Chapter 3

Drawing to a Display

3.1 CMX11 Drawing

The CMX11 parallel drawing routines are extensions of the standard Xlib drawing routines. These routines allow you to draw large sets of points, lines, arcs, rectangles, filled polygons, text strings, or an image array with a single subroutine call. Specifically, CMX11 provides the following drawing primitives:

- **`CMXDrawPoint`**, **`CMXDrawCPoint`**
- **`CMXDrawLine`**, **`CMXDrawCLine`**
- **`CMXDrawRectangle`**, **`CMXDrawCRectangle`**
- **`CMXFillRectangle`**, **`CMXFillCRectangle`**
- **`CMXFillPolygon`**, **`CMXFillCPolygon`**
- **`CMXDrawArc`**, **`CMXDrawCArc`**
- **`CMXFillArc`**, **`CMXFillCArc`**
- **`CMXDrawString`**, **`CMXDrawCString`**
- **`CMXPutImage`**, **`CMXPutImage1`**

In addition, a utility routine, **`CMXClearDisplay`**, is available to fill a display window with the background color or pixmap.

Copyright © 1993 Thinking Machines Corporation

3.1.1 Scalar Drawing

For the convenience of CM Fortran programmers, CMX11 also provides wrappers for several scalar X11 drawing routines. These are:

- **XDrawPoint**
- **XDrawLine**
- **XDrawRectangle**
- **XDrawArc**
- **XDrawString**

The scalar drawing routines work much like the corresponding parallel versions described in this chapter. Man pages for these routines appear in Chapter 6.

3.1.2 Parallel Drawing

These CMX11 routines extend the X routines in two ways. First, where the X routines take single coordinate arguments, the CMX11 routines accept parallel arrays of coordinates and draw all the objects defined in the arrays in a single call. For example, **XDrawPoint** takes an (x, y) coordinate and draws a single point in the window's foreground color; **CMXDrawPoint** takes conformable x and y parallel arrays and draws a point for each data element in the arrays.

NOTE

Primitives are drawn in the "natural" axis order of the geometry: axis 0 varies the most quickly, followed by axes 1 through n. For a one-dimensional geometry, primitive (1) is followed by primitives ($2-n$), in order.

Second, another set of the CMX11 routines allows you to specify a parallel array of color values so that each object has its own color value. For example, **CMXDrawCPoint** adds a color array that conforms to the x and y

Copyright © 1993 Thinking Machines Corporation

coordinate arrays. Each (x, y) point is drawn in the color specified by the corresponding element of the color array.

3.1.3 The X Identifiers

All the drawing primitives require arguments for the X Window System data structures **Display**, **Drawable**, and **GC** (graphics context). CMX11 provides routines that return identifiers for these structures from the CMX11 display identifier returned by **CMXCreateSimpleDisplay** or **CMXCreateDisplayClass**.

The display specifies the connection to an X server. In practice this argument identifies the screen on which you wish to draw the primitive. To get the identifier to use for this argument, use **CMXGetDisplay**.

The drawable specifies the window on the display where you wish to draw the primitive. It is returned for a CMX11 display by **CMXGetDrawable**.

The graphics context controls how the primitive is to be drawn. It provides values for the drawing characteristics to be used. For example, the graphics context defines the foreground and background colors, line width, line style, etc. When you create a CMX11 display with **CMXCreateSimpleDisplay** or **CMXCreateDisplayClass**, these routines create a graphics context for the display window. The foreground color is determined by the display, the background color is set to black, and the other values are set to the standard X settings.

CMX11 contains convenience functions, **CMXSetForeground** and **CMXSetBackground**, for setting the foreground and background colors in the graphics context. The other graphics context values must be set through the X Window System functions. See your X documentation for details on creating and setting a graphics context.

3.1.4 Coordinates

The coordinates passed to the drawing routines are always interpreted as integer distances in pixels relative to the display window. The origin (0,0) is at the upper left corner of the window inside any window border. x is the horizontal distance increasing to the right, and y is the vertical distance increasing downward. Coordinates must be given as integers between 0 and

Copyright © 1993 Thinking Machines Corporation

65535. The practical limit of the coordinate system is determined by the dimensions of the window. If the drawing routines are given coordinates outside the window boundaries, the portions of the objects outside the boundaries are clipped appropriately.

3.1.5 Colors

Color values are integer indices into the current colormap of the display. The range and interpretation of the specific values are determined by the depth of the display and its visual class. See Section 2.3 for a detailed discussion of how to manage colors on the display.

You can set the graphic context's foreground and background colors from your CM Fortran or C* program with the CMX11 routines **`CMXSetForeground`** and **`CMXSetBackground`**.

Remember: If you allocate a range of colors, your color values must include the *base* value returned by **`CMXAllocateColors`** as an offset to your colormap.

3.2 Points

The parallel point-drawing routines light a pixel at each position on the window defined by the (x, y) coordinates:

From CM Fortran:

```
SUBROUTINE CMXDrawPoint(dpy, drawable, gc, x, y)

SUBROUTINE CMXDrawCPoint(dpy, drawable, gc, x, y, color)

INTEGER dpy
INTEGER drawable
INTEGER gc
INTEGER x(array_size), y(array_size)
INTEGER color(array_size)
```


Copyright © 1993 Thinking Machines Corporation

From C*:

```
void CMXDrawPoint(Display *dpy, Drawable drawable,
        GC gc, int:current x, int:current y);

void CMXDrawCPoint(Display *dpy, Drawable drawable,
        GC gc, int:current x, int:current y,
        int:current color);
```

The *x* and *y* arguments are parallel arrays containing the coordinates of the points to be drawn. The *array_size* determines the number of points to be drawn. The arrays must be conformable; that is, they must be of the same size and shape. Both routines draw a point for each data element in the arrays; the first point is drawn at ($x(1), y(1)$); the second at ($x(2), y(2)$), etc.

CMXDrawPoint draws the points in the foreground color specified in the display's graphics context. For **CMXDrawCPoint**, the *color* array is an array of color values conforming to *x* and *y*. Each point is drawn in the color specified in the corresponding element of the *color* array.

Points are drawn using the graphics context (*gc*) values for:

function	**subwindow_mode**	**clip_x_origin**
plane_mask	**clip_y_origin**	**clip_mask**
foreground		

3.3 Lines

The parallel line-drawing routines draw a line between each pair of (x, y) coordinates:

From CM Fortran:

```
CMXDrawLine(dpy, drawable, gc, x1, y1, x2, y2)

CMXDrawCLine(dpy, drawable, gc, x1, y1, x2, y2, color)

INTEGER dpy
INTEGER drawable
INTEGER gc
INTEGER x1(array_size), y1(array_size)
```


Copyright © 1993 Thinking Machines Corporation

```
INTEGER x2(array_size), y2(array_size)
INTEGER color(array_size)
```

From C*:

```
void CMXDrawLine(Display *dpy, Drawable drawable,
        GC gc, int:current x1, int:current y1,
        int:current x2, int:current y2);

void CMXDrawCLine(Display *dpy, Drawable drawable,
        GC gc, int:current x1, int:current y1,
        int:current x2, int:current y2,
        int:current color);
```

CMXDrawLine and **CMXDrawCLine** draw a line for each data element in the coordinate arrays. The *array_size* determines the number of lines to be drawn. The corresponding elements of the *x1* and *y1* arrays supply the beginning points of the lines and the *x2* and *y2* arrays supply the end points. So, the first line is drawn from (*x1*(1), *y1*(1)) to (*x2*(1), *y2*(1)), the second from (*x1*(2), *y1*(2)) to (*x2*(2), *y2*(2)), etc. Unlike **XDrawLines**, the lines defined in **CMXDrawLine** and **CMXDrawCLine** need not be consecutive; that is, they need not be joined. No pixel is drawn more than once.

CMXDrawLine draws in the *gc* foreground color. **CMXDrawCLine** allows each line to have its own color; the *color* array provides the color for the line defined by the corresponding data elements in the coordinate arrays.

Lines are drawn using the values for the following graphics context (*gc*) data elements:

function	**subwindow_mode**	**tile**
plane_mask	**clip_x_origin**	**stipple**
line_width	**clip_y_origin**	**ts_x_origin**
line_style	**clip_mask**	**ts_y_origin**
cap_style	**foreground**	**dash_offset**
fill_style	**background**	**dash_list**

3.4 Rectangles

In the same way, the parallel rectangle routines define a set of rectangles to be drawn:

Version 1.6, October 1993
Copyright © 1993 Thinking Machines Corporation

From CM Fortran:

```
CMXDrawRectangle(dpy, drawable, gc, x, y, width, height)

CMXDrawCRectangle(dpy, drawable, gc, x, y, width, height,
&                 color)

CMXFillRectangle(dpy, drawable, gc, x, y, width, height)

CMXFillCRectangle(dpy, drawable, gc, x, y, width, height,
&                 color)

INTEGER dpy
INTEGER drawable
INTEGER gc
INTEGER x(array_size), y(array_size)
INTEGER width(array_size), height(array_size)
INTEGER color(array_size)
```

From C*:

```
void CMXDrawRectangle(Display *dpy,
         Drawable drawable, GC gc, int:current x,
         int:current y, int:current width,
         int:current height);

void CMXDrawCRectangle(Display *dpy,
         Drawable drawable, GC gc, int:current x,
         int:current y, int:current width,
         int:current height, int:current color);

void CMXFillRectangle(Display *dpy,
         Drawable drawable, GC gc, int:current x,
         int:current y, int:current width,
         int:current height);

void CMXFillCRectangle(Display *dpy,
         Drawable drawable, GC gc, int:current x,
         int:current y, int:current width,
         int:current height, int:current color);
```

The *array_size* determines the number of rectangles to be drawn. The *x* and *y* arrays define the upper left corner of each rectangle to be drawn relative to the origin of the window. The corresponding elements of the *width* and *height* arrays specify the size of each rectangle from the point (*x*, *y*). The

Copyright © 1993 Thinking Machines Corporation

width argument is the number of pixels the rectangle extends to the right from (*x*, *y*), and *height* is the number of pixels downward.

CMXDrawRectangle draws the outline of a rectangle in the foreground color for each data element in the parallel arrays. **CMXFillRectangle** draws a filled rectangle. Note that **CMXFillRectangle** does not light pixels along the right and bottom edge of the area lit by **CMXDrawRectangle**. A drawn rectangle outlines a rectangle from (*x*, *y*) at the upper left corner to (*x* + *width*, *y* + *height*) at the lower right corner. A filled rectangle is a solid rectangle from (*x*, *y*) to (*x* + (*width* − 1), *y* + (*height* − 1)).

CMXDrawCRectangle and **CMXFillCRectangle** allow specification of a color for each rectangle.

Rectangles are drawn using the values for the following graphics context (*gc*) data elements:

function	**subwindow_mode**	**tile**
plane_mask	**clip_x_origin**	**stipple**
line_width	**clip_y_origin**	**ts_x_origin**
line_style	**clip_mask**	**ts_y_origin**
cap_style	**foreground**	**dash_offset**
join_style	**background**	**dash_list**
fill_style		

3.5 Polygons

CMXFillPolygon and **CMXFillCPolygon** draw filled polygons defined by parallel arrays of coordinates. All polygons must contain the same number of points.

From CM Fortran:

```
CMXFillPolygon(dpy, drawable, gc, x, y, shape, mode)

CMXFillCPolygon(dpy, drawable, gc, x, y, shape, mode,
&                    color)

INTEGER dpy
INTEGER drawable
INTEGER gc
INTEGER x(num_points, num_polys)
```


Copyright © 1993 Thinking Machines Corporation

```
CMF$     LAYOUT x(:SERIAL, )
    INTEGER   y(num_points, num_polys)
CMF$     LAYOUT y(:SERIAL, )
    INTEGER   shape
    INTEGER   mode
    INTEGER   color(num_polys)
```

From C*:

```
void CMXFillPolygon (Display *dpy, Drawable drawable,
         GC gc, Xpoint:current points[], int npoints,
         int _shape, int mode);

void CMXFillCPolygon (Display *dpy,
         Drawable drawable, GC gc,
         Xpoint:current points[], int npoints,
         int _shape, int mode, int:current color);
```

The *x* and *y* arguments must be conformable arrays. The corresponding elements of *x* and *y* define the coordinates of the points of the polygons. So, for example, if each polygon is defined by three points, the first polygon is drawn between ($x(1,1)$, $y(1,1)$), ($x(2,1)$, $y(2,1)$), and ($x(3,1)$, $y(3,1)$). **CMXFillPolygon** draws in the *gc*'s foreground color. For **CMXFillCPolygon**, the *color* array supplies the color value for each polygon. So, for example, the first polygon is drawn in the color defined by the *color* array element (1,1).

The *shape* and *mode* arguments are the same as in the X11 routine **XFillPolygon**. The *shape* argument classes the shape of the polygons, allowing the system to use faster algorithms to draw and fill polygons with a simpler shape. Valid values for *shape* are:

Complex The path may self-intersect. The **fill_rule** (**EvenOddRule** or **WindingRule**) defined in the graphics context (*gc*) for the window determines how the shape is filled. (See your X Window System documentation for more information.)

Nonconvex The path does not self-intersect, but the shape is not wholly convex. Specifying **Nonconvex** instead of **Complex** may improve performance. However, if the path is self-intersecting, the graphics results are undefined.

Copyright © 1993 Thinking Machines Corporation

Convex The path is wholly convex. Specifying **Convex** may improve performance. However, if the path is not convex, the graphics results are undefined.

The *mode* argument specifies a coordinate mode, that is, how the sequence of coordinates is to be interpreted. Valid values for *mode* are

CoordModeOrigin
All coordinate values are interpreted in relation to the origin of the window. That is, the coordinates define positions as the number of pixels from the upper left corner of the window.

CoordModePrevious
The first point is defined in relation to the origin of the window, but each succeeding point is defined in relation to the one that precedes it. Thus, each point coordinate is an offset from the position of the previous point. For example, if the first point is located at the window coordinates (20, 30) and the second point coordinates are (10, 15), the second point is drawn at (30, 45).

Polygons are drawn and filled using the values for the following graphics context (*gc*) data elements:

function	**clip_x_origin**	**tile**
plane_mask	**clip_y_origin**	**stipple**
fill_style	**clip_mask**	**ts_x_origin**
fill_rule	**foreground**	**ts_y_origin**
subwindow_mode	**background**	

3.6 Arcs

CMXDrawArc, **CMXDrawCArc**, **CMXFillArc** and **CMXFillCArc** draw or fill a circular or elliptical arc for each data element of the *x, y, width, height, angle1,* and *angle2* parallel arrays.

Version 1.6, October 1993
Copyright © 1993 Thinking Machines Corporation

From CM Fortran:

```
 CMXDrawArc(dpy, drawable, gc, x, y, width, height, angle1,
&           angle2)

 CMXDrawCArc(dpy, drawable, gc, x, y, width, height, angle1,
&           angle2, color)

 CMXFillArc(dpy, drawable, gc, x, y, width, height, angle1,
&           angle2)

 CMXFillCArc(dpy, drawable, gc, x, y, width, height, angle1,
&           angle2, color)

 INTEGER dpy
 INTEGER drawable
 INTEGER gc
 INTEGER x(array_size), y(array_size)
 INTEGER width(array_size), height(array_size)
 INTEGER angle1(array_size), angle2(array_size)
 INTEGER color(array_size)
```

From C*:

```
void CMXDrawArc(Display *dpy, Drawable drawable,
          GC gc, int:current x, int:current y,
          int:current width, int:current height,
          int:current angle1, int:current angle2);

void CMXDrawCArc(Display *dpy, Drawable drawable,
          GC gc, int:current x, int:current y,
          int:current width, int:current height,
          int:current angle1, int:current angle2,
          int:current color);

void CMXFillArc(Display *dpy, Drawable drawable,
          GC gc, int:current x, int:current y,
          int:current width, int:current height,
          int:current angle1, int:current angle2);

void CMXFillCArc(Display *dpy, Drawable drawable,
          GC gc, int:current x, int:current y,
          int:current width, int:current height,
          int:current angle1, int:current angle2,
          int:current color);
```


Copyright © 1993 Thinking Machines Corporation

CMXDrawArc and **CMXFillArc** draw in the current foreground color set in the graphics context *gc*. **CMXFillCArc** and **CMXDrawCArc** take an additional *color* parallel array that specifies the color individually for each arc.

As illustrated in Figure 5, *x*, *y*, *width*, and *height* define the bounding box of the arc, that is, the rectangle that contains the arc. The upper left corner of the rectangle is defined by (*x*, *y*) and the rectangle extends to the right *width* pixels and downward *height* pixels. The center of the arc is at the center of this rectangle, and the curve of the arc is drawn to intersect the sides of the rectangle. If *width* and *height* are the same (the rectangle is square), the arc is circular; if they are different, the arc is elliptical.

The length of the arc is defined by *angle1* and *angle2*. The *angle1* argument specifies the start of the arc relative to the 3:00 position from the center of the bounding rectangle. The *angle2* argument specifies the end of the arc relative to *angle1*. The angles are given in sixty-fourths of a degree; positive values move counterclockwise. 360 * 64 is a complete circle. So, to draw the angle illustrated in Figure 5, we set *angle1* to −5760 (90 x 64) and *angle2* to 144,000 (225 x 64).

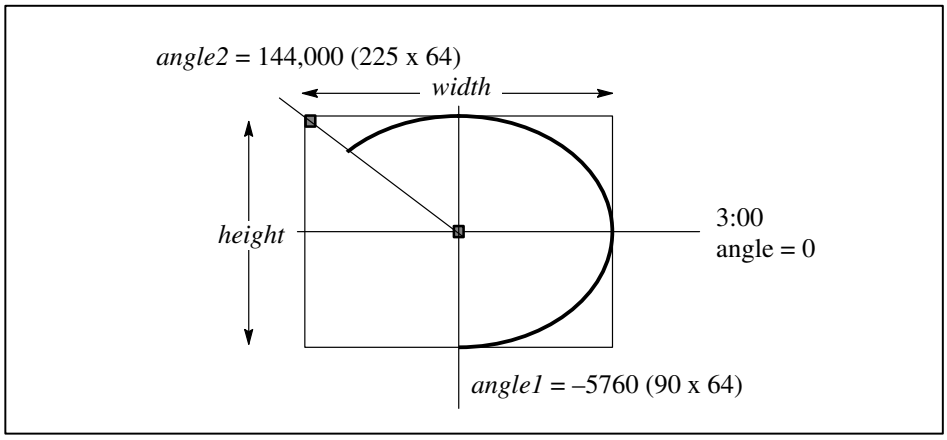


Figure 5. Measuring an arc.

CMXFillArc and **CMXFillCArc** fill the area enclosed by the arc. The exact area is determined by the **arc_mode** set in the graphics context. As shown in Figure 6, if the **arc_mode** is **ArcChord**, the arc is filled to a line joining the endpoints of the arc. If **arc_mode** is **ArcPieSlice**, the boundary of the arc is defined by line segments from the endpoints of the arc to its center point.

Copyright © 1993 Thinking Machines Corporation

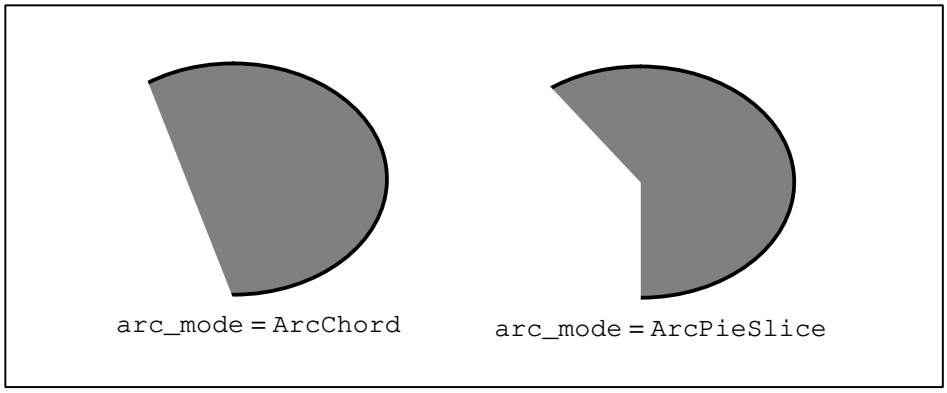


Figure 6. Arc fill patterns.

CMXDrawArc and **CMXDrawCArc** draw arcs using the values for the following graphics context (*gc*) data elements:

function	**subwindow_mode**	**tile**
plane_mask	**clip_x_origin**	**stipple**
line_width	**clip_y_origin**	**ts_x_origin**
line_style	**clip_mask**	**ts_y_origin**
cap_style	**foreground**	**dash_offset**
join_style	**background**	**dash_list**
fill_style		

CMXFillArc and **CMXFillCArc** fill arcs using the values for the following graphics context (*gc*) data elements:

function	**clip_x_origin**	**tile**
plane_mask	**clip_y_origin**	**stipple**
fill_style	**clip_mask**	**ts_x_origin**
arc_mode	**foreground**	**ts_y_origin**
subwindow_mode	**background**	

3.7 Strings

CMXDrawString writes a set of strings to the specified drawable in the foreground color. **CMXDrawCString** draws each string in the color defined by the corresponding element of the *color* array.

Copyright © 1993 Thinking Machines Corporation

From CM Fortran:

```
CMXDrawString(dpy, drawable, gc, x, y, string, slen)

CMXDrawCString(dpy, drawable, gc, x, y, string, slen, color)

INTEGER dpy
INTEGER drawable
INTEGER gc
INTEGER x(array_size), y(array_size)
INTEGER string(array_size)
INTEGER slen
INTEGER color(array_size)
```

From C*:

```
void CMXDrawString(Display *dpy, Drawable drawable,
    GC gc, int:current x, int:current y,
    char:current *string, int slen);

void CMXDrawCString(Display *dpy, Drawable drawable,
    GC gc, int:current x, int:current y,
    char:current *string, int slen, int:current color);
```

The coordinates *(x, y)* define the origin of the text string; *y* is the baseline row of pixels and *x* is the point from which the text string's lbearing, rbearing, and width are measured. The *array_size* determines the number of strings to be drawn.

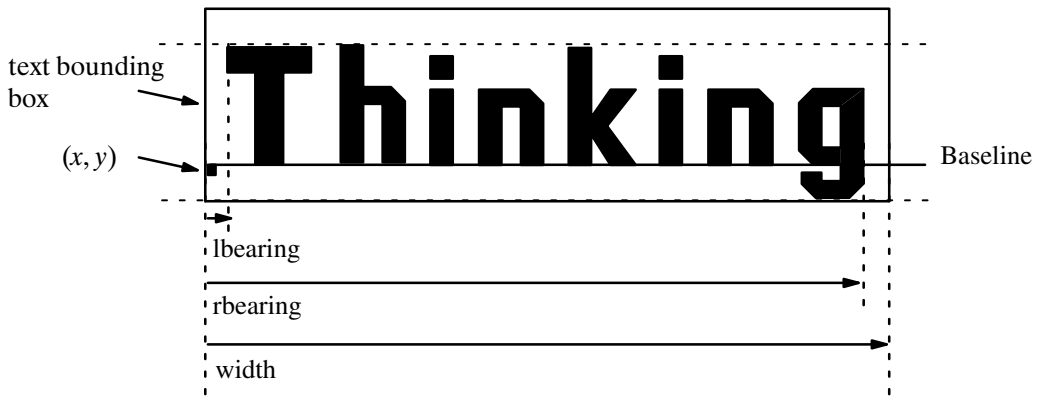


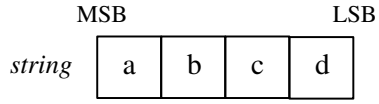
Figure 7. Positioning a text string.

Copyright © 1993 Thinking Machines Corporation

The CM Fortran programmer must load the individual characters into the *string* parallel array. For example, the following lines of CM Fortran code pack the string “abcd” into *string*:

```
          INTEGER STRING(1,100)
CMF$           LAYOUT STRING(:SERIAL,:NEWS)
          STRING(1,:) = ICHAR("a") * 65536 * 256
     &                  + ICHAR("b") * 65536
     &                  + ICHAR("c") * 256 + ICHAR("d")
```

This results in a character being loaded into each byte of the *string* variable in all 100 elements:



From C*, individual string characters can be set normally, since the array elements are characters:

```
char:current string[4];

string[0] = 'a';
string[1] = 'b';
string[2] = 'c';
string[3] = 'd';
```

CMXDrawString draws the given string into the drawable using only the *gc* **foreground** color to draw bits set in the font defined in *gc*. **CMXDrawCString** draws each string in the color defined by the corresponding element of the *color* array. **CMXDrawString** and **CMXDrawCString** draw the string using the values for the following graphics context (*gc*) data elements:

function	**clip_x_origin**	**tile**
plane_mask	**clip_y_origin**	**stipple**
fill_style	**clip_mask**	**ts_x_origin**
font	**foreground**	**ts_y_origin**
subwindow_mode		

Copyright © 1993 Thinking Machines Corporation

3.8 Images

CMXPutImage and **CMXPutImage1** transfer an image stored in the *data* parallel array to the X display. Each element of the array corresponds to a pixel of the displayed image.

From CM Fortran:

```
 CMXPutImage(dpy, drawable, gc, data, depth, src_x, src_y,
&                dst_x, dst_y, width, height)

 CMXPutImage1(dpy, drawable, gc, data, depth, src_x, src_y,
&                dst_x, dst_y, width, height)

 INTEGER dpy
 INTEGER drawable
 INTEGER gc
 INTEGER data(array_size, array_size)
 INTEGER depth
 INTEGER src_x, src_y
 INTEGER dst_x, dst_y
 INTEGER width, height
```

From C*:

```
void CMXPutImage(Display *dpy, Drawable drawable,
    GC gc, int:current data, int depth, int src_x,
    int src_y, int dst_x, int dst_y, int width,
    int height);

void CMXPutImage1(Display *dpy, Drawable drawable,
    GC gc, bool:current data, int depth, int src_x,
    int src_y, int dst_x, int dst_y, int width,
    int height);
```

The *data* describes an image on the CM that is to be combined with a rectangle in the display window.

The *data* elements are integer color values. The depth of the display drawable must be 1 or 8 bits, and the *data* array elements must be 8 or 32 bits.

CMXPutImage combines the color values in the *data* array with the pixel values already present in the display according to the value of **function** in the *gc* graphics context (which, by default, replaces the destination pixel with the source). The first data element (1, 1) will be combined with the pixel value

Copyright © 1993 Thinking Machines Corporation

in the display window at (*dst_x, dst_y*); element (2, 1) is combined with the value at (*dst_x* +1, *dst_y*), and so on. Each successive data element is written to the pixel in the display window corresponding to its position in the *data* array.

CMXPutImage1 transfers the least significant bit of each *data* element to the display. If the least significant bit is on, the corresponding pixel in the image is set to the foreground color set in the graphics context. If the least significant bit is not set, the pixel is set to the background color set in the graphics context.

CMXPutImage and **CMXPutImage1** draw the image into the display using the values for the following graphics context (*gc*) data elements:

function	**clip_x_origin**	**foreground**
plane_mask	**clip_y_origin**	**background**
subwindow_mode	**clip_mask**	

3.9 Clearing a Display Window

CMXClearDisplay allows you to clear a window to the window's background color or pixmap.

From CM Fortran:

```
CMXClearDisplay(display, drawable)

INTEGER display
INTEGER drawable
```

From C*:

```
void CMXClearDisplay(Display *display,
                     Drawable drawable);
```


Copyright © 1993 Thinking Machines Corporation

NOTE

When *drawable* is a window, **CMXClearDisplay** is equivalent to the X11 routine **XClearWindow**, and C* programmers can just call the X11 routine.

CMXClearDisplay clears the window on *display* specified by *drawable*. If *drawable* is a window, it is filled with *drawable*'s **background** color or pixmap. If *drawable* is a pixmap, it is filled with color 0. This routine overwrites all previous contents of the drawable. Note that *drawable*'s background color is set by **XSetWindowBackground**, not **CMXSetBackground**.

Version 1.6, October 1993
Copyright © 1993 Thinking Machines Corporation