International Journal of Smart Computing and Artificial Intelligence
International Institute of Applied Informatics
2017, Vol.1, No.2, P.59 - 75

Developing an Automatic Window Manipulation System Considering Content on Application Windows and User's Behavior

Keisuke Yoshida, Tadachika Ozono, Toramatsu Shintani *

Abstract

Manual window managements become increased with PCs gaining larger display areas, greater processing power, and more applications. Therefore, manipulating windows accordingly and automatically helps us to perform tasks on PCs. In this study, we explored a prototype window management system called FoXpace. FoXpace uses content on application windows and user's behavior in order to move background windows to optimal positions according to an active window. FoXpace employs edges in a desktop and user's work history to evaluate content on application windows and user's behavior, respectively. The edges can be obtained by using image-processing technique. The user's work history consists of mouse clicks, keyboard inputs, and application-switches. We developed an algorithm to determine the optimal position of each windows based on edge detection and the user's work behavior. This paper shows the development of FoXpace and its evaluations. We conducted experiments on our algorithm with questionnaires and an eye-tracking device. We concluded that our window manipulating system can reduce the cost of window manipulation while performing tasks on PCs.

Keywords: edge detection, eye-movements, user's work behavior, window management

1 Introduction

In this study, we explored a prototype window manipulating system called the Flexible Optimizing Extraction space (FoXpace). FoXpace finds an empty space on the worker's display by estimating the importance of the window regions based on the user's work behavior. By allocating the application window to the space, FoXpace reduces the costs of manipulating windows in workspace design. The workspace is changed to reflect the number and type of application windows. When using a personal computer (PC) to perform tasks, workers typically have many application windows open. To perform their tasks effectively, a worker has to design a workspace by manually managing the windows. Window management has become more complex with PCs gaining larger display areas and greater processing power. As the worker performs different tasks, the number and type of applications being used

* Department of Computer Science, Graduate School of Engineering, Nagoya Institute of Technology, Gokiso-cho, Showa-ku, Nagoya, Aichi, 466-8555, Japan

change and the design of the workspace must be managed accordingly. To improve work efficiency, a more flexible mechanism of workspace design is required.

One approach to achieving window opacity or event-transparency is to reflect the available information[1][2]. This approach increases the visibility of the window contents. The cost of manual window management increases if recognition is difficult. Therefore, automated window management is desirable. This paper presents a definition of an effective workspace, and investigates the use of the user's work behavior in the implementation of automated workspace design.

In this paper, we present a study that investigates user's evaluations of workspaces and user's eye-movement when our system manipulates windows. We have proposed FoXpace in the past, but we had not been got feedbacks of it[3]. We carried out questionnaire surveys to analyze effectiveness of workspaces designed by our workspace design mechanism. We analyzed user's eye-movement data to discuss user's cognitive loads.

The remainder of this paper is organized as follows. In Section II, we give a brief definition of a satisfactory workspace. We discuss the use of information on the user's work behavior in Section III. Section IV addresses the implementation of the workspace design mechanism and identifies future research priorities. We note that designs, results, and discussions of experiments about workspaces designed by our system in Section V. Finally, Section VI presents our conclusions.

2 Requirements for Workspace Design

Workers often preform multiple tasks in parallel on a PC. Two factors dominate workspace design: the combination of applications being used for different tasks and whether window management can be automated. In this study, we did not address the combination of applications, but focused on window management.

Recent operating systems (OSs) such as OS X and Windows 10 include a virtual desktop, designed to represent multiple workspaces on the PC. Workers must allocate windows on the virtual desktop manually, requiring extrapolation of the combination of applications needed for the tasks in hand.

2.1 Discussion toward Manipulating Windows Automatically

Automated window management has two aspects: 1) analyzing the information contained in the application window has. 2) Allowing the user to switch applications by directly accessing the window. To address the first point, the Window Distinction, M_{wd}, shows the information being displayed, and the visibility of the window is a measure of the display space management activity[4]. Therefore, M_{wd} can be used as a measure of workspace design. Note that M_{wd} does not depend on the language or execution environment being used on the PC. 2) A user can switch application by direct window access if such a switch is possible using a minimal mouse operation. Switching application methods by mouse operation can be achieved by clicking a window directly or by clicking the application icon in the taskbar or in Docs[5].

There are two main types of window layouts. One is a tiled window layout, and the other is a overlapping window layout. The tiled window layout is any open window is fully visible; windows are not allowed overlap. The overlapping window layout is any open window is allowed overlap and manipulated position in user's operations. A tiled window

layout satisfies both requirements 1) and 2). However, this layout is inflexible with regard to the number of windows. Automated window management should therefore take into account the interest of the user in each window. DFW[6] is one way of addressing this. We focus on an overlapping window layout because this layout has more potential of maximizing visibility than a tiled window layout[7]. A user should learn techniques to satisfy both requirements 1) and 2) on an overlapping window layout. These techniques are difficult for beginners of this layout. Therefore, we discuss an automated window management on an overlapping window layout.

2.2 Quantity of Information displayed on the workspace

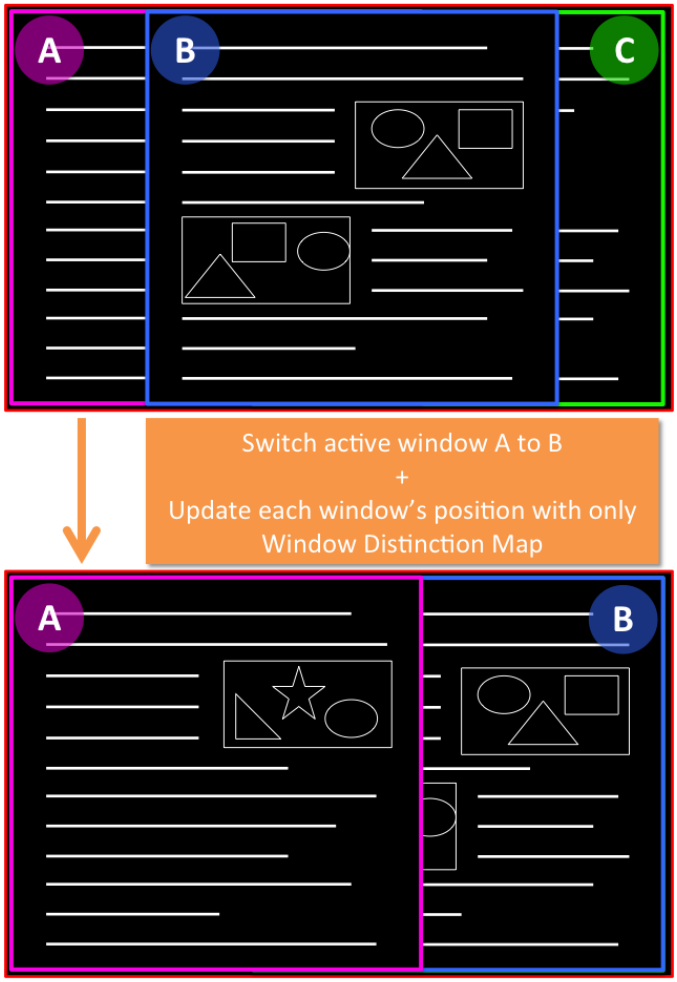

Figure 1: A method of in which only M_{wd} is used for caliculating a value of information.

Figure 1 shows an example of a method in which only M_{wd} is used for automating window management. In Figure 1, A, B, and C denote the application windows. In the top part of Figure 1, B is the active window before the user switches active applications. All three windows are visible. A common technique is to make a portion of C visible, to allow

direct access by mouse click[5][8]. In the bottom part of Figure 1, A is the active window after the user has switched active applications and updated the position of each window. A and B are now visible. If the purpose of window management is to increase only M_{wd} as the evaluation value, C is hidden behind B. A technique in which that portion of the window is displayed in automatic window management is not available using only M_{wd}. For better automated window management, a method of detecting the importance of the window is required.

The information that the user is interested in is defined in the Window Interest Map (denoted as M_{wi}). This can identify the important region within a window, and the user can make just a small part of the window visible to allow application switching. The user can therefore minimize the visible window and optimize the contents displayed in the window.

2.3 Related Works

Many previous studies have investigated ways of increasing the amount of simultaneously visible information and its accessibility on the screen.

WinCuts allows arbitrary regions of existing of windows to be replicated in separate windows[9]. WinCuts improves the efficiency of the workspace by manipulating the window size. This technique provides a user workspace in which only meaningful windows are displayed. FST and IC are techniques for increasing the visible region by manipulating the opacity of windows[1][2], allowing users to see the window that is placed behind other windows. Switchback Cursor is a technique in which the cursor can be moved behind a window[10], providing the user with a 3D mouse operation. While the active window is normally in the foreground, Switchback Cursor can operate the window in the background. Fold-and-drop is a technique based on paper sheet metaphor, in which users can make the window appear to be folded back[11]. All these approaches attempt to make the workspace visible and accessible. These studies indicate candidates of parameters manipulated automaticaly: position, size, opacity, and event-transparency. We discuss how to set these parameters automatically to increase the window visibility and accessibility.

2.4 Evaluation Value for Workspace Design

In this study, we develop a workspace design to meet the two key requirements set out above. The importance of pixels consists of level of user interest and distinctiveness of information. Automatic window manipulation requires a measure to evaluate workspaces. Therefore, we introduce $M_{dev}(x,y,t_c)$ as a measure of the value of information at pixel (x,y) and time t_c. M_{dev} is the sum of M_{wd} and M_{wi}. Thus, $M_{dev}(x,y,t_c)$ balances the amount of information displayed and the user interest in the information. To improve the workspace design, M_{dev} should be maximized. The effectiveness of the workspace is given by

$$Evaluation(t_c) = \sum_{x}^{DW} \sum_{y}^{DH} M_{dev}(x,y,t_c) \tag{1}$$

The larger the value of $Evaluation(t_c)$, the more effective the workspace at time t_c. $M_{dev}(x,y,t)$ represents the value of information at pixel (x,y) and time t, and DW and DH are the display width and height, respectively. Windows are manipulated to increase the sum of M_{wd} and M_{wi}, so that $Evaluation$ increases. M_{wd} and M_{wi} are measures of the distinctiveness of the information and its interest to the user.

We focus on position that is one of the candidate of four parameters: position, size, opacity, and event-transparency. The positioning of the windows takes into account the user's work behavior. Our system manipulates all window of the active widow and inactive windows. The system operates flexibly, dynamically changing the number and type of application windows displayed.

3 Importance of Information for Workspace Design

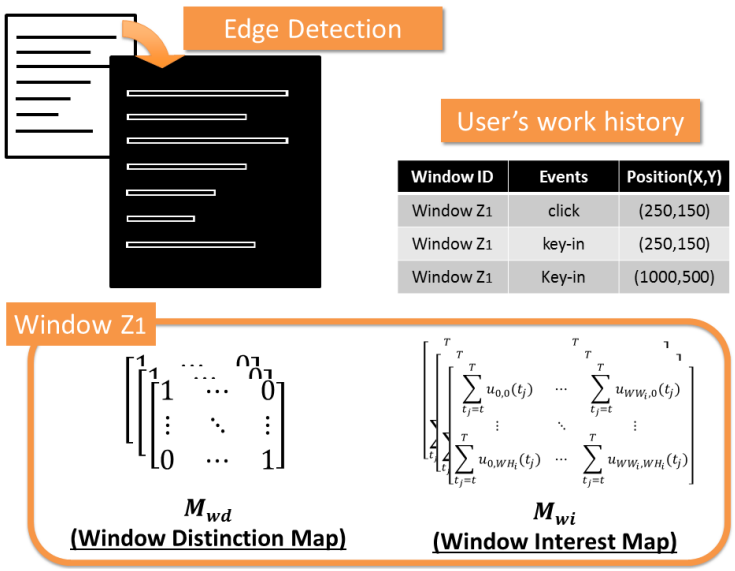

Figure 2: A summary of M_{wi} and M_{wd} for calculating the value of information.

3.1 Information Value at Pixels Regarding Content on Application Windows

As noted above, $M_{dev}(x,y,z,t)$ is a measure of the value of information to the user and the distinctiveness of that information. $M_{dev}(x,y,z,t)$ in Equation 2 presents the value of the information displayed at (x,y)-in the foreground window, and is derived as follows:

$$M_{dev}(x,y,z,t) = \delta(x,y,z)\Big\{\alpha M_{wi}(x,y,z,t) + (1-\alpha)M_{wd}(x,y,z,t)\Big\} \tag{2}$$

$\delta(x,y,z)$ outputs 1 or 0, showing whether an application window z is displayed in the foreground at (x,y), and α is a weight based on the importance of the information displayed. In our system, we set $\alpha = 0.5$ in our experience. We assume that M_{wi} and M_{wd} are equally important to understand the importance of the information displayed. M_{wi}, given as Equation 3, calculates the user's level of interest in the information, based on the work behavior $u(t_c)$ representing the user's work behavior at the time t_c. M_{dev} is the sum of M_{wd} and M_{wi}. A summary of the operation of M_{wd} and M_{wi} is given in Figure 2 and 3. M_{wi} is a matrix of the sum of the user's work behavior. We call M_{wi} Window Interest Map. $M_{wi}(x,y,z,u(t)) \geq 0$ shows their level of interest in the information presented in a window

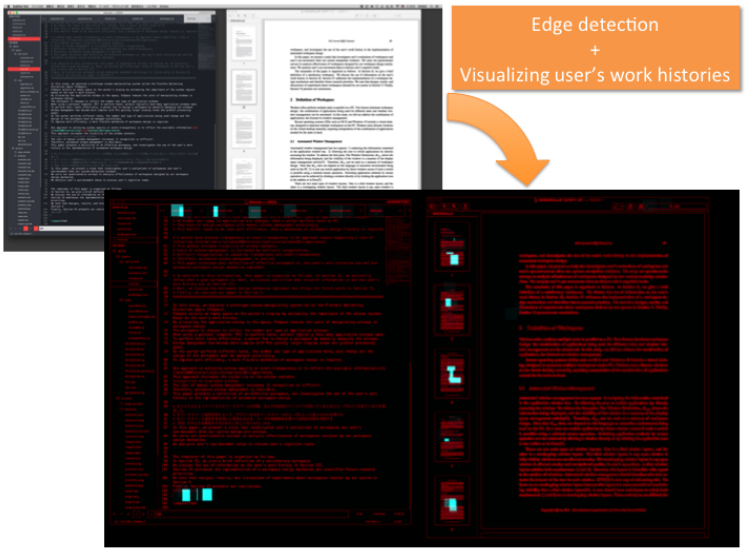

Figure 3: Visualization of M_{wi} and M_{wd} when evaluating pixel (x,y).

positioned in the z-th foreground at pixel (x,y) on the display. $M_{wd}(x,y,z,t) \geq 0$ shows the distinctiveness of the information in the same window, and represents a matrix of the information windows displayed. We call M_{wd} Window Distinction Map.

3.2 Level of User Interest Based on User's Behavior

M_{wi} derives the level of user interest in the information at (x,y). When a user actively interacts with a window, this suggests an increase in interest. We use M_{wi} to reflect a change in user interest and to restrict the significance of the work behavior. M_{wi} derives the value of a pixel from the user's switching applications, from a mouse click, or from a keyboard input. M_{wi} is the sum of $u_{x,y,z}$, as follows:

$$M_{wi}(x,y,z,u(t)) = \sum_{t_j=t}^{T} (u_{x,y,z}(t_j)) \tag{3}$$

Equation 3 derives a value for the user's interest based on the work behavior. $M_{wi}(x,y,z,u(t))$ is the sum of $u_{x,y,z}(t)$ for time T, where $u_{x,y,z}(t)$ is the user's work behavior at time t from the list of events on the window z, Figure 2. (x,y) shows a pixel in the window z.

In this study, we use mouse and keyboard operation events to represent a user's work behavior M_{wi}. There are three types of events: switching applications, a mouse click, and a keyboard input. Application switching events comprise clicking the window directory, clicking an application in the taskbar or Docs, and using a keyboard shortcut (Windows: Alt+Tab, OS X: Cmd+Tab).

When an application switching is recorded, the M_{wi} of all new active windows increased. Mouse clicks are recorded unless they involve application switching, and the M_{wi} near the pixel at which the event happened increased. Keyboard inputs are also disregarded if they trigger an application switching. Otherwise, a keyboard input increases M_{wi} near the pixel of the most recent mouse click.

Algorithm 1 setting new window position algorithm

```
1: Input: w_new, t_c
2: evaluation ← Evaluation(t_c)
3: i ← 0
4: while isCovered(w_new, w ∈ W) do
5:     setPosition(evaluation, w_new, i)
6:     i ← i + 1
7: end while
8: for w' ∈ {w|isOverlap(w_new, w), w ∈ W} do
9:     updatePosition(evaluation, w)
10: end for
```

Figure 3 shows a before-after visualization of the interest level of the information. Interest in information in indistinct positions is indicated in light blue. Interest in information resembles a tab or search form in this example.

3.3 Distinctiveness of Information

M_{wd} shows the distinctiveness of the information at (x,y). Our system uses an edge detection technique to extract M_{wd}, as follows:

$$M_{wd}(x,y,z) = EdgeDetection(WindowImage_z(x,y)) \tag{4}$$

$EdgeDetection(WindowImage_z(x,y))$ takes a gray-scale transformation of $WindowImage_z$ and applys edge detection with spatial filtering using the Laplacian filter. Figure 3 shows a before-after view of edge detection. Areas in red show the pixel of the edges. The system recognizes both characters and graphics as information. As these have a different color from the background, differences in color are used to detect borders. Edge detection shows areas where information is present.

4 Implementation

4.1 Algorithms

In designing the workspace, candidate parameters for automated window management are position, size, opacity, and event-transparency[1][2][9][14]. Our design takes account of the legibility of the window, and the position and size are manipulated manually to ensure that the windows do not overlay each other. An Integrated Development Environment (IDE) requires the area of the desktop to provide a comfortable work environment. Automatic manipulation of the size parameter disturbs the user's work pattern. Opacity and event-transparency are challenging as the user may become confused. In this study, we focus on automated manipulation of the window position to optimize its visibility and accessibility.

Our system updates the window position when a new application window is created or an active application is switched. When a user creates a new application window, the workspace and M_{wd} change significantly, reflecting a turning point in the task. Thus, the

Algorithm 2 $setPosition(evaluation, w_{new}, i)$

```
1: Input: evaluation, w_new, i
2: checkRow ← screenWidth/w_new.size.width
3: checkCol ← screenHeight/w_new.size.height
4: CA ← ⟨⟩
5: for j = 0 to checkRow do
6:    for k = 0 to checkCol do
7:       for dir ∈ {TL, TR, UL, UR} do
8:          w_candidate ← candidate(w_new, dir, j, k)
9:          w_candidate.score ← evalPortion(w_candidate)
10:         CA ← append(CA, w_candidate)
11:      end for
12:   end for
13: end for
14: CA ← sort(CA)
15: w_new.pos ← CA[i].pos
```

TL:Top Left	TR:Top Right	BL:Bottom Left	BR:Bottom Right

user needs to manipulate the window significantly. If a user switches an active application, M_{wi} again changes significantly. Concretely, the user is now less interested in the formerly active windows and more interested in the currently active window. If two windows are activated in alternate shifts, M_{wi} treats these as representing a group for the performance of a task. When applications are used in parallel, each application window should be maximally visible. M_{wd} and M_{wi} can use the progress of work on a task to reflect the information needed.

Algorithm 1 gives the setting of a new window position. It ensures that one application window is not masked by another, and derives the position that maximizes M_{dev}. The input values are the creation of a new window W_{new} and the current time t_c. The algorithm sets W_{new} in an empty space and operates in two steps.

Step 1 sets up a new window in an empty space unless this would mask existing windows. The function $isCovered(w_{new}, w \in W)$ outputs 1 if this is the case and, 0 otherwise. The function $setPosition(evaluation, w_{new}, i)$ places the new window in the i-th emptiest space in the separated displays on the grid. Step 2 detects existing windows that overlap with the new window. The function $isOverlap(w_{new}, w)$ outputs 1 if an existing window overlaps the new window, and 0 otherwise. Algorithm 1 has the function $updatePosition(evaluation, w)$. This sets the existing window at a position that maximizes M_{dev}. These steps take into account that when switching windows with a mouse operation, the user tends to click on the window directly[5].

When a user switches an active window, the sum of the previously active windows, M_{wi}, is increased and that of the newly active windows, M_{wi}, is reduced. The algorithm that updates the existing window position when the active window is switched calls-
$updatePosition(evaluation, w)$ in each window that overlaps with the newly active window. This approach supports the performance of tasks requiring multiple windows. The algorithm allows each window position to reflect the changed status of the task.

Algorithm 2 sets the window position at the i-th emptiest space in the separated displays

Algorithm 3 $updatePosition(evaluation, w_{target})$

```
 1: Input: evaluation, w_target
 2: stopCount ← 0
 3: current ← evaluation
 4: while stopCount < 5 do
 5:    distance ← random()
 6:    CA ← ⟨⟩
 7:    for dir ∈ {T,B,L,R,TL,TR,BL,BR} do
 8:       w_candidate.pos ← move(w_target, dir, distance)
 9:       w_candidate.score ← evalAll(w_candidate)
10:       CA ← append(CA, w_candidate)
11:    end for
12:    w_candidate ← MAX(CA)
13:    if current > w_candidate.score then
14:       stopCount ← stopCount + 1
15:    else
16:       stopCount ← 0
17:       w_target.pos ← w_candidate.pos
18:    end if
19: end while
```

T:Top	B:Bottom	L:Left	R:Right
TL:Top Left	TR:Top Right	BL:Bottom Left	BR:Bottom Right

in the grid. The inputs are the sum of M_{dev} at the current time *evaluation*, a new window w_{new}, and a number of order i. In lines 2 and 3, *checkRow* and *checkCol* are calculated to separate the displays in the grid. In lines 5 to 13, candidates are selected for the new window position and given scores. This algorithm separates the display in grids using basing points dir, which are elements of the top left, top right, bottom left, and bottom right. $candidate(w_{new}, dir, j, k)$ outputs a candidate for the separated display in the j-th row and k-th column from dir as $w_{candidate}$. $evalPortion(w_{candidate})$ outputs the sum of a portion of M_{dev} giving a position for $w_{candidate}$ as a score. $append(CA, w_{candidate})$ outputs an array that is CA appended to $w_{candidate}$. Lines 14 and 15, sort CA with these scores and set w_{new} to the i-th position that has the lowest score in CA.

Algorithm 3 updates the window position. The purpose of Algorithm 3 is to set each window as part of a group to perform a common task. The inputs to this algorithm are the sum of M_{def} at the current time *evaluation*, and a window w_{target}. This algorithm searches for a local maximize M_{div} using hill climbing. It sets the window at a position that is local maximize of M_{dir}. In line 5, distance is assigned a random number by $random()$ as the moving distance. Lines 7 to 11, line up candidates for new window positions and score them. $move(w_{target}, dir, distance)$ outputs a candidate to be moved from dir and *distance*, and $evalAll(w_{candidate})$ outputs the sum of M_{dev} as a score. The candidate with the maximum score in CA is selected as $w_{candidate}$. If $w_{candidate}$ has a larger score than the current score *current*, w_{target} is updated with $w_{candidate}$. If $w_{candidate}$ has a lower score than the current score, *stopCount* is incremented. The termination condition is that *stopCount* exceeds a value of five, indicating that w_{target} has not been recently updated.

4.2 Example

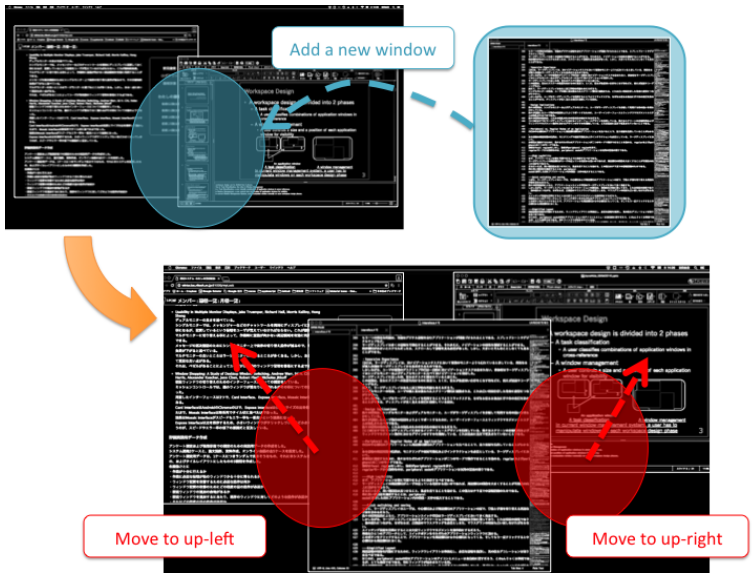

Figure 4: An example of manipulating windows on a new window created.

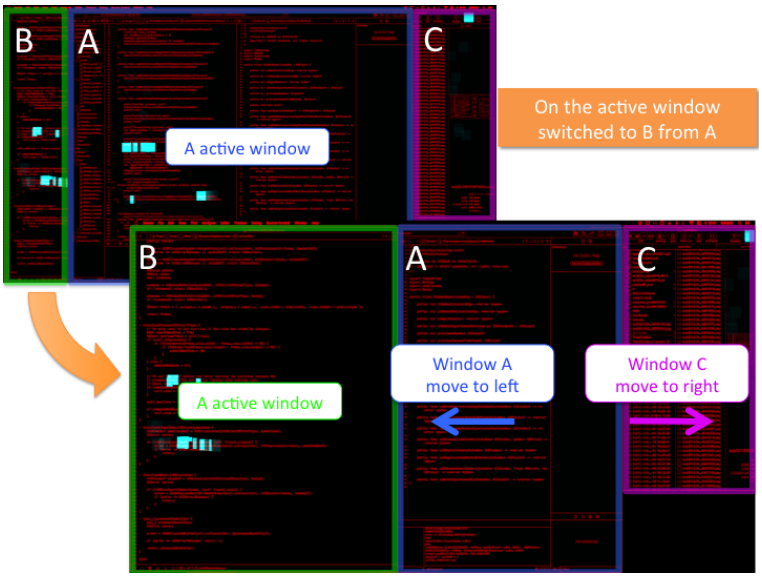

Figure 5: An example of manipulating windows on an active window switched.

A before-after example of the execution is given in Figure 4 and 5. In Figure 4, window A and B existing windows showing a web browser and a PDF viewer. Window C is a new window showing a text editor. When the new window is created, it has the least information position as its initial position. At this time, the new window is set in a position that overlaps with the existing windows, and the overlaid portion is moved to position M_{dev}, representing the local maximum. In Figure 4, window A moves to the top left and window B moves to the top right. This example shows our system locally maximizing the amount of information by manipulating a single window position. In Figure 5, there are four windows on the desktop.

Window A and B are existing windows showing a text editor with two columns and a PDF viewer. When a user switches the active window from window A to window B, window A is overlaid by window B and moves to the left to maximize M_{dev}. The level of interest in the information in the indistinct position (indicated in light blue) is six. There is interest in the information to the right of window A, so window A is moved to the left to show more information behind the two windows. This example shows how our system retains the possibility of direct access using a mouse click.

These examples of the execution demonstrate the more effective use of the display area. Our system increases the amount of information shown in the display, and allows each window to be directly accessed.

5 Experiment

In this section, we describe our experiments and results on our system. The goal of the experiments is to evaluate visibility and accessibility of windows in workspaces determined by our system. First, we explain our experimental setup, then we discuss the results.

5.1 Experimental Setup

We evaluated that our system can improve visibility and accessibility of windows in workspaces. We conducted questionnaire surveys and eye-movement analysis. In our experiments, subjects evaluated some pictures of workspaces with an eye-tracking device and answered questionnaire. The subjects were 11 students, consisting of 6 university students and 3 graduated students, 21-23 years old, who are used to managing windows in workspaces.

Table 1: Workspaces.

Workspace A	Our system: generated by our system from Workspace B
Workspace B	Manual layout: generated manucally
Workspace C	Tiled layout: a representative layout method for minimizing empty space

Table 2: Tasks.

	Description	Applications
Task 1	developing a web application	a Web browser, a text editor, and a Terminal
Task 2	developing a web page	a Web browser, a text editor, and a Finder
Task 3	reading a paper	two Web browsers, a text editor, a dictionaly, a PDF viewer, and a Finder
Task 4	making a presentation slide	a Web browser, a text editor, a PowerPoint, and a Finder
Task 5	meeting with online chat	a Web browser, and a chat tool

We used the pictures because of applying same experimental environments for all subjects. The size of pictures were 2,560 x 1,440 pixels and the size of the display was 27 inches. The subjects evaluated 15 pictures, consisting 3 pictures of workspaces in Table 1 on every 5 tasks in Table 2. Table 1 shows the three types of the workspaces. Workspace A was determined by our system, Workspace B was arranged manually, and Workspace C

was a tiled layout. Table 2 shows the five tasks. In the tasks, users needed to refer multiple windows in the workspaces and the tasks were often observed in our laboratory. We told a main window of each task to the subjects in order to evaluate the usefulness of the layouts. The main window is the most important window on a task.

Table 3: Questions.

Q1	How useful is this workspace to perform the task?
Q2	How much information do you view for performing the task?
Q3	Which window operations are required to improve this workspace?
Q4	How many window operations are required to improve this workspace?
Q5	How many spaces are found to put new windows?
Q6	Which window operations are required before you create a new window?
Q7	How many window operations are required before you create a new window?

The subjects answered the seven questions in Table 3 and free answers during the evaluation. Q1, Q2 and Q5 were scored from 1 (worst) to 7 (best). In Q3 and Q6, the window operations were “move”, “resize”, and “maximize or minimize”. In Q4 and Q7, the options were “0”, “1 or 2”, “3 or 4”, “5 or 6”, and “7 or more”.

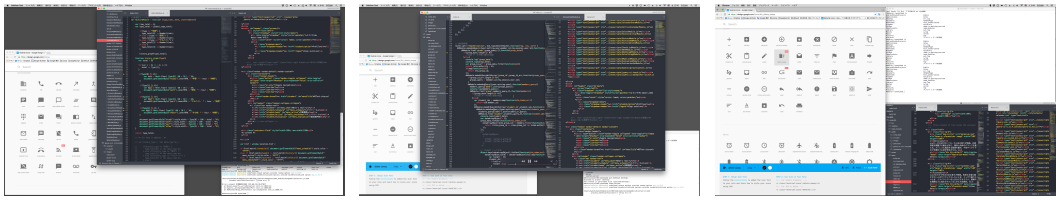

Figure 6: Examples of workspaces: Workspace A, B, C from left to right.

Figure 6 shows an example of the picture workspaces. Workspace A, B, and C in Table 1 are shown from left to right in Figure 6. We shows results of the questionnaire in Figure 7 to 12.

We investigated user’s cognitive loads with user’s eye-movement during automatic window manipulation because the manipulation may increase the load raised by losing windows from the user’s sight. We collected eye motion data by using an eye-tracking device Tobii Eye X[1], and then visualized user’s eye movements. In this evaluation, we showed movies of window management by using our system to the subjects in order to control the experiments. First, we explained performed tasks in the five movies to the subjects. The tasks contained cross-reference tasks. Second, the subjects watched the movies with the eye-tracking device. Finally, we generated maps of the collected eye movements on workspaces.

5.2 Results

Figure 7 to 12 show results of questionnaires. Figure 13 is an eye-movement map that represents the frequency of user gazes on locations in a workspace.

Figure 7 shows questionaire results, average scores on Q1, Q2, and Q3. In Figure 7, the X-axis and Y-axis show questions and an average score of questions, respectively. The three bars for each question are the average scores for Workspace A (proposed), B (manual), and C (tiled) from left to right. Workspace A is best for Q5, and it means our method can make

[1]https://tobiigaming.com/product/tobii-eyex/

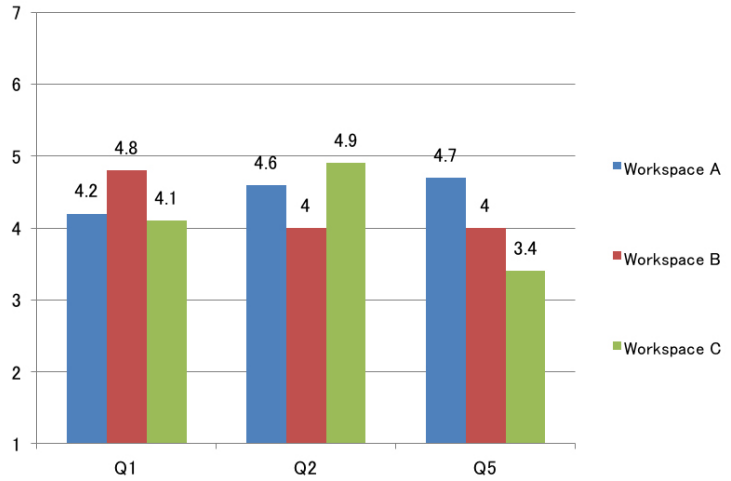

Figure 7: Average scores on Q1, Q2, and Q5.

more space for new windows than the others. The result of Q2 means that Workspace A is good for the amount of information that users can view. However, the result of Q1 means the manual window management is better than our method. We discuss the reason why Workspace A has less score than Workspace B on Q1. A total of nine subjects answered "I do not want a partially visible main window during the window is active" for Q2 and Q4. We investigated the difference of Q2 and Q4.

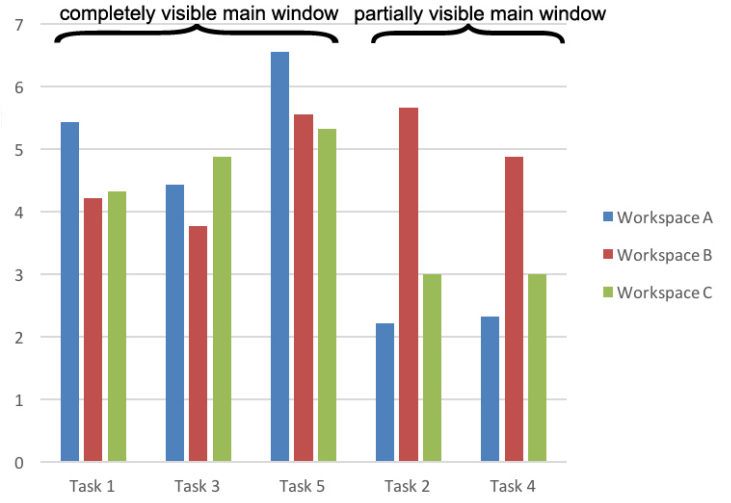

Figure 8: Average scores on each task.

The system made a main window partially visible in Task 2 and 4, a human did not. In Task 2 and 4, our system made main windows partially visible. The average scores on Task 2 and 4 for Q1 are 2.2 and 2.3, respectively. Human PC users move a part of a window to out of a display if the window is not a main window, it makes windows partially visible.

Because our system tries to maximize visible content of background windows, the system also makes partially visible windows. However, our system cannot consider a main window. On the other hand, the average scores on Task 1, 3 and 5 are 5.4, 4.4 and 6.6, respectively (Figure 8). In those cases, the workspaces had completely visible main windows. On Task 1, 3 and 5, our method outperforms the manual layout. If we can detect a main window in a workspace, we can improve our system on partially visible windows.

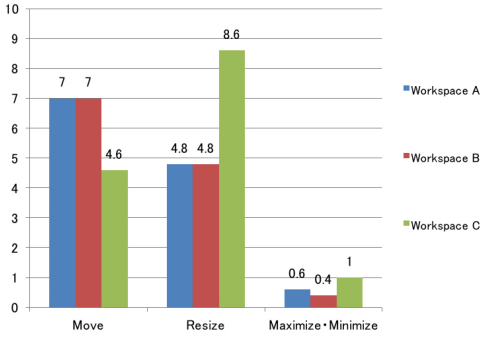

Figure 9: Averages of participants required to make a workspace better.

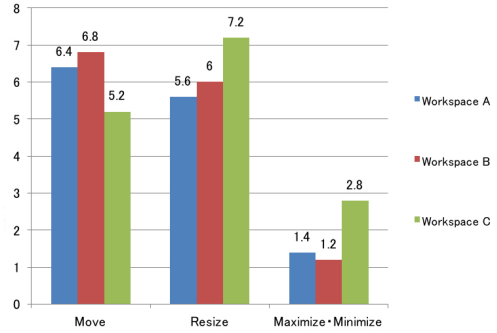

Figure 10: Averages of participants required to make a workspace better when a new application window is created.

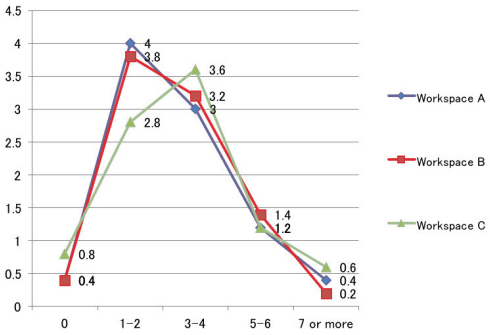

Figure 11: Average numbers of window control required to make a workspace better.

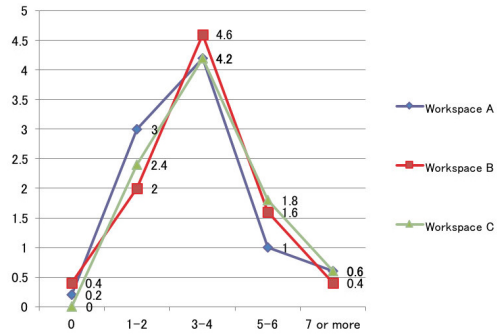

Figure 12: Average numbers of window control required to make a workspace better when a new application window is created.

Figure 9 and 10 show averages of numbers of subjects on Q4 and Q6. In this graph, X-axis shows types of window operations and Y-axis shows an average of numbers of subjects who answered the operation is required. Three bars for each question represent Workspace A, B, and C from left to right. ”Resize” is required in Workspace C in Figure 9 and 10. Tiled layout sets each window to default position and size, however, it ignores application functions. It may be good for users because Tiled layout is easy to predict a window position and size and to manage empty space for them. Tiled layout depends on only display resolution and a number of windows. Figure 9 and 10 indicate that the size of application window depends on an application function. Our method may treat application functions.

Figure 11 and 12 show average numbers of window operations on Q5 and Q7. In this graph, X-axis is kinds of window control and Y-axis is a average of participants require the kind of window control. The average numbers of window control are 2.46 on Workspace A, and 2.9 on Workspace B, and 3.1 on Workspace C in Figure 11. These are calculated with

X-axis items as 0, 1.5, 3.5, 5.5, and 7.5 from left to right in Figure 11. Similarly, average numbers of window operations are 3.26 on Workspace A, 3.43 on Workspace B, 3.53 on Workspace C in Figure 12. In Figure 11 and Figure 12, Workspace A is least number of window operations to improve manually. Therefore, we conclude that our system can automatically improve workspaces for users.

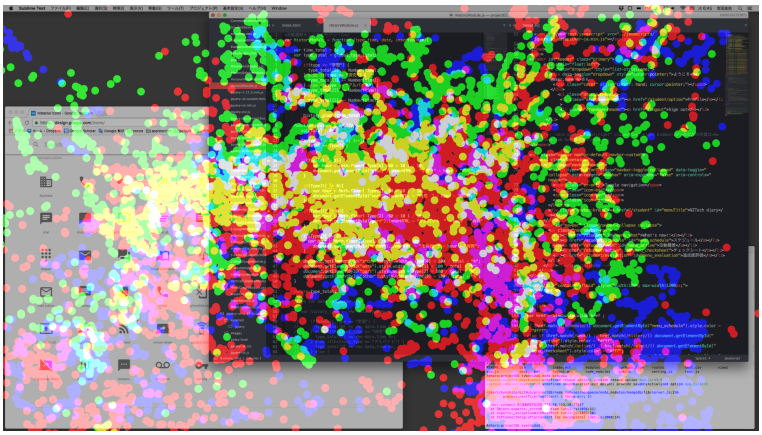

Figure 13: A map of eye-movements.

Figure 13 shows eye-movements of three participants on our system. The system manipulated windows during application switches. They saw closed position from the center of a display, while windows positioned bottom-left, top-right, and top-left. They had same images about the window layout, while our system manipulated windows. Therefore, user's cognitive load is less in manipulating windows automatically.

A map of Figure 13 is similar to 81% of mouse events occurred at the center region of display[12]. We got a feedback "I want to put a main application window in center of display" from a participant. We indicate that an active window in center of display is preferred by users.

5.3 Discussion

In our system, M_{dev} is the sum of M_{wd} and M_{wi}. This allows windows to be accessed directly. However, M_{wd} is equally reflected in meaning less information. For example, in an editor like Microsoft Excel, part of the region may represent an empty space to the user, but M_{wd} gives a large value from the edge detection of the ruled region. M_{dev} requires a method of eliminating regular patterns like ruled spaces. Moreover, we should consider which application window is main to perform a task from results of experiments. We predict that the efficiency of M_{dev} can be improved by introducing the product of M_{wd} and M_{wi}.

Peripheral vision acuity is weaker than central visual acuity. There is a limit to the information that can be seen when windows are displayed simultaneously, even when a large monitor or multiple monitors are used. When windows are widely separated on a large display, switching the gaze between windows alternately is difficult. We closely group related windows may improve the user's work efficiency. Figure 12 shows users prefers to the center of a display. M_{wi} measures user interest in information based on the user's work history, and thus, M_{dev} may be used to vary M_{wi} (VAR_m). As VAR_m decreases, each interest of information is closely.

Files, folders, shortcuts, and applications can be placed directly on the desktop (DesktopFolder). DesktopFolder covers almost the complete region of the desktop, and is a special folder that can be easily accessed. Users often place temporary files and folders of projects in progress in DesktopFolder. These are easily accessed by clicking. NMs leave desktop icons uncovered, allowing direct access[13]. We assume that desktop icons are important when performing tasks and M_{dev} should be expanded to include a Desktop Icons Map M_{di}, showing the position of the desktop icons.

In our proposed system, we specifically focused on the position of windows. As noted above, size, opacity, and event-transparency are also important. A window management system should understand the user when manipulating the window size automatically. Figure 8 and 9 show that each application window has proper size in their function. This should take into account not only resizing of the window but also rescaling. M_{wd} is changed by window rescaling, giving an indication of how a window is rescaled.

M_{dev} is useful for automating the control of window's opacity and event-transparency. However, this may also reduce legibility, for example, when translucent characters overlay other characters. Thus, the automatic opacity and event-transparency need to be controlled in a way that improves legibility[14].

6 Conclusion

In this paper, we explored a prototype window manipulating system. We presented a definition of an effective workspace, and discussed the use of a user's work history to automate workspace design. We used M_{wi} to set values from the user's work history based on application switching, mouse clicks, keyboard inputs. We set M_{wd} based on edge detection. We defined M_{dev} as the sum of M_{wi} and M_{wd}, and used it to automate window management. We noted that our questionnaire surveys and eye-movements analysis. In the experiments, our window manipulation system can reduce the cost of window manipulations on PCs. We demonstrated that user's cognitive loads are negligible on automatic window manipulations. We indicate potential future improvements to automated workspace design.

Acknowledgments

This work was supported in part by JSPS KAKENHI Grant Number 15K00422, 16K00420.

References

[1] E. W. Ishak and S. K. Feiner, "Interacting with hidden content using contentaware free-space transparency," in Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology, ser. UIST '04, 2004, pp. 189-192.

[2] M. Waldner, M. Steinberger, R. Grasset, and D. Schmalstieg, "Importance-driven compositing window management," in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ser. CHI '11. ACM, 2011, pp. 959-968.

[3] K. Yoshida, T. Ozono, T. Shiramatsu, "FoXpace: Manipulating Windows Based on the User's Work History," in Advanced Applied Informatics (IIAI-AAI), 2016 5th IIAI International Congress on, 2016, pp. 698–703.

[4] G. Robertson, M. Czerwinski, P. Baudisch, B. Meyers, D. Robbins, G. Smith, and D. Tan, “The large-display user experience,” Computer Graphics and Applications, IEEE, vol. 25, no. 4, 2005, pp. 44-51.

[5] D. R. Hutchings, G. Smith, B. Meyers, M. Czerwinski, and G. Robertson, “Display space usage and window management operation comparsons between single monitor and multiple monitor users,” in Proceedings of the Working Conference on Advanced Visual Interfaces, ser. AVI ’04. ACM, 2004, pp. 32-39.

[6] H. Shibata and K. Omura, “Reducing the cost of window operations by docking windows,” International Journal of Innovative Computing Information and Control, vol. 9, no. 12, 2013, pp. 4665-4679.

[7] S. A. Bly and J. K. Rosenberg, “A comparison of tiled and overlapping windows,” in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ser. CHI ’86. ACM, 1986, pp. 101-106.

[8] A. Warr, Ed H. Chi, H. Harris, A. Kusher, J. Chen, R. Flack, N. Jitkoff, “Window Shopping: A Study of Desktop Window Switching,” in Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, 2016, pp. 3335–3338.

[9] D. S. Tan, B. Meyers, and M. Czerwinski, “Wincuts: Manipulating arbitrary window regions for more effective use of screen space,” in Proceedings of the CHI ’04 Extended Abstracts on Human Factors in Computing Systems, ser. CHI EA ’04. ACM, 2004, pp. 1525-1528.

[10] S. Yamanaka and H. Miyashita, “Switchback cursor: mouse cursor operation for overlapped windowing,” in Human-Computer Interaction, INTERACT 2013. Springer, 2013, pp. 746-753.

[11] P. Dragicevic, “Combining crossing-based and paper-based interaction paradigms for dragging and dropping between overlapping windows,” in Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology, ser. UIST ’04. ACM, 2004, pp. 193-196.

[12] X. Bi and R. Balakrishnan, “Comparing Usage of a Large High-Resolution Display to Single or Dual Desktop Displays for Daily Work,” in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, 2009, pp. 1005–1014.

[13] D. R. Hutchings and J. Stasko, “Revisiting display space management: Understanding current practice to inform next-generation design,” in Proceedings of Graphics Interface 2004, ser. GI ’04. Canadian Human- Computer Communications Society, 2004, pp. 127-134.

[14] K. Yoshida, Y. Niwa, T. Ozono, and T. Shintani, “A method for improving the legibility of overlay web browsing,” in The Institute of Electronics, Information and Communication Engineers, vol. 115, no. 381, 2015, pp. 25-30(in Japanese).