Efficient Image Transfer Rate Control and Compression Using Pyramidal Layer Format

Andrei Tigora, Mihai Zaharescu

***Abstract*—**The pyramidal layer format's main purpose is to store images in an efficient manner for both transfer rate control and compression purposes. It involves lossless compression and is elegantly designed to ensure a proper balance between the user experience and the bandwidth constraints, as well as between texture information and edge quality.

***Keywords*—** frequency separation, image compression, PLF image file format, progressive transfer.

I. Introduction

OVER the last decade wired networks have become extremely advanced, being able to satisfy the needs of even the most demanding clients. However, with the surge of smart-phones, tablets and other mobile devices, the attention has shifted towards wireless networks. People expect to be able to read the news or check their social media account any time and any place. Yet, coverage is not uniform and available bandwidth varies greatly, even with the high-end services.

Delivering all the content over this relatively unreliable medium can be challenging, especially when talking about multimedia. Users may accept that streaming videos to handheld devices is under certain circumstances unfeasible, but they probably would not have the same attitude towards viewing images. Still, the latter can be as demanding as the former, especially in the case of social media sites where image sharing is the central activity of the users.

Fortunately, given that most mobile devices have relatively small screens, it is usually impossible for a user to see the details contained in images; therefore, initially it would suffice to transfer to the client only a reduced version of the images. Later, if the user requires it, details can be obtained. However, the browser should not request a whole new image; instead it should build upon the information it already has to limit the bandwidth consumption as much as possible.

The pyramidal layer format aims to solve the problems described above. By repeatedly down-sampling and up-sampling a bitmap image and compressing the resulting residual images a relatively compact image format can be obtained from which increasingly more detailed images can be computed.

The rest of the paper is structured as follows. Section 2 is dedicated to previous work. The next section covers the Pyramidal Layer Format construction algorithm and the image format. Section 4 describes the results and performance evaluation, whereas the last sections are reserved for future work and conclusions.

II. Background

The pyramidal layer format is built upon research performed in the field of image down-sampling and pyramidal representations [1]-[5]. In many instances it is required to obtain reduced versions of a certain image, either for technical or aesthetic reasons. Deciding how to select or combine pixels of the original image to obtain a version as accurate as possible is in itself a challenging task. But restoring the down-sampled image to its original size is where the things really become interesting. For the pyramidal layer format, the reconstruction is done with the help of residual layers that preserve the difference between the original layer and the artificially up-sampled version of the reduced layer.

Modern image data formats include some form of data compression for a better management of the relatively large data volumes. This may be done in a lossless manner, as it is the case of the PNG format, or lossy, like in JPEG [6]. No matter what the case is, they all take advantage of the redundancy displayed in most images. The compression techniques are also diverse, such as LZV in the case of PNG or TIFF and Huffman for JPEG. The PLF also employs compression to save space, encoding the residual layers where there is more likelihood for redundancy.

In visual applications, data transfer of images should allow the display of partially received contents, as this offers the feeling of responsiveness from the application [7]. With the advent of progressive image formats such as JPEG 2000 [6] and Progressive Graphics File, this is now possible. Just like in the case of these two formats, it is not necessary to have access to the entire file to display an image; displaying a reduced version of the image becomes possible as soon as the bottom layer is available, while the full image can only be displayed once the last residual layer arrives.

III. Problem Solution

Bitmap images are large compared to the text that is usually transferred during http communication. Also, with today's relatively high resolution phone cameras, it means that the average uploaded image is in the megabyte size range. This is indeed a problem from the bandwidth point of view, but it also offers some opportunity. Large images will most likely have highly redundant regions, areas where pixel values are

identical or vary only slightly. Each such region could be represented as a reference value and each individual pixel as variation from the reference. In the end, the image will be made up of relatively different reference values, and low ranging differences, which compression algorithms handle the best.

This is performed in a manner similar to the Laplacian Pyramid [8], in the current paper the focus being set on the study of downsampling-upsampling successive operation for residue computation and on the discussion about the importance of the filters used in the resample process [9].

A. Bitmap Conversion to PLF

When including a subsection, the pyramidal layer format presented here defines a region as a group of four neighboring pixels. The algorithm for obtaining such an image is described below.

Step I

A given bitmap image becomes the first layer of the format.

Step II

The current layer, *C*, is down-sampled, resulting a new layer, *N*, half the length and half the width of its predecessor.

Step III

The previously down-sampled layer, *N*, is up-sampled, resulting an up-sampled layer, *U*, that is a rough approximation of the *C* layer.

Step IV

The residual layer R[i] is obtained by subtracting the values of U from C.

Step V

If the size of the layer N image is not beneath a certain threshold, then N becomes the current layer, and the process restarts from Step II. Otherwise all the residual layers are analyzed, compressed and the image in its pyramidal layer is written to the disk.

Fig. 1 illustrates actual image decomposition.

Fig. 1 128 x 128 grayscale Lena; residual image used for reconstruction and the down-sampled 64 x 64 version; residual image used for reconstructing the 64x64 image and down-sampled 32 x 32 version

B. File Format

The pyramidal layer format comes in two forms: permanent and temporary. The permanent format stores the entire data, starting from the lowest uncompressed layer and all residual layers. This file would normally be stored on a server. The temporary counterpart contains only one layer, corresponding to the data that is to be displayed in the browser, the decompression key and various metadata. In the following paragraphs, a more detailed description of the two formats is presented.

The PLF is split into a header and a body section. The body is the more simple structure of the two; for the permanent form, it contains, in order, a serialization of the Huffman tree used for compressing the residual layers, the raw base layer and a sequence of encrypted residual layers starting from the one corresponding to the smallest residual all the way to the largest. The temporary form lacks the encrypted residual layers as it is only used to display the bitmap contained within and to construct more detailed images with the help of residual layers it might receive from the server.

The header contains a magic field, a distinct value for each form, the dimensions of the full sized image, the start and end layer, a numeric identifier of the up-sampling method and a list of offsets relative to the beginning of the file for the serialized Huffman tree, bottom layer and encoded residual images. The number of bytes assigned to each of these fields is detailed in Fig. 2.

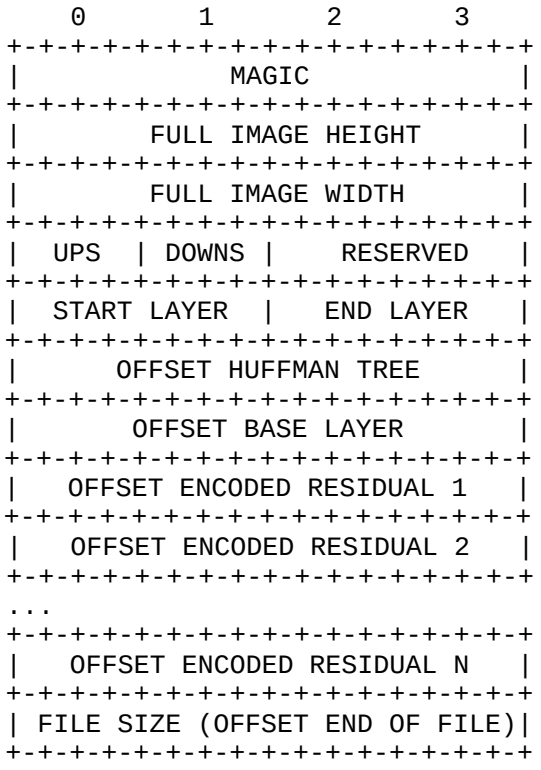

Fig. 2 PLF Header Byte Allocations

It should be noted that once the image has been encoded in PLF, the down-sampling method used to obtain this is no longer relevant. One of the reasons why the down-sampling field was introduced was to keep track of different down-sampling methods therefore which would allow comparing storage requirements later. The other one would be symmetry with the up-sampling field.

The 'reserved' field is currently not used. As a side effect of its usage, it allows access to the fields as 4 byte aligned entities.

In the case of a temporary form, the "start layer" specifies which of the layers it is currently storing as a full image, whereas the 'end layer' indicates the index of the maximum layer that may be obtained to expand the image. For the full permanent form, the start layer field is set to 1 and the 'end layer' to the number of residuals plus one, the base unencoded layer.

Another difference is that temporary forms do not contain offset entries for encoded residuals, because they do not store encoded residual layers. Therefore, in this particular case, only three offset are stored: the offset to the serialized Huffman tree, the offset to the base layer, and the file size.

The 'file size', or 'offset to the end of file' was introduced for symmetry reasons, to allow for a uniform reading of all residual layers, from its starting offset and to the offset of the next layer.

C. Usage Scenario

When including a subsection, the way these two forms are used is described in the following usage scenario. A client with a handheld device connects to the server and asks for a certain page and all its resources. The text content is transferred in its original form, but the images, being stored as PLF are only partially transferred. As a result of the initial transfer, the client will receive an image in temporary form, containing only the base layer of the permanent form of the image stored on the server. This way, the impact on the server is minimized, with very little computation required. On the client side, the browser now has access to a file that has the image stored directly in bitmap format that it only needs to load.

If the connection is good enough, and perhaps there is no cost penalty to the user, the browser could continue downloading more data to improve the quality of the image. If however this is not the case, more data could be loaded as a result of direct user interaction such as zooming in; after all, images are not necessarily displayed at their true size in html pages and, on the relatively small displays, the details are not always visible. In order to improve the quality of an image, the browser has to request more residual layers from the permanent PLF image from the server. One or more layers may be requested at a time and it is the client's duty to compute the more detailed image. The newly computed layer is stored in the temporary PLF image. Once again, the server has minimum involvement, other than reading a specific section of the requested file and delivering that section to the client.

IV. Experimental Results

Two sets of tests were run to evaluate the Pyramidal Layer Format: one focused on storage size, the other on image transfer and display performance.

The tests were performed on a machine with a 2GHz Intel Core2 Dup processor, 2GB of RAM running 32bit Ubuntu.

A. Size Measurements

The size measurements were performed on grayscale images of the following sizes: 256x256, 512x512 and 1024x1024. The images were converted to permanent PLF, with 4 layers.

The down-sampling phase was done by halving the images' width and height, and choosing the value of a single pixel for each down-sampled group of four pixels. For up-sampling, several interpolation variants were chosen: box, triangle, Hermite, Bell, B-Splines, Lanczos and Mitchell.

As can be seen in Table 1, up-sampling methods based on triangle filter and Hermite interpolation produce the best results in terms of image size. Although the files have the same size for the two up-sampling methods, the actual contents in the two cases is different.

It should be noted that file sizes vary quite a lot, depending on the chosen up-sampling method. Also, there is no way to predict how strong the size reduction will be for any particular method. For example, through Hermite interpolation, the data compression ratios obtained are 1.651, 2.112 and 1.677 respectively. This is not as efficient as the compression obtained by the original Laplacian Pyramid [8], but the

original image can be accurately recovered without incurring any penalty to image quality.

Upsampling	Image Resolution		
	256x256	512x512	1024x1024
Box	40885	147914	684455
Hermite	39689	124100	625235
Triangle	39689	124100	625235
Bell	52310	153951	701230
B-Spline	54654	163092	719755
Lanczos	47313	126926	633700
Mitchell	50021	138617	665004

Table I PLF image sizes with 4 layers and various up-sampling approaches

B. Transfer Measurements

In order to evaluate the performance of both the transfer and display time of the images a test setup was created consisting of a browser, a proxy server and the actual http server in charge of delivering the required resources.

The proxy server and the browser were meant to be used as a single entity, with the proxy simply forwarding most of the requests to the actual server, and only intervening to interpret special requests related to PLF images. More exactly, the proxy ran as a browser extension, correctly interpreting the PLF and delivering to the browser contents that it can actually display.

Whenever the browser asks for a certain layer, the proxy will determine if the requested layer or a higher already exists. If that is the case, then an equivalent representation of the layer in an displayable image format already exists and it is read and returned to the browser. If the required layer does not exist, the proxy asks the server to deliver the missing compressed residual layers. When the answer arrives, the layers are decompressed and with the help of the bitmap layer already present on the proxy, larger layers are computed one by one. The last layer is written to the temporary form image and also written in a displayable image format. The web browser eventually receives this latter file which it can display straight away.

The tests were performed on 1024x1024 grayscale images, with the number of layers, including the bottom uncompressed layer ranging from 2 to 9. Column 2 of Table 2 illustrates how big a temporary form PLF has to be to store an initial image of the specified size. The last column indicates the size of the compressed residual layer that is used following an up-sampling to obtain a layer of the specified dimensions.

Base Layer Size	Temporary PLF size	Compressed Residual Size
4 x 4	753	-
8 x 8	785	65
16 x 16	993	216
32 x 32	1761	764
64 x 64	4833	2778
128 x 128	17121	10300
256 x 256	66273	37916
512 x 512	262881	138248
1024 x 1024	1049313	493160

Table II Temporary PLF form sizes and corresponding compressed residuals necessary for obtaining original image

Based on these measurements, it was decided not to create PLF images with a base layer smaller than 32x32. This is the last dimension for which using another down-sampled layer does not make it more efficient in terms of storage requirements. Moreover, going any lower would result in producing an image having hardly distinguishable contents.

Sent content	Browser Response Time (ms)	Proxy Execution Time (ms)	Server Extraction Time (ms)	Transfer Time (ms)
Initial Temp. PLF	5	0.9	0.1	0.4
Residual 64x64	8	1.7	0.1	0.5
Residual 128x128	30	5.6	0.1	0.6
Residual 256x356	37	15.3	0.2	0.7
Residual 512x512	70	56.4	0.4	1.5
Residual 1024x1024	221	203.3	1.4	9.6

Table III Distribution of response times between the various components of the experimental setup

The first thing that can be observed in Table 3 is that of the total time, the time spent on the server is the least significant, being up to two orders of magnitude smaller than the time spent on the client side for reconstructing the image. In fact, as the size of compressed residual layer grows, the image reconstruction ends up becoming the most time consuming part, even more so that displaying the image, which is the case for the smaller layers. Another thing that should be noticed is that the transfer time also has limited impact, but this is only due to the fact that all components were running on the same machine; in a real environment, this would probably consume the most time.

V. Future Work

The main focus in the near future will be to improve the compression ratio of the image. Rough initial tests show that by simply changing the choice made during the down-sampling phase image size can be reduced by approximately 10%. More algorithms should be tested and evaluate how image contents impact compression rates. Alternatives to the compression mechanism and filtering [10] should also be studied, as here too could be room for improvement.

A lossy compression mechanism can be inferred without interfering with the file format or the decompression stages. The residual layers from the pyramid hold data necessary to reconstruct details in the higher resolution layer. Very faint details can then be eliminated just by thresholding elements with small amplitude in the residues. Having fewer values, the residues can therefore be compressed better using Huffman. Blocking artifacts, like those generated in jpg compression, are not a problem, since the image is not split in regions. With an intelligent combination between residue pixel amplitude and placement in space, the lossy compression mechanism can prove to offer visually good quality. This has to be tested, as pixels of greater value can be eliminated from regions with high variance, as their neighbors will mask them, whilst in uniform regions even faint residues are noticeable.

Another potential research topic would be using multi-channel images. Although both the encoding and decoding algorithms should be easily extendable to multi-channel images, it would be interesting to determine whether using a single compression key for all channels or one for each channel is the better choice. We also did a similar test using hypercubes for video encoding with encouraging results[11].

It may be useful to try to split the image first using segmentation or binarization[12], in order to obtain similar regions, or even using a locality-globality approach [13] (relative to how much information is in a region) and compress each region separately using the pyramidal approach. The residual images contain high frequency detail elements. These residual images could themselves be filtered in order to send sharp edges from the start and texture details at a later time. This would mean splitting the residual images in two. Such a filtering can be done in Radon space [14], [15], in order to separate connected edges from uniform noise (texture) background.

Despite performing decently, the current implementation only proves that it is feasible to use PLF. For better performance and a more accurate picture of the true potential of the format, it should be implemented so that it can be integrated with current technologies. More precisely, managing the PLF images should be done by plugins or Native Client applications on the browser side and Apache modules on the server side.

VI. Conclusion

This paper introduced a new image storage format: the pyramidal layer format. This format has relatively low storage requirements, with a potential for improvement. The main advantage is that it can be used as a progressive data format, allowing the recipient of the file to evaluate its content while it is being transferred. This also allows for selective transmission of the content, responding strictly to the end user's requirements: initially only a small portion of the file is sent, and if there is a requirement for a more detailed version, additional data may be sent. With these features in mind, the pyramidal layer format presents itself as a strong contender for image representation in the client server paradigm.

References

[1] X. Song, Y. Neuvo, "Image compression using nonlinear pyramid vector quantization," *Multidimensional Systems and Signal Processing*, vol. 5, 1994, pp. 133-149.

[2] M. S. Kishore, K. V. Rao, "A study of correlation technique on pyramid processed images," *Sadhana*, vol. 25, 2000, pp. 37-43.

[3] J. B. T. M. Roerdink, "Morphological pyramids in multiresolution MIP rendering of large volume data: survey and new results," *Journal of Mathematical Imaging and Vision*, vol. 22, 2005, pp. 143-157.

[4] V. Swathi M. Tech, K. Ashok Babu, "Low bit-rate image compression using adaptive down-sampling technique," *International Journal of Computer Technology and Applications*, vol. 15, 2002, pp. 1679-1689.

[5] L. Williams, "Pyramidal parametrics," *Computer Graphics*, vol. 17, no. 3, July 1983, pp. 1-11.

[6] M. Rabbani, R. Joshi, "An overview of the JPEG 2000 still image compression standard," *Signal Processing: Image Communication*, vol. 17, 2002, pp. 3-48.

[7] L. Vandendorpe, B. Macq, "Optimum quality and progressive resolution of video signals," *Annales Des Télécommunications*, vol. 45, 1990, pp. 487-502.

[8] P. J. Burt, E. H. Adelson, "The Laplacian pyramid as a compact image code," *Communications, IEEE Transactions on*, vol. 31, no. 4, pp. 532-540.

[9] A. Tigora, M. Zaharescu, "Pyramidal layer format – transfer rate control and compression", *in Proc. 1st WSEAS International Conference on Image Processing and Pattern Recognition* (IPPR '13), Budapest, Hungary December 10-12, 2013, pp. 110-115.

[10] C.-A. Boiangiu, A. I. Dvornic, "Methods of bitonal image conversion for modern and classic documents," *WSEAS Transactions on Computers*, vol. 7, no. 7, July 2008, pp. 1081–1090.

[11] A. Enache, C.-A. Boiangiu, "Adaptive video streaming using residue hypercubes," *in Proc. 12th International Conference on Circuits, Systems, Electronics, Control & Signal Processing* (CSECS '13), Budapest, Hungary, December 10-12, 2013, WSEAS Publishing Volume "Recent Advances in Circuits, Systems and Automatic Control", pp. 173-179.

[12] C.-A. Boiangiu, A. I. Dvornic, "Bitonal image creation for automatic content conversion," in *Proc. 9th WSEAS International Conference on Automation and Information*, Bucharest, Romania, June 24-26, 2008, WSEAS Press, pp. 454–459.

[13] C.-A. Boiangiu, A. Olteanu, A. V. Stefanescu, D. Rosner, A. I. Egner, "Local thresholding image binarization using variable-window standard deviation response," in *Proc. 21st International DAAAM Symposium, 20-23 October 2010*, Zadar, Croatia, pp. 133-134.

[14] C.-A. Boiangiu, B. Raducanu, "Robust line detection methods," in *Proc. 9th WSEAS International Conference on Automation and Information*, WSEAS Press, Bucharest, Romania, June 24-26, 2008, pp. 464–467.

[15] C.-A. Boiangiu, B. Raducanu, "Effects of data filtering techniques in line detection", in *Proc. 19th International DAAAM Symposium*, DAAAM International, Vienna, Austria, 2008, pp. 0125–0126.