Design of Platform Independent, Collaborative Graphical Environment Using Edge Detection

Ramakrishnan D, Shiyam P and Raghuveera T
College Of Engineering Guindy,
Anna University, Chennai

Abstract— **Graphical authors face a plethora of challenges in authoring images collaboratively. Their tasks involve work by different authors at each and every stage of the design process. Some of the challenges [1] in such a collaborative environment are time and space, awareness, communication, intellectual property, simultaneity and locking, protection, workflow, security and platform independence. This paper aims at providing a design environment, which is platform independent and collaborative through Layer masking based on Canny's edge detection algorithm.**

***Index Terms*—Collaborative, Common User Access Area, Convolution Mask, Edge Detection, Graphics, Logical User Area Mapper.**

I. INTRODUCTION

REAL world application involves synchronization of several tasks and Co-Authoring on any single entity say an image or a document etc. In this paper we will give the design of an architecture neutral collaborative drawing environment, which uses the browser as a drawing canvas. This kind of collaborative design environment helps authors to create their own designs or simultaneously edit their work with the help of other authors.

In order to collaboratively author an image it would be easier to divide the image into stochastic units and allocate them to multiple users. But it would be more meaningful to divide the image logically than our traditional way of random piecing. For example in scenery with mountain and a background; instead of dividing it into stochastic units it would be more meaningful to divide the scenery into two pieces, mountain and the background. This logical division can prevent the usage of different colour combinations for the same unit by different authors. For example a part of blue sky in a stochastic unit may be painted using B value 60 by one author and 61 by another author. Such a logical division can be achieved using *edge detection* techniques.

II. RELATED WORK

Cliki [2] is a collaborative drawing tool based on the same principle as the Wiki web. In this anyone can access a Cliki-based drawing and edit them inside their Web browser. GRACE (GRAphics Collaborative Editing) is a project aimed at supporting real-time, synchronous collaborative drawing over the Internet [3]. Their approach is to replicate the shared document at each user's site so that editing is done on the local copy, and then propagated to the remote copies. Moksha [4] is a prototype collaborative system. It incorporates sound to help promote awareness of actions that may be happening outside of the user's visual focus. The Global Information Systems Group at the University of Zurich is developing the Universal Information Platform, a distributed information management system. One of the applications being built for the UIP is a collaborative graphical editor [5]. In addition to the basic operations offered by other collaborative drawing tools their editor offers grouping and ungrouping operations. OPEN_Studio [6] is a Web-based drawing environment that allows groups to work on and share drawings. It uses a Java applet to let users define a color (by controlling the amount of red, green, and blue), select a texture and brush size. The Poietic Generator [7] is a device where many users collaboratively author the image at the same time. The working of Poietic Generator is simple. An image is first divided into many stochastic squares. Each user controls only one square of the entire image at any point of time and the other squares are locked from the user. The only way for the user to swap squares was to disconnect and reconnect. This idea has the following drawbacks

- As the image is divided randomly into stochastic square units, nebulous outcome results.
- A single entity say may be divided among two squares and two users may give different colors to the same entity.

In this paper we provide a lucid way to collaboratively author an image. We detect all the edges in an image and divide the image into meaningful entities and allocate them to users instead of just randomly dividing them. Such a design environment is free from the drawbacks stated above.

III. CHOOSING AN EFFICIENT EDGE DETECTION ALGORITHM

Edge detection refers to the process of identifying and locating sharp discontinuities in an image. The discontinuities are abrupt changes in pixel intensity which characterize boundaries of objects in a scene. Classical methods of edge detection involve convolving the image with an operator (a 2-

D filter), which is constructed to be sensitive to large gradients in the image while returning values of zero in uniform regions. There are an extremely large number of edge detection operators available, each designed to be sensitive to certain types of edges. Variables involved in the selection of an edge detection operator include edge orientation, noise environment and edge structure.

- The most apparent aim is to have low error rate.
- It is important that edges occurring in images should not be missed and that there be no responses to non-edges.
- The next criterion is that the edge points be well localized, that is the distance between the edge pixels as found by the detector and the actual edge is to be at a minimum so that logical divisions are intelligible.
- The above mentioned criteria were not substantial enough to completely eliminate the possibility of multiple responses to an edge. Therefore it necessitates having only one response to a single edge.

Some of the popular edge detection techniques are Sobel, Prewitt, Canny and Roberts. Gradient-based algorithms such as the Prewitt filter have a major drawback of being very sensitive to noise. The size of the kernel filter and coefficients are fixed and cannot be adapted to a given image. An adaptive edge-detection algorithm is necessary to provide a robust solution. Such an algorithm should be adaptable to the varying noise levels of the images to help distinguish valid image contents from visual artifacts introduced by noise. One such algorithm is Canny's Edge detection algorithm. Even though Canny's edge detection algorithm is computationally more expensive compared to Sobel, Prewitt and Robert's operator it performs better than all these operators under almost all scenarios.

In order to implement the Canny's edge detector algorithm [8], a series of steps must be followed. The first step is to filter out any noise in the original image. Once a suitable mask has been calculated, the Gaussian smoothing can be performed using standard convolution methods. A convolution mask is usually much smaller than the actual image. As a result, the mask is slid over the image, manipulating a square of pixels at a time. The larger the width of the Gaussian mask, the lower is the detector's sensitivity to noise**.** After smoothing the image and eliminating the noise, the next step is to find the edge strength by taking the gradient of the image. The Sobel operator performs a 2-D spatial gradient measurement on an image. Then, the approximate absolute gradient magnitude (edge strength) at each point can be found. The Sobel operator uses a pair of 3x3 convolution masks, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows) which are shown below

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Fig.1. Convolution Mask

The magnitude, or edge strength, of the gradient is then approximated using the formula:

$$|G| = |Gx| + |Gy|$$

The direction of the edge is computed using the gradient in the x and y directions.

$$\theta = \tan^{-1} (Gy / Gx)$$

Once the edge direction is known, the next step is to relate the edge direction to a direction that can be traced in an image.

After the edge directions are known, non-maximum suppression now has to be applied. Non-maximum suppression is used to trace along the edge in the edge direction and suppress any pixel value (sets it equal to 0) that is not considered to be an edge. We modify this part of Canny's algorithm to assign our own id for the logical units as explained in the next section. Finally, hysteresis is used as a means of eliminating streaking.

IV. WORKING SCENARIO

Our design architecture is shown in Fig.7. We explain our environment's working with the scenery shown in Fig.2, the image to be edited collaboratively. We first convert the RGB image to a gray scale image as shown in Fig.3. We then store the image into a two dimensional array. Each element in the array corresponds to a pixel in the image.

Fig.2. Image to be edited.

Fig.3. Image after graying.

This two dimensional array is the instance to Canny's algorithm and the edges are detected. For example the instance (based on K values in CMYK) to Canny's algorithm for the image shown in Fig.3 is shown in Fig.4.

31	31	31	31	31	31	31	31
31	31	31	31	31	31	31	31
31	31	31	31	31	31	31	31
31	31	86	86	31	31	31	31
31	86	86	86	86	31	31	31
86	86	86	86	86	86	31	31
86	86	86	86	86	86	86	86
86	86	86	86	86	86	86	86

Fig.4. Image Instance to Canny's Algorithm for the image in Fig.3

This array in Fig.4 contains values where 86 corresponds to mountain and 31 correspond to the sky. When this array is passed to our system, the image is split into two logical units i.e. mountain forms one logical unit and sky forms another logical unit. These logical units are replicated layers of the original image.

In order to identify which layer a pixel corresponds to we use a *mutex table.* A mutex table is a table, whose size is as same as that of the original image. This table gives information about which logical unit a specific pixel correspond to. We name it *mutex table* because when one user accesses one logical unit, that unit will be masked from other users. In the example shown in Fig.2 let the mountain regions correspond to logical unit1 and the blue sky correspond to logical unit2. Let their ids be 1 and 2 respectively. The generated Mutex table is shown in Fig.5.

2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	1	1	2	2	2	2
2	1	1	1	1	2	2	2
1	1	1	1	1	1	2	2
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

Fig.5. Mutex Table

V. Layer Masking

Layer Masking is a process in which the output is a logical unit that the other users cannot access until the working user releases it. The logical units thus produced are

Fig.6. Replicated layers after logical splitting

These layers are then placed in a *Common User Access Area* (CUAA). Let CUAA be present at the image server as shown in Fig.7. *We assume that no two users access the same logical unit simultaneously.* As and when the authors pick their logical image units they acquire a lock over theirs from others i.e. the image is masked from other authors. Current working authors and their corresponding logical unit ids are stored in *Logical User Area Mapper* (LUAM) table.

VI. Platform Independence

Proprietary collaborative authoring software is limited to the platforms on which it is being developed. On the other hand, network based collaborative drawing tools lets users work on different platforms. In particular web-based collaborative environment can be promoted as an architecturally neutral environment by using a "standard" web browser as the clients drawing canvas, although browser capabilities varies significantly. Our design environment uses the java capability, so that the system is platform independent, sufficiently flexible and user friendly with a good graphical user interface.

VII. Architectural Design

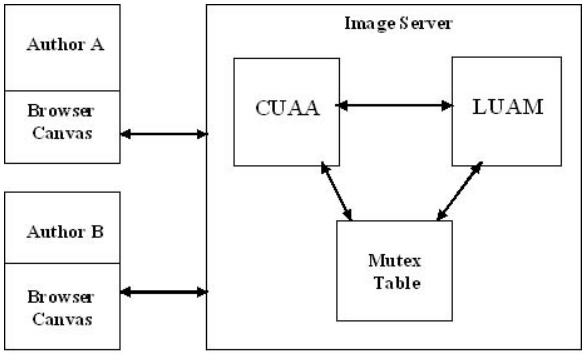

Fig.7. Design Architecture

The architecture proposed here is easy to implement. The advantages of using browser as drawing canvas are

- Good Graphical User Interface to the end users through use of good GUI web design techniques at the browser.
- Technology adaptation i.e. the design environment can be improvised just by changing the web technology involved.

Such a design environment also produces software that is just a *download and use* form and avoids installations.

The design of Logical User Area Mapper, Common User Access Area and Mutex Table should be done carefully. These three entities form the heart of our design environment. The existing software can be modified to elevate its performance just by adding these units into it.

VIII. Example Scenario

Let us assume the two authors are located at different geographic locations. Let author A uses logical unit 1 i.e. sky part of the image and the author B uses logical unit 2 i.e. mountain part of the image shown in Fig.2. The contents of LUAM will be

User	Current Unit	Next Requested Unit
A	**1**	*
B	**2**	*

Fig.8. LUAM table

Now if author A wants to access the logical unit from CUAA, LUAM will be checked now. It finds that the requested unit is allocated to author B. So it provides author A with two options, either wait for author B to complete and release the unit by setting the request in the *next requested unit* of *LUAM* or work on some other logical unit available in the CUAA.

After the completion of editing by all the authors i.e. if the *current unit* column is free the layers are merged together to produce a final copy of the image. The authors can access the final image from the image server.

IX. Further Evaluations

The design environment proposed in this paper needs to be evaluated. This can be done by adding the LUAM, CUAA and Mutex Table functionalities to the traditional collaborative image editing tools. This helps in attaining a lucid, logical division of images rather than stochastic division of the images. We intend to modify any of the open source collaborative image editing tools by adding LUAM, CUAA and Mutex Table to test our proposed architecture through simulation. We would then be able to quantify the lucidity and efficiency of our proposed architecture.

X. Conclusion and Future Work

In this paper, we have presented the application of image processing in collaborative world. We have designed a collaborative drawing environment that can be leveraged to obtain performance gains over existing architectures. It extends traditional collaborative environments. We have illustrated the graphical environment with an edge detection algorithm. We strongly feel that edge detection techniques be introduced in such environments. However, further work needs to be done if two authors simultaneously access a logical image unit from *Common User Access Area.* An independent work done at CEG, Anna University by our team has derived a software architecture for collaborative graphical design environment.

References

[1] Andy Adler, John C. Nash, Sylvie Noël, "Challenges in Collaborative Authoring Software".

[2] D. Gordon, Cliki, (2004, Mar.) [Online], http://www.mcs.vuw.ac.nz/ ~donald/?Cliki

[3] Sun and D. Chen, "Consistency maintenance in real-time collaborative graphics editing systems". ACM Trans. Computer-Human Interaction, Vol. 9, pp. 1-41. 2002.

[4] R. Ramloll and J. Mariani, "Do localised auditory cues in group drawing environments matter?", Proc. ICAD'98, Glasgow, UK, 1998.

[5] C.-L. Ignat and M.C. Norrie, "Grouping/ungrouping in graphical collaborative editing systems", Proc. 5th International Workshop on Collaborative Editing Systems, Helsinki, Finland, 2003.

[6] OPEN_Studio, (2004, Mar) [Online]. http://draw.artcontext.net

[7] Poietic Generator [Online] http://poietic-generator.net/wikini/wakka.php?wiki=Welcome

[8] J. F. Canny, "A computational approach to edge detection", IEEE Trans. Pattern Analysis and Machine Intelligence, pages 679-698, 1986.