Therefore, windows are treated as "black boxes", which does not leave much room for *overlap-avoidance* [3] if the screen is cluttered with a large number of windows. On the other hand, transparency can be employed to reveal the content of occluded windows. However, transparency alone does not ensure that important hidden content is actually revealed. If highly salient window content overlaps, visual differentiation between window layers becomes difficult for the user. These situations can be resolved with an optimized spatial layout which minimizes information overlap.

In this paper, we therefore propose importance-driven compositing – a new window management approach for see-through user interfaces. We base our work on recently established compositing window managers, which have access to the window textures after the window content has been rendered into video memory. Importance-driven compositing window management analyzes these window textures to identify perceptually important window regions to optimize the spatial window layout for see-through compositing (Figure 1b). To identify the importance of a window and its content, we rely on a visual attention model describing how much a location in an image stands out from its surroundings [17]. In summary, our contributions are:

- Using visual saliency as an importance measure of application window content and the concept of *importance maps* as unified, image-based importance representation for compositing window managers,
- a window management approach combining spatial window layout optimization with see-through compositing for maximum information visibility and interactivity,
- a hardware-accelerated implementation for real-time usage in an established windowing system, and
- a validation of our approach through an initial user study indicating a benefit of our technique for accessing occluded content compared to traditional Alt+Tab window switching and *free-space transparency* [16].

We will present novel window management functions based on importance-driven compositing to improve the access to occluded information by a new window switching approach and a semi-automatic window layout technique. We also introduce functions to decrease occlusion of window content by pop-up windows or floating menus.

RELATED WORK

With the steadily increasing number of open, overlapping application windows, users are required to apply some sort of *occlusion management* technique [8] to discover and access occluded window content. To bring occluded windows to the front, window managers provide window lists, sequential window switching techniques (such as Alt+Tab or *stack leaving* [9]), or space-filling window layouts (such as Apple's Exposé[1]). However, explicit window switching has been described as tedious [10]. Subsequently, we discuss two window management approaches alleviating the need to switch windows explicitly by increasing the amount of simultaneously visible information on the screen: see-through windows and automatic spatial window layouts.

See-Through Windows

See-through user interfaces were first introduced by Bier *et al.* [5] as movable filters operating on the underlying graphics object. Today, advances in hardware-accelerated window management (such as *Metisse* [6]) have made enhanced window rendering techniques, such as transparencies, geometric deformations, or change visualizations [4], widely available. Most up-to-date window managers support half-transparent rendering of application windows to show information hidden underneath. However, experiments have shown that simple alpha blending either causes readability problems with foreground items or decreases the perception of the background image – depending on the chosen alpha level [12, 2]. Outlines of text and icons [12] and *multiblending* [2] have been shown to improve recognizability of blended content. *Dynamic transparencies* [11] help to increase background visibility and were demonstrated to reduce targeting performance only marginally. For see-through window management, Ishak and Feiner [16] apply *free-space transparency* only to "unimportant" (white) window regions. This guarantees that important content in top-level windows is preserved. However, it does not guarantee that important content in occluded windows is actually revealed. If important content in the occluded windows is located underneath the important regions of the overlay window, the user has to manually re-arrange windows to access hidden content. This limitation is the main motivation for our approach: we use window importance not only for see-through compositing, but also for an optimized spatial window layout which minimizes overlaps of important regions.

Spatial Window Layout

Window layout techniques like overlap-avoiding dragging [3] or constraint-enabled window management [1] spatially arrange windows to avoid window overlaps. For both techniques, a sufficiently large screen is required to accommodate for all the non-overlapping windows in their original size. Other techniques only temporarily change the window layout for certain situations, for instance if a physical occlusion of user interface elements has been detected [27] or if the user wants to copy-and-paste between overlapping windows [7]. All of these examples treat windows as simple rectangular shapes – irrespective of their content. For our window layout routine, we were inspired by the field of augmented reality, where a common task is to annotate real-world objects in video images with virtual labels. To find the optimal placement for the label, the video images are analyzed (*e.g.*, [19, 24]). We extended this approach for compositing window management where we treat windows as image templates with the objective of finding the window layout with the least information overlap.

To deal with limited screen space, window layout techniques often apply some sort of deformation to the windows. For instance, tiled window managers (*e.g.*, [18]) have to resize windows to keep all windows visible without an overlap. Automatic down-sizing introduces the risk of clipping important content at the window's boundary. Therefore, others crop windows to preserve only the most relevant regions of context windows (*e.g.*, [15, 20]). However, in these systems the user has to define the relevant regions manually, which can be a tedious task. In contrast, importance-driven compositing automatically identifies perceptually important window regions, by image-based analysis of window textures.

Task management systems (*e.g.*, [23, 22]) apply perspective transformations or down-scaling to context windows, while the focus windows remain in the center of the screen. How-

[1] `http://www.apple.com/macosx/what-is-macosx/expose.html`

ever, content can be too small or skewed to allow for details to be perceived. Furthermore, interaction with transformed windows is not possible in these examples. Thus, users have to perform explicit window switches to access and manipulate information in transformed windows. A main distinguishing aspect of our work is that we use unimportant screen space – potentially embedded within windows – to show important elements of occluded windows in original size. Content is neither distorted nor shrunk and fully interactive, if visible. This allows the user to look up and interact with the fine-grained content in occluded windows without explicitly bringing them to the front.

IMPORTANCE-DRIVEN COMPOSITING

Importance-driven compositing analyses the importance of window content to find an optimal spatial window layout – in terms of visibility and interactivity of occluded content – for see-through window interfaces. Our approach is composed of four steps:

1. The creation of desktop and window *importance maps* containing image-based descriptions of important regions,
2. a *window layout* routine, placing occluded windows to minimize the overlap between important regions,
3. importance-driven see-through *compositing*, applying per-pixel transparencies to reveal important content of occluded windows, and
4. *interaction* techniques allowing the user to access and manipulate content in occluded windows.

In the following, we will discuss these steps in more detail.

Importance Map

We introduce importance maps as unified image-based representation of importance in windowing systems: *window importance maps* are extracted from each window's content using a model of saliency-based visual attention. In our system, the window's content is given as an image – more specifically as a *texture*, as we are relying on a hardware-accelerated window rendering (similar as described in [6]).

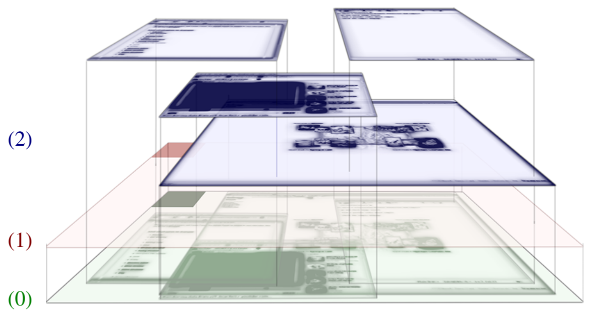


Figure 2. Window importance maps (2, blue) are created for each window individually and accumulated with high-level information (1, red) to a common desktop importance map (0, green). Dark areas represent high importance.

Window importance maps are accumulated into a common *desktop importance map*, which can be extended by high-level information provided by the user or derived from the physical screen setup. Windows are added to the desktop importance map from front to back – ensuring that recently used windows are prioritized for layout and compositing. Figure 2 shows individual window importance maps, high-level information, and the merged desktop importance map. The resulting desktop composition is depicted in Figure 3(8).

Visual Attention

Saliency is a measure of how much a location visually stands out from the surrounding image regions [17, 21]. Typically used bottom-up saliency features are regions of high changes in luminance, color opposition, orientation changes, and motion. For our window importance maps, we apply the conspicuity analysis proposed by Mendez *et al.* [21], which extracts pixel-wise saliency values based on an analysis of lightness, red-green and blue-yellow color opposition, to the (dynamic) textures of the windows. In addition, we measure visual changes in occluded windows and temporarily increase importance in regions where content has changed. Visual changes caused by user interaction, such as scrolling the window content, are ignored. The importance maps in Figure 2 show that user interface elements and information content, such as text or images, are assigned high importance values. In particular, the video in the center window of Figure 2 is highly salient due to additional motion. Homogeneous background regions have low importance – independent of their background color.

High-Level Importance

In addition to the accumulated window importance maps describing importance based on visual perception, the desktop importance map can also contain high-level information. For instance, a user might want to keep desktop regions uncovered from application windows to have access to frequently used desktop icons [14] or to avoid windows spanning across physical monitor bezels on multi-monitor settings [10]. This importance information can be accumulated into the desktop importance map – for instance by automatically adding a one-pixel line of high importance along monitor bezels. In the example map of Figure 2, the user manually selected the upper left corner of the screen as high-importance desktop region, where window placement should be avoided.

Window Layout

The aim of the window layout routine is to spatially arrange windows so that important regions of occluded windows are moved to desktop areas of little importance. Therefore we consider both, each window's individual importance map (Figure 3(2)) and the desktop importance map describing the current distribution of information over the desktop (Figure 3(3)), to find the optimal window placement. Figure 3 shows the layout step for a single window and the resulting desktop composition. The optimal window placement is determined by considering three influencing factors described in the following (*c.f.*, Figure 3(4-6)).

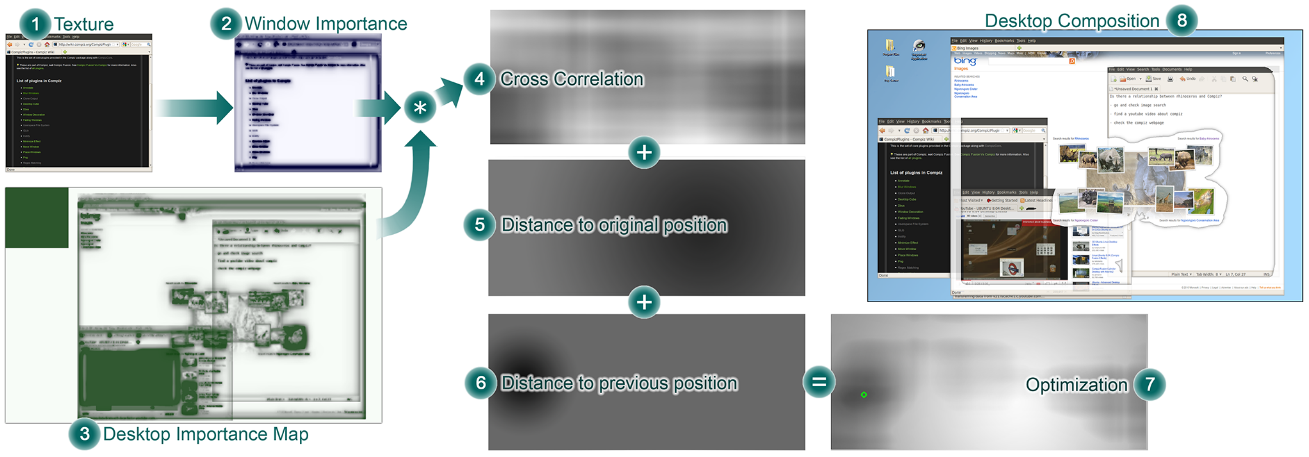


Figure 3. Overview of importance-based compositing for a single window: (1) from the window's texture, (2) the window importance map is created. (3) Given the desktop importance map, the window layout routine finds the optimal placement considering (4) the information overlap, (5) the distance to the original window location, and (6) the distance to the previous location. (7) The combination of these factors gives an optimization problem with the solution marked as a green dot (window center). (8) Finally, the window is rendered to the existing desktop content.

Firstly, the overlap of information in the window (I_w) and the already existing information on the desktop (I_d) shall be kept low. The information overlap can be formulated as a cross-correlation of the two importance maps I_w and I_d:

$$J_i(\vec{p}) = \sum_{(x,y)\in\Omega_w} I_w(x,y) \cdot I_d\left(\vec{p} + (x,y)\right),$$

where (x,y) visit all discrete positions in the window texture Ω_w, and $\vec{p}$ is the window location. Secondly, the location $\vec{p}$ should have little displacement from the original window location $\vec{p}_o$, which is defined by the location where the window has been mapped or manually positioned by the user. Thus, this term is responsible for maintaining a certain degree of spatial stability. Thirdly, the resulting window location $\vec{p}$ should vary minimally from the location in the previous frame $\vec{p}_p$. In other words, jitter should be minimized.

These requirements can be formulated as an optimization problem over all possible window locations $\vec{p}$ (Figure 3(7)):

$$J(\vec{p}) = \omega_i J_i(\vec{p}) + \omega_d D(\vec{p}_o, \vec{p}) + \omega_j D(\vec{p}_p, \vec{p}),$$

where J is the associated cost function to be minimized, composed by a weighted sum of the information overlap J_i, and the distance D to the original location (window displacement) and previous position (jitter). The individual weights (ω) vary for the presented window manager functions described further below in the paper.

To reduce the number of potential window locations ($\vec{p}$), the search space can be decreased by additional constraints. For instance, windows can be bound to certain screen regions or to "parent" windows. In addition, we limit the maximal window movement over time. This introduces smooth frame-to-frame animations and helps the user keep track of window movements. Window content is never placed outside the screen boundaries. This decreases the search space for large windows, while maximized windows will not be re-positioned at all.

Our algorithm treats multiple windows sequentially in a greedy manner. For each window, the best placement is determined by solving the optimization problem as stated above. Subsequently, the window's importance map is added to the desktop importance map at the determined location. The modified desktop importance map then serves as an input for the next window's placement. As the windows being traversed first have more freedom in finding a good placement, more recently used windows are implicitly prioritized.

Compositing

Importance-driven see-through compositing reveals occluded content by applying pixel-wise transparencies. We implemented two well-known compositing techniques from the field of technical illustrations and volume rendering [26]: *ghosting* and *cut-aways*.

Ghosting determines each window pixel's alpha value $\alpha_w(x,y)$ by evaluating the importance ratio of the window's importance map $I_w(x,y)$ and the desktop importance map $I_d(x,y)$ at the respective pixel:

$$\alpha_w(x,y) = \frac{I_w(x,y)}{I_w(x,y) + I_d(x,y)}.$$

Thereby, it ensures that the most important features of each window are visually preserved (Figure 4a). However, if the window layout routine cannot spatially separate important regions (for instance due to high information density), important features in overlapping windows compete for visual prominence.

Cut-aways put more emphasis on the windows' stacking order: they ensure that the most prominent features of the overlay windows are preserved. Only if the desktop's importance map value is below a certain threshold, the obscured window's content is revealed. Smooth blurring and desaturation of obscured window content provide subtle depth cues. This approach is similar to free-space transparency [16], where

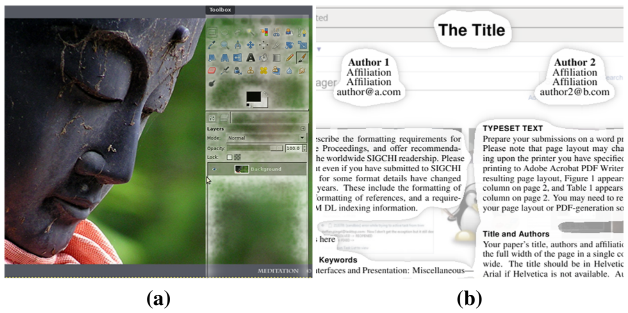


Figure 4. See-through compositing: (a) Ghosting and (b) cut-aways.

transparency is only applied to white regions in overlay windows. Our cut-away technique differs from free-space transparency, as we apply hard boundaries between the foreground and the background and add shadows to visually indicate depth layers (Figure 4b).

Interaction

Importance-driven compositing window management allows users to interact with visible portions of occluded windows – even if located within the boundaries of an overlay window. We rely on the simple assumption that the user aims to interact with the visually most prominent window at the current mouse pointer location. We therefore set the input focus according to the compositing result: the window with the overall highest contribution to the pixel's color below the mouse pointer is activated.

WINDOW MANAGEMENT FUNCTIONS

Based on the concept of importance-driven compositing, we subsequently present novel window management functions for overlapping windows. Clearly, applying see-through window compositing to all application windows at all times interferes with focused attention [12] – the ability to focus entirely on a single item without visual interference from other items. Therefore, our suggested functions apply importance-driven compositing either temporarily or solely to a subset of windows.

Uncovering Windows

Hutchings *et al.* [13] found that window switching is a frequent activity and that most windows are activated for less than four seconds. This indicates that users often only need to skim the content of a secondary window to resume their primary task, such as reading a bit of documentation, checking if a new e-mail has arrived, or retrieving the result of a calculation. With sequential window switching techniques, the user has to perform multiple operations: initiate the window switch, identify and select the window of interest, perform the actual operation, and repeat the window switching step to return to the previous window. Not surprisingly, "the need to Alt-Tab" has been described as tedious [10].

To reduce the number of necessary activities to quickly access occluded content, we use importance-driven compositing to uncover information in occluded windows on demand. By pressing a keyboard combination, spatial window layout

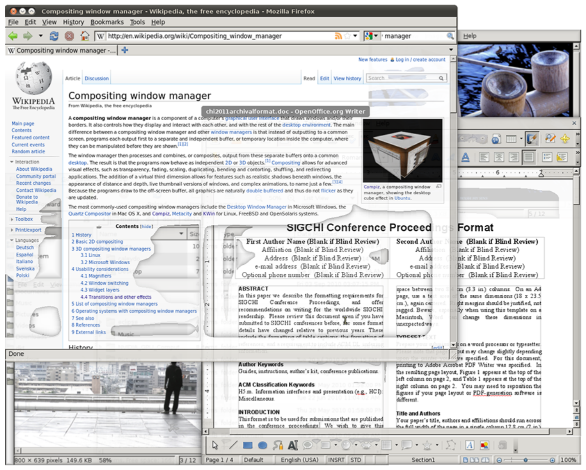


Figure 5. Unimportant window regions are temporarily cut away and occluded windows are spatially arranged to reveal obscured content. The user directly interacts with the otherwise invisible document content uncovered below the browser window.

and cut-away compositing is applied to all occluded (unminimized) desktop windows (*c.f.*, Figure 5). Top-level windows are not re-positioned. Penalties on window displacement for the layout algorithm are kept low to give more emphasis on minimizing the information overlap than a stabilized window layout. We discard the window title bars from occluded windows, as they have a high visual saliency but little benefit for user interaction in this situation. As input is redirected to the most salient window at the cursor location, simple operations in occluded windows – such as pressing a button – can be accomplished while exposing occluded content without actually bringing the window to the front. To signal which window currently holds the input focus, we increase the active window's overall alpha value and show its title at the left upper window corner (as for the document editor in Figure 5). When releasing the key combination, the active window under the pointer is brought to the front and the other occluded windows return to their original locations.

Semi-Automatic Window Layout

When working with large-scale monitors, users rarely maximize windows. Instead, they often "carefully coordinate" their windows and keep a small portion of occluded windows visible for direct access [14]. This window arrangement alleviates the need for explicit window switching. However, manually arranging the windows is rather time-consuming and keeping important elements visible for easy window identification requires frequent re-adjustment as other windows are moved.

We use importance-driven compositing as a semi-automatic window layout technique which is initiated when the user starts dragging a window. All windows – except for the window currently being dragged – become subject to semi-automatic window layout. As the user drags a window, underlying windows are re-positioned to avoid information overlap (Figure 6). In addition, ghosting allows the user to see the content underneath while keeping the title bar

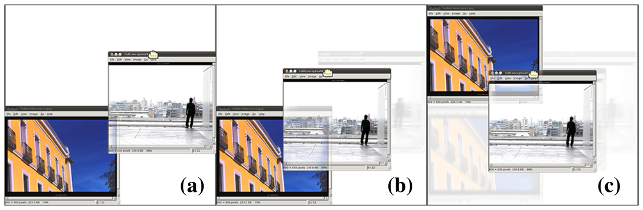


Figure 6. Semi-automatic window layout: (a-b) the user drags a window towards the left, which causes an overlap with another window. The window layout persists as long as the amount of covered information is low. (c) If more information is occluded, the obscured window is re-positioned to leave the most important window content uncovered. Window trails were added for illustration purposes.

pressed. As the user releases the mouse button, the applied layout persists and the windows become fully opaque again.

In contrast to the previously described window uncovering function, we treat the window being dragged as a black box (*i.e.*, the window importance map is uniformly high) as the aim is to optimize the layout for opaque windows. However, we do consider the importance of content in the windows underneath. In case the dragged window is moved on top of other windows, the layout function re-positions these occluded windows so that only unimportant regions are covered. Penalties for window displacement and jitter are high to keep the spatial window layout as persistent as possible.

Users can furthermore influence the resulting layout by providing high-level desktop importance information. They can define desktop regions where window placement should be avoided or prioritize certain regions.

Pop-Up Window Placement

Windows popping up without the user triggering it usually signals events which require the user's attention. Examples are dialogs indicating a new e-mail, instant messages, or a completed file operation. Immediate user activity is not always necessary. Yet, dialog windows popping up on top of the window stack – sometimes even obscuring the main interaction item – necessitate the user to interrupt the current task and undertake immediate action.

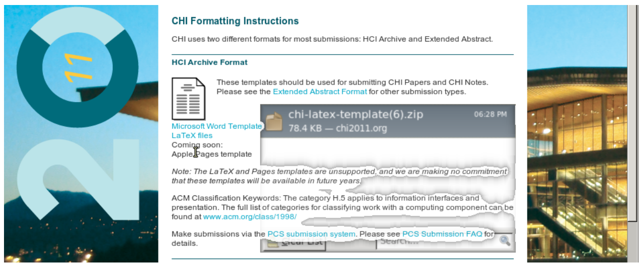
To minimize user interruption, we apply importance-driven compositing to pop-up dialogs. Newly created windows from the class "dialog" are automatically positioned to avoid an overlap with the content of other windows. Cut-aways are employed to ensure the readability of foreground information, as depicted in Figure 7. Users can directly interact with dialog windows wherever visibility is given. To bring the dialog window to the front as a conventional opaque window, the user employs a simple shortcut. If the user does not interact with the dialog within 10 seconds after creation, the window moves to the back of the window stack as a conventional occluded window.

Intelligent Floating Toolbars

Floating toolbars or tear-off menus are commonplace in many applications, such as word processors or image editing software. Instead of docking a menu to the window's boundary, the user can detach it and re-position it anywhere on the screen. Floating toolbars usually stay on top of the main application window to enable quick access to important user interface elements. Thereby, toolbars can lead to an occlusion of the main window content.

We allow the user to interactively pick floating toolbars for importance-driven compositing using a shortcut or a context menu entry. Once selected, the toolbar window is re-stacked behind the main application window and re-positioned to avoid an overlap with the main content. Ghosting maintains the most important content of both, toolbar and main window. We constrain the layout algorithm to apply only minimal displacement from the menu's previous and original location. As a result, the toolbar will minimally adjust its position as the user manipulates the main window's content, offering a trade-off between spatial stability and avoiding occlusion of important content. Consider, for instance, the floating menu shown in Figure 8: as the user scrolls the content, the menu is set to empty document spots but is never moved too far from its original location.

Users can bind a toolbar to a main window by using a context menu entry or by dragging it within the boundaries of the parent window and not moving it for a second. Automatic toolbar placement is then restricted to the interior and the immediate surroundings of the parent window. If the main window is moved, the toolbar is re-placed accordingly. To detach the toolbar from its parent, the user drags it outside the parent window's boundaries. Visual feedback of the binding/unbinding operation is provided by animated icons.


CHI Formatting Instructions

CHI uses two different formats for most submissions: HCI Archive and Extended Abstract.

HCI Archive Format

These templates should be used for submitting CHI Papers and CHI Notes. Please see the Extended Abstract Format for other submission types.

Microsoft Word Template
LaTeX files
Coming soon:
Apple Pages template

Note: The LaTeX and Pages templates are unsupported, and we are making no commitment that these templates will be available in future years.

ACM Classification Keywords: The category H.5 applies to information interfaces and presentation. The full list of categories for classifying work with a computing component can be found at www.acm.org/class/1998/

Make submissions via the PCS submission system. Please see PCS Submission FAQ for details.

chi-latex-template(6).zip 06:28 PM
78.4 KB — chi2011.org

Figure 7. Importance-driven compositing takes care that the pop-up dialog does not occlude important website content while keeping the most important elements of the dialog visible.

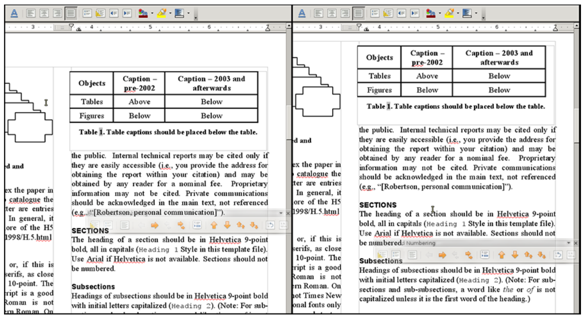

Objects	Caption – pre-2002	Caption – 2003 and afterwards
Tables	Above	Below
Figures	Below	Below

Table 1. Table captions should be placed below the table.

the public. Internal technical reports may be cited only if they are easily accessible (i.e., you provide the address for obtaining the report within your citation) and may be obtained by any reader for a nominal fee. Proprietary information may not be cited. Private communications should be acknowledged in the main text, not referenced (e.g., "[Robertson, personal communication]").

SECTIONS
The heading of a section should be in Helvetica 9-point bold, all in capitals (Heading 1 Style in this template file). Use Arial if Helvetica is not available. Sections should not be numbered.

Subsections
Headings of subsections should be in Helvetica 9-point bold with initial letters capitalized (Heading 2). (Note: For sub-sections and sub-subsections, a word like *the* or *of* is not capitalized unless it is the first word of the heading.)

Figure 8. A floating menu is re-positioned automatically while scrolling the document to avoid overlap with textual content.

Our system supports independent application windows to reach similar functionality as floating application toolbars – similar to user interface "holes" to access auxiliary applications [25]. For instance, the user can attach a calculator to a programming environment or a sticky note to a document editor and access them like conventional user interface elements of the main application.

IMPLEMENTATION

Tremendous processing capabilities are nowadays available in form of graphics processing units (GPUs) in virtually every PC system. Yet, they remain mostly unused for common window operations. In our system, we formulate computationally expensive tasks (such as the saliency computation or the window layout routine) using advanced GPU languages - namely, the OpenGL Shading Language[2] (GLSL) and the Open Computing Language[3] (OpenCL) - to support real-time interaction. For our prototype, we extended the OpenGL-based compositing window manager Compiz[4] for the X Window System of Linux. On a Quad-Core 2.80 Ghz CPU and NVIDIA GeForce GTX 480, for a desktop resolution of 1280x1024, placing and rendering a window of approximately 500x300 pixels requires 6ms. For a conventional office scenario with five managed windows, we obtain an average frame rate of 20 fps.

Window importance maps are extracted from the individual windows' textures by using a GLSL implementation of Itti's visual attention model [17], as proposed by Mendez *et al.* [21]. Visual window changes are monitored through window *damage* events, *i.e.*, notifications of window region updates provided by the X Window System. These damage regions are merged into the window's importance map.

For the computationally expensive evaluation of the window layout cost function, we execute a two level parallel search in OpenCL. The best location found on a lower resolution is refined locally to give the new window location.

The final window compositing step is realized as a GLSL fragment shader, which is applied to the translated windows. The compositing shader determines per-pixel alpha values and evaluates each pixel's neighborhood for blurring and shadowing, according to the chosen compositing technique. The shader is also responsible for merging the window's importance map with the desktop importance map and to determine the most salient window at the respective pixel. This information is queried each time the mouse is moved, to find the active window at the current mouse pointer location. To redirect the mouse pointer input to an obscured window, we temporarily raise the window in the window manager's stacking order (refer to Stürzlinger *et al.* [25] for technical solutions and limitations for input redirection in the X Window System). Although this modifies the window manager's stacking order, we do not alter our traversal order for layout and compositing to keep the desktop visually consistent.

[2]`http://www.opengl.org/documentation/glsl/`

[3]`http://www.khronos.org/opencl/`

[4]`http://www.compiz.org`

EXPLORATORY USER STUDY

We conducted a preliminary user study to judge the usability of importance-driven compositing for accessing content in occluded windows. For that purpose, we compared it with two other window management techniques for three different tasks. The tasks were designed to simulate real information work situations, where users have to skim through information in occluded windows or quickly interact with obscured content before resuming the main task.

We hypothesized that importance-driven compositing will be advantageous compared to sequential window switching for quickly looking at information in occluded windows, as the number of necessary steps to reveal occluded content is reduced. Furthermore, we expected a performance benefit of importance-driven compositing for simple interaction tasks in occluded windows, as we provide the facility to directly interact with user interface elements in occluded windows.

We compared the following three window management techniques for revealing occluded window content:
Alt+tab (AT) in combination with conventional overlapping windows is a standard window switching technique provided by all major operating systems and served as a control condition. The employed Alt+Tab switcher by the Compiz window manager shows small previews of all windows when activated by the Alt+Tab sequence (Figure 9a), and allows for sequential window selection by pressing the tab key.
Free-space transparency (FST), proposed by Ishak and Feiner [16], applies transparency to unimportant window regions of overlay windows using a smooth gradient between transparent and opaque regions. We simulated FST using importance-driven compositing without the layout routine (Figure 9b). Instead of the original notion of unimportant window content (white pixels), we used our importance maps to define transparency values. FST does not allow users to directly interact with the occluded content. Therefore, we provided the "pie menu" proposed by the authors, which shows a circular menu of all the windows lying underneath the current pointer location, to bring occluded windows to the front. Participants had to press Start+tab to retrieve this menu and then click on a window preview to bring the desired window to the front.
Importance-driven compositing (IC) was employed to uncover occluded windows on demand, as described further above. To initiate IC, participants had to press the key combination Start+w. As long as the keys were pressed, occluded windows were spatially arranged, cut-aways were applied and interactivity for the most salient window underneath the pointer was ensured (Figure 9c).

Participants were asked to solve a visual search task. Questions were presented in a textual format and the users had to identify specific items in occluded windows. They were presented with a maximized main window and five small object windows behind the main window, which were arranged in a cascaded fashion. The object windows always contained an image of a 2D geometric primitive and – depending on the task type – a small textual label and a button. The main window contained the question the users had to answer and

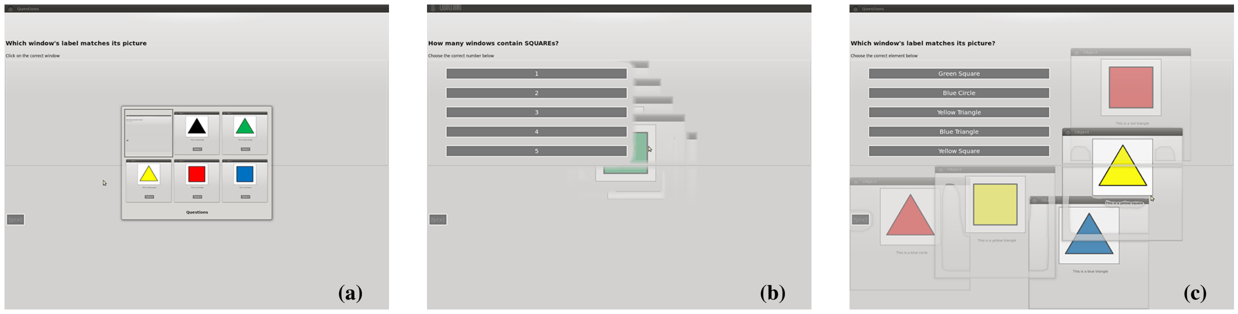

Which window's label matches its picture
Click on the correct window
Questions
(a)
How many windows contain SQUAREs?
Choose the correct number below
1
2
3
4
5
(b)
Which window's label matches its picture?
Choose the correct element below
Green Square
Blue Circle
Yellow Triangle
Blue Triangle
Yellow Square
(c)

Figure 9. The three window management techniques and the three task types in our experiment: (a) overlapping windows with the Alt+Tab menu for the interact task, (b) free-space transparency with the count task, and (c) importance-driven compositing with the read task.

a list of solutions (in two task types). The following three task types had to be solved:

Count: Participants were asked to search for a certain 2D geometric primitive (*e.g.*, square) in the five object windows, count its occurrences, and select the number of occurrences from the list of solutions in the main window (*c.f.*, Figure 9b). This represents a scenario where users have to get an overview of all the windows and identify them based on a strong visual feature, which is also clearly visible in a scaled window representation.
Read: Participants had to find the only object window which contained a textual label (*e.g.*, *"This is a yellow triangle"*) that matched the associated picture of a geometric primitive (*c.f.*, Figure 9c). Subsequently, they had to select the same label from the list in the main window. This represents a scenario where the user has to switch to a window containing textual information required for the main task. Note that the text labels were too small to be readable in the FST pie menu and AT preview menu.
Interact: The task was similar to the read task, except that the validation of the matching label was selected by a push button directly in the object window (*c.f.*, Figure 9a). The task was designed to represent situations where the user has to shortly interact with an occluded window before resuming the main task.

We recruited 15 participants (4 female, aged 15 to 32) from a local high school and university. All participants were experienced computer users and tested for color-blindness. Nine participants are using Microsoft Windows as a primary operating system, three employ Linux and three use Mac OS X. Twelve of the participants use Alt+Tab "often" to "very often" to switch windows, followed by the window list in the task bar, which is employed frequently by eleven participants. None of the participants stated using one window switching technique exclusively.

The study was conducted on a PC running Linux Ubuntu 10.04 with a 1280x1024 17" monitor. We measured the completion times (time between appearance of a question and selecting an answer) and error rates for each task item. Participants were also handed out a preference questionnaire at the end of the experiment. A semi-structured interview was conducted to collect subjective feedback. For each technique, participants had a short practice session (6 questions).

The order of the techniques and tasks was counter-balanced. Each user had to complete four repetitions for each task in every technique. For the interact task and read task, the stacking position of the window containing the correct label was balanced across the repetitions.

Results

We conducted a 3 (Technique: AT, FST, IC) x 3 (Task: Count, Read, Interact) repeated measures ANOVA ($\alpha = .05$) to evaluate completion times. Bonferroni adjustments were applied for post-hoc comparisons. We found a main effect for Technique ($F_{2,28} = 82.231, p < .001$) and Task ($F_{2,28} = 18.182, p < .001$), as well as a borderline significant interaction between the two factors ($F_{4,56} = 2.601, p = .046$). Post-hoc comparisons showed that IC (6.2s) was significantly faster than both, FST and AT (13.9s and 8.4s). AT was also faster than FST. However, IC was only faster than AT for the read and interact tasks. For the count task, there is no significant difference between IC and AT, but both techniques were performing better than FST. Figure 10 illustrates the completion time results.

Participants generally committed few errors with 97% of the questions being answered correctly. The highest error rates were collected for FST in the count and read tasks (10.0% and 6.7%, respectively).

Participants were asked to rank the three techniques on a seven-point Likert scale. A Friedman non-parametric test revealed a significant difference between user preferences ($\chi^2(2) = 23.414, p < .001$). Post-hoc comparisons using

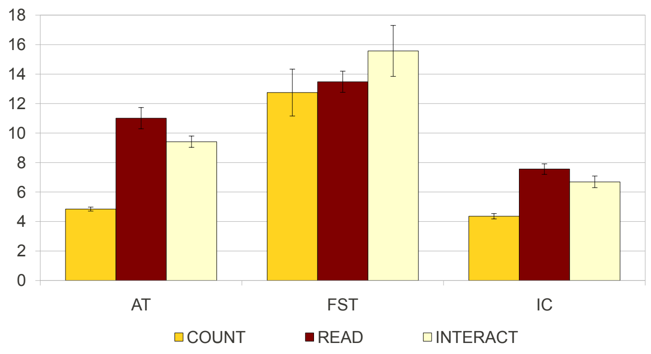


Figure 10. Completion times (seconds) for the three conditions (AT, FST, and IC).

Wilcoxon Signed Rank tests with Bonferroni adjustments showed that IC (6.33) was rated significantly higher than AT (4.73), and that both techniques were evaluated higher than FST (2.47). In the interview, participants mentioned the readability of small text and the ability to interact with obscured user interface elements as main reasons to rank IC higher than FST and AT. FST was primarily disliked for the pie menu to access occluded windows, which was described as hard to use, because only the windows located underneath the pointer were shown. Due to the initial cascaded window layout, only the top-most object window was fully visible (*c.f.*, Figure 9b). Therefore, several users commented that they did not know which windows were located beneath the mouse pointer and, as a consequence, which windows were included in the menu. One user summarized interaction with FST as: *"Transparency and not being able to interact [with occluded content] is very exhausting"*.

We also asked the participants which technique they preferred for the three tasks. For the interact task and read task, IC was chosen by the majority (15 and 12 participants out of 15, respectively). In the interview, most participants stated that IC was most appropriate for the read task as the text in obscured windows was readable, in contrast to the small menus of AT or FST. For the interact task, participants mentioned the ability to directly interact with occluded content without explicitly selecting the corresponding window as exceptionally useful. For the count task, AT was the most preferred technique (selected by 8). Participants commented that the menu of AT provided a good overview, so they could immediately see which images were available.

The results partially support our hypotheses: IC indeed provides an advantage for quickly looking at content in occluded windows. However, compared to AT, the performance advantage is only significant for accessing fine-grained information (as in the read task). The benefit of IC for simple interaction in occluded windows was confirmed by the results of the interact task.

DISCUSSION AND FUTURE WORK

The results of our experiment indicate that importance-driven compositing supports users in skimming through information in occluded application windows. Users described the access to occluded information as "more fluid" compared to sequential window switching menus, as only one activity was necessary to extract information from an occluded window. Another distinguishing aspect is the ability to directly interact with content of occluded windows without explicitly selecting the window in a separated user interface. Participants commented that this feature created a sense of "direct" interaction on the occluded content.

We consider the encouraging results of our exploratory user study as starting point for more empirical evaluations on importance-driven compositing. We will address potential limiting factors of our system which have not been captured by our experiment, such as the effect of automatic window re-arrangements on spatial memory or the effect of a dense information display on focused attention. To assess the suitability of visual saliency as primary measure of importance, we still need to thoroughly investigate whether traditional bottom-up saliency features are sufficient to describe the importance of window content – from user interface elements to various content like text, images, videos, or complex visualizations. Longitudinal studies will help to assess the benefits and limitations of our proposed window management functions in more realistic settings.

With importance-driven compositing, the desktop is not just a collection of (window) rectangles but rather a rich information map. We showed that we can leverage this importance measure for new interaction techniques on window manager level, which would otherwise require access to the application's content. It can also be employed as unified interface to implement previously suggested window management techniques, such as *free-space transparency* [16], *multiblending* [2], or *clipping lists* [20], as fully functional window manager extensions. Image-based representations of physical occlusions can serve as influence to the desktop's importance and thus generate *occlusion-aware* window layouts [27] automatically.

In the future, we will investigate how our importance model can be enriched by techniques for automatic extraction of individual user interface components [25], user-defined window constraints [1], or *fine-grained window management* [7], which takes into account the context of a user's action on a window. We also consider to add scaling as additional optimization factor to our layout routine. Shrinking and expanding regions of low importance can lead to a more flexible content arrangement, especially for large windows. Finally, our prototype has to be considered as proof-of-concept implementation. We demonstrated the feasibility of using state-of-the-art GPU languages in window managers. Yet, system performance can – and should – be further enhanced. Besides low-level optimizations for the GPU code, high-level observations of window changes could be used to trigger only partial importance map updates.

CONCLUSION

We presented importance-driven compositing window management which considers the importance of window content for an optimized spatial window layout in combination with see-through compositing. The aim of the technique is to optimize visibility and interactivity of important content contained in multiple overlapping application windows. Importance is defined by an image-based analysis of visually salient features within window textures. Based on importance-driven compositing, we presented new window management functions to minimize information overlap and to ease the access to occluded content. Results of a preliminary experiment indicate that users could extract fine-grained information and perform easy interaction tasks in occluded application windows faster and with greater subjective satisfaction compared to overlapping windows with sequential Alt+Tab switching. In comparison to free-space transparency [16], users appreciated the increased content visibility due to a more appropriate spatial window layout and the ability to directly interact with occluded content.

ACKNOWLEDGEMENTS
The authors would like to acknowledge Erick Mendez for providing the GLSL implementation for the saliency computation. This work was funded by the Austrian Research Promotion Agency (FFG) BRIDGE 822716.

REFERENCES
1. G. J. Badros, J. Nichols, and A. Borning. Scwm: An extensible constraint-enabled window manager. In *Proc. USENIX 2001*, pages 225–234. USENIX Association, 2001.
2. P. Baudisch and C. Gutwin. Multiblending: displaying overlapping windows simultaneously without the drawbacks of alpha blending. In *Proc. CHI 2004*, pages 367–374. ACM, 2004.
3. B. A. Bell and S. K. Feiner. Dynamic space management for user interfaces. In *Proc. UIST 2000*, pages 239–248. ACM, 2000.
4. A. Bezerianos, P. Dragicevic, and R. Balakrishnan. Mnemonic rendering: an image-based approach for exposing hidden changes in dynamic displays. In *Proc. UIST 2006*, pages 159–168. ACM, 2006.
5. E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses: the see-through interface. In *Proc. SIGGRAPH 1993*, pages 73–80. ACM, 1993.
6. O. Chapuis and N. Roussel. Metisse is not a 3D Desktop! In *Proc. UIST 2005*, pages 13–22. ACM, 2005.
7. O. Chapuis and N. Roussel. Copy-and-paste between overlapping windows. In *Proc. CHI 2007*, pages 201–210. ACM, 2007.
8. N. Elmqvist and P. Tsigas. A taxonomy of 3d occlusion management for visualization. *IEEE TVCG*, 14:1095–1109, September 2008.
9. G. Faure, O. Chapuis, and N. Roussel. Power tools for copying and moving: useful stuff for your desktop. In *Proc. CHI 2009*, pages 1675–1678. ACM, 2009.
10. J. Grudin. Partitioning digital worlds: focal and peripheral awareness in multiple monitor use. In *Proc. CHI 2001*, pages 458–465. ACM, 2001.
11. C. Gutwin, J. Dyck, and C. Fedak. The effects of dynamic transparency on targeting performance. In *Proc. GI 2003*, pages 105–112, 2003.
12. B. L. Harrison, H. Ishii, K. J. Vicente, and W. Buxton. Transparent layered user interfaces: an evaluation of a display design to enhance focused and divided attention. In *Proc. CHI 1995*, pages 317–324, 1995.
13. D. R. Hutchings, G. Smith, B. Meyers, M. Czerwinski, and G. Robertson. Display space usage and window management operation comparisons between single monitor and multiple monitor users. In *Proc. AVI 2004*, pages 32–39. ACM, 2004.
14. D. R. Hutchings and J. Stasko. Revisiting display space management: understanding current practice to inform next-generation design. In *Proc. GI 2004*, pages 127–134, 2004.
15. D. R. Hutchings and J. Stasko. Shrinking window operations for expanding display space. In *Proc. AVI 2004*, pages 350–353. ACM, 2004.
16. E. W. Ishak and S. K. Feiner. Interacting with hidden content using content-aware free-space transparency. In *Proc. UIST 2004*, pages 189–192. ACM, 2004.
17. L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *PAMI*, 20(11):1254–1259, 1998.
18. E. Kandogan and B. Shneiderman. Elastic windows: evaluation of multi-window operations. In *Proc. CHI 1997*, pages 250–257. ACM, 1997.
19. A. Leykin and M. Tuceryan. Automatic determination of text readability over textured backgrounds for augmented reality systems. In *Proc. ISMAR 2004*, pages 224–230. IEEE Computer Society, 2004.
20. T. Matthews, M. Czerwinski, G. Robertson, and D. Tan. Clipping lists and change borders: improving multitasking efficiency with peripheral information design. In *Proc. CHI 2006*, pages 989–998. ACM, 2006.
21. E. Mendez, S. K. Feiner, and D. Schmalstieg. Focus and context in mixed reality by modulating first order salient features. In *Smart Graphics 2010*. Springer, 2010.
22. G. Robertson, E. Horvitz, M. Czerwinski, P. Baudisch, D. R. Hutchings, B. Meyers, D. Robbins, and G. Smith. Scalable fabric: flexible task management. In *Proc. AVI 2004*, pages 85–89. ACM, 2004.
23. G. Robertson, M. van Dantzich, D. Robbins, M. Czerwinski, K. Hinckley, K. Risden, D. Thiel, and V. Gorokhovsky. The Task Gallery: A 3D Window Manager. In *Proc. CHI 2000*, pages 494–501, 2000.
24. E. Rosten, G. Reitmayr, and T. Drummond. Real-time video annotations for augmented reality. In *Proc. ISVC 2005*, pages 294–302. Springer, 2005.
25. W. Stürzlinger, O. Chapuis, D. Phillips, and N. Roussel. User Interface Façades: Towards Fully Adaptable User Interfaces. In *Proc. UIST 2006*, pages 309–318. ACM, 2006.
26. I. Viola, A. Kanitsar, and M. E. Groller. Importance-driven volume rendering. In *Proc. VIS 2004*, pages 139–146. IEEE Computer Society, 2004.
27. D. Vogel and R. Balakrishnan. Occlusion-aware interfaces. In *Proc. CHI 2010*, pages 263–272. ACM, 2010.