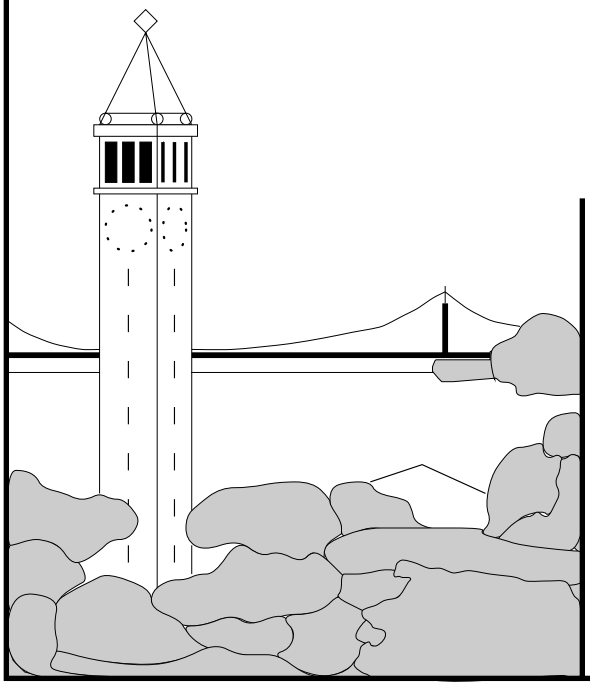
GridPix—Presenting Large Image Files Over the Internet

Satoshi Asami and David A. Patterson



Report No. UCB/CSD-00-1099

May 2000

Computer Science Division (EECS)
University of California
Berkeley, California 94720

GridPix—Presenting Large Image Files Over the Internet

Satoshi Asami and David A. Patterson
Tertiary Disk Project*
University of California at Berkeley
Berkeley, CA 94720-1776
asami@cs.berkeley.edu

Abstract

GRIDPIX is a web-based image presentation system that allows users to zoom in to portions of a large image and scroll around to view different parts by presenting the image as a layered, tiled collection of grids. GRIDPIX is a collection of server-side scripts and does not require the user to download a plugin or an applet. The client-side implementation of GRIDPIX is done entirely in HTML, and thus can be viewed on any graphical web browser, regardless of support for advanced features. While an image presented by GRIDPIX will utilize the memory cache on both the server and client computers, GRIDPIX itself is immune to memory leak bugs, as caching of images are done entirely by the operating system and the browser.

1 Introduction

This paper describes the design and implementation of GRIDPIX, a new way of presenting large images on the Internet. GRIDPIX allows users to zoom in to portions of a large image and scroll around to view different parts.

The Tertiary Disk Project, in cooperation with the Fine Arts Museums of San Francisco, has built a large database of photographs of artwork. GRIDPIX was chosen as the method to present the images, some of which are over 6,000 by 4,000 pixels in their native size. The image database has been available online with GRIDPIX since March 1998.

There are other available methods for image presentation but neither of them achieve the balance of simplicity and power of GRIDPIX. The GRIDPIX implementation is unique in that it uses only standard HTML features to present the tiled images, so the user does not have to download a plugin or an applet before viewing GRIDPIX images. Also, GRIDPIX is immune to memory leak bugs, as the caching of images are done by the operating system and the browser—in other words, GRIDPIX images benefit fully from memory caching, while not having to implement a complex and error-prone caching system by itself.

The rest of this paper is organized as follows. Section 2 briefly introduces the readers to GRIDPIX and Section 3 describes the image database and our architecture. Section 4 compares two currently available solutions with GRIDPIX. Section 5 describes the GRIDPIX design and implementation. The GRIDPIX image concept, file format, and access programs are explained in detail. Finally, Section 6 discusses a few design issues and Section 7 concludes the paper.

2 GRIDPIX Usage

Figure 1 is a sample GRIDPIX screen. The grids are not visible but the image is actually loaded in 15 pieces, 3 rows and 5 columns. There are three main functions: zooming in, zooming out, and scrolling.

Zoom in The user can zoom in by clicking the mouse on any part of the image itself. The

*This research was funded by DARPA Roboline grant N00600-93-K-2481 and the State of California MICRO program.

Figure 1: Sample GridPix Screen

new image will be four times the resolution of the original, i.e., twice the width and twice the height. The size of the window will not change, so the user will be seeing the area one quarter of the original image. The new image will be centered around the tile clicked by the user. The user can repeatedly click on the image to zoom in past the original size to the maximum resolution, which is set to 1,600% by default.

Zoom out One of the buttons on the right is the "zoom-out" button. The image will be one quarter the size, centered at the same spot. The user can repeatedly click on the zoom-out button to shrink the image down to a minimum defined at image conversion time, which is 12.5% by default.

Scroll The user can click on the arrows on the horizontal or vertical scrollbars to scroll the image one tile in that direction. In addition, the user can scroll the image all the way to one end by clicking on the special "scroll to edge" arrows at the end of the scrollbars.

Note that since GRIDPIX is implemented entirely in HTML, it is not possible to grab and drag on the "thumb" of the scrollbar or the image itself. All navigation actions are invoked by clicking.

3 The Imagebase

Over the past few years, The Fine Arts Museums of San Francisco have photographed over 70,000 objects of art and stored the images on PhotoCDs to present them on their web site (`http://www.thinker.org/`). Visitors to the web site can search the image database by artist, title, or description of an art work. The description is a set of keywords that were compiled by a group of museum volunteers. The image database is the largest on-line art collection in the world.

Because of limited storage, the museum offered only low-resolution versions of the images, JPEG images from 2KB thumbnails up to about 100KB, 500×500 pixels, online. The larger versions of the images were kept only on CD-ROMs. In the original form, many images are 3,072 $\times$ 2,048 pixels and about 4.5MB in size, or 6,144 $\times$ 4,096 pixels and occupy 18MB. They shrink somewhat after cropping the borders created by the photographing.

3.1 Problems

Since the web site's launch in 1995, many users have requested larger versions of the images. There are four problems with presenting large images over the Internet:

3.1.1 Storage Space

The first problem is the simple matter of storage space. Each image, even converted to a space-efficient format such as JPEG at a cost of image quality, is still about 1MB in size for the smaller images and 4MB for the larger variant. That would require a storage space of about 70GB just for the images. The original PhotoCDs, as well as an intermediate high-quality (full 24-bit) format such as TIFFs, will easily increase the space demand to more than a terabyte.

3.1.2 Network Bandwidth

Another problem is available network bandwidth. It is not reasonable to simple dump a large file to the user, which averages 1MB as mentioned previously. If the user is on a dial-up network, the situation is even worse, as it will take more than a few minutes to download the entire image.

3.1.3 Copyright

The third problem, which is a unique situation for us, is copyright of art works. There are many current artists whose art works are photographed by the museum. It is not appropriate for the museum to distribute full-size images of those art works, even in relatively low-quality JPEG format.

3.1.4 Usability

The last problem is presentability. Nobody owns a monitor that can display 6,000 by 4,000 pixels on the screen. If the full image is presented as a single HTML page, the user will only see part of it, and will have to scroll around to view the entire image. One alternative is to present the images as a sequence of linked HTML pages, i.e., have the user click on an image to retrieve a larger version. This approach will increase the total size of data transfer over the network.

3.2 Solutions

The first problem is solved by Tertiary Disk, a 3TB disk storage system developed by the Tertiary Disk Project[1]. By connecting 400 9GB disks to 20 PCs over a high-speed network, it allows all the original PhotoCDs as well as human-processed TIFFs to be stored on-line. The TIFFs can be used to generate images in other formats, as well as being used for some other projects in the university, such as content-based search. It can also be used to print high-quality pictures.

GRIDPIX addresses the other three, as we will describe in detail in the rest of this paper.

4 Previous Solutions

When we started development of GRIDPIX, two other implementations of tile-based image viewers existed. They are called FlashPix[2] and GIS Viewer[3].

4.1 FlashPix

FlashPix, developed by Eastman Kodak and others, is a very powerful standard but is overkill for our purposes. It contains support for much more than simple tiled image storage. For instance, it can store information about the brand of film that was used to photograph the image.

Its current implementation requires a client-side plugin, so its use is limited to those with the operating system/browser pair which is supported by the vendor. The size of the plugin is over 1MB. In addition, its complexity makes the server very complicated and large. The source code for the server is not freely available, so there also is a problem of operating system/web server pair being supported by the vendor.

GRIDPIX is just a collection of server-side scripts and does not require the user to download a large plugin or an applet. Also, GRIDPIX is implemented using only standard HTML commands, any thus can be viewed on any graphical web browser.

4.2 GIS Viewer

The GIS Viewer is developed by Prof. Wilensky's Digital Library Project at the University of California at Berkeley. It was originally written for viewing geographic data, hence its name. It is implemented as a Java Applet and thus runs on Netscape Navigator, version 4 or later, and Microsoft Internet Explorer, version 4 or later.

It has a nice interface, not unlike that of FlashPix. It also has several interesting capabilities, such as "layering" several images on top of each other and allowing the user to "annotate" comments on parts of the images, which will be stored on disk for later use.

The biggest problem with the GIS Viewer is not the viewer itself, but the web browsers. Both browsers we tested, Netscape Navigator and Internet Explorer, had a huge memory leak problem when Java applets were used. This caused the web browser to quickly bloat to a few dozen megabytes just by zooming in and scrolling around in a single image.

The applet itself is about 250KB.

5 Implementation

This section illustrates the implementation of GRIDPIX. We will first describe the layering and how the picture is broken up into tiles. Subsequently, the file format will be described in detail, as well as access functions. Finally, the usage and algorithm of the script to generate the HTML files is presented.

5.1 GRIDPIX Images

Figure 2 shows a GRIDPIX file consisting of three layers. Conceptually, GRIDPIX images are similar to Quadtree images. Each layer represents the image at a certain resolution. In the current implementation, the difference in resolution between adjacent layers is 2 in both dimensions; i.e., 1 pixel in one layer becomes four pixels in the layer underneath. This factor is not inherent to the design of GRIDPIX.

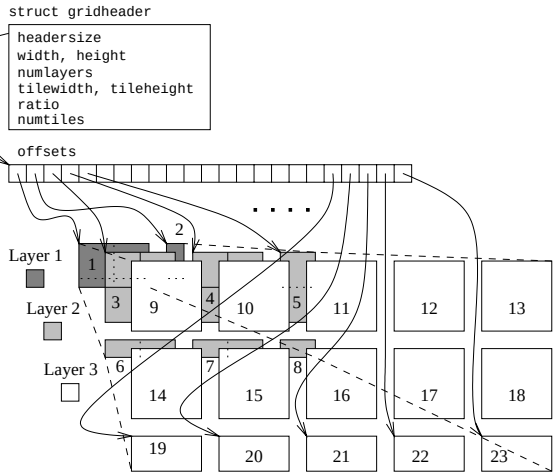


Figure 2: Three-layer GRIDPIX Image

Within each layer, the image is further broken up into tiles. The tiles are all squares of the same size except for the right and bottom edges, which are usually smaller due to the image width and height being not exactly integer multiples of the tile width and height.

In Figure 2, the small numbers inside or adjacent to each tile is the tile's number. Tile numbers start from 1 in the smallest layer and continues between adjacent layers. In our sample, layer 1 consists of tiles 1 and 2, layer 2 consists of tiles 3 through 8, and layer 3 consists of tiles 9 through 23. In this example, tiles 1, 2, 5-8, and 19-23 are not "full" tiles. The tile numbers have no bearing on the image themselves; they are used for fast tile retrieval, as the GRIDPIX server does not have to compute the index of the tile in the file based on image resolutions and other information. Normally, the server only sees the file as a table of indices which tells it where it can find a tile; it does not have to understand that they are image files.

The tiles are numbered in breadth-first order to encourage caching of file blocks by the operating system. As the user scrolls around in a certain resolution, all the tiles are stored on adjacent disk blocks and will very likely to be cached. They will also benefit from read-aheads if the filesystem supports it.

The largest layer stored in a GRIDPIX file is the original size. When the client requests a tile of a layer larger than the original, the tile retrieval program creates those tiles on-the-fly. This is the only case in which the GRIDPIX server has to decode or encode JPEG images. The process is described in detail in Section 5.4.

5.2 File Format

The GRIDPIX file format has evolved over the years. The original format was byte-order and word-length dependent, so it was not possible to share GRIDPIX files between some architectures. The revised format is byte-order and word-length independent. Programs that read GRIDPIX files can deal with either format, which are distinguished by their four-byte magic numbers.

A GRIDPIX file consists of a header and data. The header consists of information about the original image such as width and height, parameters of the GRIDPIX conversion such as number of layers and ratio between layers, and an array of offsets of actual tile images. Each tile is stored as a JFIF format JPEG file[4], immediately starting after the header. The header is variable-sized, and its size is stored near the beginning of the header itself, right after the magic number. The hidden purpose of that design was to easily allow future extensions of the file format, which was utilized when the format was revised.

5.2.1 Original GRIDPIX Format

The header and the rest of the original GRIDPIX file is shown in Figure 3. The magic number is the string "`Grid`". Since the entire header was written from

memory to disk in one piece, there is no difference between the file and memory formats. The drawback of this design is that the file format was architecture dependent.

There is virtually no limit in either the number of layers or the image size. The only limits are the numbers representable by 32-bit signed integers used in various fields. For instance, it can't store an image that is more than 2,147,483,647 pixels in either width or height. Unless the image is extremely narrow, the space required for any single such image will surely exceed the capacity of any storage system currently on this planet.

```
#define Magic "Grid"
#define MaxLayers 20

struct gridheader {
  unsigned char magic[4] ;
    /* size of this struct */
  int headersize ;
  int width, height ;
  int layers ;
  int tilewidth, tileheight ;
    /* currently always 2 */
  int ratio ;
    /* num of tiles in file */
  int numfiles ;
    /* num of tiles in each layer */
  int layersize[MaxLayers] ;
} ;

struct gridpix {
  struct gridheader header ;
    /* actually an array */
  off_t offsets[1] ;
} ;
```

Figure 3: Original GRIDPIX File Format

The header includes some information that is redundant, e.g., the number of tiles in the file, which can be computed by image sizes and number of layers. These are included just as a matter of convenience.

5.2.2 Revised GRIDPIX Format

To make the GRIDPIX file architecture independent, the format was revised to decouple the memory and file layouts. The magic number of this new format is the string "`GPX2`". Numbers are stored in 64-bit integers (`int64_t`) in memory, and in an array of 8 bytes (`unsigned char [8]`) on file. The new format is shown in Figure 4.

To aid conversion between the memory and file formats, two functions shown in Figure 5 are provided to convert the numbers from memory to file formats and from file to memory formats, respectively.

5.3 GRIDPIX File Generation

GRIDPIX files are created by the program `gridpack`. `gridpack` takes a Portable Pixmap (PPM) file, a 24-bit full-color format, and converts it to GRIDPIX. It takes a few optional arguments, such as tile size ("`-t` *size*") and JPEG image quality ("`-q` *quality*"). The `netpbm` package can be used to convert many image formats, including TIFF and Targa, to PPM.

5.4 Tile Retrieval

The GRIDPIX tile retriever, `gettile`, is usually called with just two arguments, the GRIDPIX file name and the tile number. In fact, `gettile` does not even parse the header past the second field, the header size, in that case. Figure 6 shows pseudocode that illustrates a typical tile retrieval operation.

There are two disk accesses; one to read the header, including the offset table, and one to read the actual data. As various tiles in an image are requested by the client, the whole header will very likely be in the server filesystem's memory cache. For instance, there are up to about 300 tiles in a standard GRIDPIX file created from a $3,072 \times 2,048$ pixel PhotoCD files; the size of the header is about 2KB. Also, the breadth-first ordering clusters the tiles of the same layer close to each other, thus encouraging the filesystem cache to hold them in memory.

```
#define Magic2 "GPX2"

typedef
  unsigned char g_int64_t[8];
  /* on file */
#define f_int_t g_int64_t
  /* in memory */
#define m_int_t int64_t

/* New header type on file */
struct gridheader2 {
  unsigned char magic[4] ;
  f_int_t headersize ;
  f_int_t width, height ;
  f_int_t layers ;
  f_int_t tilewidth, tileheight ;
  f_int_t ratio ;
  f_int_t numfiles ;
  f_int_t layersize[MaxLayers] ;
} ;

/* New header type in memory */
struct gridheader {
  unsigned char magic[4] ;
  m_int_t headersize ;
  m_int_t width, height ;
  m_int_t layers ;
  m_int_t tilewidth, tileheight ;
  m_int_t ratio ;
  m_int_t numfiles ;
  m_int_t layersize[MaxLayers] ;
} ;

struct gridpix {
  struct gridheader header ;
    /* actually an array */
  m_int_t offsets[1] ;
} ;
```

Figure 4: Revised GRIDPIX File Format

```
void fromfint(m_int_t *target,
  f_int_t source) ;
void tofint(f_int_t *target,
  m_int_t source) ;
```

Figure 5: GRIDPIX Converter Functions

```
gettile(gridpix, tilenum)
{
    read header;
    verify header.magic;
    tilepos = header.headersize
        + sizeof(pointer) * tilenum;
    seek to tilepos;
    read two words and store them
        in offset1 and offset2;
    seek to offset1;
    read (offset2 - offset1) bytes
        and return it;
}
```

Figure 6: The `gettile` function

When the user zooms in past the original resolution, the request is given to `gettile` as a retrieval of a tile whose tile number is larger than that of the last tile. When `gettile` sees such a request, it will calculate the exact location of where that tile would be if it were part of the GRIDPIX file, and slices up one of the JPEG tiles to return the imaginary tile.

Figure 7 illustrates how such a request is handled. Layer 3, the largest layer of the same GRIDPIX file as in Figure 2, is shown in solid squares. Suppose there is a request for tile 34. Since this is larger than the total number of tiles in this GRIDPIX file, 23, `gettile` recognizes it as a subtile request and calculates the exact magnification and location of the tile. According to the dimensions of this GRIDPIX file, tile 34 would be in layer 4, shown as shaded squares, in the lower left quarter of tile 9—tiles 24, 25, 34 and 35 comprise tile 9. Thus, `gettile` will decode the JPEG file in tile 9, re-encode its lower left quarter, and return it as tile 34.

As mentioned earlier, handling requests for subtiles is the only time `gettile` needs to recognize the tiles as JPEG files; otherwise, it will just return the entire file. `gettile` can provide subtiles of arbitrary resolution. The number of magnification levels `gettile` provides past the original resolution is a compile-time option.

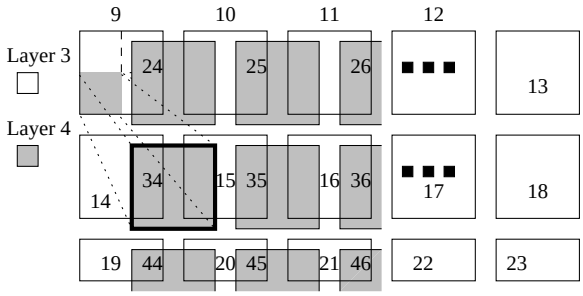


Figure 7: Subtile Retrieval

5.5 HTML Page Generation

The HTML pages are generated by the program `mkhtml`. In essence, `mkhtml` is the heart and soul of GRIDPIX. In addition to reading the GRIDPIX file itself, it takes arguments such as resolution (layer number), width and height of user's screen, x and y coordinates of the upper-left corner, and returns a complete HTML page for that portion of the image.

Figure 8 shows the gist of the HTML source for a simple GRIDPIX image. The image is arranged in a 3×2 grid of tiles. Each image is a hyper-link button. The images are supplied by the `gettile` program in `<img>` tags and the `<a>` tags point to calls to `mkhtml`. The arrows on the scrollbars are also hyper-links to `mkhtml`. In addition, there are buttons on the right side used for zooming out, displaying a list of images and displaying a help page.

The `mkhtml` command takes 8 arguments. They are image number, zoom level, horizontal and vertical sizes of the view area, x and y offsets of the upper-left corner of the window, smoothing option and display style, in that order. Smoothing option is explained in Section 6.3. The display style argument selects between various page style options. This is necessary when the same `mkhtml` command is used to serve multiple sets of images with differing display needs.

6 Discussion

In this section, we will discuss several design issues that came up during the implementation of GRIDPIX.

6.1 Tile Boundaries

One question that comes to most people's mind when hearing that GRIDPIX just lays out tiles using HTML is: "will the tile boundaries bother me?" The answer is "no" or at least "not any more than the full image". It is because the JPEG encryption is inherently tile-based; the *encryption unit* of JPEG images is 8×8 by default. Thus, as long as the tile height and width are integer multiples of 8, the boundaries of tiles will not be any more visible than the 8×8 grids in the underlying JPEG file.

6.2 Freedom of Scrolling

GRIDPIX does not allow people to scroll in units other than the tile size. Part of it is inherent to the pure-HTML implementation; it is impossible to have the user grab and drag either the image or the scroll-bar thumb. However, it *is* possible to have a different kind of interface, such as a "half-arrow" button, that allows the user to specify scroll units of less than one tile. The tile server will then generate fractions of tiles, and they will be stitched together using an HTML file that specifies the fractional tile size.

There are three reasons why I did not implement this in the current version of GRIDPIX. One is complexity. The second is clarity of user interface. I could not imagine an interface that is obvious to anyone how to scroll half a tile. The third is caching. For instance, if the user scrolls a half-tile, then another half-tile, the server will either have to send a whole tile for the second scrolling action, thus wasting the half-tile already fetched, or send images always in half-tile increments, causing the image to be very fragmented.

This is a weakness in the HTML-based implementation, as other methods, with more intelligence in the client, can handle such cases without a problem. HTML does not allowed us to specify "display the top half of this image" or some such.

6.3 Image Magnification

In GRIDPIX, users are allowed to zoom in past the original resolutions of images. By default, `get-`

```
<table border=0>
 <tr><td><table border=1>
    <tr><td><table border=0> <!-- 3 x 2 array of tiles -->
       <tr><td width=450 height=288 align=left valign=top>
         <a href="/cgi-bin/mkhtml.cgi?23191021154700943&432&288&0&0&0&1">
          <img width=144 height=144 border=0
           src="/cgi-bin/gettile.cgi?23191021154700947&0"></a>
         [two more images in <a> tags]
         <br>
         <a href="/cgi-bin/mkhtml.cgi?23191021154700943&432&288&0&2&0&1">
          <img width=144 height=144 border=0
           src="/cgi-bin/gettile.cgi?231910211547009412&0"></a>
         [two more images in <a> tags]
         <br>
        </td></tr>
       </table></td>
     <td><table height=292 border=0> <!-- vertical scrollbar -->
       <tr><td width=14 border=0>
         <img width=14 height=14 src="/images/sb/vbardtt.png">
         <img width=14 height=102 src="/images/sb/vbard.png">
         <img width=14 height=14 src="/images/sb/vbardb.png">
         <a href="/cgi-bin/mkhtml.cgi?23191021154700942&432&288&0&1&0&1">
          <img width=14 height=130 border=0 src="/images/sb/vbarl.png">
          <br>
          <img width=14 height=14 border=0 src="/images/sb/downarrow.png"></a>
         <br>
         <a href="/cgi-bin/mkhtml.cgi?23191021154700942&432&288&0&2&0&1">
          <img width=14 height=14 border=0 src="/images/sb/downarrows.png"></a>
         <br></td></tr>
       </table></td></tr>
    <tr height=14><td>
      <table width=436 border=0>
      [horizontal scrollbar]
      </table>
     </td><td width=14 height=14 align=center border=0></td></tr>
   </table></td>
  <td valign=top> <!-- buttons on right side -->
   <table>
    <tr><td>25\%</td></tr>
    <tr><td><a href="/gridpix/">
       <img width=40 height=40 border=0 src="/images/dir.png"></a></td></tr>
    <tr><td><a href="/cgi-bin/mkhtml.cgi?23191021154700941&432&288&0&0&0&1">
       <img width=40 height=40 border=0 src="/images/zoomout.png"></a></td></tr>
    <tr><td><a href="/cgi-bin/gpxhelp.cgi?231910211547009458045642581">
       <img width=40 height=40 border=0 src="/images/help.png"></a></td></tr>
   </table></td></tr>
</table>
```

Figure 8: An HTML page generated by `mkhtml`

`tile` will return a subtile with fewer pixels, which the web browser will expand it to the requisite size because of the `width` and `height` attributes in the `<img>` tags.

This will cause each pixel in the original image to be magnified as the user keeps zooming in, first to 2×2, then 4×4, and so on, causing an annoying aliasing effect. In order to reduce the aliasing, by specifying the smooth option to `mkhtml`, the user can get a subtile with a smoothing filter. This increases the processing load and transfer size, since the new subtile will be first expanded to the original size and then blurred. The effect of this is mixed; some images look better with smoothing, others look worse. Overall, it does not seem like it is worth the additional calculation and transfer speed penalty.

7 Conclusion

GRIDPIX is a web-based image presentation system that allows users to zoom in and scroll around to view portions of a large image. As a collection of server-side scripts with the client side implemented entirely in HTML, GRIDPIX does not require the user to download a plugin or an applet, and thus can be viewed on any graphical web browser regardless of support for advanced features. The feasibility of GRIDPIX has been proved by presenting over 70,000 images of art objects over the Internet since March 1998.

References

[1] Nisha Talagala, Satoshi Asami, David Patterson, Bob Futernick, and Dakin Hart. The Berkeley-San Francisco Fine Arts Database. In *Proceedings of the Fifteenth IEEE Symposium on Mass Storage Systems*, pages 163–167, March 1998.

[2] Eastman Kodak, Microsoft, Hewlett-Packard, and Live Picture. *FlashPix Format Specification, Version 1.0.1*. 1997.

[3] UC Berkeley Digital Library Project. *The GIS viewer*. `http://dlp.CS.Berkeley.EDU/gis/`.

[4] The Independent JPEG Group. `http://www.jpeg.org/public/jpeghomepage.htm`.