Optimal Linear Interpolation Coding for Server-based Computing

Fei Li and Jason Nieh
Department of Computer Science
Columbia University
New York, NY, 10027, U.S.A.
{fl200, nieh}@cs.columbia.edu

Abstract— **Server-based computing (SBC) is becoming a popular approach to deliver computational services across the network due to its reduced administrative costs and better resource utilization. In SBC, all application processing is done on servers while only screen updates are sent to clients. While many SBC encoding techniques have been explored for transmitting these screen updates efficiently, existing screen update approaches do not effectively support multimedia applications. To address this problem, we propose optimal linear interpolation (OLI), a new pixel-based SBC screen update coding algorithm. With OLI, the server selects and transmits only a small sample of pixels to represent a screen update. The client recovers the complete screen update from these samples using piecewise linear interpolation to achieve the best visual quality. OLI can be used to provide lossless or lossy compression to adaptively trade-off between network bandwidth and processing time requirements. We further propose and evaluate 2-D lossless linear interpolation (2DLI), which is based on OLI but additionally provides lower encoding complexity for lossless compression. Our experimental results show that when compared with other compression methods, 2DLI provides good data compression ratio with modest computational overhead, for both servers and clients.**

I. Introduction

In recent years, there is a growing trend away from the distributed model of desktop computing toward a more centralized server-based computing (SBC) model. In SBC, all application processing is carried out by a set of shared server machines. Clients connect to the servers for all their computing needs. Since SBC servers maintain the full persistent state of user sessions, the only functionality required for the client is to be able to send keyboard and mouse input to the server and receive graphical display updates from the server. By employing a more centralized computing model, SBC offers the potential of reducing total cost of computational services through reduced system management cost and better utilization of shared hardware resources. SBC is being deployed to deliver computational services in a wide range of environments, from LAN-based work group environments to Internet ASPs [4] [5] [8] [9] [10].

The key enabling technology underlying the SBC approach is the remote display protocol, which enables graphical displays to be served across a network to a client device while applications and even window systems are executed at the server side. Because sending the display information as raw pixels alone would be impractical due to network bandwidth limitations, the display information is sent as encoded screen updates. A number of SBC encoding techniques have been developed, ranging from using higher-level graphics primitives to using lower-level pixel-based compression techniques. However, existing SBC encoding techniques have been shown to not be effective in supporting the display demands of multimedia applications [6] [7].

To improve support for multimedia applications in SBC environments, we have developed new encoding algorithms that can provide screen updates with less data. In this paper, we focus on an optimal linear interpolation (OLI) algorithm. The underlying idea in OLI is to treat screen updates as linear pixel arrays and to take pixel values at the server side as sample values of a curve. A re-sampling of this curve is performed to capture as much essential information on the original curve as possible. Pixel values chosen from this re-sampling curve are transmitted from the server to the client to represent screen updates. The client then employs piecewise linear interpolation to recover the curve. The sampling rate can be varied to provide lossless or lossy compression to trade-off different resource constraints, such as network bandwidth and processing time.

In particular, we also present a 2-D lossless linear interpolation (2DLI) algorithm based on OLI that provides good lossless compression of screen updates with low coding complexity. We have implemented 2DLI and compared its performance against other screen update encoding techniques on various display workloads, including smooth-toned images, web pages, desktop screen dumps, and instructional video content. Our experimental results show that when compared with other compression methods, 2DLI provides good data compression ratio with modest computational overhead, for both servers and clients.

This paper describes both OLI and 2DLI and is organized as follows. Section II describes related work on SBC coding techniques. Section III discusses SBC coding requirements and formulates the SBC coding problem in further detail. Section IV presents the OLI algorithm. Section V presents the 2DLI algorithm based on OLI. Section VI shows the results of our experimental measurements comparing 2DLI against other encoding algorithms for different display workloads. Finally, we present some concluding remarks.

II. Related Work

Previous approaches in encoding screen updates can be loosely classified into two categories, graphics-based and pixel-based. Graphics-based approaches employ a variety of higher-level graphics primitives for representing screen updates in terms of fonts, lines, bitmaps, etc. These approaches are used in systems such as X, Windows Terminal Services [5], Citrix MetaFrame [4], and Tarantella [9]. Despite the range of available encoding primitives, the screen updates associated with multimedia applications such as images and video are typically encoded as raw pixel bitmaps. In some cases, an additional run-length encoding step is applied. Because run-length encoding does not perform well on multimedia display workloads, little compression is achieved on these screen updates. Furthermore, the higher-level graphics-based encoding primitives require more client complexity, potentially resulting in higher client decoding time.

Pixel-based approaches are simpler and treat a screen update as just a region of pixels. These approaches are used in systems such as Sun Ray [10] and Virtual Network Computing (VNC) [8] [11]. These approaches do not employ higher-level primitives such as fonts and lines,

thereby avoiding the client complexity associated with graphics-based approaches. However, they employ similar bitmap encoding primitives for multimedia display workloads, which limits the achievable compression on screen updates for such applications.

More recently, some SBC pixel-based encoding techniques have been developed that take advantage of the common image characteristics of screen updates. A new mixed method called *PWC* was proposed in [1]. Based on characteristics of color of a pixel is statistically related to its surrounding colors for palette images, a piecewise-constant image model was given to describe boundaries between domains of palette images. Another method that has been proposed separates background from foreground and uses *PWC* to encode the marks in foreground as a codebook [2] [3]. Both of these methods achieve improved compression ratios, but at the cost of higher encoding and decoding complexity. These coding costs limit the utility of these techniques for supporting multimedia applications in SBC environments.

III. Problem Formulation

A. Coding Requirements

We summarize the coding requirements for SBC model as follows:

1. *Low complexity*: As encoding and decoding processing must occur on frequent screen updates, it is important for a coding algorithm to have low enough coding complexity to operate in real-time on current hardware. For example, to guarantee showing video with size of 352×240 at the rate of 25 frames/second, more than 2M true color pixels should be processed per second.
2. *Good compression*: Given the limited availability of bandwidth in current broadband environments, it is important for a coding algorithm to provide effective compression of screen updates for multimedia applications. For example, existing SBC encoding techniques begin to perform poorly in deliver 320×240 resolution multimedia video when available bandwidth drops below 10 Mbps [6].
3. *Universal suitability*: Given that a wide range of applications may be used in an SBC environment, it is desirable for a coding algorithm to provide good performance across different display workloads, including both smooth-toned and discrete-toned images.

B. Problem Formulation

In SBC, a region to be updated at the client side is represented by a sequence of rectangles of pixels. Each pixel is typically represented by 8-bit or 24-bit data for RGB color values as supported and agreed to by both server and clients. We can regard a rectangle of size $w \times h$ as a 2-dimensional plane. The x-coordinates range from 0 to w (h), and are made up by the cardinal number of pixels in the rectangle traversed left-right (top-down) for any pixel. The y-coordinates represent the R, G, or B value of the corresponding pixel. Hence, R/G/B values for the rectangle can be treated as samples from 3 individual curves in this 2-dimension space. Fig. 1 shows the B values of the first $1,000$ pixels from a typical update rectangle of 752×420 pixels using the xy-coordinate system as described above.

As we can see from Fig. 1, standard desktop update rectangles are very amenable to piece-wise linear interpolation. From the 2D space representation of an update rectangle, a K-segment piece-wise linear interpolation function $f(x)$ as given in Equation 1 can be used to capture the update rectangle:

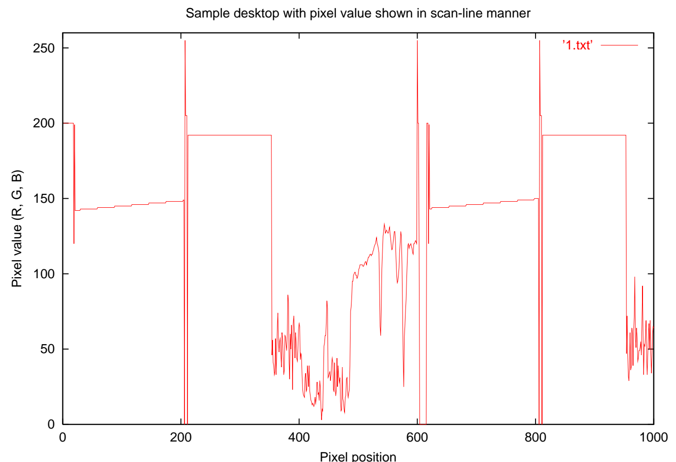

Fig. 1. Sample desktop with blue value shown in scan-line manner

$$f(x) = \sum_{i=1}^{K} \psi_i(a_i x + b_i) \quad (1)$$

where a_i and b_i are *slope* and *y-intercept* for the ith segment respectively, and ψ_i is the *sign function* defined as:

$$\psi_i = \begin{cases} 0; & \text{if } x_i \le x < x_{i+1} \\ 1; & \text{else.} \end{cases}$$

While pixel values can be linear interpolated horizontally or vertically, we consider the horizontal case to formalize the problem; vertical interpolating pixel values can be done in the same way. With minimal overhead, the client can use $f(x)$ to reconstruct the update region. The reconstructed curve is defined with K (value, position) pairs of pixels transmitted from the server, where the value of K depends on the bandwidth available. If we use set S to denote the data set from server to client, for a K segment piece-wise curve, we have:

$$S = \bigcup_{i=1}^{K} \{v_i, p_i\} \quad (2)$$

where v_i is pixel value and p_i is the relative offset for the v_i. Our goal is to minimize the size of S while maintaining the same visual quality or maximize the visual quality under the same bandwidth limitation.

Desktop applications in server-based computing are different from multimedia-only applications, as they tend to have more console-like applications. Exact values for images of console applications are more preferred than interpolation with approximate value. Because we use a piecewise curve to reconstruct the pixel values, a curve covering as many original pixels as possible is preferred. Therefore, metrics such as *least square* cannot be used to govern the selection criteria for interpolation methods in that they do not reflect any resulting client image quality with respect to non-multimedia applications.

Problem 1: Given rectangle R of dimensions $w \times h$ with original pixel value $p_i, (0 \le i < w \times h)$, an interpolation method resulting in K segments will have reconstructed pixel value c_i. Suppose S contains the (value, position) pairs to represent a K-segment piece-wise linear interpolation function, then there will be $C^K_{w \times h}$ many of S.

In each set S, for each pixel value, p_i, we have a sign function:

$$s_i = \begin{cases} 1; \text{ if } |p_i - c_i| \le T \\ 0; \text{ else.} \end{cases}$$

where T is the threshold to guarantee that satisfactory image quality is reconstructed by the client. $\|S\|$ is used to denote the number of pixel values reconstructed under threshold T:

$$\|S\| = \sum_{i=1}^{w \times h} s_i \tag{3}$$

Now we have formulated the problem into choosing S with the maximum $\|S\|$ as the re-sampled pixel set to be transmitted to the client.

IV. Optimal Linear Interpolation Algorithm – OLI

We illustrate OLI algorithm according to the problem formulation to reflect the idea of optimal interpolation. For a given bandwidth available B, OLI finds an optimal piece-wise linear interpolation function of S with $K \propto B$ pixels in that $\|S\|$ is maximal.

A. Algorithm

OLI recursively iterates through all combinations of pixels in search of an optimal set for piece-wise linear interpolation. For each iteration with k pixels to be chosen from remaining pixels, if the jth pixel is chosen to be in the resulting S, $k-1$ pixels are to be chosen from the $(j+1)^{th}$ pixel to the $(w \times h - 1)^{th}$ pixel. We use $v[0, \cdots, w \times h - 1]$ to contain the pixel values in scan-line manner.

Algorithm 1: OLI($begin$, end, K)

1: **if** $K = 1$ **then**
2: **for** any pixel pair (i, j) between $begin$ and end **do**
3: $slope = \frac{v[j]-v[i]}{j-i}$;
4: $intercept = \frac{v[i] \times j - v[j] \times i}{j-i}$;
5: calculate $\|S_{i,j}\|$;
6: **end for**
7: $\|S\| = \max(\|S_{i,j}\|)$;
8: /* a, b are subscripts $S_{i,j}$ chosen as $\|S\|$ */
9: Put a and b into S;
10: return(S);
11: **else**
12: **for** $begin \le i \le end$ **do**
13: S^i = OLI($begin$, i, 1);
14: **end for**
15: $\|S\| = \max(\|S^i\| + OLI(i, end, K-1))$;
16: $S = S^i \bigcup OLI(i, end, K-1)$;
17: return(S);
18: **end if**

B. Analysis

The OLI algorithm computes the optimal pixel set under bandwidth limitation B. Although ways can be devised to improve OLI, it is still too expensive to be useful because of its intrinsic computational complexity. If we have A pixel values ($A = w \times h$) and denote computational complexity $F(A, K)$ under the bandwidth limitation K, we have:

$$F(A, K) = \sum_{m=2}^{A} (m \times C_m^2 + F(A-m, K-1)). \tag{4}$$

Therefore, we know OLI has an exponential computational complexity on encoding, which makes it inappropriate for SBC applications. The exponential complexity is due to the selecting of re-sampled pixels under threshold. This may result in a linear algorithm if the threshold is 0, which is a special case for lossless compression. Based on this idea, we give a 2-D lossless solution for local interpolation in the following section.

V. 2-D Lossless Linear Interpolation Algorithm – 2DLI

To be practical for SBC applications, an algorithm to select pixel values for interpolation usage by the client should have linear running time with respect to the number of pixels in the screen update rectangle. In this section, we propose a greedy algorithm, which does 2D interpolation (2D lossless linear interpolation – 2DLI). It tries to minimize the number of pixels that can not be interpolated by gradients with a constant δ through its neighboring pixels in the rectangular region. Those pixel values that cannot be interpolated are delivered to the clients for recovery of the rectangular region through 2-dimensional linear interpolation. This solution give a low-complexity encoding algorithm for SBC.

A. Algorithm

Given a rectangle of dimensions $w \times h$ with original pixel value $p_i, (0 \le i < w \times h)$, an interpolation method results in S 3-element pair $(p_i, \delta_x, \delta_y)$. For those pixels with value of p_i, we call them *isolated pixel*s. Pixel values that can be horizontally interpolated or vertically interpolated by going from p_i are called *dependent pixel*s. For isolated pixel p_i, its dependent pixel values can be calculated through,

$$p_j = p_i + \delta_x \times (x_j - x_i) + b \tag{5}$$

and

$$p_k = p_i + \delta_y \times (y_k - y_i) + c \tag{6}$$

where b and c are horizontal and vertical intercepts respectively, p_j and p_k are pixel values recovered horizontally and vertically, $1 \le j \le w$ and $1 \le k \le h$.

The algorithm goes as follows. Beginning from an image with all pixel direction value of I (isolated), we find as many as dependent pixel values that can be recovered through its δ_x and δ_y. After denoting $direction$ values for those dependent pixel values as V (vertically interpolated) or H (horizontally interpolated), we search the next I pixel and find its dependent pixel values until all the pixel are traversed. 2DLI results in a minimal *isolated* pixel values set to be delivered to clients. Hence, we have:

Algorithm 2: 2DLI($pixel$, $index$, $pair$)

1: For each R, G, B value, do
2: **for** i in (1, $height$) **do**
3: **for** j in (1, $width$) **do**
4: **if** $index[i][j] = I$ **then**
5: $\delta_x = pixel[i][j+1] - pixel[i][j]$;
6: $\delta_y = pixel[i+1][j] - pixel[i][j]$;
7: **for** k in $(j+1, width)$ **do**
8: $A = pixel[i][k] - pixel[i][j] - (k-j) \times \delta_x$

```
9:            if A = 0 then
10:              Mark index[i][k] = V;
11:           end if
12:         end for
13:         for l in (i + 1, height) do
14:           B = pixel[l][j] − pixel[i][j] − (l − i) × δ_y
15:           if B = 0 then
16:             Mark index[l][j] = H;
17:           end if
18:         end for
19:       end if
20:     end for
21: end for
22: for i in (0, height) and j in (0, width) do
23:   if index[i][j] = I then
24:     S = S ⋃ (pixel[i][j], δ_x, δ_y)
25:   end if
26: end for
27: Using universal data compression for index and S;
```

B. Analysis

2DLI algorithm has a running complexity of $O(n)$ for both encoding and decoding, where n is the number of pixels. Also, it could be lossy once the difference between the interpolated pixel value and original image pixel value is under some threshold. We employ $index$ because this extracted information could provide more information on spatial image transformation.

VI. Experiments

As an initial step in evaluating the effectiveness of linear interpolation algorithms for SBC, we implemented 2DLI and compared its performance with several popular coding methods on various display workloads. The coding methods we used for comparison with 2DLI were JPEG [13] in lossless compression mode, Lempel-Ziv coding [12] as implemented in Gzip [14], and the hextile coding method used in VNC [11]. We compared these methods on four different types of display workloads: 11 smooth-toned images from [15], 5 web pages with different combinations of image and text content, 7 desktop screen dumps from [16], and content from an instructional video. Figures 2 to 5 show example display content from each of the four respective workloads.

For each coding method, we measured the compression ratios achieved for each class of display workload versus using raw pixels to represent the respective display content. For each coding method, we also measured the encoding and decoding performance for each class of display workload. These measurements were performed on an IBM NetVista PC with a 1 GHz AMD Athlon CPU and 256 MB RAM, running RedHat Linux 7.1. Due to space limitations, we only report averages of our measurements here. Table I shows the average compression ratio for each coding method and Table II shows the encoding and decoding performance for each coding method as measured in the average number of pixels processed per second.

Our results show that 2DLI algorithm performs very well across different display workloads that appear in multimedia SBC environments. Table I shows that 2DLI provides better lossless display compression than any of the other approaches for discrete-toned images such as web pages, desktop screen dumps, and instructional video content. For smooth-toned images, 2DLI performs only second to JPEG. Although JPEG has slightly better compression performance on smooth-toned images, Table II shows that JPEG takes almost three times as much processing time to encoding the display data as 2DLI. Overall, 2DLI encodes on average 2.41 times faster than JPEG and decodes on average 1.75 times faster than JPEG. While 2DLI is not as fast as Gzip and VNC hextile, it provides superior compression than both techniques across all four different display workloads.

VII. Conclusion

In this paper, we have developed a family of linear interpolation algorithms for encoding SBC screen updates. We developed an OLI algorithm and 2DLI with linear encoding and decoding computational complexity. These algorithms represent a region of pixels as a piecewise linear function of a small number of values, and can be used to provide lossless or lossy compression. We have implemented our linear interpolation algorithm and compared its performance with other approaches on discrete-toned and smoothed-toned images. Our results show that 2DLI provides much better compression than JPEG, gzip or VNC on web pages, screen dumps, and instructional videos, and performs second only to JPEG on smooth-toned images, but with much lower coding time. This combination of low coding complexity and good compression performance across a wide range of images makes the linear interpolation method a viable universal technique for effective encoding of SBC screen updates for multimedia applications.

References

[1] P. J. Ausbeck, *Context Models for Palette Images* Proceedings of Data Compression Conference, March 1998.

[2] B. O. Christiansen, K. E. Schauser and M. Munke, *A Novel Codec for Thin-client Computing* Proceedings of Data Compression Conference, March 2000.

[3] B. O. Christiansen, K. E. Schauser and M. Munke, *Streaming Thin Client Compression* Proceedings of Data Compression Conference, March 2001.

[4] Citrix Systems, *Citrix MetaFrame 1.8 Backgrounder* Citrix While Paper, June 1998.

[5] B. C. Cumberland and G. Carius, *Microsoft Windows NT Server 4.0, Terminal Server Edition: Technical Reference* Microsoft Press, Redmond, WA, August 1999.

[6] J. Nieh and S. J. Yang, *Measuring the Multimedia Performance of Server-Based Computing* Proceedings of the Tenth International Workshop on Network and Operating System Support for Digital Audio and Video, Chapel Hill, NC, June 2000.

[7] J. Nieh, S. J. Yang and N. Novik, *A Comparison of Thin-Client Computing Architecture* Technical Report CUCS-022-00, Network Computing Laboratory, Columbia University, November 2000.

[8] T. Richardson, Q. Stafford-Fraser, K. R. Wood and A. Hopper, *Virtual Network Computing* IEEE Internet Computing, Vol. 2, No. 1, January/February 1998.

[9] The Santa Cruz Operation, *Tarantella Web-Enabling Software: The Adaptive Internet Protocol* A SCO Technical While Paper, December 1998.

[10] Sun Ray 1 Enterprise Appliance, *Sun Microsystems* http://www.sun.com/products/sunray1

[11] T. Richardson and K. R. Wood, *The RFB Protocol* ORL, Cambridge, January 1998.

[12] J. Ziv and A. Lempel, *Compression of Individual Sequences Via Variable-Rate Coding* IEEE Transactions on Information Theory, Vol. 24, pp. 530 - 536, 1978.

[13] Lossless JPEG Software, *ftp.cs.cornell.edu in /pub/multimed/ljpg.tar.Z*

[14] Gzip Home Page, *http://www.gzip.org/*

[15] Standard Test Images, *http://www.geocities.com/SiliconValley/Lakes/6686/test-images/*

[16] VNC Screenshots, *http://www.uk.research.att.com/vnc/screenshots.html*

Fig. 2. Smooth-toned image

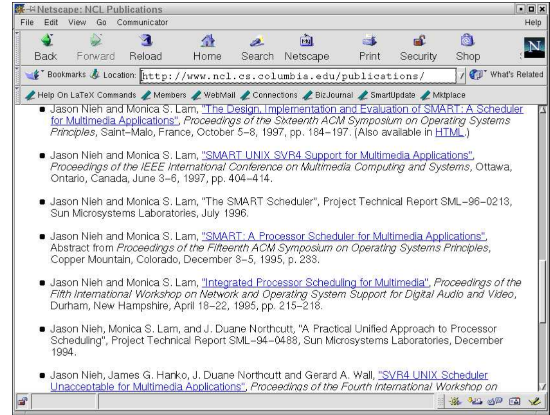

Fig. 3. Web page

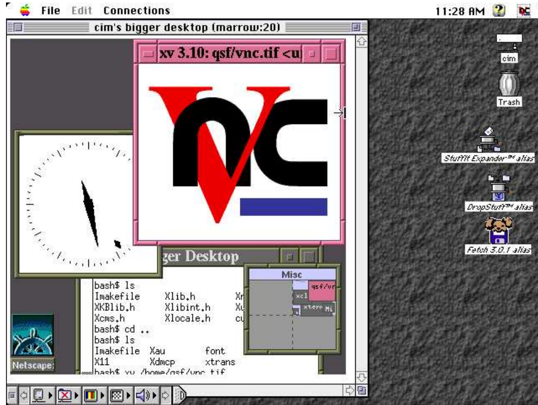

Fig. 4. Desktop screen dump

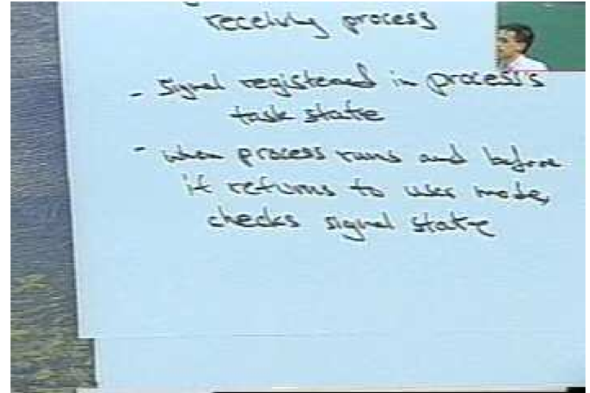

Fig. 5. Instructional video

TABLE I

COMPARISON ON AVERAGE COMPRESSION RATIO AMONG 2DLI, JPEG, GZIP AND HEXTILE

Compression Ratio vs. Images	Smooth-toned	Web page	Screen dump	Instructional video
JPEG	1.67	2.49	1.58	1.61
Gzip	1.16	16.38	19.32	1.14
Hextile	1.08	6.14	17.93	1.00
2DLI	1.40	20.04	23.14	2.15
2DLI Compression Ratio Range	1.04 - 2.70	9.72 - 33.32	7.20 - 65.73	2.15

TABLE II

COMPARISON ON CODING COMPLEXITY ($Mpixels/sec$) AMONG 2DLI, JPEG, GZIP AND HEXTILE

Coding	Time vs. Images	Smooth-toned	Web page	Screen dump	Instructional video
Encoding	JPEG	1.00	1.16	1.03	1.05
	Gzip	8.94	9.72	8.99	8.64
	Hextile	6.12	7.74	9.57	5.31
	2DLI	2.77	1.98	1.96	3.52
Decoding	JPEG	1.91	1.96	2.00	1.96
	Gunzip	9.04	12.75	12.28	9.34
	Hextile	5.12	9.77	11.12	11.55
	2DLI	2.52	4.31	3.09	3.80