xScribble: A Generalized Scheme for String-Encoding Graphical Data in Multiuser Graphical Chat

Rahul Anand, Joshy Joseph, P. Dipin Dev, Hegina Alex, and P.C. Rafeeque

Department of Computer Science and Engineering,
Government College of Engineering Kannur,
Kannur, India
rahulanand.fine@gmail.com, rafeeqpc@yahoo.co.in

Abstract. Multiuser graphical chat enables two or more users to communicate user generated graphical data in real time. It is most commonly used in online whiteboards where users can interact simultaneously. In this paper, we introduce xScribble: a generalized scheme for encoding graphical data for real time network communication. The paper discusses how to encode graphical data from various drawing tools into string format flexible enough to be used with any text chat system. The memory efficiency and performance of the xScribble scheme is also analysed.

Keywords: graphical chat, string-encoding, online whiteboard, real-time interaction.

1 Introduction

Usage of graphical aids like diagrams is often necessary to communicate certain concepts in a clear and concise manner. But exchange of graphical data over the Internet is limited by the overhead of available techniques like e-mail or file sharing. This makes Graphical chat an important functionality that is missing in todays instant messaging services. Available closed source online whiteboard applications have implementations apparently dependent on their programming environment [1].

Most of the popular chat clients like Google Talk [2] provide full fledged text chat service together with voice, video chatting and even file sharing. In this scenario, an online whiteboard scheme which can be integrated to the existing system of frequently used clients can be an attractive notion in terms of online communication. Such a scheme has the additional advantage of being able to be used with other available services like video call. This is desirable to the end users in that graphical chat can be quite naturally integrated into their usual choice of chat client. Such a scheme can be implemented by a generalized algorithm which efficiently converts graphical data into a representation compatible with any text chat protocol. Moreover, this scheme realizes graphical chat with minimal additions to the underlying protocol, and optimizes the added memory requirements for communication.

In this paper, we describe a scheme for efficiently encoding graphical data into string representation in real time. We call the complete scheme 'xScribble'. The scheme provides a general algorithm to encode graphics into text representation which can be communicated over a text chat protocol.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 evaluates the basic features of graphical data from various drawing tools. Section 4 describes encoding and decoding of certain popular class of drawing tools. Section 5 presents the general algorithms used for encoding and decoding graphical data. Section 6 and Section 7 respectively discusses notions for extension of the encoding format to account for other drawing tools, and actual results of implementing the scheme. Section 8 concludes the paper with remarks on its future scope.

2 Related Work

Existing online whiteboard systems use web based technology like Java applet and Adobe Flash [3]. The actual protocol used comes under a closed specification. Scriblink [4] is an online whiteboard with real-time collaboration. The site is built using Java. Groupboard [5] hosts multiple whiteboards with multi-user interaction capability. The underlying technology includes Java applet and Ajax. Twiddla [6] is another browser based whiteboard application intended to support web meetings. It provides inbuilt voice support using Java plug-in. iScribble [7] features an online artist collaboration built in Adobe Flash. openCanvas [8] is a desktop image editing software with additional networking feature. A maximum of 3 users can connect to a running instance of the software using TCP connections.

3 Components of Graphical Data

In this section, we analyse the basic information required to represent and/or reproduce graphical data. The paper discusses the data components associated with two main classes of drawing tools - Free hand tools and Fixed point tools - and other tools which are variations of the above classes. In any drawing environment, deciding where and how to produce the graphical output essentially involves retrieving the coordinates of the user input device. Additional parameters may be supplied to the method - most commonly the selected colour, line style, fill style (for closed figures) etc. We analyse the minimal information required to represent major classes of drawing tools.

3.1 Free-Hand Tools

The sequence of line segments connecting all the mouse coordinates during the free hand drawing can form a smooth curve. A mouse drag is thus represented by a connected sequence of line segments called a polyline [9]. Parameters associated with free hand tools can be the stroke thickness and stroke style.

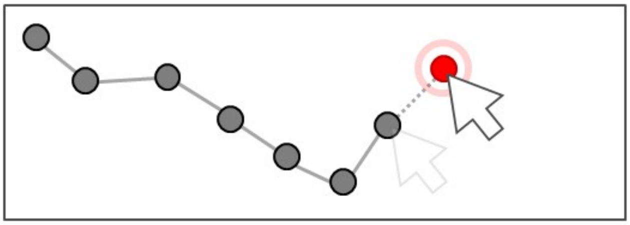

Fig. 1. Free hand drawing starts once the user clicks the mouse inside the drawing area. The point of mouse click is retrieved and stored. As the mouse is dragged from this point, the next mouse coordinate is retrieved. A line segment is drawn from the last stored point to the new point with preferred colour and thickness properties. The new point now becomes the last stored point. The process continues as long as the mouse is dragged. Free hand drawing stops when the mouse release action is detected.

3.2 Fixed-Point Tools

Most common fixed point tools include Rectangle, Oval, and Straight line tool. In these tools, the point of first click of mouse is taken as a fixed point. From here, the user can drag the mouse to decide the dimensions and orientation of the graphic output. When the mouse is released, the final drawing is saved on the drawing board. The user is provided with graphical feedback by painting intermediate shapes during the mouse drag. This gives a sensation of scaling the graphic object by the dragging of the mouse. Anyhow, the final drawing depends only on the point of initial mouse click and the point where the mouse is finally released.

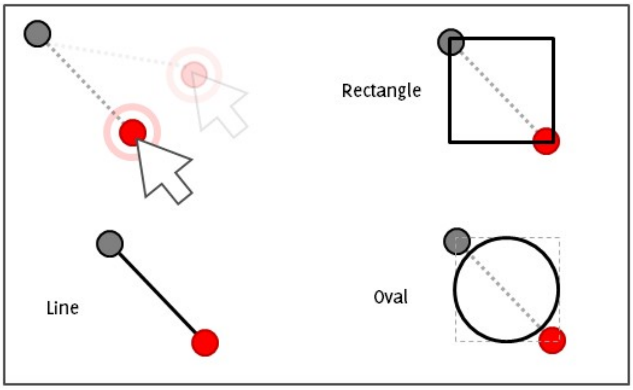

Fig. 2. In Rectangle tool, the points are considered to be forming a diagonal of the rectangle. The required parameters for the drawing method to draw a rectangle are computed from these two points using simple geometry. In Oval tool, the two points again form the diagonal of a rectangle. Here, an ellipse is drawn whose major and minor axes are respectively the width and height of the enclosing rectangle. In Straight line tool, a line segment is drawn between the initial and final points. The closed figures have a parameter determining whether or not they are filled with a selected fill colour.

3.3 Other Tools

This section discusses the component information for three other drawing tools. Most of the drawing tools are variations of the above two categories.

Air Brush. The Air brush tool is similar to a free hand tool in that the user can draw with an air brush by dragging the mouse over the drawing area. But instead of a single stroke, the air brush provides a sensation of spraying the colour over the mouse path within a fixed thickness. The logic behind the Air brush tool is to create a spraying effect at each passing coordinate.

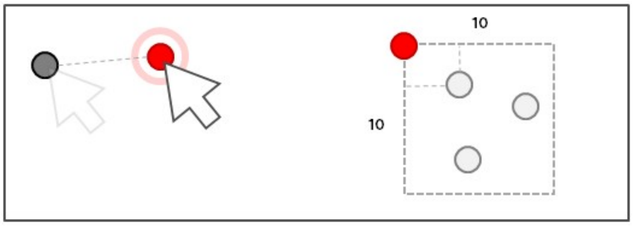

Fig. 3. At each mouse coordinate, a square of dimensions 10 pixel units is assumed adjacent to the mouse coordinate. The algorithm now generate and paint 15 random pixels such that their coordinates fall within this square.

Pseudo code for Air brush

```
Air-brush (x, y, selected-color)
  Repeat 15 times:
    Set rx to an random integer between 0 and 10
    Set ry to an random integer between 0 and 10
    Draw a node of selected-color at (x+rx, y+ry)
```

Flood Fill. The Bucket fill tool uses the flood fill algorithm [10] to fill closed areas in the drawing. It first determines the colour at the pixel where the user clicks. This becomes the target colour. The selected colour is filled in this target pixel. Now the colouring spreads to the top, bottom, left and right pixels of the target pixel. These pixels are painted if their colour is same as the target colour. In case any new pixels are painted, the colouring spreads from them again. Thus, all the pixels in the region formed by the target colour is filled with the selected colour.

Pseudo code for Flood fill

```
Flood-fill (x, y, replacement-color):
  Set Q to the empty queue
  Set node to Node(x,y)
  Set target-color to colour of node
  Add node to the end of Q.
  While Q is not empty:
    Remove first element from Q, let it be n
    If the color of n is equal to target-color
      Set the color of n to replacement-color
      Add the node to the west of n to the end of Q
      Add the node to the east of n to the end of Q
      Add the node to the north of n to the end of Q
      Add the node to the south of n to the end of Q
```

Text Tool. The Text tool writes reads a string of text from the user and paint it on the drawing board at specified coordinates [11]. Clicking the mouse on the drawing board allows the user to input the desired string at this location. The entered string is now painted over the canvas in the required colour.

4 Encoding the Graphical Data

At the heart of xScribble scheme is the technique of encoding the variety of graphical data into individual strings, which can be efficiently decoded at the receiver side. The string should encode the kind of tool used for drawing, its properties and the set of associated mouse coordinates. In order to represent the data in the most optimized format, the scheme uses a comma separated list of decimal integers. Since the data is essentially a text string it can be communicated with HTTP in the web or any other chat protocols like XMPP [12].

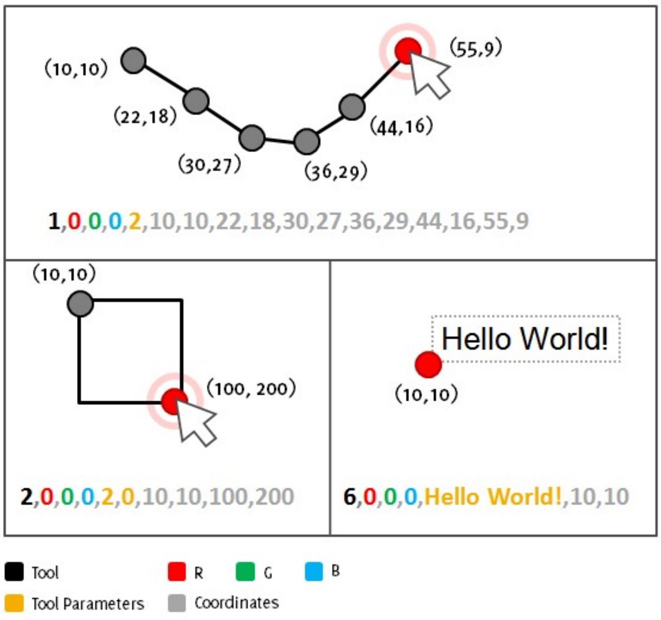

Fig. 4. The integer components in the string are separated by commas. The first component identifies the tool used for creating the graphics. The tool specific parameters follow this. The colour is represented in the form of its Red, Blue, and Green (RGB) components, each which is encoded as integers between 0 and 255 [13]. Other parameters will follow these values. Here colour is only a tool-specific parameter and may be omitted for tools or drawing commands which doesn't require colour information. Finally, the required coordinates which defines the graphic are appended. These integers are processed in pairs as they are alternatively the x and y coordinates of each point. Again, the number of coordinates can vary from a single point to a series of points according to the type of the tool used.

4.1 Free-Hand Tools

Most of the common free hand tools can be represented under a single category and thereby by a single integer in our encoding scheme. As discussed, the parameters for a free hand tool other than colour are line thickness and line style (dotted, dashed, solid) [14]. These class of tools require the entire series of points through which the user has performed the free hand stroke. Thus all the coordinates from the beginning to the the end of drawing are encoded into the string.

4.2 Fixed-Point Tools

The most common fixed point tools in popular drawing applications are Rectangle, Oval and Straight line. Although the technique of producing the graphics differs for each of these tools, they require only a pair of coordinates to represent the position and size of the resulting graphic. These are respectively the fixed point and the final position of the dynamic point where the user stops drawing. The parameters for these tools are also common, which generally include the colour, outline thickness and the fill style for Rectangle and Oval. Although the encoding scheme is structurally similar we should account for the fact that the coordinates are processed in a different way for each of the tools.

4.3 Other Tools

Air Brush. According to the discussed drawing logic of Air brush, a large number of pixels are painted on the canvas as the user draw with the air brush tool in free hand style. Here we may strike a compromise between data size and accuracy of data reproduction by encoding only the coordinates through which the user moves the mouse. At the receiver end, random pixels are generated around each of these path coordinates. Actual test cases show that the random pixels restricted to a small area and in a dense fashion show only negligible differences between the graphic outputs at the two ends.

Flood Fill. Flood fill tool demands only minimal information to produce its graphical output. These information are namely the selected fill colour and the coordinates from where the flood fill procedure is to commence. Major part in producing the graphical output is done by the flood fill algorithm. However the performance of flood fill is critical in the dynamic environment of graphical chat since other graphical data may be added to the canvas before the flood fill is completed.

Text Tool. Text entry tool is a special case in string encoding of graphical data in that this tool uses a character set different from other tools in the encoding. Encoded data of text tool consists of the actual text string entered by the user and the coordinates showing the relative position where the text is to be painted. The character set may even include a wide variety of Unicode symbols if user input in native languages is to be provided.

Drawing Commands. The peculiarity of drawing commands, such as Clear Canvas, is that they do not usually need any graphical information such as colour or any coordinates. When such commands are encoded, basically they only contain the first component integer, which identifies the command itself. Even in an environment where a larger set of commands are used, the same unified encoding scheme can be used.

5 Encoding and Decoding Algorithms

This section provides the actual algorithms for encoding graphical data to text string and for decoding a text string to reproduce original graphics respectively. The algorithms account for the above mentioned categories of drawing tools. The encoding algorithm is not sequentially run for a stream of input data. Rather it spans the entire drawing process from when a user starts drawing on the canvas to when he stops drawing. The encoding algorithm consider drawing in a general context. It may be noted that the ongoing state of drawing is represented by the continuous dragging of the mouse pointer in a classic interface. Most of the other common tools can be accounted for in the algorithm as special cases of existing tools or with minimal additions.

Pseudo code for String encoding

```
if drawing starts:
  Set encode-string to the empty string
  Append current-tool, colour R,G,B values, tool
   parameters, x-coordinate and y-coordinate of current
   tool location to encode-string separated by commas

while drawing:
  if current-tool is a free hand tool
    Append x-coordinate and y-coordinate of current tool
     location to encode-string separated by commas

if drawing ends:
  if current-tool is not bucket-fill or text-input:
    Append x-coordinate and y-coordinate of current tool
     location to encode-string separated by commas
  else if current-tool is text-input:
    Append entered text to encode-string
  Send encode-string to the network
```

Pseudo code for String decoding

```
String-Decode (decode-string)
  Split decode-string at commas and store the elements
   in decode-array

  Set current-tool to the first element of decode-array

  Set R,G,B to the next three elements of decode array
  Set selected-color to Color(R,G,B)

  for each element corresponding to tool parameters:
    Set variables corresponding to the parameter values

  Set x0, y0 to the next two elements in decode array
```

```
if current-tool is a free hand tool:
  while the end of decode-array is not reached:
    Set x1, y1 to the next two elements in decode array
    Call appropriate free hand tool method with
     parameters x0, y0, x1, y1, selected-color
     and tool parameters
    Set x0 = x1, y0 = y1

if current-tool is a fixed point tool:
  Set x1, y1 to the next two elements in decode array
  Call appropriate fixed point tool method with
   parameters x0, y0, x1, y1, selected-color
   and tool parameters

if current-tool is bucket-fill:
  Call Flood-Fill(x0, y0, selected-color)

if current-tool is text-input:
  Set user-text to the last tool parameter
  Call Text-input(x0, y0, user-text, selected-color)
```

6 Implementation Details

xScribble scheme is designed to be implemented in any networking environment which supports sending and receiving text data. While integrating xScribble scheme in existing chat servers there is virtually no property which distinguishes graphical chat data from normal text chat. The encoded graphical data might be prepended with a special header string for this purpose. The application can recognize a graphical chat string when it detects the header string at its beginning.

The design of the encoding format supports efficient compression. For instance, the most commonly occurring character in the encoded string is the comma because it is used as a separator for other components. If Huffman Coding is used for compression, a comma may be represented in a single bit. Also the most frequent character set includes integers from 0 to 9. These characters can also be mapped to bit strings of size not exceeding 3 bits. Thus the size of the encoded data is considerably reduced with the minimal character set [15]. This can be a great advantage when we need to keep track of the drawing board state at a centralized server.

The method of producing the actual graphics on the canvas is implementation specific. The graphical data may be used to construct static graphics or it can be used to create separate graphical objects. Graphical objects, such as in the case of vector image editors like Inkscape [16], allow more flexibility in editing the drawing board. In this case additional information regarding the identifier assigned to the graphical object need to be encoded in the communicated data.

7 Experimental Results

We implemented an integration of the encoding scheme with Google Talk in xScribble instant messaging application [17]. Here the xScribble scheme was implemented with minimal drawing tools using Python over the XMPP protocol

[18]. Users could perform graphical chat using their Google account. The graphical data could be communicated with the same efficiency as in the case of text data. The group chat feature in Google Talk can be utilized to enable any number of users to simultaneously draw on a shared canvas. The size of graphical data was limited only by the inbuilt quota on amount of text data passing through the server. In all the cases, the size of graphics in transit is highly optimized by the encoding scheme.

We have also implemented the xScribble scheme using Java applets at client side and a server running under Tomcat. Text chat functionality was implemented in the server using HTTP messages. The scheme was tested in the Internet at connection speeds of 100 to 200 KBps. The average size of the encoded data from above mentioned drawing tools is 50 to 100 bytes, which is comparable to usual text chat data. The scheme employs an additional overhead at the client ends alone, where an extra processing is done for encoding and decoding purposes. This encoding of graphics in real-time and decoding of received messages could be performed without any noticeable delay. Simultaneous drawing over the network was tested with more than 5 users. This number can be conveniently as high as 20 or more; the scalability of the system (determined by the number of possible simultaneous users) is limited only by the capability of the server. While a basic server can have limitations regarding the maximum number of users using a particular drawing board, integration with fully developed server systems poses no perceivable delay from the side of the string encoding scheme.

8 Conclusion

Our aim in this research was to develop a scheme to efficiently communicate user generated graphics over a multi-user network in real-time. Initial studies found that existing systems have closed and implementation specific communication protocols which cannot be integrated with other server systems. We developed the xScribble scheme which provides a generalized and optimized encoding scheme for graphics. The efficient encoding of graphics to text strings with minimal character set enabled the integration of the scheme into any existing text chat server. Thus the scheme virtually enables graphical chat to be implemented as a feature in all instant messengers. The scheme may be extended further to account for vector graphic objects providing more control over the canvas. Additional exclusive locking features may be included in the encoding when users are allowed to alter the graphic objects on the canvas. We hope our work will contribute to more productive and expressive network communication.

References

1. Oekaki Central, `http://www.oekakicentral.com/fp/fp1/index2.php`
2. Getting started with Google Talk, `http://www.google.com/talk/start.html`
3. Paint Chat, `http://en.wikipedia.org/wiki/Paint_Chat`
4. Scriblink online whiteboard, `http://www.scriblink.com/index.jsp?act=about`

5. Groupboard, http://www.groupboard.com/products/
6. Twiddla, http://www.twiddla.com/
7. iScribble, http://www.iscribble.net/
8. OpenCanvas image editor, http://en.wikipedia.org/wiki/OpenCanvas
9. Foley, J.D.: Computer graphics: principles and practice, 2nd edn., p. 27. Addison-Wesley Publishing Company, Inc. (1996)
10. Dav Data. A non recursive Flood fill algorithm, http://www.davdata.nl/math/floodfill.html
11. Klawonn, F.: Introduction to Computer Graphics: Using Java 2D and 3D, p. 96. Springer, London (2008)
12. The Extensible Messaging and Presence Protocol (XMPP). RFCs, http://xmpp.org/xmpp-protocols/rfcs/
13. Prez-Quiones, M.A.: Media Computation CS2984-S07 (January 2007), http://happy.cs.vt.edu/~manuel/courses/cs2984/slides/05Colors.html
14. Klawonn, F.: Introduction to Computer Graphics: Using Java 2D and 3D, p. 63. Springer, London (2008)
15. Astrachan, O.L.: From ASCII Coding to Huffman Coding (February 2004), http://www.cs.duke.edu/csed/poop/huff/info/
16. Inkscape vector editor, http://wiki.inkscape.org/wiki/index.php/Inkscape
17. Anand, R., et al.: xScribble instant messaging client for Google Talk, http://code.google.com/p/xscribble/
18. Package xmpp API documentation, http://xmpppy.sourceforge.net/apidocs/index.html