X Image Extension Overview

J. Ben Fahy, Ph.D. & Robert N.C. Shelley

Abstract

The X Image Extension provides a powerful mechanism for the transfer and display of virtually any image on any X-capable hardware. While not intended for use as a general purpose image-processing engine, XIE does provide a robust set of image rendition and enhancement primitives that can be combined into arbitrarily complex expressions. XIE also provides import and export facilities for moving images between client and server or core X and XIE and for accessing images as resources.

XIE Design Goals

The X Image Extension (XIE) was designed to facilitate efficient and robust image display on X Window System servers. XIE provides tools for rapidly transferring an image from client to server and for converting the image format to match the server's hardware characteristics. XIE does not attempt to provide tools for general-purpose image processing. However, simple image enhancement and filtering operations such as contrast enhancement and convolution are available, as well as dithering, geometric transformations, histogram generation, and so on.

The X Window client-server architecture was based on the assumption that data on the wire would be of a relatively high level. Therefore, a high-bandwidth connection is not an absolute requirement for X. However, image transfer requires sending large amounts of low-level data, which may take an unacceptable amount of time on slow or heavily loaded networks. This image transport bottleneck makes the X Window System less than optimal for supporting image-intensive applications.

XIE solves the image transport problem by supporting the transmission of compressed images across the wire and providing image decompression facilities in the server. Compression ratios on the order of 20:1 or more are common for bitonal images. The client can send a relatively modest amount of data to the server where it is decompressed into a much larger data structure. Since the image may not fit completely on the server's display monitor at full resolution, XIE supports geometric operations, such as scale and rotation, to allow an image to be transformed to fit in a reasonable window size as is depicted in Figure 1.

Rendition in XIE is defined as the process of changing the format of an image in order to make it compatible with the server's frame buffer, its associated lookup table(s), and other limitations. Once an image has been rendered, it may be transferred directly to the hardware and viewed on the screen. Rendering a complicated image to be compatible with highly limited hardware can be quite challenging and may require a series of manipulations to be performed on the image. It may be necessary to convert the image from trichromatic to monochromatic (gray scale or bitonal). A scaling operation can be used to reduce the image's size to fit the screen dimensions. Convolution provides a means for sharpening the image or performing other general filtering operations. Dithering may be used to reduce the number of quantization levels to comply with the number of colormap entries available, and so on.

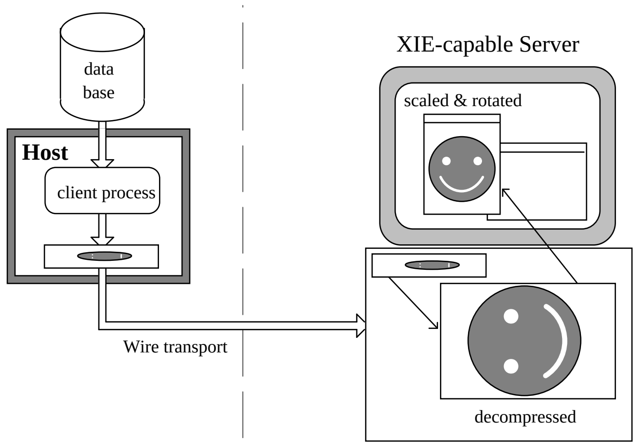


Figure 1. Compressed transport from client to server, followed by decompression, scaling, and display.

The primitives of XIE that are needed to support image transfer and rendition define a new class of virtual display hardware. In some cases, it may be possible for server implementors to map these functions directly onto physical hardware accelerators. For example, the Convolution operator in XIE might be implemented directly using a convolution board or chip. Providing a vehicle for hardware vendors to upgrade the performance of X servers for image-specific operations was an explicit design goal of XIE.

To summarize the main goals of XIE:

- Display any image on any server
- Reduce the time it takes to move images from client to server
- Support the development and utilization of image-specific hardware
- Do not cover all of image processing

XIE Historical Summary

The XIE project was initiated in 1988 by Joe Mauro of Digital Equipment Corporation. The first formal proposal was submitted to the X Consortium in 1990. At that time, it was voted not to proceed to technical review, "because of a lack of sufficient resources to review the proposal". Digital continued to work with interested Consortium members in revising the protocol and produced a portable sample implementation of XIE (Version 3) to demonstrate proof of concept. In March 1991, XIE was officially moved into technical review. After more than a year and a half of discussion by many interested companies, a substantially modified XIE (version 4) was promoted from technical review to public review at the end of 1992. In January 1993 AGE Logic was contracted to produce a new sample implementation that was completed in January 1994. Several more enhancements were made to the protocol during the public review and sample implementation period; the final release of the XIE-SI was version 5. The major changes to the XIE protocol between versions 3 and 5 included:

- More direct control over server behavior given to the client
- Generalization and refinement of the computational model
- Elements that do not perform implicit data type conversions
- Tools for creating a processing engine moved to the client side
- Unification of all geometric operators into a single transformation
- Enhanced support for color spaces and color operations
- Cleaner interface to Core X
- Support for more compression techniques
- Identification of a Document Imaging Subset

XIE Architecture

The image extension is designed to add new functionality to X, but, at the same time, it avoids making any fundamental changes to the X architecture. XIE receives requests and sends errors, replies, and events through the standard mechanisms defined for extensions. XIE resources must be registered with the Core X resource manager so that at client shutdown all XIE resources can be cleanly deallocated. All data in and out of XIE must pass through X, either by the standard extension mechanism or by explicit import and export. Note in Figure 2 that all utilization of resources in XIE must go through a Photoflo Manager. Data is imported into the Photoflo Manager and exported out from it. The Photoflo Manager, which also runs the computational engine of XIE, is discussed in the next section.

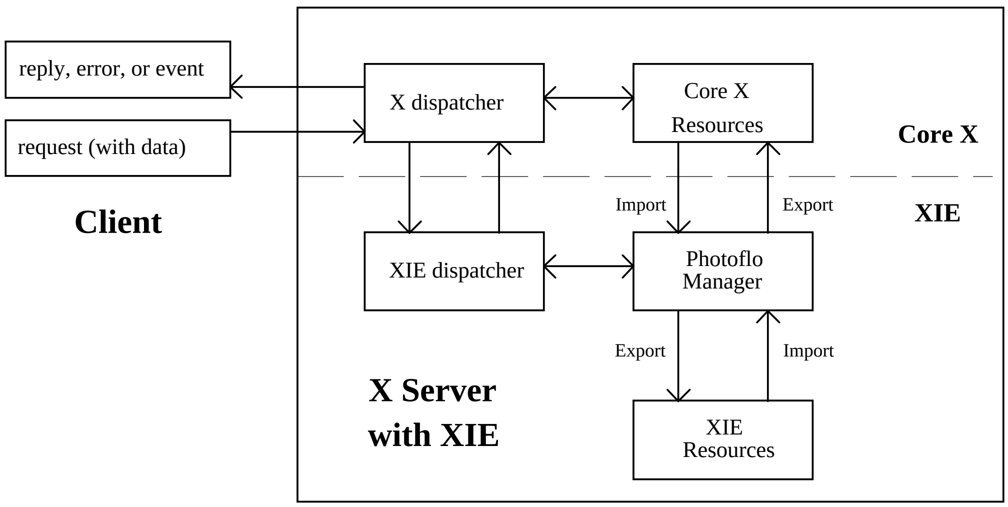


Figure 2. High-level view of an X server containing XIE support.

XIE's Computational Model

Typical XIE processing employs a sequence of operations. For example:

- Transport an image from client to server
- Decompress the image
- Scale the image
- Render and enhance the image
- Display (export image data to a Core X drawable)

A client could ask the server to perform each of these operations one step at a time — send the complete image to the server, then decompress it, then scale it, and so on, and finally display the result. One disadvantage of this approach is that the user may have to wait a long time before seeing the image on the screen. Furthermore, the memory requirements of holding the full decompressed image in the server may be severe. The necessity of producing full intermediate images between steps further aggravates this situation. Consider the simple sequence of operations {transfer image, decompress, scale, export to X} that is depicted in Figure 3:

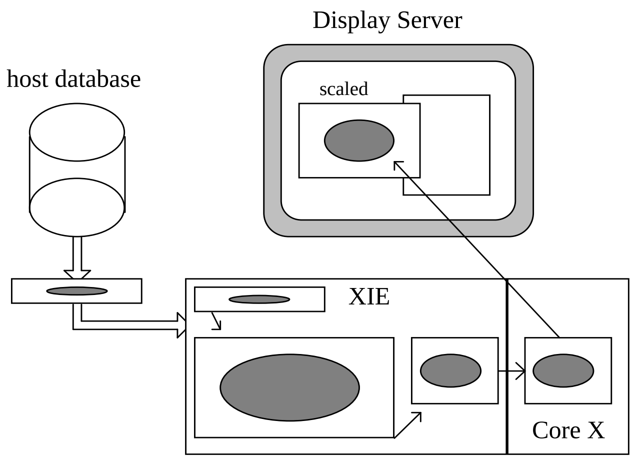


Figure 3. Processing each step completely before moving data to the next element in the sequence requires allocating a large amount of buffer space in the server.

When the image is processed one step at a time, large buffers must be allocated to hold the decompressed image in the server. This may require more memory than is available in the server, in which case XIE would have to return an allocation error, thereby preventing the user from viewing the image.

Suppose instead that the client could specify all operations in the sequence to the server at once. The server could then choose to perform only part of each operation before passing the output data to the next processing step in the sequence. For example, it might decompress half of the incoming image, scale that half, and export it to Core X. At that point the user would see the top of the image displayed on the screen (Figure 4):

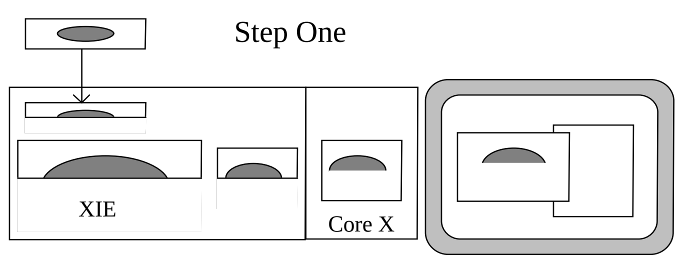


Figure 4. Processing only half of each step before moving data to the next element.

After the top half of the image has been displayed, the server could decompress the remainder of the image, scale it, and export the result to Core X, thus completing the display of the whole image (Figure 5). Two benefits have been realized. First, the user was able to see something appear on the screen in roughly half the time. Second, the server did not have to allocate as much buffer space. In fact, the amount of buffer space required will be inversely proportional to the number of steps into which the process is subdivided (until some minimum level is reached).

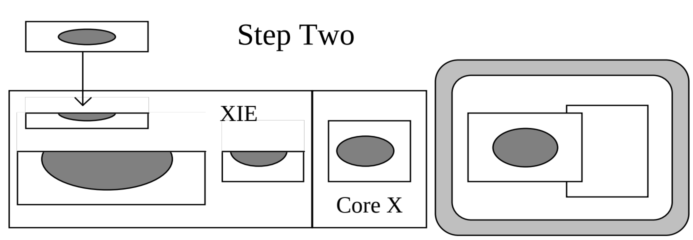
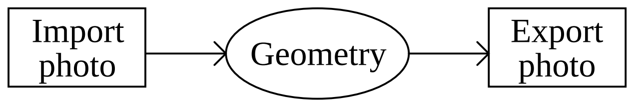


Figure 5. Processing the second half of each step.

In XIE, the ability to define a sequence of operations to be performed is referred to as constructing a *Photoflo*. Photoflos are so named because the image data (sometimes *photo*graphic) is viewed as *flo*wing through the sequence of elements. The Photoflo Manager depicted in Figure 2 obtains data via *import* elements, processes the data using *process* elements, and outputs the final result via *export* elements. A Photoflo for the sequence just discussed would look like:



No explicit decompression step is required. The Import element describes the format of the data to be transferred, and the server knows it must implicitly decompress the image data prior to passing it to the Geometry element. Similarly, the client may specify that an image is to be compressed before it is stored in a Photomap or returned to the client.

Photoflos do not have to be linear. The client is allowed to construct any Directed Acyclic Graph (DAG) as long as the way the elements are connected makes sense. For example, the Photoflo in Figure 6 computes an unsharp masked enhancement of the input image by first blurring a subregion of the image, using a convolution operation, and then adding the difference to the original image:

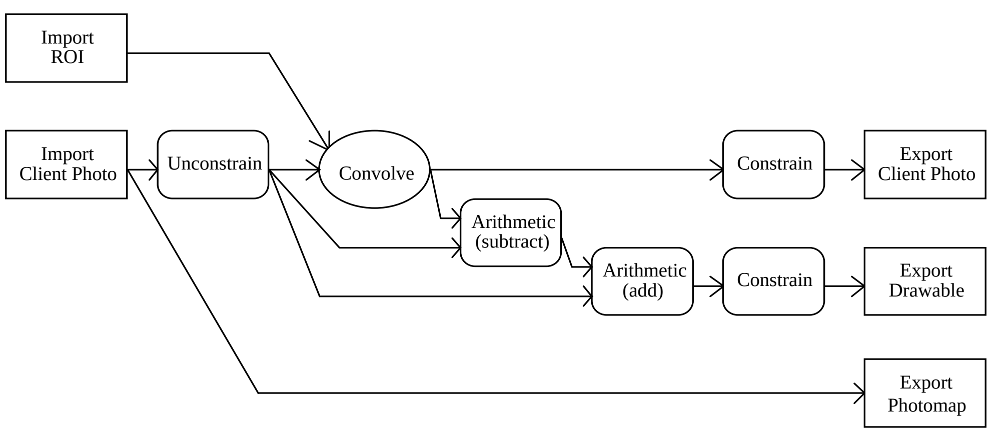


Figure 6. Photoflo to compute unsharp masked enhancement with rectangles of interest.

The client can specify this DAG to the server all at once, with either **CreatePhotoflo** (to create a permanent Photoflo that can be reused) or **ExecuteImmediate** (to create a temporary Photoflo that will be destroyed immediately upon completion). The Photoflo Manager creates the Photoflo and then waits for further instruction. Once the Photoflo is *activated* (performed by **ExecutePhotoflo** for permanent Photoflos, or performed implicitly for temporary Photoflos), the Photoflo Manager will asynchronously execute the Photoflo, processing input as it becomes available. The client pushes data into the Photoflo by using a **PutClientData** request, specifying the Photoflo and the appropriate Import element. The client reads data out of the Photoflo by sending a **GetClientData** request, specifying the Photoflo and the appropriate Export element. It is not unusual for the Photoflo to block further processing pending either additional input from the client or retrieval of available data by the client.

Several operations in XIE permit control over the image *processing domain* by specifying a *control-plane* or a set of *rectangles-of-interest* (ROI). This allows portions of the image to be included or excluded from the rendition or enhancement operation. In Figure 6, the Convolve operation is restricted to a subset of the pixels in the image by importing a set of ROI)s to modify the processing domain of the Convolve element. Pixels that fall within the area described by the set of rectangles are processed; those which do not are passed through unchanged. This allows the user to reduce the computation time in areas of the image that are not interesting.

XIE Resources

XIE defines six permanent resources that can be created and destroyed: ColorLists, LUTs, Photoflos, Photomaps, Photospaces, and ROIs. Some of these have already been introduced.

A *Photoflo* is a computational engine that the client specifies using either a **CreatePhotoflo** or an **ExecuteImmediate** request. A Photoflo created by **CreatePhotoflo** is considered permanent and exists until an explicit **DestroyPhotoflo** request or client shutdown. The resource ID specified at Photoflo creation time is a normal core X resource ID. A Photoflo created by **ExecuteImmediate** is a temporary resource that is created in a separate resource ID space, called a *Photospace*. Before creating the first temporary Photoflo, a Photospace must be created to hold it using the **CreatePhotospace** request. If this Photospace is destroyed, all Photoflos in the space are immediately destroyed along with it.

*LUT*s, *ROI*s, and *Photomap*s are used as Photoflo inputs by using ImportLUT, ImportROI, and ImportPhotomap elements, respectively. A Photomap is a handle for image data stored in the server. Photomaps are created with no attributes: they inherit data and descriptive information from ExportPhotomap elements in Photoflos. The attributes of a Photomap may be queried by using the **QueryPhotomap** request. A LUT is a handle for lookup table data for the Point operation. A LUT resource receives data and attributes from an ExportLUT element in a Photoflo. A ROI resource is a handle for a set of rectangles-of-interest. It is populated using the ExportROI element.

ColorLists are used to store colors allocated from a colormap. When a colormap and a color or gray scale image is passed to the ConvertToIndex element, the colormap identifier and the index of each colormap cell allocated by ConvertToIndex are stored in the ColorList. The contents of a ColorList can be queried using the **QueryColorList** request.

Element Definitions

This section briefly discusses the elements with which one can create computational engines (Photoflos). The number of individual Photoflo elements listed below is deceptively modest, in that there are many different ways in which several of the elements could have been specified. For example, in defining a geometric transformation for an image, it is quite simple to specify the transform in idealized coordinates, but impossible to choose one “optimal” resampling method. This is because evaluation of the algorithm's performance depends on factors that are largely subjective, and processor or data dependent. It is unreasonable to pick a single algorithm for a given element when clearly dozens of other algorithms are available of equal or better quality. It was decided to build flexibility into XIE's basic element definitions, giving both server implementors and client writers leeway in selecting the algorithms that best meet their needs.

The XIE protocol incorporates this flexibility by defining a *technique* parameter for several of its elements that acts as a modifier of the those elements behavior. The techniques that are supplied to import and export elements generally identify image-compression schemes, whereas the techniques specified for processing elements identify various image-processing algorithms that provide slightly different methods for doing the same basic operation. In some cases techniques offer a trade-off between execution time and the fidelity of the results, while in other cases different techniques are desirable because of the image class or content, or to achieve a special effect.

For some techniques the server implementor is required to provide a default algorithm. This allows an application to select an operation in a generic way without concerning itself with the details of the particular algorithm being used. The protocol document specifies that some techniques must be implemented in all servers, whereas other techniques are considered optional. The server implementor is also permitted to extend XIE with additional proprietary techniques as is seen fit. XIE includes a query request to determine what techniques are supported by a particular implementation and to determine what algorithms are being used as defaults.

Import Elements

Import elements may be classified according to the source of their data, which must be either the client, Core X, or XIE resources.

Client Import Elements

ImportClientLut — reads in lookup table data from the client. Since the client will send raw data using PutClientData, this element is used to specify the format of the expected data stream.

ImportClientPhoto — reads in image data from the client. Since the client will be sending raw image data using PutClientData, this element is provided to specify the format of the data stream. A *decode* technique, and its associated arguments, are used to specify which compression algorithm has been applied to the input data.

ImportClientROI — used to read in a list of rectangles for process domain control. Since a fixed format is defined, only the number of rectangles needs to be provided.

Core X Resource Import Elements

ImportDrawable — reads in image data from a window or pixmap.

ImportDrawablePlane — reads in a selected image bitplane from a window or pixmap.

XIE Resource Import Elements

ImportLUT — import existing lookup table data, typically to be used by the Point processing element.

ImportPhotomap — import previously stored image data. Since the Photomap maintains the attributes of the stored image, no decoding parameters are required (as was the case with ImportClientPhoto).

ImportROI — import a list of rectangles that was previously stored in the server, typically used for process domain control.

Process Elements

Processing elements can be roughly grouped by their complexity, functionality, or the aspect of the image that they work with. One such grouping follows.

Simple Algebraic Processing Elements

Arithmetic — allows arithmetic operations, such as addition or subtraction, to be performed between a pair of images or an image and a constant value. Other operations, such as multiplication or division, can only be performed between an image and a constant value.

Blend — allows an alpha-blend operation between two images or an image and a constant value. The weights for the blend operation can be determined by a constant value or on a per-pixel basis through the use of an *alpha-plane*.

Compare — compares two images or an image and a constant value to produce a bit plane with ones where the comparison is satisfied.

Logical — performs per-pixel bit-wise operations on images, or between an image and a constant.

Math — performs mathematical operations, such as square root or natural logarithm, on a single source image.

Format Conversion Elements

BandCombine — accepts three SingleBand images and merges them to form one TripleBand image.

BandExtract — produces a SingleBand image from a TripleBand image by summing a percentage of the pixel values from each input band with a bias value.

BandSelect — selects a single image band from a TripleBand image.

Constrain — converts unconstrained pixel values into integer pixel values (certain operations in XIE can be performed using unconstrained pixel values which are represented within the server as either floating-point or fixed-point values).

ConvertFromIndex — takes a colormap and an image as input and produces a new image whose pixel values are taken from the colormap.

ConvertFromRGB — converts RGB image data to another colorspace.

ConvertToIndex — allocates image colors from a colormap or matches colors in the image against those already existing in the colormap. Each pixel in the output image is the colormap index of the colormap cell that most closely represents the corresponding input image pixel. The action taken (allocate, match, and so on) is dependent on the color allocation technique given to ConvertToIndex. Each colormap cell that is allocated by ConvertToIndex is stored in a ColorList resource.

ConvertToRGB — converts non-RGB image data to RGB (from another color space).

Dither — reduces the number of quantization levels in an image by spreading the information contained in a pixel over the surrounding area.

Unconstrain — converts integer pixel values into unconstrained pixel values (certain operations in XIE can be performed using unconstrained pixel values which are represented within the server as either floating-point or fixed-point values.)

Simple Enhancement and Filtering Elements

Convolve — performs a convolution on an image; each output pixel is computed from an area of input pixels, as determined by the size and contents of a convolution kernel.

MatchHistogram — computes the histogram of the input image and remaps pixel data so as to match, as closely as possible, the specified histogram shape.

Point — applies a lookup table operation to each pixel in the source image. Point is so named because the output pixel value depends only on the input value at that "point".

Domain-Based Operations

Geometry — performs a geometric transformation on image data. Special cases include scale, crop, mirror, shear, rotate, and translate. The operation is specified as a mapping of the coordinates of the output image back to the coordinate space of the input image. Resampling and anti-aliasing algorithms can be specified by using technique parameters.

PasteUp — allows multiple image "tiles" to be combined into a single image. A tile is an image with an offset attached that specifies where the tile is to be placed in the output image. Overlapping tiles are resolved by a defined stacking order. If the tiles do not completely cover the output image, uncovered regions are filled by a constant pixel value.

Export Elements

As with import elements, export elements may be classified according to the destination of their data, which must be either the client, Core X, or XIE resources.

Client Export Elements

ExportClientHistogram — computes the histogram of an image and makes it available to the client via the **GetClientData** request.

ExportClientLUT — makes lookup table data available to the client. The client can specify the entire LUT or a subrange of the LUT.

ExportClientPhoto — makes image data available to the client. The client can specify the format it wants by using an *encode* technique and its associated parameters.

ExportClientROI — makes rectangle-of-interest data available to the client.

Core X Resource Export Elements

ExportDrawable — writes image data into a pixmap or window. The client must provide a graphics context to direct the insertion of data.

ExportDrawablePlane — writes a bitonal image into a pixmap or window. A client-specified graphics context is used to convert the pixels to foreground and background colors and to perform optional stippling.

XIE Resource Export Elements

ExportLUT — exports lookup table data to a LUT resource. ExportLUT obtains its input from a previous element such as ImportClientLUT or ImportLUT.

ExportPhotomap — stores an image in a Photomap resource. Encoding parameters can be supplied to tell the server what format to store the data in. Alternatively, the client can allow the server to choose the most efficient format for image storage; the client can hint as to whether image storage space or image retrieval time is the desired efficiency metric.

ExportROI — stores a list of rectangles in an ROI resource. The exported data must come from either ImportClientROI or ImportROI.

Subsetting

The full XIE protocol defines a very powerful flexible system. It is recognized, however, that many application writers will wish to use XIE features to perform relatively simple tasks on simple (for example, bitonal) images. They may never deal with color spaces or processing domains or want to do anything more complex than simply import, scale, and export. They will be interested in using a server that is tuned specifically for the needs of document imaging, ignoring all other concerns. In fact, they can do without many of XIE's capabilities, as long as the server provides the simple operations they require and it obeys the same protocol as the full XIE specification for those operations.

The XIE Document Imaging Subset (DIS) was designed to respond to this demand. DIS drops the concepts of Color Lists and processing domains completely and supports only two processing elements: Geometry and Point. Geometry is required in order to do scaling and rotation. Since the DIS client is responsible for managing the colormap, Point is needed to convert pixel values into colormap indexes.

The only required image-compression schemes are those associated with bitonal images. DIS also provides the ability to import a full-depth drawable, scale it using Geometry, and then export the result back to Core X. This functionality may be useful for those who want to be able to scale images produced by other applications. One could imagine using this feature to resize a drawing in order to include it in a document.

Summary

The X Window System defines a standard architecture that has provided a stable platform-independent display environment for application developers to build upon. By adhering to the same basic principles, XIE enhances the image display capabilities of the core protocol by providing:

- Efficient image transport between client and server
- Efficient caching of images on the server
- Image rendition and enhancement capabilities on the server
- Support for the development and utilization of image-specific hardware
- Consistent policy-free support for image display across all platforms

References

1. *X Window System,* Scheifler, Gettys, Digital Press.
2. *X Image Extension Protocol*, Shelley et al., Digital Equipment Corporation and AGE Logic, Inc.
3. *X Image Extension Library,* Rogers AGE Logic, Inc.
4. *XIE Sample Implementation Architecture,* Shelley, Verheiden & Fahy, AGE Logic, Inc.
5. *X Image Extension Overview,* Fahy & Shelley, The X Resource — Special Issue C, 1/93 O'Reilly & Associates, Inc.