1991

XS: A Hardware Independent Graphics and Windows Library

Vinod Anupam
Chanderjit L. Bajaj
Andrew Burnett
Malcolm C. Fields
Andrew V. Royappa

Report Number:
91-062
XS: A HARDWARE INDEPENDENT GRAPHICS AND WINDOWS LIBRARY

Vinod Anupam, Chanderjit L. Bajaj
Andrew Burnett, Malcolm C. Fields
Andrew V. Royappa, Daniel R. Schikore

CSD-TR-91-062
August 1991
XS: A Hardware Independent Graphics and Windows Library*

Vinod Anupam
Chanderjit L. Bajaj
Andrew Burnett
Malcolm C. Fields
Andrew V. Royappa
Daniel R. Schikore

Department of Computer Science,
Purdue University,
West Lafayette, IN 47907

Abstract

The XS Graphics Libraries are a suite of 3D graphics libraries that access system-dependent graphics facilities (and hardware) in a uniform, system-independent manner. Each system supported is represented by a single library in the suite. All libraries in the suite present the same function-call interface. In this way, an application program can maintain source-level portability across several systems by simply linking with appropriate members of the XS suite. The XS suite is now used by all SHASTRA applications. SHASTRA is a collection of toolkits that deals with the construction and manipulation of geometric objects. By its very nature it relies heavily on high-performance computer graphics. The SHASTRA environment aims to provide distributed, collaborative geometric design across a heterogeneous workstation environment. It was therefore necessary to achieve truly portable computer graphics without suffering the usual loss in performance. The XS suite was designed and built as a solution to this problem. The current members of the XS suite consist of the X Window System (X11 Release 4) on SUN workstations and other X11 platforms, the GL graphics library specialized for SGI workstations, the STARBASE graphics library on HP systems and Windows 3.0 on IBM compatible PCs.

*Supported in part by NSF grant CCR 90-02228 and AFSOR contract 91-0276
CONTENTS

Contents

1 Introduction......................................................... 3

2 Design Objectives.................................................. 3

3 Library Functionality............................................... 4
   3.1 Preliminary Definitions...................................... 4
   3.2 Initialization Routines...................................... 4
   3.3 Window Manipulation and Input............................... 6
   3.4 Matrix Stack Operations..................................... 8
   3.5 Modeling Transformations.................................... 9
   3.6 Drawing Functions........................................... 10
   3.7 Viewing and Perspective Transformations.................. 12
   3.8 Shading and Rendering....................................... 13
   3.9 Screen and Window Dumps.................................... 13

4 Library Suite..................................................... 14

5 Example Applications.............................................. 19

A Appendix: C Code with XS calls for the Cubes demo........... 24
1 Introduction

At this time, many graphics systems support the X11 graphics libraries, toolkits and widget sets. The widget sets provide powerful, system-independent facilities for building user interfaces. However, each system generally has its own 3D graphics library. These libraries are often system dependent, usually because they access specialized graphics hardware. Under these circumstances, an application that requires a sophisticated user interface and high-powered graphics cannot maintain source-level portability across different systems.

The XS Graphics Libraries are a suite of 3D graphics and windows libraries that access system-dependent graphics facilities (and hardware) in a uniform, system-independent manner. Each system supported is represented by a single library in the suite. All libraries in the suite present the same function-call interface. In this way, an application program can maintain source-level portability across several systems by simply linking with appropriate members of the XS suite.

The XS libraries can be used in conjunction with any user interface management system (e.g. the X Window System). The client will, in almost all ways, be a normal toolkit application, except for any windows created through the XS library. These will be created by the underlying graphics system. Through the functions in the XS library, the application can perform 3D graphics operations in these windows, and utilize native graphics facilities. The XS libraries thus present a standard interface to 3D graphics on a variety of powerful graphics platforms.

The SHASTRA project[2] is an attempt to integrate three separate toolkits[1, 4, 5] that address different aspects of this field. The primary goal of this project is true distributed geometric design across a heterogeneous workstation environment: it was therefore necessary to achieve truly portable computer graphics without suffering the usual loss in performance. The XS suite was our solution to this problem. Our experiences in designing and using XS are presented in [3].

This report details the architecture and the functionality of the XS suite in section 3. Example applications utilizing XS are presented in section 5. The C-code of a simple example illustrating the use and portability of XS is given in the Appendix.

2 Design Objectives

1. [Standard Interface]

XS aims to provide an interface such that any application that requires a sophisticated user interface and high-powered graphics can maintain source-level portability across different systems.

The primary goal of the system is to provide access to system-dependent graphics facilities (and hardware) in a uniform, system-independent manner, and thus present a standard interface to 3D graphics on a variety of powerful graphics platforms.

2. [Desired Functionality]
Graphics platforms usually provide a very rich instruction set, which is finely tuned to their hardware. Ideally, we would like to have much or all of this functionality available through a device independent interface. However, the essence of the idea of device independence necessitates the abstracting out of highly device specific functionality, and providing functionality which is likely to be available and consistent on all systems. Minimally a device independent 3D graphics interface should support the following operations:

(a) **Multiple Window Manipulation and Input**: The system should provide applications with a method of creating and manipulating multiple graphics windows. Also, it should a low level event handling mechanism for input through those windows.

(b) **Drawing Functions**: A 3D graphics system will, minimally, provide a substrate for drawing points, lines and polygons.

(c) **Viewing and Perspective Transformations**: Applications can use this mechanism to specify orthographic or perspective projection, and the viewing volume.

(d) **Modeling Transformations**: These should let an application translate, rotate, and scale the current view space.

(e) **Shading and Rendering**: This will provide the application with methods to control the lighting model and other factors for shaded display.

(f) **Screen and Window Dumps**: This will enable an application to store images in the 3D graphics windows, for possible future redisplay and hardcopying.

### 3 Library Functionality

#### 3.1 Preliminary Definitions

- **Matrix** – A transformation matrix in XS is a 4x4 array of type `double`.

- **Angle** – Angles in XS are given by integers which represent tenths of degrees.

- **Screen coordinate system** – The XS screen coordinate system has its origin at the lower left of the screen, with X and Y coordinates increasing to the right and upwards, respectively. Each XS window has its own coordinate system with the origin at the lower left corner, with X and Y increasing to the right and upwards, respectively. The units of X and Y for screen coordinates are pixels. The only use for screen coordinates in XS is to determine window placement on the screen, and mouse cursor placement within a window.

#### 3.2 Initialization Routines

- **Initialize XS**
Figure 1: The XS library architecture
void xsInitialize(xac, wg_top)
XtAppContext xac;
Widget wg_top;

Initializes the XS variables. Should be called at the beginning of any XS application.

3.3 Window Manipulation and Input
XS graphics can only be drawn in XS windows. The following functions allow for creation and management of XS windows.

- Adding a Window
  void xsAddWindow(width, height, allow_resize, keep_aspect, name, wid)
  int width, height, allow_resize, keep_aspect, *wid;
  char *name;

  Open a new XS Window with the given width and height, specifying whether the user can resize, and whether the aspect ratio should be maintained. The window title is set to name, and the XS window id is returned in wid.

- Setting Current Window
  void xsWinSet(wid)
  int wid;

  Sets the current XS window to the window with the given wid.

- Getting Window id
  void xsWinGet(wid)
  int *wid;

  Returns the id of the current XS window in wid.

- Changing Window Title
  void xsWinTitle(title, wid)
  char *title;
  int wid;

  Change the title of the XS window with the given wid.

- Window Information
void xsGetWinInfo(width, height, x, y)
int *width, *height, *x, *y;

Returns the width, height, x coordinate and y coordinate of the current window in
the screen coordinate system.

- Removing a Window

void xsRemoveWindow(wid)
int wid;

Removes the XS window with the given wid. If wid is the current window, the current
window becomes undefined.

- Notification of Window Change

void xsRegisterWindowChange(func)
void (*func)(int);

XS will call the function func whenever the mouse moves into or out of an XS window.
If the mouse moves into an XS window, the id of that window will be passed to the
function. If it moves into the root window, -1 will be passed.

- Notification of Mouse Movement

void xsQueueMotionMode(flag)
int flag;

If flag is TRUE, then mouse motions will trigger calls to the mouse callback function.
Otherwise, only mouse button changes will cause the callback function to be called.

- Notification of Mouse Activity

void xsRegisterMouse(func)
void (*func)(int, int, int, int, int);

XS will call the function func whenever a button is pressed or released in an XS
window. Depending on the last call to xsQueueMotionMode(), this function may be
called every time that mouse motion is detected in an XS window. The parameters
passed to the function are the id of the XS window, the event that occurred (defined by
the constants XSMOTION, XSBUTTONUP, and XSBUTTONDOWN), the X and
Y coordinates of the mouse, and if the event was a button event, the last para-
ter indicates which button was detected called (defined by the constants XSLEFT,
XSMIDDLE, XSRIGHT).
3 LIBRARY FUNCTIONALITY

- Notification of Window Exposure

```c
void xsRegisterRedisplay(func)
void (*func)(int);
```

XS will call the function `func` whenever a change in the windows exposes a part of an XS window. The parameter passed to the function is the id of the XS window which was exposed.

- Find position of Mouse

```c
void xsGetMousePos(x, y)
ing *x, *y;
```

Returns the x and y coordinates of the mouse in the current window’s coordinate system.

3.4 Matrix Stack Operations

All graphics in XS are drawn with respect to the top matrix in a stack of transformation matrices. The following functions allow the user to manipulate the stack.

- Make an Identity Matrix

```c
void xsIdMatrix(M)
double M[4][4];
```

Copies the identity matrix into the matrix `M`.

- Multiply onto stack

```c
void xsMultMatrix(M)
double M[4][4];
```

Left multiplies the matrix `M` with the matrix on the top of the matrix stack. The result replaces what was the top of the matrix stack.

- Duplicate Top Matrix

```c
void xsDupMatrix()
```

Makes a duplicate of the top matrix on the stack and pushes it onto the stack.

- Replace Top Matrix
3 LIBRARY FUNCTIONALITY

void xsReplaceMatrix(M)
double M[4][4];

Replaces the top matrix of the stack with the matrix M.

• Pop Matrix

void xsPopMatrix()

Pops the top matrix off of the matrix stack.

• Get Top Matrix

void xsGetMatrix(M)
double M[4][4];

Copies the top matrix of the stack into the matrix M without affecting the contents of
the stack.

• Load Matrix onto Stack

void xsLoadMatrix(M)
double M[4][4];

If the argument is NULL, nothing is done. Otherwise, the matrix M is pushed onto
the stack.

• Unload Matrix off Stack

void xsUnloadMatrix(M)
double M[4][4];

If the argument is NULL, nothing is done. Otherwise, pops the top matrix of the
stack into the matrix M.

3.5 Modeling Transformations

The following functions perform modeling transformations by multiplying a new transfor-
mation matrix with the matrix on the top of the matrix stack. If the matrix argument M
is not NULL, then it is pushed onto the stack before the multiplication, and popped off of
the stack after and returned. In this case the contents of the stack remain unchanged.

• Translation
void xsTranslate(tx, ty, tz, M)
double tx, ty, tz;
double M[4][4];

Translate along the X axis by tx, along the Y axis by ty, and along the Z axis by tz.

• Scaling

void xsScale(sx, sy, sz, M)
double sx, sy, sz;
double M[4][4];

Scale along the X axis by sx, along the Y axis by sy, and along the Z axis by sz.

• Rotation

void xsRotate(theta, A, M)
int theta;
char A;
double M[4][4];

Rotate around the axis A by an angle of theta. The axis must be one of the characters 'x', 'y', or 'z'.

3.6 Drawing Functions

The following functions produce graphic output in the current XS window. If the matrix M is not NULL, then it is pushed onto the stack before the drawing operation, and popped off after. All drawing functions use the top matrix of the stack as the transformation matrix.

• Drawing a Point

void xsPoint(p, M)
double p[3];
double M[4][4];

Draw the point p given by (p[0], p[1], p[2]).

• Drawing a Line

void xsLine(p0, p1, M)
double p0[3], p1[3];
double M[4][4];
Draw the line from point $p_0$ to point $p_1$.

- **Drawing a Polygon**

  ```c
  void xsPolygon(n, vertices, normals, shade, wireframe, M)
  int n, shade, wireframe;
  double vertices[][3], normals[][3];
  double M[4][4];
  ```

  Draw a polygon with $n$ vertices. If shade is TRUE, the polygon will be filled and shaded according to the current lighting model. If wireframe is TRUE, the outline of the polygon will be drawn. Each vertex in `vertices` should have a corresponding normal in `normals`. The normals need not be of unit length but must be non-zero.

- **Clearing the Window**

  ```c
  void xsClear()
  ```

  Clears the current XS window in the current drawing color.

- **Setting the Drawing Color**

  ```c
  void xsSetRGB(r, g, b)
  double r, g, b;
  ```

  Sets the current drawing color given the red, green, and blue components, each ranging from 0.0 - 1.0. Subsequent points, lines and polygon outlines will be drawn in this color. This color does not affect the interior of polygons: that depends on whether a polygon is to be shaded, and if so, on the current shading parameters (see `xsInitShade,xsShade`).

The following functions are intended to enclose a series of drawing commands.

- **Begin Drawing**

  ```c
  void xsBeginDraw(M)
  double M[4][4];
  ```

  Prepares XS to perform graphics calls. If $M$ is not NULL, it is pushed onto the matrix stack.

- **End Drawing**

  ```c
  void xsEndDraw(M)
  double M[4][4];
  ```

  `xsEndDraw` signifies the end of a series of drawing commands. If the matrix $M$ is not NULL, the top matrix of the stack is popped into $M$, otherwise the stack is unchanged. If double buffering is supported and active, the buffers are swapped at this time.
3.7 Viewing and Perspective Transformations

The following functions define the type of projection and view volume. The viewer is assumed to be in standard position, located at the origin and looking in the direction of the −Z axis. The twist angle is 0 degrees from the Y axis (hence, the Y axis is the standard "up" direction).

- **Orthographic Projection**
  ```c
  void xsOrthographic(left, right, bottom, top, near, far)
  double left, right, bottom, top, near, far;
  ```
  Defines a view volume with clipping planes at \( X=\text{left} \) and \( X=\text{right} \), \( Y=\text{top} \), \( Y=\text{bottom} \), \( Z=-\text{near} \) and \( Z=-\text{far} \), relative the viewer in standard position.

- **Perspective Projection**
  ```c
  void xsPerspective(fovy, aspect, near, far)
  int fovy;
  double aspect, near, far;
  ```
  Defines a view volume with an angle of fovy in the Y direction, and the given aspect ratio between \( X \) and \( Y \), with clipping planes at the distances near and far which are positive distances from the viewer. The value for far must be greater than the value of near.

The following functions transform the viewer from an arbitrary position to standard position.

- **Line-of-sight positioning**
  ```c
  void xsView(eye, ref, twist, M)
  double eye[3], ref[3];
  int twist;
  double M[4][4];
  ```
  Position the viewer to look from point eye to point ref, with an angle of twist about the resulting line-of-sight.

- **Polar positioning**
  ```c
  void xsPolarView(dist, azim, inc, twist, M)
  double dist, M[4][4];
  int azim, inc, twist;
  ```
  The position of the viewer is defined to be at the distance dist from the origin, with an azimuthal angle of azim in the XY plane, measured from the y axis (\( \theta \) in spherical coordinates), an angle of incidence of inc, measured from the Z axis (\( \phi \) in spherical coordinates), and an angle of twist about the line-of-sight.
3.8 Shading and Rendering

• Lighting

void xsDefaultLight(index)
int index;

This initializes the lighting model for the current window. Each window must have a unique index as an argument to this function, which specifies one of the predefined lighting models.

• Initializing the Table of Shades

void xsInitShade(ptab, pn)
char ***ptab;
int *pn;

This function will initialize hardware-dependent polygon shading facilities, if any. It will then return in ptab an array of strings. The i'th member of the array will be the name of the i'th shade defined. A user interface could allow the user to select a particular shade. The number of shades is returned via the integer pointer pn, and will be zero when shading is not available. In this case, polygons will have empty interiors even when shading is requested.

• Setting the Current Shade

void xsShade(index)
int index;

The argument index to this function specifies one of the predefined shading models. Displayed polygons will use this shading model until it is altered by another xsShade() call.

3.9 Screen and Window Dumps

• Dump to PostScript File

void xsDumpToFile(file)
char *file;

In the SGI version of XS, a grf format dump of the current XS window will be written to the file file. In the X version, the user can choose the window to dump by clicking the mouse. A full screen dump can be chosen by clicking in the root window. The file created by the X version of XS will be a postscript file.
4 Library Suite

xs/

|------ xs.h: general XS header file |
| xInitialize(xacAppContext, wgTopWidget) |
| xGetOtherConnection(pfd, pf) |
| xRegisterMouse(fn) |
| xRegisterRedisplay(fn) |
| xRegisterWindowChange(fn) |
| xPoint(P) |
| xLabelledPoint(P) |
| xLine(P1, P2) |
| xMultiLine(number, vertices) |
| xPolygon(vc, Vertices, Normals) |
| xRotate(angle, axis) |
| xTranslate(tx, ty, tz) |
| xScale(sx, sy, sz) |
| xPerspective(FoV, aspect, Near, Far) |
| xOrthographic(Left, Right, Bottom, Top, Near, Far) |
| xPolarView(Dist, Azim, Inc, Twist) |
| xView(Eye, Reference, Twist) |
| xBeginDraw() |
| xEndDraw() |
| xIdMatrix(Matrix) |
| xLoadMatrix(Matrix) |
| xReplaceMatrix(Matrix) |
| xUnloadMatrix(Matrix) |
| xMultMatrix(Matrix) |
| xDupMatrix() |
| xPopMatrix() |
| xGetMatrix(Matrix) |
| xClear() |
| xGetMousePos(px, py) |
| xWinSet(WinId) |
| xWinGet(pWinId) |
| xAddWindow(width, height, resize, aspect, name, pwid) |
| xRemoveWindow(WinId) |
| xWinTitle(title, WinID) |
| xQueueMotionMode(flag) |
| xGetWinInfo(pWidth, pHeight, pxOrigin, pyOrigin) |
| xSetRGB(r, g, b) |
Figure 2: The XS library suite
xsInitializeConfig()
xsDefaultLight(index)
xsInitShade(pnames, pnumber)
xsShade(index)
xsDumpToFile(fname)
xsHandleXEvent()

--- SGI/
|-- xsa.h: SGI-specific header files

|-- xsa_init.c: SGI-specific initialization routines:
   | xsAlternateInitialize(iLength, iWidth)

|-- xsa_xinit.c: XS initialization routines:

|-- xsa_xevent.c: Handle X events in XS routines:

|-- xsa_event.c: SGI-specific event handler routines:

|-- xsa_calls.c: SGI-specific drawing routines:
   | xsSetup(width, height, resize, aspect, name, pwid)
   | xsInitializeConfig()
   | findGLWindow(i)
   | HandleSGIEvent()

|-- xsa_light.c: SGI-specific shading/lighting routines:
   | LightsOff()
   | DefineDefaultLight()
   | UseDefaultLight()
   | DefineSkinLight()
   | UseSkinLight()
   | DefineCyanPlasticLight()
   | UseCyanPlasticLight()
   | DefineGreyPlasticLight()
   | UseGreyPlasticLight()
   | DefineYellowPlasticLight()
   | UseYellowPlasticLight()
   | DefineRedPlasticLight()
   | UseRedPlasticLight()
DefineGreenPlasticLight()
UseGreenPlasticLight()
DefineBluePlasticLight()
UseBluePlasticLight()
DefinePinkPlasticLight()
UsePinkPlasticLight()
DefineLapisLight()
UseLapisLight()
DefineBrownStoneLight()
UseBrownStoneLight()
DefineBronzeLight()
UseBronzeLight()
DefineRedRubberLight()
UseRedRubberLight()
DefinePewterLight()
UsePewterLight()
DefinePlasterLight()
UsePlasterLight()
DefineSilverLight()
UseSilverLight()
DefineGoldLight()
UseGoldLight()
DefineMetalLight()
UseMetalLight()
DefineBrassLight()
UseBrassLight()
DefineLightingModels()

-- xsa_print.c: SGI-specific window dump routines:

-- colorquant.c: color quantization for 24-bit image
    colorquant(r, g, b, pixels, colormap, colors, bits, rgbmap, fast)
    bzero(pa, n)
    QuantHistogram(r, g, b, box)
    CutBoxes(boxes, colors)
    GreatestVariance(boxes, n)
    BoxStats(box)
    CutBox(box, newbox)
    FindCutpoint(box, color, newbox1, newbox2)
    UpdateFrequencies(box1, box2)
    ComputeRGBMap(boxes, colors, rgbmap, bits, colormap, fast)
    SetRGBMap(boxnum, box, rgbmap, bits)
find_colors(boxes, colors, rgbmap, bits, colormap)
getneighbors(boxes, num, neighbors, colors, colormap)
makenearest(boxes, boxnum, nneighbors, neighbors, rgbmap, bits, colormap)

--- HP/

| xsa.h: X-specific header files and definitions
| xsa_calls.c: HP starbase graphics routines
| RotateUnknownAngle(sin, cos, axis, Matrix)
| SBsetup(win)
| XSActionProc(w, xevp, params, nparams)
| XSError(proc, msg)

| xsa_matrix.c: matrix manipulation routines:
| LightSources()
| UseCopperLight()
| UseRubberLight()
| UsePlasticLight()
| UseObsidianLight()
| UsePotteryLight()
| UseBrassLight()

--- X11/

| xsa.h: X-specific header files and definitions
| xsa_calls.c: X-specific drawing routines:
| ClipPoint(p)
| ClipSegment(p1, p2)
| RotateUnknownAngle(sin, cos, axis, mat)
| XSActionProc(w, xevp, params, nparams)
| XSError(proc, msg)

| xsa_matrix.c: matrix manipulation routines:
| InitMatrixStack()
| MultMatrix(mat1, mat2, mat3)
5

Example Applications

- **Cubes** is a demonstration of using XS graphics in more than one window. Cubes are displayed in each of the two windows in this demonstration. One of the cubes is shaded, while the other is not. See figure 5. Each cube has its own transformation and they can be resized and rotated independently, using the mouse.

- **Graph** is a parametric surface plotter. It uses a predefined grid size and calculates the values of the function over the grid. The surface can be displayed shaded or in wireframe. The mouse can be used to resize and rotate the surface. A new function can be chosen by editing the parameters and clicking the "OK" button.

- **Solid** allows the user to create a shape and extrude it into a solid. The shape is formed by clicking the mouse in the window area, and lines are drawn between the points clicked. When the extrude button is clicked, the first and last points are joined to form a polygonal shape, and the polygon is extruded in the z direction to form a solid. The solid can be rotated using the mouse.

- **Spiro** is an example demonstrating uses of rotation. It is a computer version of a spirograph. First, the user is allowed to resize the two circles, using the left and
Figure 3: Wireframe and Shaded Display using XS

Figure 4: Curve and Shaded Surface Display using XS in GANITH
Figure 5: Wireframe and Shaded Solid Object Display using XS in SHILP

Figure 6: Shaded Display of Human Anatomy Models using XS in VAIDAK
middle buttons to resize the outer and inner circles, respectively. By clicking the "Begin Demo" button, the demonstration will start. The inner circle rolls around the inside of the outer circle, while a point on the inner circle is traced. A total of 12 revolutions are drawn, and then the user can start over.

- **Zbuff** is an example of intersecting polygons which are shaded, demonstrating zbuffering features of some computers. The polygons can be rotated using the mouse.

- **GANITH** is an algebraic geometry toolkit, part of the SHAstra system [2], which manipulates multivariate polynomials and power series [5]. It can be used to solve a system of algebraic equations and visualize its multiple solutions. Example applications of this for geometric modeling and computer graphics are curve and surface display, curve-curve intersections, surface-surface intersections, global and local parameterizations, implicitizations, and inversions. It also incorporates techniques for multivariate interpolation and least-squares approximation to an arbitrary collection of points and curves. See figure 4.

- **SHILP** is another toolkit which is part of the SHAstra system, for the geometric design of solids defined with algebraic surface boundaries [1]. Curves and surfaces can be represented in both implicit and rational parametric form, in either power or Bernstein polynomial bases. See figure 5. The current functionality of the toolkit includes restricted extrude, revolve and offset operations, edit operations on planar lamina and solids, fleshing of wireframes with interpolating algebraic surfaces, and color rendering and animation of solids. For the purpose of finite element analysis, we allow the decomposition of arbitrary polyhedra with holes into convex pieces, as well as bounded aspect ratio triangulations of both the boundary and interior of solids.

- **VAIDAK** is a medical imaging and model reconstruction toolkit and part of the SHAstra system which manipulates medical image volume data and constructs accurate surface and solid models of skeletal and soft tissue structures from CT (Computed tomography), MRI (magnetic resonance imaging), or laser surface imaging data [4]. See figure 6. VAIDAK incorporates both heuristic and exact methods of contouring image data, active thresholding, tiling or polygon reconstruction, and curved surface patch reconstruction. It also incorporates a browser feature to modify the contours, a scanner to view image data and interactively pick threshold values, and a render window to change lighting and display modes.
References


A Appendix: C Code with XS calls for the Cubes demo

/---------------------------------------------------------------------------
* cubedemo.c -- two windows with cubes, one shaded, one not. 
*---------------------------------------------------------------------------
*/

#include <stdio.h>
#include <math.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Xaw/Command.h>
#include <X11/Xaw/Form.h>
#include <X11/Xaw/Box.h>
#include <X11/Shell.h>
#include "xs.h"

#define X 0
#define Y 1
#define Z 2

#define RESIZE 0
#define ROTATE 1

static
 Widget
 wgt, /* top level widget */
 wgtControlShell, wgControl, /* control panel shell and form */
 wgresize, wgtrotate, wgtquit, wgm[20];

int DebugLevel = 100;

XtAppContext xac;

/---------------------------------------------------------------------------
* Main program. 
*---------------------------------------------------------------------------
*/
int winid1, winid2, oldx, oldy, first=TRUE, mouseaction;

double facepoints[6][4][3] = {
  {1,1,1}, {1,1,1}, {1,1,1}, {1,1,1},
  {1,1,-1}, {1,1,-1}, {1,1,-1}, {1,1,-1},
  {1,-1,1}, {1,-1,1}, {1,-1,1}, {1,-1,1},
  {-1,1,1}, {-1,1,1}, {-1,1,1}, {-1,1,1},
  {-1,-1,1}, {-1,-1,1}, {-1,-1,1}, {-1,-1,1},
  {-1,-1,-1}, {-1,-1,-1}, {-1,-1,-1}, {-1,-1,-1},
};

double facenormals[6][4][3] = {
  {-.577, -.577, -.577}, {-.577, -.577, .577}, {-.577, .577, .577}, {-.577, .577, -.577},
  {-.577, -.577, -.577}, {-.577, -.577, .577}, {.577, -.577, .577}, {.577, -.577, -.577},
  {.577, -.577, -.577}, {.577, -.577, .577}, {.577, .577, -.577}, {.577, .577, .577},
  {.577, .577, -.577}, {.577, .577, .577}, {-.577, .577, -.577}, {-.577, .577, .577},
  {-.577, -.577, -.577}, {-.577, -.577, .577}, {.577, -.577, .577}, {.577, -.577, -.577},
  {-.577, -.577, -.577}, {-.577, -.577, .577}, {-.577, .577, -.577}, {-.577, .577, .577},
};

#define BLACK  0.0,0.0,0.0
#define RED    1.0,0.0,0.0
#define GREEN  0.0,1.0,0.0
#define BLUE   0.0,0.0,1.0
#define GRAY   0.8,0.8,0.8
#define WHITE  1.0,1.0,1.0

/* Draws a cube centered at the origin, shaded==TRUE */
/* will cause the cube to be shaded, otherwise it will be drawn wireframe */
DrawCube(shaded)
  int shaded;
  {
    int face;

    xsBeginDraw(NULL);
    xsSetRGB(WHITE);
    xsClear();
    xsSetRGB(BLUE);
    for (face=0; face<6; face++)
      xsPolygon(4, facepoints[face], facenormals[face], shaded, !shaded, NULL);
    xsEndDraw(NULL);
  }
APPENDIX: C CODE WITH XS CALLS FOR THE CUBES DEMO

/* window 1 is shaded, while window 2 is wireframe */
Drawcubes()
{
    xsWinSet(winid1);
    DrawCube(TRUE);
    xsWinSet(winid2);
    DrawCube(FALSE);
}

SetProjection(wid)
int wid;
{
    int width, height, x,y;
    double neww, newh;

    xsWinSet(wid);
    xsGetWinInfo(&width, &height, &x, &y);

    if (width > height) {
        newh=6.0;
        neww=(6.0*width)/height;
    } else {
        neww=6.0;
        newh=(6.0*height)/width;
    }

    xsOrthographic(-neww, neww, -newh, newh, -100.0, 100.0);
}

RedrawCB(wid)
int wid;
{
    SetProjection(wid);
    DrawCube(wid==winid1);
}

/* callback for resize button */
static void ResizeCB(wg, xpClient, xpCall)
    Widget wg;
APPENDIX: C CODE WITH XS CALLS FOR THE CUBES DEMO

```c
XtPointer xpClient;
XtPointer xpCall;
{
    mouseaction=RESIZE;
}

/* callback for rotate button */
static void RotateCB(vg, xpClient, xpCall)
    Widget vg;
    XtPointer xpClient;
    XtPointer xpCall;
{
    mouseaction=ROTATE;
}

/* callback for quit button */
static void QuitCB(vg, xpClient, xpCall)
    Widget vg;
    XtPointer xpClient;
    XtPointer xpCall;
{
    exit(0);
}

static void SetShadeCB(vg, xpClient, xpCall)
    Widget vg;
    XtPointer xpClient;
    XtPointer xpCall;
{
    xsShade((int)xpClient);
    DrawCube();
}

/* Handle rotations and scaling */
MouseCB(id, event, x, y, index)
    int id, event, x, y, index;
{
    int dx, dy;
    double factor;

    /* begin tracking mouse at buttondown event */
    if (event==XSBUTTONDOWN) {
```
A  APPENDIX:  C  CODE  WITH  XS CALLS  FOR  THE  CUBES DEMO

```c
oldx=x;
oldy=y;
first=FALSE;
}
/* done tracking mouse when button is released */
else if (event==XSBUTTONUP) {
    first=TRUE;
}
/* when flag is set, dragging of mouse causes transformations */
else if (!first) {
    xsWinSet(id);
    dx=x-oldx;
    dy=y-oldy;
    if (mouseaction == RESIZE) {
        factor=1.0+dx/50.0;
        xsScale(factor, factor, factor, NULL);
    }
    else if (mouseaction==ROTATE) {
        xsRotate(dx, 'y', NULL);
        xsRotate(dy, 'x', NULL);
    }
    /* set the new mouse position */
    oldx=x;
    oldy=y;
    DrawCube(id==winid1);
}
}
handleX() {
    XEvent xev, xevNext;;

    while (XtAppPending(xac)) {
        XtAppNextEvent(xac, &xev);
        if (xev.type==MotionNotify) {
            while (XtAppPending(xac)) {
                XtAppPeekEvent(xac, &xevNext);
                if (xevNext.type != MotionNotify)
```
A  APPENDIX:  C  CODE  WITH  XS  CALLS  FOR  THE  CUBES  DEMO

break;
if (xevNext.xmotion.window != xev.xmotion.window)
    break;
    XtAppNextEvent(xac, &xev);
}
}
    XtDispatchEvent(&xev);
}
    return(0);
}

/* Initialize windows and add callbacks */
main(argc, argv)
int argc;
char *argv[];
{
    double eye[3], ref[3];
    char **lmodels;
    int i, numlmodels, pfd;
    void (*pf)();

    /* Create the top-level widget and initialize application context */
    wgTop = XtAppInitialize(&xac, "Tester", NULL, 0, &argc, argv,
                  NULL, NULL, 0);

    /* Initializations */
    xsInitialize(xac, wgTop);
    xsInitShade(&lmodels, &numlmodels);

    /* Create the control panel */
    wgControlShell = XtCreatePopupShell("control-sh", topLevelShellWidgetClass,
                        wgTop, NULL, 0);
    wgControl = XtCreateManagedWidget("control", boxWidgetClass,
                        wgControlShell, NULL, 0);

    for (i=0; i<numlmodels; i++)
        wgmodels[i]= XtCreateManagedWidget(lmodels[i], commandWidgetClass,
                        wgControl, NULL, 0);

    /* Create the buttons on control panel */
    wgresize = XtCreateManagedWidget("resize", commandWidgetClass,
                        wgControl, NULL, 0);
A  APPENDIX:  C  CODE  WITH XS CALLS FOR THE CUBES DEMO

```c
wgrotate = XtCreateManagedWidget("rotate", commandWidgetClass,
    wgControl, NULL, 0);
wgquit = XtCreateManagedWidget("exit", commandWidgetClass,
    wgControl, NULL, 0);

/* Add callbacks */
XtAddCallback(wgresize, XtNcallback, ResizeCB, NULL);
XtAddCallback(wgrotate, XtNcallback, RotateCB, NULL);
XtAddCallback(wgquit, XtNcallback, QuitCB, NULL);

for (i=0; i<numlmodels; i++)
    XtAddCallback(wgmodels[i], XtNcallback, SetShadeCB, (XtPointer)i);

/* Pop up all widgets in the tree */
XtPopup(wgControlShell, XtGrabNone);

/* first window will be shaded */
xsAddWindow(37, 12, TRUE, FALSE, "Shaded", &winid1);

SetProjection(winid1);
eye[0]=0.0; eye[1]=0.0; eye[2]=6.0;
ref[0]=0.0; ref[1]=0.0; ref[2]=0.0;
xsView(eye,ref,0,NULL);
xsShade(0);

DrawCube(TRUE);

/* this window will just be wireframe */
xsAddWindow(37,12, TRUE, FALSE, "Wireframe", &winid2);

SetProjection(winid2);
eye[0]=0.0; eye[1]=0.0; eye[2]=6.0;
ref[0]=0.0; ref[1]=0.0; ref[2]=0.0;
xsView(eye,ref,0,NULL);

DrawCube(FALSE);

mouseaction=ROTATE;
xsRegisterMouse(MouseCB);
xsQueueMotionMode(TRUE);
xsRegisterRedisplay(RedrawCB);
```
xsGetOtherConnection(&pfd, &pf);

while (1) {
    if (pf != NULL)
        (*pf)();
    handleX();
}
}