The
Connection Machine
System

*Render Reference Manual for Paris

Version 2.0

November 1991

Thinking Machines Corporation
Cambridge, Massachusetts

First printing, November 1991

****

The information in this document is subject to change without notice and should not be construed as a commitment by Thinking Machines Corporation. Thinking Machines Corporation reserves the right to make changes to any products described herein to improve functioning or design. Although the information in this document has been reviewed and is believed to be reliable, Thinking Machines Corporation does not assume responsibility or liability for any errors that may appear in this document. Thinking Machines Corporation does not assume any liability arising from the application or use of any information or product described herein.

****

Connection Machine® is a registered trademark of Thinking Machines Corporation.
CM, CM-1, CM-2, CM-200, and DataVault are trademarks of Thinking Machines Corporation.
C*® is a registered trademark of Thinking Machines Corporation.
Paris, *Lisp, and CM Fortran are trademarks of Thinking Machines Corporation.
C/Paris, Lisp/Paris, and Fortran/Paris are trademarks of Thinking Machines Corporation.
In Parallel® is a registered trademark of Thinking Machines Corporation.
Thinking Machines is a trademark of Thinking Machines Corporation.
Microsoft is a trademark of Microsoft Corporation.
Sun, Sun-4, and Sun Workstation are registered trademarks of Sun Microsystems, Inc.
Symbolics, Symbolics 3600, and Genera are trademarks of Symbolics, Inc.
UNIX is a registered trademark of AT&T Bell Laboratories.
VAX, ULTRIX, and VAXBI are trademarks of Digital Equipment Corporation.
The X Window System is a trademark of the Massachusetts Institute of Technology.

Copyright © 1991 by Thinking Machines Corporation. All rights reserved.

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts 02142–1264
(617) 234–1000/876–1111

Contents

About This Manual

Objectives of This Manual

This manual provides detailed reference information about the Paris interface to the *Render library routines. Separate *Render manuals are available for the C* and CM Fortran interfaces.

Intended Audience

This manual is intended for programmers using *Render to support graphics or visualization applications on the Connection Machine.

It is assumed that the reader has a basic understanding of Paris programming on the Connection Machine System.

Revision Information

This manual documents *Render, Version 2.0.

This manual replaces the **Render Reference Manual, Version 5.2.*

Organization of This Manual

Chapter 1 **Introduction to *Render**
A brief overview of the *Render library and its use.

Chapter 2 **Drawing Routines**
Detailed documentation of the *Render point, line, sphere, and array drawing routines.

Chapter 3 **Graphics Math Routines**
Detailed documentation of the *Render graphics math routines.

These routines provide utilities for performing common graphics math operations on vectors and matrices in front-end arrays or CM fields.

Chapter 4 **Dithering Routines**
*Render's halftone routines convert a grayscale image of floating-point or double-floating-point values to a 1-bit-per-pixel image suitable for displaying on a black and white monitor. In addition, the library includes two routines that convert color RGB images to grayscale

Related Documents

This manual is one of three that make up the Connection Machine Visualization Programming documentation set. The other two are:

- *Generic Display Interface Reference Manual*
- *Image File Interface Reference Manual*

Notation Conventions

The table below displays the notation conventions observed in this manual.

Convention	Meaning
`bold typewriter`	C/Paris, Fortran/Paris, and Lisp/Paris language elements, such as operators, keywords, and function names, when they appear embedded in text or in syntax lines. Also UNIX and CM System Software commands, command options, and file names.
italics	Argument or parameter names and placeholders, when they appear embedded in text or syntax lines.
`typewriter`	Code examples and code fragments.
`%` **`bold typewriter`** `typewriter`	In interactive examples, user input is shown in **`bold typewriter`** and system output is shown in `regular typewriter` font.

Customer Support

Thinking Machines Customer Support encourages customers to report errors in Connection Machine operation and to suggest improvements in our products.

When reporting an error, please provide as much information as possible to help us identify and correct the problem. A code example that failed to execute, a session transcript, the record of a backtrace, or other such information can greatly reduce the time it takes Thinking Machines to respond to the report.

If your site has an Applications Engineer or a local site coordinator, please contact that person directly for support. Otherwise, please contact Thinking Machines' home office customer support staff:

U.S. Mail:	Thinking Machines Corporation Customer Support 245 First Street Cambridge, Massachusetts 02142–1264
Internet Electronic Mail:	customer–support@think.com
uucp Electronic Mail:	ames!think!customer–support
Telephone:	(617) 234–4000 (617) 876–1111

For Symbolics Users Only

The Symbolics Lisp machine, when connected to the Internet network, provides a special mail facility for automatic reporting of Connection Machine system errors. When such an error occurs, simply press Ctrl–M to create a report. In the mail window that appears, the `To:` field should be addressed as follows:

`To:` customer–support@think.com

Please supplement the automatic report with any further pertinent information.

Chapter 1

Introduction to *Render

The *Render library is a set of Paris-level utilities for drawing simple graphics primitives into an image buffer field in the Connection Machine memory. This image may then be transferred to an X Window System display or CM framebuffer for display.

This chapter provides a brief introduction to the *Render library. The remaining chapters give full descriptions of the routines.

1.1 The CM Visualization Libraries

*Render is one of three libraries that support visualization programming on the CM. The other two libraries are the Generic Display Interface and the Image File Interface.

As illustrated in Figure 1, these three libraries provide the basic tools for building visualization applications on the CM. With *Render you can process the data produced by your application to create an image in an *image buffer* in CM memory. With the Generic Display Interface you can create and control a display space and write the image buffer to it. Finally, the Image File Interface enables you to store images for future display or processing, or to transfer the image to other graphics environments.

The image buffer is a CM field or variable in a 2D Paris VP set allocated in the size and shape of the image to be displayed. The image buffer is used to collect and store pixel values for display. Each virtual processor in the image buffer VP set contains a color value and, if 3D, a z coordinate for the pixel at the corresponding (x, y) location on the display. The image buffer is discussed in detail in the introduction to Chapter 2.

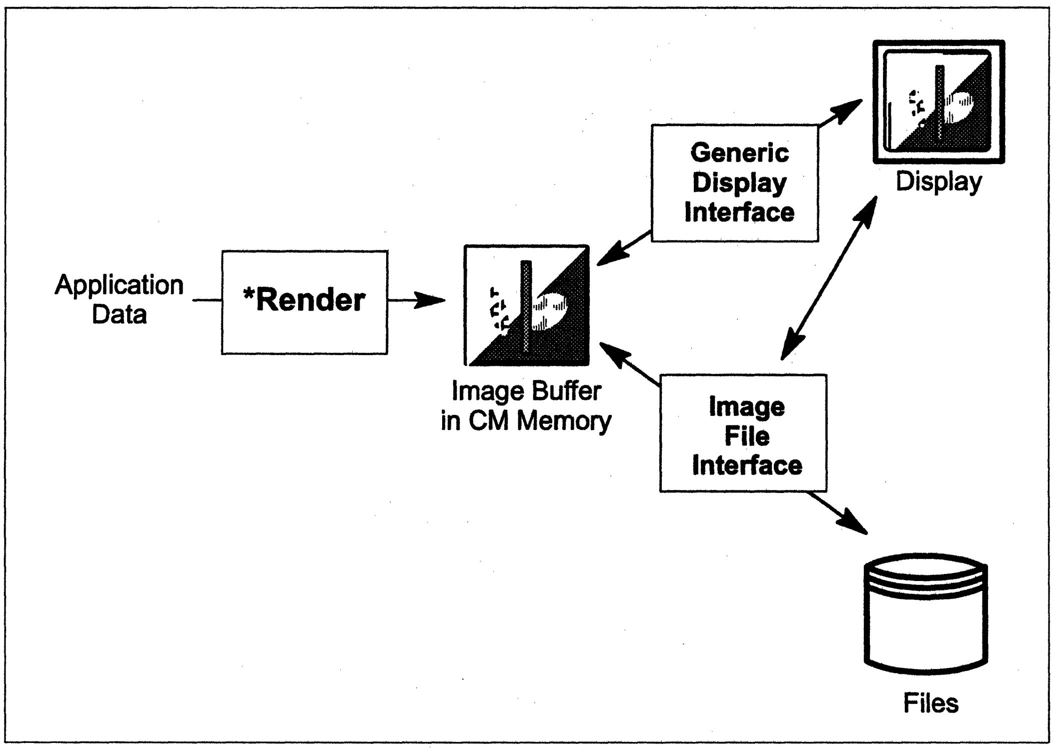


Figure 1. Basic data flow in Connection Machine visualization.

1.1.1 The Generic Display Interface

The Generic Display Interface is a library of routines that provide a single simple interface through which your application can

- create and initialize Generic Display workstations and displays by having the user select them from a menu (the display provides a display space for images from CM memory; the workstation provides resources to support text and cursor routines)
- transfer image data from CM memory to different types of displays without specialized routines
- query and modify the characteristics of the physical displays from the Generic Display Interface, including the display color maps
- display text strings to any selected generic display

- display, track, and interact with a cursor on the generic display with your workstation mouse

The Generic Display Interface simplifies image display and interaction and allows you to write *device-independent* applications that can be moved to different displays at run time without changing your application. It is documented in the *Generic Display Reference Manual for Paris* included with the CM visualization document set.

1.1.2 The Image File Interface

The Image File Interface supports the transfer of images to files in TIFF (Tagged Image File Format), a standard specification for image data files. TIFF is supported by many other graphics software packages, so you can easily move CM images stored with the Image File Interface to other graphics environments for editing or display. The TIFF format also provides for compression of the image data in the file and stores information about the image that can be used when reading the file back into the Connection Machine system or into some other TIFF reader.

The Image File Interface transfers images between files and an image buffer on the CM, a scalar array on the front-end computer, or even directly to or from a Generic Display Interface display. It is documented in the *Image File Interface Reference Manual for Paris*.

1.2 *Render

*Render is made up of the following major components:

- **Drawing Routines**

  The *Render drawing routines write 2D and 3D points, 2D lines, and image arrays into an image buffer field in CM memory. Some routines, those with *_fe_* in their names, draw a single point or line specified by coordinates stored in 1D arrays on the front-end computer. Other routines take a field of coordinates and draw the set of lines or points specified in a single operation.

  Simple sphere drawing is also supported by a routine that draws shaded spheres at specified locations in the image buffer.

  In addition, *Render includes clipping operations that allow you to clip a set of line coordinates in a CM field to a specified clipping range.

These routines are described in Chapter 2 of this manual.

- **Graphics Math Routines**

  *Render's Graphics Math routines support the basic graphics math operations on vectors and matrices. As with the drawing routines, there are math routines to operate on a single vector or matrix in front-end memory, or on a field of vectors or matrices in CM memory.

  The vector routines include basic operations such as copying, adding, subtracting, normalizing, negating, taking the cosine, dot product, cross product, or perpendicular of two vectors, and applying transformation matrix to a vector. More specialized routines include determining reflectance and transmittance vectors for ray-tracing and radiosity applications.

  Included with the graphics math routines are color conversion routines to transform color vectors between different color models.

  These routines are described in Chapter 3.

- **Dithering Routines**

  The dithering routines allow you to move a color image to a grayscale or monochrome scale with a minimum loss of image detail.

  Two routines convert a color RGB image to an 8-bit grayscale image. That image may then be given to one of a set of 6 dithering images that reduce the grayscale to a 1-bit monochrome image. The dithering images support either integer or floating point color values and allow you to apply either dot diffusion or error propagation methods to produce the monochrome image.

  These routines are described in Chapter 4.

1.3 Using *Render

To use the *Render routines you must include the appropriate header file in your program and link with the supporting libraries when compiling.

1.3.1 C/Paris

To use the *Render drawing and dither routines you must include the header file **`cmsr-draw.h`** as follows:

```
#include <cm/cmsr-draw.h>
```

To use the *Render math routines you must include the header file **`cmsr-math.h`** as follows:

```
#include <cm/cmsr-math.h>
```

For all the *Render routines, you must use the following links when compiling:

```
cc prog.c -lcmsr -lx11 -lparis -lm
```

1.3.2 Fortran/Paris

To use the *Render drawing and dither routines you must include the header file **`cmsr-draw-fort.h`** as follows:

```
INCLUDE '/usr/include/cm/cmsr-draw-fort.h'
```

To use the *Render math routines you must include the header file **`cmsr-math-fort.h`** as follows:

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
```

Note

This directory path, **`/usr/include/cm/cmsr-math-fort.h`**, is the location for these header files recommended by the installation script for this software. However, you should check with your system administrator for the exact location at your site.

For all the *Render routines, you must use the following links when compiling:

```
cc prog.c -lcmsr -lx11 -lparisfort -lparis -lm
```

1.3.3 Lisp

For Lisp programs you must use a band in which the graphics package has been loaded. If necessary, you can load it by entering:

```
(lcmw:load-optional-system 'graphics)
```

This will make all the graphics library functions available.

Chapter 2

Drawing Routines

2.1 Overview

Render helps you create and manipulate an image in an image buffer in Connection Machine memory. The drawing routines draw points, lines, arrays and spheres into an *image buffer* in CM memory by writing color values into the appropriate locations.

2.1.1 The Image Buffer Field

The image buffer is a Paris field in a 2D VP set allocated in the size and shape of the image to be displayed. The image buffer is the destination field for the *Render drawing operations and the source field for the Generic Display Interface's display routines.

You allocate the image buffer so that the length of the axes of the image buffer VP set corresponds to the resolution of the image to be displayed, 1 virtual processor to each pixel. Axis 0 of the geometry maps to the display's x (horizontal) axis, and axis 1 of the geometry maps to the display's y (vertical) axis. Each virtual processor in the image buffer VP set contains a color value and, if 3D, a z coordinate for the pixel at the corresponding (x, y) location on the display.

*Render and the Generic Display Interface allow you to operate on the image buffer like a virtual display space by specifying locations in screen coordinates. The visualization libraries assume the right-handed screen coordinate system shown in Figure 2. The origin (0,0) is at the upper left corner of the image, positive x increases to the right, positive y increases toward the bottom of the screen, and positive z increases into the screen. The coordinate values are specified in terms of pixels.

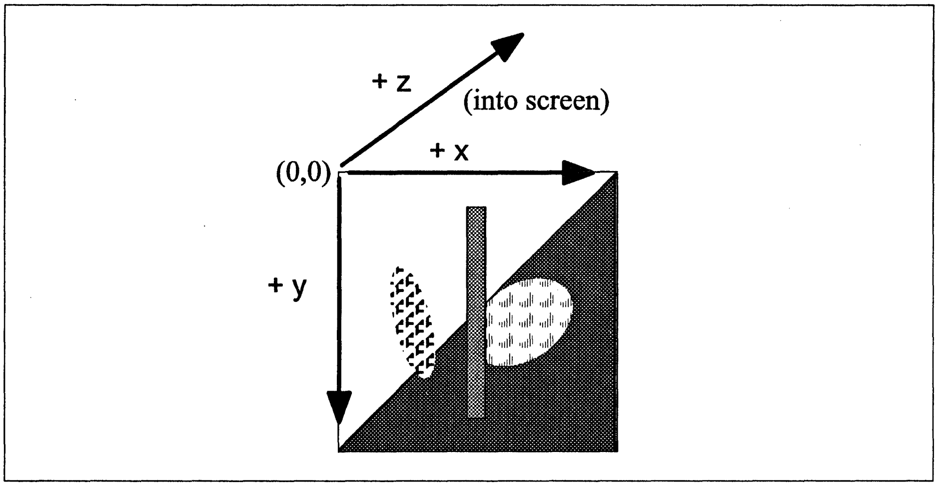


Figure 2. The image buffer coordinate system.

The 2D *Render drawing routines specify the location of the drawing primitives in *x* and *y* coordinate pairs that correspond to pixel/processor locations in the image buffer field. The routines "draw" into the image buffer field by loading a specified color value into the appropriate processor location.

The image buffer is then displayed by transferring the color data from CM memory to a generic display, as with the Generic Display Interface routine **`CMSR_write_to_display`**. The origin of the image buffer field (0,0) is displayed at the upper left corner of the display and the color value in each virtual processor is assigned to the corresponding pixel of the display. The length of the image buffer field allocated for the color data should be the same as the depth of the display. If the field length is longer, only the low order, least significant bits are displayed. If the field length is shorter than the depth of the window, an error is signaled.

2.1.2 The Z Buffer

For *Render's 3D drawing routines, a *z-buffer* field is allocated containing two subfields, one for color data and one for *z* coordinate data. The *z* value occupies the most significant bits, and the color value occupies the least significant bits. The 3D drawing routines specify *x*, *y*, and *z* coordinates, and color values. *Render includes a utility routine, **`CMSR_initialize_z_buffer`**, which prepares an allocated *z*-buffer field for use by initializing the z coordinate portion of the z-buffer field to the largest value that can be represented and the color portion to a specified background color.

As with the 2D routines, the *x* and *y* coordinates determine the location in the z-buffer field VP set that will receive the color value. But before writing the color value, the system performs a z-buffer comparison between the incoming *z* coordinate and the z-buffer value already stored at that location. If the incoming *z* coordinate is smaller (that is, "nearer" the viewer), the color value associated with it is written to the field and the incoming *z* coordinate becomes the z-buffer value at that point. If the incoming *z* coordinate is larger (that is, "farther" from the viewer) than the current z-buffer value, neither the color nor *z* coordinate is changed for that location. Thus, the point stored is the visible point nearest the viewer.

2.1.3 Framebuffer-Ordered Geometries

The transfer of fields of color data between CM memory and the CM framebuffer can be optimized by using image buffer geometries created with *framebuffer ordering*. I/O performance to X Window System or Symbolics generic displays is unaffected by the choice of ordering.

The function **`CMFB-create-cmfb-geometry`** allocates and returns a 2D geometry of a specified *width* and *height*. *Width* specifies the length of axis 0 of the geometry and maps to the screen's *x* (horizontal) axis. *Height* specifies the length of axis 1 and maps to the screen's *y* (vertical) axis. Both axes are created with framebuffer ordering.

Framebuffer-ordered geometries are intended to be used only as image buffers. While image transfers to the CM framebuffer are faster, Paris NEWS communication functions operate much more slowly on a framebuffer-ordered VP set. The NEWS function must perform a send to reorder a framebuffer-ordered geometry before the NEWS operation can be completed.

If you do not use NEWS functions in the image buffer, it is recommended that you do *not* use a normal grid-ordered geometry as an image buffer. The Generic Display Interface I/O functions will accept a NEWS-ordered geometry as an image buffer, but performance is slowed significantly. These operations must perform a send to "shuffle" the field into framebuffer order before transferring it to the CM framebuffer.

2.1.4 The Combiner Parameter

*Render routines that draw into the image buffer use a *combiner* parameter to define the method used to combine the array values being transferred from the source field with the values already in the image buffer field. Valid values for this parameter are:

`DEFAULT`	No combiner method specified.
`OVERWRITE`	Replace existing image buffer value with source value.
`LOGIOR`	Combine using bitwise logical inclusive OR.
`LOGAND`	Combine using bitwise logical AND.
`LOGXOR`	Combine using bitwise logical exclusive OR.
`U-ADD`	Combine using unsigned integer addition.
`S-ADD`	Combine using signed integer addition.
`U-MIN`	Combine using unsigned integer minimum operation.
`S-MIN`	Combine using signed integer minimum operation.
`U-MAX`	Combine using unsigned integer maximum operation.
`S-MAX`	Combine using signed integer maximum operation.

These values correspond to the appropriate versions of the Paris **`send`** functions. For example, specifying a *combiner* of `U-MAX` will call **`send-with-u-max`** to write into the image buffer. The `DEFAULT` setting corresponds to the **`send-1L`** Paris function.

Note that the combiner parameter also controls how multiple values sent to the same image buffer location are to be combined. For example, if two or more color values are written to a single location in the image buffer field and the *combiner* operation is `ADD`, each color value is added to the current color at that location as it arrives at the processor. If the *combiner* operation is `MAX`, the largest of the arriving values or the original value is saved.

If more than one value is received at a single location when *combiner* is set to `DEFAULT` or `OVERWRITE`, the result is unpredictable. The `OVERWRITE` operation discards the original value, but does not predict which of the incoming values will be saved. The `DEFAULT` operation overwrites the original value, but an unpredictable ordering of bits will be saved; that is, none of the incoming messages will be saved intact. You should use these operations only when you are sure that only one value will be sent to any location.

2.1.5 Drawing Points and Lines

The 2D point drawing routines load a color value into a specific location in the image buffer field. The 3D point drawing routines write to a specific (x, y) location in the z-buffer field in the same way as the 2D routines. But the 3D routines also perform a z coordinate comparison as described in Section 1.3.2 above. The color value associated with the smaller z ("nearer" the viewer) is chosen over the color value with a larger z. The line drawing routines draw a color value into the image buffer field along a line between specified endpoints.

As summarized in Table 1, different versions of the *Render operations support either front-end variables or other Connection Machine fields as the source for coordinate and color values. Similarly, in different *Render operations the coordinate and color values can be either floating-point or signed integer values.

Table 1. *Render point and line drawing operations.

	Uses Front-End Source Variable	**Uses CM Source Field**
Signed Integer Values	`CMSR_fe_s_draw_line` `CMSR_fe_s_draw_point`	`CMSR_s_draw_line` `CMSR_s_draw_point`

Floating-Point Values	`CMSR_fe_f_draw_line`
	`CMSR_fe_f_draw_point`
	`CMSR_fe_f_draw_point_3d`
		`CMSR_f_draw_line`
		`CMSR_f_draw_point`
		`CMSR_f_draw_point_3d`

The *Render operations that use front-end variables specify a single coordinate pair and color value. These operations draw a single point or line with each call of the routine.

The *Render operations that use CM source fields operate in parallel on the set of coordinate pairs and color values specified in the fields. With each call of these routines, one point or line is drawn for each active virtual processor in the current VP set.

The source fields must be in the current VP set when the *Render operation is called, but the source fields need not be in the same VP set as the image buffer field.

Floating-Point Coordinates

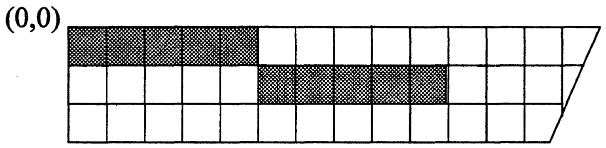
Floating-point coordinate values must, of course, be reduced to integer values to determine which discrete pixels are actually turned on.

When using floating-point coordinate values, the *Render routines round the floating-point values to integral pixel values by using the function

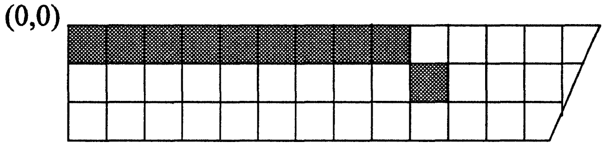
`round(`*value*`) = floor(`*value* `+ 0.5)`

This means that the area of a pixel in floating-point coordinates is (x–0.5, x+0.5) by (y–0.5, y+0.5). For example, the first pixel is lit by the coordinates from (–.05, –0.5) to (0.5, 0.5), and a display space of size 128 by 128 has a floating-point extent of (–0.5, –0.5) to (127.5, 127.5).

This convention has been adopted because it allows more accurate line drawing. For example, if a line is drawn from (0.0, 0.0) to (9.0, 1.0) the pixels that will be lit are as follows:



If a simple **floor** function where used, the less intuitive result would be:



Note

Line drawing using floating-point coordinates with **CMSR_fe_f_draw_line** or **CMSR_f_draw_line** is significantly slower than the line-drawing routines that use integer coordinates. If you are hampered by the speed of the floating-point routines, you may want to convert the coordinates to integer values and then use **CMSR_fe_s_draw_line** or **CMSR_s_draw_line**.

The floating-point routines are slower because of added processing needed to draw fractional slopes accurately.

Clipping

The *Render line and point drawing operations optionally clip the primitives to the coordinate range of the image buffer field. If the *clip_p* parameter for these routines is true, points and portions of lines with coordinates outside the image buffer field will not be drawn. These operations do not change the coordinate values specified by the user in CM source fields or front-end variables.

In addition, two clipping operations, **`CMSR_f_clip_lines`** and **`CMSR_s_clip_lines`**, clip source fields containing 2D floating-point or integer coordinate values, respectively, to a user-defined coordinate range.

If a line falls completely outside the clipping range, these routines clear the test flag of the corresponding virtual processor. If only a portion of a line is within the clipping range, these routines set the virtual processor test flag and clip the lines by interpolating new endpoints at the boundary of the clipping range, overwriting the original line coordinates specified in the source field. If a line is entirely within the clipping range, these routines set the virtual processor test flag and leave the coordinates unchanged.

You may then use the Paris instruction **`CM_logand_context_with_test`** to load the test flag values into the context flags of the source field VP set and use these fields as source fields for **`CMSR_s_draw_line`** or **`CMSR_f_draw_line`**.

2.1.6 Sphere Drawing

`CMSR_s_draw_sphere` provides a simple interface for drawing shaded spheres into an image buffer in CM memory.

`CMSR_s_draw_sphere` takes six CM fields as arguments:

- the image buffer field into which the spheres are to be drawn
- a vector field specifying the 3D coordinates of each sphere's center
- a field giving the radius of each sphere
- two fields giving minimum and maximum color values defining the range of values from the color map that the sphere can take on
- an optional information field that can be used as you wish.

The spheres are shaded as if a light source was placed at negative infinity along the *z* axis, and anti-aliasing may be performed to smooth the sphere edges.

2.1.7 Transferring Image Arrays

*Render also includes routines to transfer arrays of image data from one CM field to another, and between a front-end array and a CM field.

From a CM Field to the Image Buffer

CMSR_draw_image allows you to transfer a portion of a source field of color values to the image buffer field. Both the source field and the image buffer field must be in two-dimensional VP sets. The source field must be in the current VP set when the operation is called. The image buffer field does *not* have to be in the current VP set.

CMSR_draw_image specifies the coordinates of a subarray of the source field and a location in the image buffer. This operation allows you to move a portion of an image into the image buffer field from another two-dimensional VP set, or to move a portion of an image to another position within an image buffer field.

Between a Front-End Array and a CM Field

The following routines use bit-packed transfers to move an image between an array on the front-end computer and a field in CM memory.

The *read* routines pack image buffer field values into a front-end array:

CMSR_read_array_from_field

CMSR_read_array_from_field_1

These routines pack the image by loading the color values from the image buffer field into the front-end array elements as closely as possible. For example, a 128 by 128 1-bit image could be packed into a 16 by 128 front-end **char** or **CHARACTER** array, 8 image array elements to a front-end array element. When **CMSR_read_array_from_field** writes this image to the front-end array, the image field in 8 CM processors fills a byte of the front-end array. If *array_element_size* is 8, each CM processor fills a byte of the front-end array elements, and if *array_element_size* is 32, each CM processor fills a word of the front-end array elements.

The write routines perform the opposite operation, loading an image array packed into a front-end array into a CM image buffer field:

CMSR_write_array_to_field

CMSR_write_array_to_field_1

The routines that end in 1 are more detailed versions. They allow you to specify a portion of the source (array or field) to be transferred rather than the entire array, and to specify offsets indicating where the image array should be placed.

2.2 *Render Drawing Routine Descriptions

This section provides individual descriptions of the *Render drawing routines:

CMSR_initialize_z_buffer

Initializes a z-buffer field for use.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
  CMSR_initialize_z_buffer
        (z_buffer_field, color_value, coord_s_len, coord_e_len, color_len)

CM_field_id_t  z_buffer_field;
int            color_value;
unsigned int   coord_s_len;
unsigned int   coord_e_len;
unsigned int   color_len;
```

Fortran Syntax

```
      INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

      SUBROUTINE CMSR_INITIALIZE_Z_BUFFER
     &          (z_buffer_field, color_value, coord_s_len, coord_e_len, color_len)

      INTEGER  z_buffer_field
      INTEGER  color_value
      INTEGER  coord_s_len
      INTEGER  coord_e_len
      INTEGER  color_len
```

Lisp Syntax

```
CMSR:initialize-z-buffer    (z-buffer-field
                             &optional (color-value 0) (coord-s-len 23)
                             (coord-e-len 8) (color-len 8))
```

ARGUMENTS

z_buffer_field	A Paris field identifier. *z_buffer_field* is a CM field with subfields for a floating-point *z* coordinate value and an unsigned integer color value. The *z* value occupies the most significant bits, and the color value occupies the least significant bits. The total length of the field must be (*coord_s_len* + *coord_e_len* + 1 + *color_len*) where *coord_s_len* is the length of the *z* coordinate significand, *coord_e_len* is the length of the *z* coordinate exponent, 1 is the sign bit for the *z* value, and *color_len* is the length of the color value.
color_value	The color value to which the z-buffer is to be initialized. In Lisp this parameter defaults to 0.
coord_s_len	The length, in bits, of the significand of the *z* coordinate value in *z_buf-fer_field.*
coord_e_len	The length, in bits, of the exponent of the *z* coordinate value in *z_buf-fer_field.*
color_len	The length, in bits, of the color value in *z_buffer_field.*

DESCRIPTION

`CMSR_initialize_z_buffer` initializes a z-buffer field for use. The *z* coordinate portion of *z_buffer_field* is initialized to the largest value that can be represented. The color portion of *z_buffer_field* is initialized to *color_value.*

This function should be used to initialize any *z_buffer_field* before use.

SEE ALSO

`CMSR_f_draw_point_3d`

`CMSR_fe_f_draw_point_3d`

CMSR_f_draw_point

Draws a set of 2D points into the CM image buffer field using floating-point coordinate values.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
CMSR_f_draw_point  (image_buffer_field, x_field, y_field, color_field,
                    coord_s_length, coord_e_length, color_length,
                    combiner, clip_p)

CM_field_id_t      image_buffer_field, x_field, y_field, color_field;
unsigned int       coord_s_length, coord_e_length, color_length;
CMSR_combiner_t    combiner;
int                clip_p;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

   SUBROUTINE CMSR_F_DRAW_POINT
&                          (image_buffer_field, x_field, y_field,color_field,
&                           coord_s_length,coord_e_length,color_length,
&                           combiner, clip_p)

   INTEGER  image_buffer_field, x_field, y_field, color_field
   INTEGER  coord_s_length, coord_e_length, color_length
   INTEGER  combiner, clip_p
```

Lisp Syntax

```
CMSR:f-draw-point  (image-buffer-field x-field y-field color-field
                    coord-s-length coord-e-length color-length
                     &key (combiner :default) (clip-p t))
```

ARGUMENTS

image_buffer_field	A Paris field identifier. The points are drawn into this field at the locations specified by the *x_field* and *y_field* coordinate pairs. The *image_buffer_field* must be in a two-dimensional VP set, and may or may not be in the same VP set as the *color_field* and coordinate fields. It need not be in the current VP set.
x_field, *y_field*	Paris field identifiers. These fields contain floating-point values that are, respectively, the x and y coordinates at which to draw the points in the image buffer field. *x_field* and *y_field* must be in the current VP set.
color_field	A Paris field identifier. This field contains the value drawn into the image buffer. *color_field* must be in the current VP set.
coord_s_length, *coord_e_length*	Unsigned integers specifying the length of the floating-point significand and exponent, respectively, in the coordinate values used for *x_field* and *y_field*.
color_length	The length, in bits, of the *color_field*.
combiner	A symbol defining the method used to combine the color values being written into the image buffer field with the values already in the image buffer field. Valid values are listed in the table below.

C Values	Fortran Values	Lisp Keywords
`CMSR_default`	`CMSR_default`	`:DEFAULT`
`CMSR_overwrite`	`CMSR_overwrite`	`:OVERWRITE`
`CMSR_logior`	`CMSR_logior`	`:LOGIOR`
`CMSR_logand`	`CMSR_logand`	`:LOGAND`
`CMSR_logxor`	`CMSR_logxor`	`:LOGXOR`
`CMSR_u_add`	`CMSR_u_add`	`:U-ADD`
`CMSR_s_add`	`CMSR_s_add`	`:S-ADD`
`CMSR_u_min`	`CMSR_u_min`	`:U-MIN`
`CMSR_s_min`	`CMSR_s_min`	`:S-MIN`
`CMSR_u_max`	`CMSR_u_max`	`:U-MAX`
`CMSR_s_max`	`CMSR_s_max`	`:S-MAX`

clip_p A symbol indicating whether the line is to be clipped or not.

If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), lines and points outside the range of the image buffer field are not drawn.

If *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), it is an error to draw outside the range of the image buffer field.

DESCRIPTION

`CMSR_f_draw_point` draws a set of points, defined with floating-point coordinates, into the specified image buffer field.

For each active processor in the current VP set, the value in *color_field* is drawn into *image_buffer_field* at the processor location (*x_field*, *y_field*].

The *Render routines round the floating-point values to integral pixel values by using the function

`round(value) = floor(value+0.5)`

This means that the area of a pixel in floating-point coordinates is (x–0.5,x+0.5) by (y–0.5,y+0.5). For example, the first pixel is lit by the coordinates from (–.05, –0.5) to (.5, .5), and a display space of size 128 x 128 has a floating-point extent of (–0.5, 127.5) x (–0.5, 127.5).

The value written into each location in the image buffer field is a combination of the value of *color_field*, the previous value at that location, and the value of any other points overwriting the same location. The method used to combine these values is controlled by the *combiner* parameter.

If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), points with coordinates outside the range of the image buffer field coordinates are not drawn. If the CM safety mode is on and *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), an error is signaled if the point is not within the boundaries of the destination image buffer field.

ERRORS

The following errors are signaled if the CM safety mode is on.

It is an error to call `CMSR_draw_f_point` with

- coordinates not within the destination image buffer field if *clip_p* is false
- an *image_buffer_field* that is not part of a two-dimensional VP set geometry
- a *color_length* that is longer than the length of the *image_buffer_field* or *color_field*
- color or coordinate fields not in the current VP set

SEE ALSO

`CMSR_fe_s_draw_point`

`CMSR_fe_f_draw_point`

`CMSR_s_draw_point`

CMSR_f_draw_point_3d

Draws a set of 3D points into the CM image buffer field using floating-point coordinate values.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
   CMSR_f_draw_point_3d(z_buffer_field, xyz_vector_field, color_field,
                        coord_s_len, coord_e_len, color_len, clip_p);

CM_field_id_t  z_buffer_field;
CM_field_id_t  xyz_vector_field;
CM_field_id_t  color_field;
unsigned int   coord_s_len;
unsigned int   coord_e_len;
unsigned int   color_len;
int            clip_p;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

   CMSR_F_DRAW_POINT_3D (z_buffer_field, xyz_vector_field, color_field,
&                        coord_s_len, coord_e_len, color_len, clip_p)

   INTEGER z_buffer_field
   INTEGER xyz_vector_field
   INTEGER color_field
   INTEGER coord_s_len
   INTEGER coord_e_len
   INTEGER color_len
   INTEGER clip_p
```

Lisp Syntax

```
CMSR:f-draw-point-3d (z-buffer-field xyz-vector-field color-field
                      coord-s-len coord-e-len color-len
                      &optional (clip-p t))
```

ARGUMENTS

z_buffer_field	A Paris field identifier. *z_buffer_field* is a CM field with subfields for a floating-point *z* coordinate value and an unsigned integer color value. The *z* value occupies the most significant bits, and the color value occupies the least significant bits. The total length of the field must be (*coord_s_len* + *coord_e_len* + 1 + *color_len*) where *coord_s_len* is the length of the *z* coordinate significand, *coord_e_len* is the length of the *z* coordinate exponent, 1 is the sign bit for the *z* value, and *color_len* is the length of the color value. The *z* subfield may be accessed by using the value **`CM_add_offset_to_field`**(*z_buffer_field*, *color_len)* and the color subfield may be accessed by using the value *z_buffer_field.*
xyz_vector_field	A Paris field indentifier specifying the field containing the coordinates, in screen coordinate space, of the points to be drawn to *z_buffer_field.* The *x* and *y* coordinates specify the location in the *z_buffer_field* VP set that will receive this processor's *z* coordinate and color value. The coordinates are floating-point values, each having a length of (*coord_s_len* + *coord_e_len* + 1) bits. The vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is (3 * (*coord_s_len* + *coord_e_len* + 1))
color_field	A Paris field indentifier identifying the field containing the integer color values to be drawn into *z_buffer_field.*
coord_s_len	The length, in bits, of the significand of the floating-point values in the *z* subfield of *z_buffer_field* and the *x*, *y*, and *z* subfields of *xyz_vector_field.*
coord_e_len	The length, in bits, of the exponent of the floating-point values in the *z* subfield of *z_buffer_field* and the *x*, *y*, and *z* subfields of *xyz_vector_field.*
color_len	The length, in bits, of the color subfield in *z_buffer_field* and the *color_field.*

clip_p

A symbol indicating whether the points are to be clipped or not.

If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), lines, or portions of lines, drawn outside the range of the *z_buffer_field* coordinates are clipped.

If *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), it is an error to draw outside the range of the *z_buffer_field.*

The range of the *z_buffer_field* is defined by the length of the 2 axes in the 2D geometry in which it is defined.

DESCRIPTION

For each active processor in the VP set containing *xyz_vector_field* and *color_field,* **`CMSR_f_draw_point_3d`** draws a z-buffer image value into *z_buffer_field* at the location specified by the x and y components of the *xyz_vector_field.* The z-buffer image value is composed of the z value from *xyz_vector_field* and the color value from *color_field.*

The *Render routines round the floating-point coordinate values to integral pixel values by using the function

```
round(value) = floor(value + 0.5)
```

This means that the area of a pixel in floating-point coordinates is $(x - 0.5, x + 0.5)$ by $(y - 0.5, y + 0.5)$. For example, the first pixel is lit by the coordinates from $(-.05, -0.5)$ to $(0.5, 0.5)$, and a display space of size 128 by 128 has a floating-point extent of $(-0.5, 127.5)$ to $(-0.5, 127.5)$.

The fields *xyz_vector_field* and *color_field* must both be in the current VP set when **`CMSR_f_draw_point_3d`** is called. The field *z_buffer_field* does not need to be in same VP set as *xyz_vector_field* and *color_field,* nor does it need to be in the current VP set.

The system performs a z-buffer comparison in *z_buffer_field* based on a right handed coordinate system, that is, positive z increases into the screen, positive y increases toward the bottom of the screen, and positive x increases to the right. The origin of the image (0,0) is the upper left corner. If a z-buffer image value is written to a point in *z_buffer_field* that already contains an image value, the color value associated with the smaller z ("nearer" the viewer) is chosen over the color value with a larger z.

ERRORS

With CM safety mode on, an error is signaled if you call `CMSR_f_draw_point_3d` with

- a *z_buffer_field* that is not part of a two-dimensional VP set geometry
- the fields *xyz_vector_field*, *color_field* not in the current VP set
- a *color_length* that is longer than the length of the *image_buffer_field* or *color_field*
- coordinates not within the destination *z-buffer* field if *clip_p* is false

SEE ALSO

`CMSR_fe_f_draw_point_3d`

`CMSR_f_draw_point`

`CMSR_s_draw_point`

`CMSR_fe_f_draw_point`

`CMSR_fe_s_draw_point`

CMSR_s_draw_point

Draws a set of points into the CM image buffer field using signed integer coordinates.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
CMSR_s_draw_point  (image_buffer_field, x_field, y_field, color_field,
                    coord_length, color_length, combiner, clip_p) ;

CM_field_id_t      image_buffer_field, x_field, y_field, color_field;
unsigned int       coord_length, color_length;
CMSR_combiner_t    combiner;
int                clip_p;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

   SUBROUTINE CMSR_S_DRAW_POINT
&                             (image_buffer_field, x_field, y_field, color_field,
&                              coord_length, color_length, combiner, clip_p)

   INTEGER  image_buffer_field, x_field, y_field, color_field
   INTEGER  coord_length, color_length, combiner, clip_p
```

Lisp Syntax

```
CMSR:s-draw-point  (image-buffer-field x-field y-field
                    color-field coord-length color-length
                    &key (combiner :default) (clip-p  t))
```

ARGUMENTS

image_buffer_field A Paris field identifier. The specified point is drawn into this field at the location specified by *x_field* and *y_field*. The *image_buffer_* *field* must be in a two-dimensional VP set, and may or may not be in the same VP set as the *color_field* or coordinate fields. It need not be in the current VP set.

x_field, y_field

Paris field identifiers. These fields contain integer values that are the *x* and *y* coordinates, respectively, at which the point is to be drawn in the image buffer field. *x_field* and *y_field* must be in the current VP set.

color_field

A Paris field identifier. This fields contains the value to be drawn into the *image_buffer_field. color_field* must be in the current VP set.

coord_length

An unsigned integer specifying the length of the coordinates used for *x_field* and *y_field.*

NOTE: In routines using signed integer coordinates, *coord_length* must include room for the sign bit.

color_length

The length, in bits, of *color_field.*

combiner

A symbol defining the method used to combine the color values being written into the image buffer field with the values already in the image buffer field. Valid values are listed in the table below.

C Values	Fortran Values	Lisp Keywords
`CMSR_default`	`CMSR_default`	`:DEFAULT`
`CMSR_overwrite`	`CMSR_overwrite`	`:OVERWRITE`
`CMSR_logior`	`CMSR_logior`	`:LOGIOR`
`CMSR_logand`	`CMSR_logand`	`:LOGAND`
`CMSR_logxor`	`CMSR_logxor`	`:LOGXOR`
`CMSR_u_add`	`CMSR_u_add`	`:U-ADD`
`CMSR_s_add`	`CMSR_s_add`	`:S-ADD`
`CMSR_u_min`	`CMSR_u_min`	`:U-MIN`
`CMSR_s_min`	`CMSR_s_min`	`:S-MIN`
`CMSR_u_max`	`CMSR_u_max`	`:U-MAX`
`CMSR_s_max`	`CMSR_s_max`	`:S-MAX`

clip_p

A symbol indicating whether the point is to be clipped or not.

If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), points outside the range of the image buffer field coordinates are not drawn.

If *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), it is an error to draw outside the range of the image buffer field.

DESCRIPTION

`CMSR_s_draw_point` draws a point, defined with signed integer coordinates, into the specified image buffer field.

For each active processor in the current VP set, the value in *color_field* is drawn into *image_buffer_field* at processor location (*x*,*y*).

The value written into each location in the image buffer is a combination of the value of *color_field*, the previous value at that location, and the value of any other points overwriting the same location. The method used to combine these values is controlled by the *combiner* parameter.

If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), points with coordinates outside the range of the image buffer field are not drawn. If the CM safety mode is on and *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), an error is signaled if the point is not within the boundaries of the destination image buffer field.

ERRORS

With CM safety mode on, an error is signaled if you call **`CMSR_draw_s_point`** with

- coordinates not within the destination image buffer if *clip_p* is false
- an *image_buffer_field* that is not part of a two-dimensional VP set geometry
- a *color_length* that is longer than the length or the *image_buffer_field* or *color_field*
- color or coordinate fields not in the current VP set

SEE ALSO

`CMSR_f_draw_point`

`CMSR_fe_s_draw_point`

`CMSR_fe_f_draw_point`

CMSR_fe_f_draw_point

Draws a point into the CM image buffer field using front-end floating-point coordinate values.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
CMSR_fe_f_draw_point(image_buffer_field, x, y, color,
                     color_length, combiner, clip_p)

CM_field_id_t       image_buffer_field;
double              x, y;
int                 color;
unsigned int        color_length;
CMSR_combiner_t     combiner;
int                 clip_p;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

SUBROUTINE CMSR_FE_F_DRAW_POINT(image_buffer_field, x, y, color,
                                color_length, combiner, clip_p)

INTEGER             image_buffer_field
DOUBLE PRECISION    x, y
INTEGER             color, color_length, combiner, clip_p
```

Lisp Syntax

```
CMSR:fe-f-draw-point  (image-buffer-field x y color color-length
                       &key (combiner :default) (clip-p t))
```

ARGUMENTS

image_buffer_field A Paris field identifier. The specified point is drawn into this field at the location specified by *x* and *y*. The *image_buffer_field* must be in a two-dimensional VP set. It need not be in the current VP set.

x, y Front-end floating-point coordinate values, defining the *x* and *y* coordinates, respectively, at which to draw the point in the image buffer field.

color_length The length, in bits, of *color*.

combiner A symbol defining the method used to combine the color values being written into the image buffer field with the values already in the image buffer field. Valid values are listed in the table below.

C Values	Fortran Values	Lisp Keywords
`CMSR_default`	`CMSR_default`	`:DEFAULT`
`CMSR_overwrite`	`CMSR_overwrite`	`:OVERWRITE`
`CMSR_logior`	`CMSR_logior`	`:LOGIOR`
`CMSR_logand`	`CMSR_logand`	`:LOGAND`
`CMSR_logxor`	`CMSR_logxor`	`:LOGXOR`
`CMSR_u_add`	`CMSR_u_add`	`:U-ADD`
`CMSR_s_add`	`CMSR_s_add`	`:S-ADD`
`CMSR_u_min`	`CMSR_u_min`	`:U-MIN`
`CMSR_s_min`	`CMSR_s_min`	`:S-MIN`
`CMSR_u_max`	`CMSR_u_max`	`:U-MAX`
`CMSR_s_max`	`CMSR_s_max`	`:S-MAX`

clip_p A symbol indicating whether the point is to be clipped or not.

If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), points outside the range of the image buffer field coordinates are not drawn.

If *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), it is an error to draw outside the range of the image buffer field.

DESCRIPTION

`CMSR_fe_f_draw_point` draws a point, defined with front-end coordinate values, into the specified image buffer field.

The value in *color_field* is drawn into *image_buffer_field* at processor location (x,y). The *Render routines round the floating-point coordinate values to integral pixel values by using the function

`round(value) = floor(value+0.5)`

This means that the area of a pixel in floating-point coordinates is (x–0.5,x+0.5) by (y–0.5,y+0.5). For example, the first pixel is lit by the coordinates from (–.05, –0.5) to (0.5, 0.5), and a display space of size 128 by 128 has a floating-point extent of (–0.5, 127.5) to (–0.5, 127.5).

The value written into the location in the image buffer is a combination of the value of *color_field*, the previous value at that location, and the value of any other points overwriting the same location. The method used to combine these values is controlled by the *combiner* parameter.

If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), points with coordinates outside the range of the image buffer field coordinates are not drawn. If the CM safety mode is on and *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), an error is signaled if the point is not within the boundaries of the destination image buffer field.

ERRORS

With CM safety mode on, an error is signaled if you call **`CMSR_fe_f_draw_point`** with

- coordinates not within the destination image buffer if *clip_p* is false
- an *image_buffer_field* that is not part of a two-dimensional VP set
- a *color_length* that is longer than the length of the *image_buffer_field*

SEE ALSO

`CMSR_f_draw_point`

`CMSR_fe_s_draw_point`

`CMSR_s_draw_point`

CMSR_fe_f_draw_point_3d

Draws a point into the CM image buffer field using 3D front-end floating-point coordinate values.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
   CMSR_fe_f_draw_point_3d (z_buffer_field, xyz_vector, color,
                            coord_s_len, coord_e_len, color_len, clip_p)

CM_field_id_t  z_buffer_field;
double         xyz_vector[3];
unsigned int   color;
unsigned int   coord_s_len;
unsigned int   coord_e_len;
unsigned int   color_len;
int            clip_p;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

   CMSR_FE_F_DRAW_POINT_3D (z_buffer_field, xyz_vector, color,
&                           coord_s_len, coord_e_len, color_len, clip_p)

   INTEGER           z_buffer_field
   DOUBLE PRECISION  xyz_vector(3)
   INTEGER           color
   INTEGER           coord_s_len
   INTEGER           coord_e_len
   INTEGER           color_len
   INTEGER           clip_p
```

Lisp Syntax

```
CMSR:fe-f-draw-point-3d (z-buffer-field xyz-vector color
                         coord-s-len coord-e-len color-len
                          &optional (clip-p t))
```

ARGUMENTS

z_buffer_field	A Paris field identifier. *z_buffer_field* is a CM field with subfields for a floating-point *z*-coordinate value and an unsigned integer color value. The *z* value occupies the most significant bits, and the *color* value occupies the least significant bits. The total length of the field must be (*coord_s_len* + *coord_e_len* + 1 + *color_len*) where *coord_s_len* is the length of the *z*-coordinate significand, *coord_e_len* is the length of the *z*-coordinate exponent, 1 is the sign bit for the *z* value, and *color_len* is the length of the *color* value. The *z* subfield may be accessed by using the value **`CM_add_offset_to_field`**(*z_buffer_field, color_len)* and the color subfield may be accessed by using the value *z_buffer_field.*
xyz_vector	An array of three double-precision floating-point values on the Connection Machine's front-end computer. The values represent the *x*, *y*, and *z* coordinates, respectively, of the point to be drawn into the *z_buffer_field* in Connection Machine memory. The *x* and *y* coordinates specify the location in the *z_buffer_field* VP set that will receive the *z* coordinate value and the *color* value.
color	An unsigned integer containing the color value to be drawn to this point in *z_buffer_field.* The number of bits used to represent the color depends on the bits per pixel to be displayed.
coord_s_len	The length, in bits, of the significand of the floating-point values in the *z* subfield of *z_buffer_field.*
coord_e_len	The length, in bits, of the exponent of the floating-point values in the *z* subfield of *z_buffer_field.*
color_len	The length, in bits, of the color subfield in *z_buffer_field.* This value specifies how many of the least significant bits of *color* to transfer to CM.
clip_p	A symbol indicating whether the points are to be clipped or not. If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), lines, or portions of lines, drawn outside the range of the z-buffer field coordinates are clipped. If *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), it is an error to draw outside the range of the *z_buffer_field.* The range

of the *z_buffer_field* is defined by the length of the 2 axes in the 2D geometry in which it is defined.

DESCRIPTION

`CMSR_fe_f_draw_point_3d` draws a z-buffer image value into *z_buffer_field* at the location specified by the *x* and *y* components of the *xyz_vector_field.* The z-buffer image value is composed of the z value from *xyz_vector_field* and the color value from the *color* argument.

The *Render routines round the floating-point coordinate values to integral pixel values by using the function

```
round(value) = floor(value+0.5)
```

This means that the area of a pixel in floating-point coordinates is (*x*–0.5,*x*+0.5) by (*y*–0.5,*y*+0.5). For example, the first pixel is lit by the coordinates from (–.05, –0.5) to (0.5, 0.5), and a display space of size 128 by 128 has a floating-point extent of (–0.5, 127.5) to (–0.5, 127.5).

The system performs a z-buffer comparison in *z_buffer_field* based on a right handed coordinate system, that is, positive *z* increases into the screen, positive *y* increases toward the bottom of the screen, and positive *x* increases to the right. The origin of the image (0,0) is the upper left corner. If a z-buffer image value is written to a point in *z_buffer_field* that already contains an image value, the color value associated with the smaller *z* (“nearer” the viewer) is chosen over the color value with a larger *z.*

ERRORS

With CM safety mode on, an error is signaled if you call **`CMSR_fe_f_draw_point_3d`** with

- coordinates not within the destination image buffer if *clip–p* is false
- a *z_buffer_field* that is not part of a two-dimensional VP set
- a *color_length* that is longer than the length of the color component of *z_buffer_field*

SEE ALSO

`CMSR_f_draw_point_3d`

`CMSR_f_draw_point`

`CMSR_s_draw_point`

`CMSR_fe_f_draw_point`

`CMSR_fe_s_draw_point`

CMSR_fe_s_draw_point

Draws a point into the CM image buffer field using front-end signed integer coordinate values.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
CMSR_fe_s_draw_point  (image_buffer_field, x, y, color,
                       color_length, combiner, clip_p)

CM_field_id_t      image_buffer_field;
int                x, y, color;
unsigned int       color_length;
CMSR_combiner_t    combiner;
int                clip_p;
```

Fortran Syntax

```
      INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

      SUBROUTINE CMSR_FE_S_DRAW_POINT (image_buffer_field, x, y, color,
     &                                 color_length, combiner, clip_p)

      INTEGER image_buffer_field x, y, color, color_length
      INTEGER combiner
      INTEGER clip_p
```

Lisp Syntax

```
CMSR:fe-s-draw-point  (image-buffer-field x y color color-length
                       &key (combiner :default) (clip-p t))
```

ARGUMENTS

image_buffer_field	A Paris field identifier. The specified point is drawn into this field at the location specified by *x* and *y*. The *image_buffer_field* must be in a two-dimensional VP set. It need not be in the current VP set.
x, *y*	Front-end integer coordinate values, defining the *x* and *y* coordinates, respectively, at which to draw the point in the image buffer field.
color	An unsigned integer containing the color value to be drawn to this point in *z_buffer_field.* The number of bits used to represent the color depends on the bits per pixel to be displayed.
color_length	The length, in bits, of *color.*
combiner	A symbol defining the method used to combine the color values being written into the image buffer field with the values already in the image buffer field. Valid values are listed in the table below.

C Values	Fortran Values	Lisp Keywords
`CMSR_default`	`CMSR_default`	`:DEFAULT`
`CMSR_overwrite`	`CMSR_overwrite`	`:OVERWRITE`
`CMSR_logior`	`CMSR_logior`	`:LOGIOR`
`CMSR_logand`	`CMSR_logand`	`:LOGAND`
`CMSR_logxor`	`CMSR_logxor`	`:LOGXOR`
`CMSR_u_add`	`CMSR_u_add`	`:U-ADD`
`CMSR_s_add`	`CMSR_s_add`	`:S-ADD`
`CMSR_u_min`	`CMSR_u_min`	`:U-MIN`
`CMSR_s_min`	`CMSR_s_min`	`:S-MIN`
`CMSR_u_max`	`CMSR_u_max`	`:U-MAX`
`CMSR_s_max`	`CMSR_s_max`	`:S-MAX`

clip_p

A symbol indicating whether the point is to be clipped or not.

If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), points outside the range of the image buffer field coordinates are not drawn.

If *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), it is an error to draw outside the range of the image buffer field.

DESCRIPTION

CMSR_fe_s_draw_point draws a point, defined with front-end coordinate values, into the specified image buffer field.

The value in *color_field* is drawn into *image_buffer_field* at the processor location (*x*,*y*).

The value written into the location in the image buffer is a combination of the value of *color_field*, the previous value at that location, and the value of any other points overwriting the same location. The method used to combine these values is controlled by the *combiner* parameter.

If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), points drawn with coordinates outside the range of the image buffer field coordinates are clipped. If the CM safety mode is on and *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), an error is signaled if the point is not within the boundaries of the destination image buffer field.

ERRORS

With CM safety mode on, an error is signaled if you call **CMSR_fe_s_draw_point** with

- coordinates not within the destination image buffer if *clip_p* is on
- an *image_buffer_field* that is not part of a two-dimensional VP set
- a *color_length* that is longer than the length of the *image_buffer_field* or *color_field.*

SEE ALSO

CMSR_f_draw_point

CMSR_fe_f_draw_point

CMSR_s_draw_point

CMSR_f_draw_line

Draws a set of lines into a CM image buffer field using floating-point coordinates.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
CMSR_f_draw_line   (image_buffer_field, x_start_field, y_start_field,
                    x_end_field, y_end_field, color_field, coord_s_length,
                    coord_e_length, color_length, combiner,
                    draw_end_point_p, clip_p)

CM_field_id_t  image_buffer_field, color_field,
               x_start_field, y_start_field,
               x_end_field, y_end_field;
unsigned int   color_length, coord_s_length,coord_e_length;
CMSR_combiner  combiner;
int            draw_end_point_p, clip_p;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

SUBROUTINE CMSR_F_DRAW_LINE
& image_buffer_field, x_start_field, y_start_field, x_end_field, y_end_field,
& color_field, coord_s_length, coord_e_length, color_length,
& combiner, draw_end_point_p, clip_p)

INTEGER  image_buffer_field, color_field
INTEGER  x_start_field, y_start_field, x_end_field, y_end_field
INTEGER  color_length, coord_s_length, coord_e_length combiner
INTEGER  draw_end_point_p, clip_p
```

Lisp Syntax

```
CMSR:f-draw-line   (image-buffer-field x-start-field y-start-field x-end-field
                   y-end-field color-field coord-s-length coord-e-length
                   color-length &key (combiner :default)
                   (draw-end-point-p t) (clip-p t))
```

ARGUMENTS

image_buffer_field A Paris field identifier. The specified lines are drawn into this field at the locations specified by the (*x_start_field*, *y_start_field*) and (*x_end_field*, *y_end_field*) coordinate pairs. The *image_buffer_field* must be at least as long as *color_length*. The *image_buffer_field* must be in a two-dimensional VP set, and may or may not be in the same VP set as the *color_field* and coordinate fields.

x_start_field, *y_start_field*

Paris field identifiers. These fields contain floating-point values that are the *x* and *y* coordinates, respectively, at which to begin drawing the lines. These fields must be in the current VP set.

x_end_field, *y_end_field*

Paris field identifiers. These fields contain floating-point values that are the *x* and *y* coordinates, respectively, at which to end the lines. These fields must be in the current VP set.

color_field A Paris field identifier. This field contains the value drawn into the image buffer. The *color_field* must be in the current VP set.

combiner A symbol defining the method used to combine the color values being written into the image buffer field with the values already in the image buffer field. Valid values are listed in the table below.

C Values	Fortran Values	Lisp Keywords
`CMSR_default`	`CMSR_default`	`:DEFAULT`
`CMSR_overwrite`	`CMSR_overwrite`	`:OVERWRITE`
`CMSR_logior`	`CMSR_logior`	`:LOGIOR`
`CMSR_logand`	`CMSR_logand`	`:LOGAND`
`CMSR_logxor`	`CMSR_logxor`	`:LOGXOR`
`CMSR_u_add`	`CMSR_u_add`	`:U-ADD`
`CMSR_s_add`	`CMSR_s_add`	`:S-ADD`
`CMSR_u_min`	`CMSR_u_min`	`:U-MIN`
`CMSR_s_min`	`CMSR_s_min`	`:S-MIN`
`CMSR_u_max`	`CMSR_u_max`	`:U-MAX`
`CMSR_s_max`	`CMSR_s_max`	`:S-MAX`

coord_s_length, coord_e_length	Unsigned integers specifying the length of the floating-point significand and exponent, respectively, in the coordinate values used for *x_start_field, y_start_field, x_end_field*, and *y_end_field.*
color_length	The length, in bits, of the *color_field.*
draw_end_point_p	A symbol indicating whether the end points of the lines (*x_end_field, y_end_field*) are to be drawn or not. The end points are only drawn if *draw_end_point_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp).
clip_p	A symbol indicating whether the lines are to be clipped or not. If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), lines, or portions of lines, drawn outside the range of the image buffer field coordinates are clipped. If *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), it is an error to draw outside the range of the image buffer field.

DESCRIPTION

`CMSR_f_draw_line` draws a set of lines, defined with floating-poing coordinates, into the specified image buffer field.

For each active processor in the current VP set, the value in *color_field* is drawn into the *image_buffer_field* at the processor locations along the line from the (*x_start_field, y_start_field*) to (*x_end_field, y_end_field*). The starting points of the lines are always drawn; the end points are only drawn if *draw_end_point_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp).

The *Render routines round the floating-point coordinate values to integral pixel values by using the function

`round(value) = floor(value+0.5)`

This means that the area of a pixel in floating-point coordinates is (x–0.5, x+0.5) by (y–0.5, y+0.5). For example, the first pixel is lit by the coordinates from (–.05, –0.5) to (0.5, 0.5), and a display space of size 128 by 128 has a floating-point extent of (–0.5, 127.5) to (–0.5, 127.5).

The value written into each location in the image buffer is a combination of the value of *color_field*, the previous value at that location, and the value of any other lines writing to the same location. The method used to combine these values is controlled by the *combiner* parameter.

If the CM safety mode is on and *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), an error is signaled if a line is not within the boundaries of the destination image buffer field. If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), lines, or portions of lines, drawn outside the range of the image buffer field coordinates are clipped.

Note

Line drawing using floating-point coordinates with **`CMSR_fe_f_draw_line`** or **`CMSR_f_draw_line`** is significantly slower than the line drawing routines that use integer coordinates. If you are hampered by the speed of the floating-point routines, you may want to convert the coordinates to integer values and then use **`CMSR_fe_s_draw_line`** or **`CMSR_s_draw_line`**.

ERRORS

With CM safety mode on, an error is signaled if you call **`CMSR_draw_f_line`** with

- coordinates not within the destination image buffer if *clip_p* is false
- an *image_buffer_field* that is not part of a two-dimensional VP set
- a *color_length* that is longer than the length of the *image_buffer_field* or *color_field*
- color or coordinate fields that are not in the current VP set

SEE ALSO

`CMSR_fe_s_draw_line`

`CMSR_fe_f_draw_line`

`CMSR_s_draw_line`

CMSR_s_draw_line

Draws a set of lines into the CM image buffer field using signed integer coordinate values.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
CMSR_s_draw_line  (image_buffer_field, x_start_field, y_start_field,
                   x_end_field, y_end_field, color_field, coord_length,
                   color_length, combiner, draw_end_point_p, clip_p)

CM_field_id_t      image_buffer_field, color_field, x_start_field,
                   y_start_field;
CM_field_id_t      x_end_field, y_end_field, color_field;
unsigned int       coord_length, color_length;
CMSR_combiner_t    combiner;
int                draw_end_point_p, clip_p;
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

    SUBROUTINE CMSR_S_DRAW_LINE
                                (image_buffer_field, x_start_field, y_start_field,
&                              x_end_field, y_end_field, color_field, coord_length,
&                              color_length, combiner, draw_end_point_p, clip_p)

    INTEGER  image_buffer_field, x_start_field, y_start_field
    INTEGER  x_end_field, y_end_field, color_field
    INTEGER  coord_length, color_length
    INTEGER  combiner
    INTEGER  draw_end_point_p, clip_p
```

Lisp Syntax

```
CMSR:s-draw-line   (image-buffer-field x-start-field y-start-field x-end-field
                   y-end-field color-field coord-length color-length
                   &key (combiner :default) (draw-end-point-p t)
                   (clip-p t))
```

ARGUMENTS

image_buffer_field	A Paris field identifier. The specified lines are drawn into this field at the locations specified by the (*x_start_field, y_start_field*) and (*x_end_field, y_end_field*) coordinate pairs. The *image_buffer_field* must be at least as long as *color_length*. The *image_buffer_field* must be in a two-dimensional VP set, and may or may not be in the same VP set as the *color_field* and coordinate fields. It need not be in the current VP set.
x_start_field, y_start_field	Paris field identifiers. These fields contain integer values that are the *x* and *y* coordinates, respectively, at which to begin drawing the lines. These fields must be in the current VP set.
x_end_field, y_end_field	Paris field identifiers. These fields contain integer values that are the *x* and *y* coordinates, respectively, at which to end the lines. These fields must be in the current VP set.
color_field	A Paris field identifier. This field contains the value drawn into the image buffer. *color_field* must be in the current VP set.
combiner	A symbol defining the method used to combine the color values being written into the image buffer field with the values already in the image buffer field. Valid values are listed in the table below.

C Values	Fortran Values	Lisp Keywords
`CMSR_default`	`CMSR_default`	`:DEFAULT`
`CMSR_overwrite`	`CMSR_overwrite`	`:OVERWRITE`
`CMSR_logior`	`CMSR_logior`	`:LOGIOR`
`CMSR_logand`	`CMSR_logand`	`:LOGAND`
`CMSR_logxor`	`CMSR_logxor`	`:LOGXOR`
`CMSR_u_add`	`CMSR_u_add`	`:U-ADD`
`CMSR_s_add`	`CMSR_s_add`	`:S-ADD`
`CMSR_u_min`	`CMSR_u_min`	`:U-MIN`
`CMSR_s_min`	`CMSR_s_min`	`:S-MIN`
`CMSR_u_max`	`CMSR_u_max`	`:U-MAX`
`CMSR_s_max`	`CMSR_s_max`	`:S-MAX`

color_length	The length, in bits, of the *color_field*.

coord_length	An unsigned integer specifying the length of the coordinates used for *x_start_field*, *y_start_field*, *x_end_field*, and *y_end_field*. **Note:** In routines using signed integer coordinates, *coord_length* must include room for the sign bit.
draw_end_point_p	A symbol indicating whether the end point of the line (*x_end_field*, *y_end_field*) is drawn or not. The end point is only drawn if *draw_end_point_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp).
clip_p	A symbol indicating whether the line is to be clipped or not. If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), lines or portions of lines drawn outside the range of the image buffer field coordinates are clipped. If *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), it is an error to draw outside the range of the image buffer field.

DESCRIPTION

`CMSR_s_draw_line` draws lines, defined with signed integer coordinates, into the specified image buffer field.

For each active processor in the current VP set, the value in *color_field* is drawn into *image_buffer_field* at the processor locations along the line from (*x_start_field*, *y_start_field*) to (*x_end_field*, *y_end_field*). The start points of the lines are always drawn; the end points are only drawn if *draw_end_point_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp).

The value written into each location in the image buffer is a combination of the value of *color_field*, the previous value at that location, and the value of any other lines overwriting the same location. The method used to combine these values is controlled by the *combiner* parameter.

If the CM safety mode is on and *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), an error is signaled if the line is not within the boundaries of the destination image buffer field. If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), lines or portions of lines drawn outside the range of the image buffer field coordinates are clipped.

ERRORS

With CM safety mode on, an error is signaled if you call **`CMSR_draw_s_line`** with

- coordinates not within the destination image buffer if *clip_p* is false
- an *image_buffer_field* that is not part of a two-dimensional VP set
- a *color_length* that is longer than the *image_buffer_field* or *color_field*
- color or coordinate fields not in the current VP set

SEE ALSO

`CMSR_f_draw_line`

`CMSR_fe_s_draw_line`

`CMSR_fe_f_draw_line`

CMSR_fe_f_draw_line

Draws a line into the CM image buffer field using front-end floating-point coordinate values.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
CMSR_fe_f_draw_line(image_buffer_field, x_start, y_start,
                    x_end, y_end, color, color_length, combiner,
                    draw_end_point_p, clip_p)

CM_field_id_t     image_buffer_field;
double            x_start, y_start, x_end, y_end;
int               color;
unsigned int      color_length;
CMSR_combiner_t   combiner;
int               draw_end_point_p, clip_p;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

   SUBROUTINE CMSR_FE_F_DRAW_LINE
&                     (image_buffer_field, x_start, y_start,
&                      x_end, y_end, color, color_length, combiner,
&                      draw_end_point_p, clip_p)

   INTEGER            image_buffer_field
   DOUBLE PRECISION   x_start, y_start, x_end, y_end
   INTEGER            color, color_length, combiner draw_end_point_p
   INTEGER            clip_p
```

Lisp Syntax

```
CMSR:fe-f-draw-line  (image-buffer-field x-start y-start x-end
                      y-end color color-length
                      &key (combiner :default)
                      (draw-end-point-p t) (clip-p t)
```

ARGUMENTS

image_buffer_field	A Paris field identifier. The specified line is drawn into this field at the location specified by (*x_start*, *y_start*) and (*x_end*, *y_end*). The *image_buffer_field* must be in a two-dimensional VP set. It need not be in the current VP set.
x_start, *y_start*	Front-end floating-point coordinate values, defining the *x* and *y* coordinates, respectively, at which to begin drawing the line in the image buffer field.
x_end, *y_end*	Front-end floating-point coordinate values, defining the *x* and *y* coordinates, respectively, at which to end the line in the image buffer field.
color	The value to be drawn into the *image_buffer_field.*
combiner	A symbol defining the method used to combine the color values being written into the image buffer field with the values already in the image buffer field. Valid values are listed in the table below.

C Values	Fortran Values	Lisp Keywords
`CMSR_default`	`CMSR_default`	`:DEFAULT`
`CMSR_overwrite`	`CMSR_overwrite`	`:OVERWRITE`
`CMSR_logior`	`CMSR_logior`	`:LOGIOR`
`CMSR_logand`	`CMSR_logand`	`:LOGAND`
`CMSR_logxor`	`CMSR_logxor`	`:LOGXOR`
`CMSR_u_add`	`CMSR_u_add`	`:U-ADD`
`CMSR_s_add`	`CMSR_s_add`	`:S-ADD`
`CMSR_u_min`	`CMSR_u_min`	`:U-MIN`
`CMSR_s_min`	`CMSR_s_min`	`:S-MIN`
`CMSR_u_max`	`CMSR_u_max`	`:U-MAX`
`CMSR_s_max`	`CMSR_s_max`	`:S-MAX`

color_length	The length of *color* in number of bits.
draw_end_point_p	A symbol indicating whether the end point of the line (*x_end*, *y_end*) is drawn or not. The end point is only drawn if *draw_end_point_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp).

clip_p	A symbol indicating whether the line is to be clipped or not. If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), lines, or portions of lines, drawn outside the range of the image buffer field coordinates are clipped. If *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), it is an error to draw outside the range of the image buffer field.

DESCRIPTION

`CMSR_fe_f_draw_line` draws a line, defined with front-end floating-point coordinate values, into the specified image buffer field.

The value in *color* is drawn into the *image_buffer_field* at the processor locations along the line from (*x_start*, *y_start*) to (*x_end*, *y_end*). The start point is always drawn; the end point is only drawn if *draw_end_point_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp).

The *Render routines round the floating-point coordinate values to integral pixel values by using the function

`round(value) = floor(value+0.5)`

This means that the area of a pixel in floating-point coordinates is $(x-0.5, x+0.5)$ by $(y-0.5, y+0.5)$. For example, the first pixel is lit by the coordinates from (–.05, –0.5) to (0.5, 0.5), and a display space of size 128 by 128 has a floating-point extent of (–0.5, 127.5) to (–0.5, 127.5).

The value written into each location in the image buffer is a combination of the value of *color*, the previous value at that location, and the value of any other lines overwriting the same location. The method used to combine these values is controlled by the *combiner* parameter.

If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), lines, or portions of lines, drawn outside the range of the image buffer field coordinates are clipped. If the CM safety mode is on and *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), an error is signaled if the line is not within the boundaries of the destination image buffer field.

Note

Line drawing using floating-point coordinates with **`CMSR_fe_f_draw_line`** or **`CMSR_f_draw_line`** is significantly slower than the line drawing routines that use integer coordinates. If you are hampered by the speed of the floating-point routines, you may want to convert the coordinates to integer values and than use **`CMSR_fe_s_draw_line`** or **`CMSR_s_draw_line`**.

ERRORS

With CM safety mode on, an error is signaled if you call **`CMSR_fe_f_draw_line`** with

- coordinates not within the destination image buffer field if *clip_p* is false
- an *image_buffer_field* that does not have a two-dimensional VP set geometry
- a *color_length* that is longer than the length of the *image_buffer_field* or *color*
- color or coordinate fields (*x_start*, *y_start* , *x_end*, *y_end*, or *color*) are not in the current VP set

SEE ALSO

`CMSR_f_draw_line`

`CMSR_fe_s_draw_line`

`CMSR_s_draw_line`

CMSR_fe_s_draw_line

Draws a line into the CM image buffer field using front-end signed integer coordinate values.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
CMSR_fe_s_draw_line
                (image_buffer_field, x_start, y_start, x_end, y_end, color,
                 color_length, combiner, draw_end_point_p, clip_p) ;

CM_field_id_t      image_buffer_field;
int                x_start, y_start, x_end, y_end, color;
unsigned int       color_length;
CMSR_combiner_t    combiner;
int                draw_end_point_p, clip_p;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

   SUBROUTINE CMSR_FE_S_DRAW_LINE
&                          (image_buffer_field, x_start, y_start, x_end,
&                           y_end, color, color_length, combiner,
&                           draw_end_point_p, clip_p)

   INTEGER  image_buffer_field
   INTEGER  x_start, y_start, x_end, y_end, color, color_length
   INTEGER  combiner, draw_end_point_p, clip_p
```

Lisp Syntax

```
CMSR:fe-s-draw-line  (image-buffer-field x-start y-start x-end
                      y-end color color-length
                      &key (combiner :default)
                      (draw-end-point-p t) (clip-p t)
```

ARGUMENTS

image_buffer_field	A Paris field identifier. The specified line is drawn into this field at the location specified by (*x_start*, *y_start*) and (*x_end*, *y_end*). The *image_buffer_field* must be in a two-dimensional VP set. It need not be in the current VP set.
x_start, *y_start*	Front-end integer coordinate values, defining the *x* and *y* coordinates, respectively, in the image buffer field at which to begin drawing the line.
x_end, *y_end*	Front-end integer coordinate values, defining the *x* and *y* coordinates, respectively, in the image buffer field at which to end the line.
color	The value to be drawn into the *image_buffer_field.*
combiner	A symbol defining the method used to combine the color values being written into the image buffer field with the values already in the image buffer field. Valid values are listed in the table below.

C Values	Fortran Values	Lisp Keywords
`CMSR_default`	`CMSR_default`	`:DEFAULT`
`CMSR_overwrite`	`CMSR_overwrite`	`:OVERWRITE`
`CMSR_logior`	`CMSR_logior`	`:LOGIOR`
`CMSR_logand`	`CMSR_logand`	`:LOGAND`
`CMSR_logxor`	`CMSR_logxor`	`:LOGXOR`
`CMSR_u_add`	`CMSR_u_add`	`:U-ADD`
`CMSR_s_add`	`CMSR_s_add`	`:S-ADD`
`CMSR_u_min`	`CMSR_u_min`	`:U-MIN`
`CMSR_s_min`	`CMSR_s_min`	`:S-MIN`
`CMSR_u_max`	`CMSR_u_max`	`:U-MAX`
`CMSR_s_max`	`CMSR_s_max`	`:S-MAX`

color_length	The length, in bits, of *color* that is to be transferred to the *image_buffer_field.*
draw_end_point_p	A symbol indicating whether the end point of the line (*x_end*, *y_end*) is drawn or not. The end point is only drawn if *draw_end_point_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp).

clip_p	A symbol indicating whether the line is to be clipped or not. If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), lines, or portions of lines, drawn outside the range of the image buffer field coordinates are clipped. If *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), it is an error to draw outside the range of the image buffer field.

DESCRIPTION

`CMSR_fe_s_draw_line` draws a line, defined with front-end signed integer coordinate values, into the specified image buffer field.

The value in *color* is drawn into the *image_buffer_field* at the processor locations along the line from (*x_start_field*, *y_start_field*) to (*x_end_field*, *y_end_field*). The start point is always drawn; the end point is only drawn if *draw_end_point_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp).

The value written into each location in the image buffer is a combination of the value of *color*, the previous value at that location, and the value of any other lines overwriting the same location. The method used to combine these values is controlled by the *combiner* parameter.

If *clip_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), lines, or portions of lines, drawn outside the range of the image buffer field coordinates are clipped. If the CM safety mode is on and *clip_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), an error is signaled if the line is not within the boundaries of the destination image buffer field.

ERRORS

With CM safety mode on, an error is signaled if you call `CMSR_fe_s_draw_line` with

- coordinates not within the destination image buffer field if *clip_p* is false
- an *image_buffer_field* that does not have a two-dimensional VP set geometry
- a *color_length* that is longer than the length of the *image_buffer_field* or *color*
- fields not in the current VP set

SEE ALSO

`CMSR_f_draw_line`

`CMSR_fe_f_draw_line`

`CMSR_s_draw_line`

CMSR_f_clip_lines

Clips floating-point line coordinates to specified boundaries.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
CMSR_f_clip_lines
                 (x_start_field, y_start_field, x_end_field, y_end_field,
                  x_min, y_min, x_max, y_max, coord_s_length,
                  coord_e_length)

CM_field_id_t  x_start_field, y_start_field, x_end_field, y_end_field;
double         x_min, y_min, x_max, y_max;
unsigned int   coord_s_length, coord_e_length;
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

    SUBROUTINE CMSR_F_CLIP_LINES
&                          (x_start_field, y_start_field, x_end_field,
&                           y_end_field, x_min, y_min, x_max, y_max,
&                           coord_s_length, coord_e_length)

    INTEGER           x_start_field, y_start_field, x_end_field, y_end_field
    DOUBLE PRECISION  x_min, y_min, x_max, y_max
    INTEGER           coord_s_length, coord_e_length
```

Lisp Syntax

```
CMSR:f-clip-lines (x-start-field y-start-field x-end-field y-end-field
                   x-min y-min x-max y-max coord-s-length
                   coord-e-length)
```

ARGUMENTS

x_start_field, y_start_field	Paris field identifiers. These fields contain floating-point values that are the *x* and *y* coordinates, respectively, of the beginning of the lines to be clipped. These fields must be in the current VP set.
x_end_field, y_end_field	Paris field identifiers. These fields contain floating-point values that are the *x* and *y* coordinates, respectively, of the end of the lines to be clipped. These fields must be in the current VP set.
x_min, y_min	Floating-point coordinates, given as front-end values, specifying the lower boundary of the clipping range.
x_max, y_max	Floating-point coordinates, given as front-end values, specifying the upper right corner of the clip range bounding box.
coord_s_length, coord_e_length	Unsigned integers specifying the length of the floating-point significand and exponent, respectively, in the coordinates used for *x_start_field, y_start_field, x_end_field,* and *y_end_field.*

DESCRIPTION

`CMSR_f_clip_lines` clips the line coordinates in the fields *x_start_field, y_start_field, x_end_field,* and *y_end_field* to the clipping range defined by *x_min, y_min, x_max,* and *y_max.* These fields may then be used in a call to **`CMSR_f_draw_line`** to draw the clipped set of lines into a CM image buffer.

`CMSR_f_clip_lines` modifies the line coordinate fields as follows:

- If a line falls completely outside the clipping range, this routine clears the test flag of the virtual processor containing that line's coordinates.
- If only a portion of a line falls within the clipping range, this routine sets the virtual processors' test flag and clips the out-of-range coordinates to the edge of the clipping box. To clip a line, this routine interpolates new endpoints for the line segment and overwrites the fields *x_start_field, y_start_field, x_end_field,* and *y_end_field* for that line.
- If the line falls completely within the clipping range, the routine sets the virtual processors' test flag and leaves the line coordinates unchanged.

NOTE: Before using these clipped fields as source fields for **`CMSR_f_draw_line`**, you must use the Paris instruction **`CM_logand_context_with_test`** to load the context flags of the coordinate fields' VP set with the modified test flag values. (See the *Paris Reference Manual* and *Getting Started in C/Paris Programming*, Chapter 4, for more information.)

ERRORS

With CM safety turned on, an error is signaled if *x_min* is not less than or equal to *x_max* and *y_min* is not less than or equal to *y_max*.

SEE ALSO

`CMSR_s_clip_lines`

CMSR_s_clip_lines

Clips and/or interpolates signed integer line coordinates to specified boundaries.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
CMSR_s_clip_lines
                   (x_start_field, y_start_field, x_end_field, y_end_field,
                    x_min, y_min, x_max, y_max, coord_length)

CM_field_id_t  x_start_field, y_start_field, x_end_field, y_end_field;
int            x_min, y_min, x_max, y_max;
unsigned int   coord_length;
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

    SUBROUTINE CMSR_S_CLIP_LINES
&                             (x_start_field, y_start_field, x_end_field,
&                              y_end_field, x_min, y_min, x_max, y_max,
&                              coord_length)

    INTEGER  x_start_field, y_start_field, x_end_field, y_end_field
    INTEGER  x_min, y_min, x_max, y_max
    INTEGER  coord_length
```

Lisp Syntax

```
CMSR:s-clip-lines
                 (x-start-field y-start-field x-end-field y-end-field x-min
                  y-min x-max y-max coord-length)
```

ARGUMENTS

x_start_field, y_start_field	Paris field identifiers. These fields contain signed integer values that are the *x* and *y* coordinates, respectively, of the beginning of the lines to be clipped. These fields must be in the current VP set.
x_end_field, y_end_field	Paris field identifiers. These fields contain signed integer values that are the *x* and *y* coordinates, respectively, of the end of the lines to be clipped. These fields must be in the current VP set.
x_min, y_min	Integer coordinates, given as front-end values, specifying the lower bounds of the clipping range for the *x* and *y* coordinates.
x_max, y_max	Integer coordinates, given as front-end values, specifying the upper bounds of the clipping range for the *x* and *y* coordinates.
coord_length	An unsigned integer specifying the length of the coordinates used for *x_start_field, y_start_field, x_end_field,* and *y_end_field.*

DESCRIPTION

`CMSR_s_clip_lines` clips the line coordinates in the fields *x_start_field, y_start_field, x_end_field,* and *y_end_field* against the clipping range defined by *x_min, y_min, x_max,* and, *y_max.* These fields may then be used in a call to `CMSR_s_draw_lines` to draw the clipped set of lines into a CM image buffer.

`CMSR_s_clip_lines` modifies the line coordinate fields as follows:

- If a line falls completely outside the clipping range, this routine clears the test flag of the virtual processor containing that line's coordinates.
- If only a portion of a line falls within the clipping range, this routine sets the virtual processors' test flag and clips the out-of-range coordinates to the edge of the clipping box. To clip a line, the routine interpolates new endpoints for the line segment and overwrites the fields *x_start_field, y_start_field, x_end_field,* and *y_end_field* for that line.
- If the line falls completely within the clipping range, the routine sets the virtual processors' test flag and leaves the line coordinates unchanged.

NOTE: Before using these clipped fields as source fields for `CMSR_s_draw_line`, you must use the Paris instruction `CM_logand_context_with_test` to load the context flags of the coordinate fields' VP set with the modified test flag values. (See the *Paris Reference Manual* and *Getting Started in C/Paris Programming*, Chapter 4, for more information.)

ERRORS

With CM safety turned on, an error is signaled if *x_min* is not less than or equal to *x_max* and *y_min* is not less than or equal to *y_max*.

SEE ALSO

`CMSR_f_clip_lines`

CMSR_s_draw_sphere

Draws a set of spheres into the CM image buffer field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
   CMSR_s_draw_sphere
    (image_buffer_field, xyz_vector_field, min_color_field,
     max_color_field,sphere_info_field, radius_field, ncomponents,
     coord_len, color_len, sphere_info_len, radius_len, anti_alias) ;

CMSR_field_id_t   image_buffer_field, xyz_vector_field;
CMSR_field_id_t   min_color_field[ncomponents] ;
CMSR_field_id_t   max_color_field[ncomponents];
CMSR_field_id_t   sphere_info_field, radius_field;
unsigned int      ncomponents, coord_len;
unsigned int      color_len[ncomponents];
unsigned int      sphere_info_len, radius_len;
CMSR_anti_alias_t anti_alias;
```

Fortran Syntax

```
   INCLUDE'/usr/include/cm/cmsr-draw-fort.h

   SUBROUTINE CMSR_S_DRAW_SPHERE
&      (image_buffer_field, xyz_vector_field, min_color_field, max_color_field,
&       sphere_info_field, radius_field, ncomponents, coord_len,
&       color_len, sphere_info_len, radius_len, anti_alias) ;

   INTEGER  image_buffer_field, xyz_vector_field
   INTEGER  min_color_field(ncomponents), max_color_field(ncomponents)
   INTEGER  sphere_info_field, radius_field, ncomponents, coord_len
   INTEGER  color_len(ncomponents), sphere_info_len, radius_len, anti_alias
```

Lisp Syntax

This routine is not available from Lisp.

ARGUMENTS

image_buffer_field	A Paris field identifier. The *image_buffer* is the field into which the spheres are drawn. The *image_buffer* field must be at least (*color_len* * *ncomponents*) + *sphere_info_len* bits.
xyz_vector_field	A Paris field identifier specifying the field containing the coordinates, in screen coordinate space, of the center point of each sphere. The coordinates are signed integer values of *coord_len* length. The length of the entire field must be 3 * *coord_len* bits. The vector field is organized so that *x* occupies the least significant *coord_len* bits, *y* occupies the next *coord_len* bits, and *z* the most significant *coord_len* bits.
min_color_field	An *ncomponent* array of Paris field identifiers specifying the fields containing the minimum color for each color component. The *min_color* field is unsigned. The minimum color is the lowest color map entry to be used to draw the sphere. Spheres are shaded from *max_color* at the center to *min_color* at the edge.
max_color_field	An *ncomponent* array of Paris field identifiers specifying the field containing the maximum color for each sphere. The *max_color* field is unsigned. The maximum color is the highest color map entry to be used to draw the sphere. Spheres are shaded from *max_color* at the center to *min_color* at the edge, or if *min_color* is 0, to black.
sphere_info_field	A Paris field identifier specifying an optional sphere information field. This field can be defined and used by the programmer in any way that is useful to the application. The sphere information is placed in the most significant *sphere_info_len* bits of *image_buffer*. The sphere_info placed in the *image_buffer* is applied to the sphere closest to each pixel.
radius_field	A Paris field identifier. The radius of each sphere to be drawn. This field must be *radius_len* bits long. The radii of the spheres must all be the same, or must decrease with increasing *z*.

ncomponents	An unsigned integer specifying the number of color components that will be used to specify color for all spheres. The number of color components must be at least 1.
coord_len	The length, in bits, of a single coordinate value as specified in *xyz_vector.*
color_len	An *ncomponent* array giving the length, in bits, of each color component in the *image_buffer* field. For example, for RGB true color, *ncomponents* is 3 and *color_len* for each component is 8.
sphere_info_len	The length, in bits of the optional sphere information field.
radius_len	The length, in bits, of the value specified in *radius.*
anti_alias	An enumerated variable indicating the method of anti-aliasing to be applied when drawing the spheres. Valid values are: ▪ **`CMSR_no_anti_alias`** No anti-aliasing is performed. ▪ **`CMSR_edge_anti_alias`** Performs anti-aliasing at the edges between spheres and the background, but leaves jagged edges where spheres interpenetrate.

DESCRIPTION

For each active processor in the current VP set, **`CMSR_s_draw_sphere`** draws a sphere into the *image_buffer* field in CM memory. The fields *xyz_vector*, *min_color*, *max_color*, *sphere_info*, and *radius* must all be in the current VP set when **`CMSR_s_draw_sphere`** is called.

Each sphere is centered at the screen coordinates specified by the *x* and *y* components of the *xyz_vector* field and is drawn with a radius of *radius*. If the center of the sphere is outside the boundaries of the image, the entire sphere is not drawn; otherwise, portions of the sphere outside the image boundaries are clipped. Where spheres intersect, the sphere with smaller *z* coordinates (nearer the viewer) overwrite spheres with larger *z* values.

NOTE

`CMSR_s_draw_sphere` calculates hidden surface removal for the set of spheres before drawing them to the image buffer. This routine does *not* write the *z* coordinate to a user accessible z-buffer field as do the 3D point drawing routines, `CMSR_f_draw_point_3d`, `CMSR_fe_f_draw_point_3d`, and `CMSR_s_draw_point_3d`.

The color components are written to the least significant bits of the *image_buffer* field, followed by the *sphere_info*. When the *image_buffer* is written to a display, the low bits of the field are interpreted as a color value to the depth (or bits per pixel) of the display and the high-order bits are left unchanged. The *sphere_info* portion of the field may be used by the application as required.

The spheres are shaded as though a light source were placed at negative infinity along the *z* axis. Shading is based on the range of colors in the display color map from *max_color* to *min_color.* The color specified by *max_color* is drawn at the center of the sphere and color values are then interpolated over the range of colors to *min_color* at the edge of the sphere.

If the *anti_alias* argument is `CMSR_edge_anti_alias`, anti-aliasing is performed to smooth the sphere edges. However, two restrictions apply to anti-aliasing:

- No anti-aliasing is performed where the edge of a sphere intersects with another sphere.
- The color map from *min_color* to *max_color* for each color component must be a single range that increases linearly. That is, if *ncomponents* is 1, the color map must be set up, like a grayscale map, as a single ramp from black to some maximum color. If *ncomponents* = 3, each component must contain a single linear ramp.

CMSR_draw_image

Transfers a subarray of a CM source field into the CM image buffer field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

    void
    CMSR_draw_image
        (image_buffer_field, source_field, source_length, x_offset, y_offset,
        x_start, y_start, x_limit, y_limit, combiner) ;

CM_field_id_t  image_buffer_field, source_field;
unsigned int   source_length;
int            x_offset, y_offset;
int            x_start, y_start, x_limit, y_limit;
CMSR_combiner  combiner;
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

    SUBROUTINE CMSR_DRAW_IMAGE
&       (image_buffer_field, source_field, source_length, x_offset, y_offset,
&        x_start, y_start, x_limit, y_limit, combiner)

    INTEGER  image_buffer_field, source_field, source_length
    INTEGER  x_offset, y_offset x_start, y_start, x_limit, y_limit
    INTEGER  combiner
```

Lisp Syntax

```
CMSR:draw-image (image-buffer-field source-field source-length
                 x-offset y-offset x-start y-start x-limit y-limit
                 &key (combiner :default))
```

ARGUMENTS

image_buffer_field	A Paris field containing unsigned integers. The *source_field* is copied into this field beginning at the location specified by *x_offset* and *y_offset*. The *image_buffer_field* must be in a two-dimensional VP set, and may or may not be in the same VP set as the *source_field*. It need not be in the current VP set.
source_field	A Paris field containing unsigned integers. This field, within the subarray specified by *x_start*, *y_start*, *x_limit*, and *y_limit*, is copied into *image_buffer_field*. The *source_field* must be in a two-dimensional VP set, and must also be in the current VP set.
source_length	An integer specifying the length of the data values in the *source_field* field. The value of *source_length* must be less than, or equal to, the length of the image buffer field.
x_offset, *y_offset*	Front-end integer values, specifying the location in the image buffer field at which to begin loading the values from the *source_field*.
x_start, *y_start*, *x_limit*, *y_limit*	Front-end integer values, defining the location of a rectangle in *source_field* from which values are taken to be loaded into *image_buffer_field*. The values moved include the value at the start location, but exclude the value at the limit location.
combiner	A symbol defining the method used to combine the array values being transferred from the source array with the values already in the image buffer field. Valid values are listed in the table below.

C Values	Fortran Values	Lisp Keywords
`CMSR_default`	`CMSR_default`	`:DEFAULT`
`CMSR_overwrite`	`CMSR_overwrite`	`:OVERWRITE`
`CMSR_logior`	`CMSR_logior`	`:LOGIOR`
`CMSR_logand`	`CMSR_logand`	`:LOGAND`
`CMSR_logxor`	`CMSR_logxor`	`:LOGXOR`
`CMSR_u_add`	`CMSR_u_add`	`:U-ADD`
`CMSR_s_add`	`CMSR_s_add`	`:S-ADD`
`CMSR_u_min`	`CMSR_u_min`	`:U-MIN`
`CMSR_s_min`	`CMSR_s_min`	`:S-MIN`
`CMSR_u_max`	`CMSR_u_max`	`:U-MAX`
`CMSR_s_max`	`CMSR_s_max`	`:S-MAX`

DESCRIPTION

`CMSR_draw_image` transfers the values in the Paris field *source_field*, within the subarray specified by *x_start*, *y_start*, *x_limit*, and *y_limit* into *image_buffer_field.* The subarray will be loaded into *image_buffer_field* beginning at the location specified by *x_offset* and *y_offset.*

Both *source_field* and *image_buffer_field* must be associated with VP sets with two-dimensional geometries.

The *source_field* coordinates (*x_start*, *y_start)* and (*x_limit*, *y_limit)* define the subarray to be moved to the image buffer field. The first element, at the virtual processor location (*x_start*, *y_start*), is moved to the location in the image buffer field specified by (*x_offset*, *y_offset).* The last source element moved is at location (*x_limit* – 1, *y_limit* – 1). The width of the the rectangle is *x_limit* – *x_start*, and the height is *y_limit* – *y_start.*

x_start must be less than *x_limit*, and *y_start* must be less than *y_limit.*

The value written into each location in the image buffer is a combination of the value of *source_field*, the previous value at that location, and the value of any other lines writing to the same location. The method used to combine these values is controlled by the *combiner* parameter.

ERRORS

With CM safety mode on, an error is signaled if you call **`CMSR_draw_image`** with

- *start* coordinate indices greater than, or equal to, the *limit* coordinates
- *start* or *limit* coordinate indices that are out of the bounds of the source field
- *offset* coordinate indices that are out of the bounds of the image buffer field
- a coordinate index sum (*offset* + (*limit* – *start*)), which is out of the bounds of the image buffer field
- a *source_field* or *image_buffer_field* that does not have a two-dimensional VP set geometry
- a *source_length* that is longer than the length of the *image_buffer_field* or *source_field*
- source field not in the current VP set

CMSR_fe_draw_rectangle

Fills a rectangle in a CM image buffer field with a specified color.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
CMSR_fe_draw_rectangle
                        (image_buffer_field, x, y, width, height, color, depth)

CM_field_id_t  image_buffer_field;
int            x;
int            y;
int            width;
int            height;
int            color;
unsigned int   depth;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

   SUBROUTINE CMSR_FE_DRAW_RECTANGLE
&                          (image_buffer_field, x, y, width, height, color, depth)

   INTEGER  field;
   INTEGER  x;
   INTEGER  y;
   INTEGER  width;
   INTEGER  height;
   INTEGER  color;
   INTEGER  depth;
```

Lisp Syntax

```
CMSR:fe-draw-rectangle
                        (image_buffer_field, x, y, width, height, color, depth)
```

ARGUMENTS

image_buffer_field	A Paris field identifier. The specified rectangle is drawn into this field. The *image_buffer_field* must be in a two-dimensional VP set. It need not be in the current VP set.
x, *y*	The position in the image buffer field at which to begin drawing the rectangle. The position is measured in pixels from the upper left corner. *x* is the horizontal distance to the right. *y* is the vertical distance down.
width, *height*	The dimensions in pixels of the rectangle to be drawn. *width* is the horizontal distance of the rectangle from (*x*, *y*). *height* is the vertical distance of the rectangle from (*x*, *y*).
color	An integer specifying the color value to be written into the image buffer field.
depth	The length, in bits, of the image buffer *field*.

DESCRIPTION

`CMSR_fe_draw_rectangle` draws a filled rectangle of the specified *color* into the image buffer *field*. The *x* and *y* arguments define the location in the image buffer at which to begin drawing the rectangle, and *width* and *height* specify the number of pixels in each dimension of the image. The rectangle fills the image buffer from (*x*, *y*) at the upper left corner to ((*x* + *width*), (*y* + *height*)) at the lower right.

SEE ALSO

`CMSR_fe_display_rectangle`

CMSR_write_array_to_field

Writes an image packed into a front-end array to a CM image buffer field.

SYNTAX

C Syntax

```
#include <cm/display.h>

void
  CMSR_write_array_to_field
              (field, array, array_width, array_height, array_element_size) ;

CM_field_id_t  field;
char           *array;
unsigned int   array_width, array_height;
unsigned int   array_element_size;
```

Fortran Syntax

```
      INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

      SUBROUTINE CMSR_WRITE_ARRAY_TO_FIELD
&                (field, array, array_width, array_height, array_element_size)

      INTEGER  field
      CHAR*(*) array
      INTEGER  array_width
      INTEGER  array_height
      INTEGER  array_element_size
```

Lisp Syntax

```
CMSR:write-array-to-field
              (field, array, &optional [array–element–size nil))
```

ARGUMENTS

field	The destination field. This field must be 2D.

array	A 2D array on the front-end computer to be copied to the field.
array_width	The number of image elements, in *array_element_size* units, along the *faster varying* dimension of the front-end array. For Fortran this is the first index; for C this is the second index. This is the axis that is mapped to axis 0 of the field. Because **`CMSR_write_array_to_field`** does packed-bit transfers, the width of the array must be byte-aligned, that is, (*array_width* * *array_element_size*) % 8 = 0 For Lisp, the array dimensions can be determined and must not be specified.
array_heigh	The number of image elements along the slower varying dimension of the front-end array. For Fortran this is the second index; for C this is the first index. This axis is mapped to axis 1 in the field. For Lisp, the array dimensions can be determined and must not be specified.
array_element_size	The length, in bits, of the image array elements packed into the front-end array, *array*. Usually this is the depth of the image to be displayed. This must be a power of two between 1 and 128. In Lisp, this defaults to the actual size of an array element.

DESCRIPTION

`CMSR_write_array_to_field` copies an image array packed into *array* on the front-end computer to *field* in CM memory. The front-end array must be a 2D array but can be any front-end data type.

The three parameters *array_width*, *array_height*, and *array_element_size* define the image array packed into the front-end array *array*. The *array_element_size* argument specifies the length in bits of each pixel value in the image array. The arguments *array_width* and *array_height* are the total number of image elements (pixels) in each dimension of the image array.

Beginning at the first element of the array, an *array_width* by *array_height* rectangle of *array_element_size* units is copied into the CM field, overwriting any pixel values that are already stored there. The array is transferred so that the fastest varying front-end dimension maps to axis 0 on the CM. Therefore, the CM field should have axis 0 of at least length *array_width* and axis 1 of length *array_height* to hold all of the array.

If the *array_width* or *array_height* is larger than the field, the image array elements beyond the field boundaries to the right and bottom are clipped.

If the *array_width* or *array_height* is smaller than the field dimensions, the portion of the field beyond the array width and height is left unchanged.

Note that *array_width*, *array_height*, and *array_element_size* refer to the image array to be transferred, not to the front-end array in which the image is stored. For example, a 128 by 128 1-bit image could be packed into a 16 by 128 front-end **`char`** or **`CHARACTER`** array, 8 image array elements to a front-end array element. When **`CMSR_write_array_to_field`** writes this image with an *array_element_size* of 1 to the field, each byte of the front-end array source, fills the 1-bit image field in 8 CM processors. If the image *array_element_size* were 8, each byte of the front-end array would go to a single CM processor.

To take one more example, 128 x 128 image that is 32 bits deep might be stored in a 512 x 128 front-end character array; each pixel's data packed into 4 front-end array elements. If this array was passed to **`CMSR_write_array_to_field_1`** with an *array_element_size* of 32 each 4 bytes of the front-end array would be stored in the field of a single CM processor.

`CMSR_write_array_to_field` uses the byte ordering of the front-end computer. So if the front-end byte ordering is MSB-first, the most significant byte and bit of the array elements go to the lowest processor address in CM memory.

SEE ALSO

`CMSR_write_array_to_field_1`

CMSR_write_array_to_field_1

Writes a specified subarray of an image packed into a front-end array to a CM image buffer field.

SYNTAX

C Syntax

```
#include <cm/cmsr-draw.h>

void
  CMSR_write_array_to_field_1
      (field, array, array_width, array_height, array_element_size,
      xoffset, yoffset, xstart, ystart, width, height,
      x_varies_fastest_p, combiner)

CM_field_id_t  field;
void           *array;
unsigned int   array_width, array_height;
unsigned int   array_element_size;
int            xoffset, yoffset;
int            xstart, ystart;
int            width, height;
int            x_varies_fastest_p;
CMSR_combiner_t    combiner;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

   SUBROUTINE CMSR_WRITE_ARRAY_TO_FIELD_1
&      (field, array, array_width, array_height, array_element_size,
&       xoffset, yoffset, xstart, ystart, width, height,
&       x_varies_fastest_p, combiner)

   INTEGER  field
   CHAR*(*) array
   INTEGER  array_width, array_height
   INTEGER  array_element_size
   INTEGER  xoffset, yoffset
   INTEGER  xstart,  ystart
   INTEGER  width, height
   INTEGER  x_varies_fastest_p
   INTEGER  combiner
```

Lisp Syntax

`CMSR:write-array-to-field_1` (*field array* **`&key`** *array–element–size*
xoffset yoffset xstart ystart width height
(*x–varies–fastest–p* **`t`**) *combiner*)

ARGUMENTS

field	The destination field. This field must be 2D.
array	An array on the front-end computer containing the image data to be copied to the field.
array_width	The number of image elements, in *array_element_size* units, along the *faster varying* dimension of the front-end array. For Fortran this is the first index; for C this is the second index. If *x_varies_fastest_p* is true, this is the axis that is mapped to axis 0 of the field. Because **`CMSR_write_array_to_field`** does packed-bit transfers, the width of the array must be byte-aligned, that is, $(\mathit{array\_width} * \mathit{array\_element\_size}) \% 8 = 0$ For Lisp, the array dimensions can be determined and must not be specified.
array_height	The number of image elements, in *array_element_size* units, along the slower varying dimension of the front-end array. For Fortran this is the second index; for C this is the first index. If *x_varies_fastest_p* is true, this axis is mapped to axis 1 in the field. For Lisp, the array dimensions can be determined and must not be specified.
array_element_size	The length, in bits, of the image array elements packed into the front-end array, *array*. Usually this is the depth of the image to be displayed. This must be a power of two between 1 and 128. In Lisp, this defaults to the actual size of an array element.
xoffset, yoffset	The location in *array* at which to begin copying data. The *xoffset* is the number of elements along the width (i.e., the faster varying) dimension of the array in units of *array_element_size*. The *yoffset*

is the number of elements along the height (i.e., the slower varying) dimension. In Lisp, this defaults to (0,0).

xoffset and *yoffset* must be non-negative.

xstart, ystart

The location in *field* at which to begin writing the image array. *xstart* is measured along axis 0; *ystart* is measured along axis 1.

width

The number of image array elements, in *array_element_size* units, to be transferred along the horizontal (i.e., the faster varying) dimension of the array. In Lisp, this defaults to *array_width.*

height

The number of image array elements to be transferred along the vertical (i.e., the slower varying) dimension of the array. In Lisp, this defaults to *array_height.*

combiner

A symbol defining the method used to combine the color values being written from the array into the field with the values already in the field. Valid values are listed in the table below.

C Values	Fortran Values	Lisp Keywords
`CMSR_default`	`CMSR_default`	`:DEFAULT`
`CMSR_overwrite`	`CMSR_overwrite`	`:OVERWRITE`
`CMSR_logior`	`CMSR_logior`	`:LOGIOR`
`CMSR_logand`	`CMSR_logand`	`:LOGAND`
`CMSR_logxor`	`CMSR_logxor`	`:LOGXOR`
`CMSR_u_add`	`CMSR_u_add`	`:U-ADD`
`CMSR_s_add`	`CMSR_s_add`	`:S-ADD`
`CMSR_u_min`	`CMSR_u_min`	`:U-MIN`
`CMSR_s_min`	`CMSR_s_min`	`:S-MIN`
`CMSR_u_max`	`CMSR_u_max`	`:U-MAX`
`CMSR_s_max`	`CMSR_s_max`	`:S-MAX`

x_varies_fastest_p

If *x_varies_fastest_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), the front-end array is mapped directly to the field, aligning the faster-varying axis of the array to axis 0 of the field. This produces the correct results for Fortran arrays and for C arrays that are referenced [*y*][*x*].

If *x_varies_fastest_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), the front-end array is transposed as it is transferred to the field; the faster-varying axis of the array is mapped to axis 1 of the field. This produces correct results for C arrays that are referenced [*x*][*y*].

DESCRIPTION

`CMSR_write_array_to_field_1` copies a specified subarray of the image packed into *array* on the front-end computer to *field* in CM memory. The front-end array must be a 2D array, but it can be any front-end data type that provides a length in bits that is a power of two between 1 and 128.

The three parameters, *array_width*, *array_height*, and *array_element_size* define the image array packed into the front-end array *array.* The *array_element_size* argument specifies the length in bits of each pixel value in the image array. This is the size of each data elements that will be transferred to a CM processor. The arguments *array_width* and *array_height* are the total number of image elements (pixels) in each dimension of the image array.

Note that *array_width*, *array_height*, and *array_element_size* refer to the image array to be transferred, not to the front-end array in which the image is stored. For example, a 128 by 128 1-bit image could be packed into a 16 by 128 front-end **`char`** or **`CHARACTER`** array, 8 image array elements to a front-end array element. When **`CMSR_write_array_to_field_1`**, with an *array_element_size* of 1, writes this image to the field, each byte of the front-end array source fills the 1-bit image field in 8 CM processors. If the image *array_element_size* were 8, each byte of the front-end array would go to a single CM processor.

To take one more example, 128 x 128 image that is 32 bits deep might be stored in a 512 x 128 front-end character array; each pixel's data packed into 4 front-end array elements. If this array was passed to **`CMSR_write_array_to_field_1`** with an *array_element_size* of 32, each 4 bytes of the front-end array would be stored in the field of a single CM processor.

The arguments *xoffset*, *yoffset*, *width*, and *height* define the subarray within the image array that is to be transferred. *xoffset* and *yoffset* define the location in the image array, in *array_element_size* units, at which the transfer should begin. *width* and *height* are the number of image array elements to be transferred in each direction. So, the portion of the image array to be transferred is the subarray from (*xoffset*, *yoffset*) at the upper left corner, to ((*xoffset* + *width*), (*yoffset* + *height*)) at the lower right corner.

Each image element of the subarray is transferred to the corresponding location in the image buffer field beginning at the point defined by (*xstart*, *ystart*). Each array element value is combined with the pixel value at the corresponding field location according to the value of *combiner*. The default value is to overwrite. If the *array_ element_size* is smaller than the depth of the field, an error is signaled. If the *array_ element_size* is larger than the depth of the field, only the lower-order bits of the array element, up to the field's depth, are used.

If the *width* or *height* of the image to be transferred is larger than the image buffer field dimensions, the portion of the array beyond the field boundaries to the right and bottom is clipped. If the *width* or *height* of the image to be transferred is smaller than the field, the portion of the field beyond the array width and height is left unchanged.

`CMSR_write_array_to_field_1` uses the byte ordering of the front-end computer. So if the front-end byte ordering is MSB-first, the most significant byte and bit of the array elements go to the lowest processor address in CM memory.

SEE ALSO

`CMSR_write_array_to_field`

CMSR_read_array_from_field

Packs an image array from a CM field into a front-end array.

SYNTAX

C Syntax

```
#include <cm/cmsr-draw.h>

void
  CMSR_read_array_from_field
            (field, array, array_width, array_height, array_element_size) ;

CM_field_id_t             field;
CMSR_generic_pointer_t    array;
unsigned int              array_width, array_height;
unsigned int              array_element_size;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

   SUBROUTINE CMSR_READ_ARRAY_FROM_FIELD
&             (field, array, array_width, array_height, array_element_size)

   INTEGER  field
   CHAR*(*) array
   INTEGER  array_width
   INTEGER  array_height
   INTEGER  array_element_size
```

Lisp Syntax

```
CMSR:write-array-to-field
            (field, array, &optional array-element-size)
```

ARGUMENTS

field	The source field. This field must be 2D.

array	A 2D front-end array into which the data from the field is to be read.
array_width	The number of elements, in *array_element_size* units, to be stored along the *faster varying* dimension of the front-end array. For Fortran this is the first index; for C this is the second index. Axis 0 of the field is mapped to this dimension of the array. Because `CMSR_read_array_from_field` does packed-bit transfers, the array width must be byte-aligned, that is, $(array\_width * array\_element\_size) \% 8 = 0$
array_height	The number of elements to be stored along the slower varying dimension of the front-end array. For Fortran this is the second index; for C this is the first index. Axis 1 of the field is mapped to this axis. For Lisp, the array dimensions can be determined and must not be specified.
array_element_size	The length, in bits, of *field*. This will also be the size of the image array elements stored in the front-end array *array*. This must be a power of two between 1 and 128. In Lisp, this defaults to the actual size of an array element.

DESCRIPTION

`CMSR_read_array_from_field` reads image values from *field* and packs them into *array* on the front-end computer.

The front-end array must be a 2D array but can be any front-end data type. If the array is not large enough to hold the entire field, the portions of the field image on the right and bottom (+*x*, +*y*) beyond the array dimensions are clipped.

The three parameters *array_width*, *array_height*, and *array_element_size* define the image array in field to be packed into the front-end array *array*. The *array_element_size* argument specifies the depth of the field. *array_width* is the length of axis 0 and *array_height* is the length of axis 1. The array is transferred so that axis 0 of *field* maps to the fastest varying dimension of the front-end array.

Note that *array_width*, *array_height*, and *array_element_size* refer to the image array to be transferred, not to the front-end array in which the image is stored. For example, a 128 by 128 1-bit image could be packed into a 16 by 128 front-end `char` or `CHARACTER` array, 8 image array elements to a front-end array element. When `CMSR_`

`read_array_from_field` writes this image to the front-end array, the image field in 8 CM processors fills a byte of the front-end array. If *array_element_size* is 8, each CM processor fills a byte of the front-end array elements, and if *array_element_size* is 32, each CM processor fills a word of the front-end array elements.

`CMSR_read_array_from_field` uses the byte ordering of the front-end computer. So if the front-end byte ordering is MSB-first, the most significant byte and bit are taken from the lowest processor address in CM memory.

SEE ALSO

`CMSR_read_array_from_field_1`

CMSR_read_array_from_field_1

Packs a specified subarray of an image in a CM image buffer field into a subarray of a front-end array.

SYNTAX

C Syntax

```
#include <cm/cmsr-draw.h>

void
   CMSR_read_array_from_field_1
        (field, array, array_width, array_height, array_element_size,
        xoffset, yoffset, xstart, ystart, width, height,
        x_varies_fastest_p, combiner)

CM_field_id_t            field;
CMSR_generic_pointer_t   array;
unsigned int             array_width, array_height;
unsigned int             array_element_size;
int                      xoffset, yoffset;
int                      xstart, ystart;
unsigned int             width, height;
int                      x_varies_fastest_p;
CMSR_combiner_t          combiner;
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-draw-fort.h'

    SUBROUTINE CMSR_READ_ARRAY_FROM_FIELD_1
&         (field, array, array_width, array_height, array_element_size,
&          xoffset, yoffset, xstart, ystart, width, height,
&          x_varies_fastest_p, combiner)

    INTEGER  field
    CHAR*(*) array
    INTEGER  array_width, array_height
    INTEGER  array_element_size
    INTEGER  xoffset, yoffset
    INTEGER  xstart, ystart
    INTEGER  width, height
    INTEGER  x_varies_fastest_p
    INTEGER  combiner
```

Lisp Syntax

`CMSR:read-array-from-field_1` (*field array* **`&key`** *array–element–size xoffset yoffset xstart ystart width height* (*x–varies–fastest–p* **`t`**) *combiner*)

ARGUMENTS

field	The source field. This field must be 2D.
array	A 2D front-end array into which the data from the field is to be read.
array_width	The number of elements, in *array_element_size* units, to be stored along the *faster varying* dimension of the front-end array. For Fortran this is the first index; for C this is the second index. Because **`CMSR_read_array_from_field`** does packed-bit transfers, the array width must be byte-aligned, that is, $(\mathit{array\_width} * \mathit{array\_element\_size}) \% 8 = 0$
array_height	The number of elements, in *array_element_size* units, to be stored along the slower varying dimension of the front-end array. For Fortran this is the second index; for C this is the first index. For Lisp, the array dimensions can be determined and must not be specified.
array_element_size	The length, in bits, of *field*. This will also be the size of the image array elements stored into the front-end array *array*. This must be a power of two between 1 and 128. In Lisp, this defaults to the actual size of an array element.
xoffset, *yoffset*	The offset into the array at which to begin writing the data from the field. The *xoffset* is the number of elements along the width (i.e., the faster varying) dimension of the array in units of *array_element_size*. The *yoffset* is the number of elements along the height (i.e., the slower varying) dimension. In Lisp, this defaults to (0,0). *xoffset* and *yoffset* must be non-negative.
xstart, *ystart*	The location in *field* at which to begin reading the image. *x_start* is measured in grid coordinates along axis 0, *y_start* along axis 1.

width

The number of image array elements, in *array_element_size* units, to be transferred along the horizontal (i.e., the faster varying) dimension of the array. In Lisp, this defaults to *array_width.*

height

The number of image array elements to be transferred along the vertical (i.e., the slower varying) dimension of the array. In Lisp, this defaults to *array_height.*

x_varies_fastest_p

Indicates whether the first or second array index varies fastest in *array.*

If *x_varies_fastest_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), the axis 0 of the field is mapped directly to the faster-varying axis of the array. This produces the correct results for Fortran arrays and for C arrays that are referenced [*y*][*x*].

If *x_varies_fastest_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), the image data is transposed as it is transferred from the field into the array so that axis 1 of the field is mapped to the faster-varying axis of the array. This produces correct results for C arrays that are referenced [*x*][*y*].

x_varies_fastest_p should be true for Fortran arrays or for C arrays that are referenced [y][x]. For C arrays referenced [*x*][*y*], *x_varies_fastest_p* should be false.

combiner

A symbol defining the method used to combine the color values being written from the field into the array with the values already in the array. Valid values are listed in the table below.

C Values	Fortran Values	Lisp Keywords
`CMSR_default`	`CMSR_default`	`:DEFAULT`
`CMSR_overwrite`	`CMSR_overwrite`	`:OVERWRITE`
`CMSR_logior`	`CMSR_logior`	`:LOGIOR`
`CMSR_logand`	`CMSR_logand`	`:LOGAND`
`CMSR_logxor`	`CMSR_logxor`	`:LOGXOR`
`CMSR_u_add`	`CMSR_u_add`	`:U-ADD`
`CMSR_s_add`	`CMSR_s_add`	`:S-ADD`
`CMSR_u_min`	`CMSR_u_min`	`:U-MIN`
`CMSR_s_min`	`CMSR_s_min`	`:S-MIN`
`CMSR_u_max`	`CMSR_u_max`	`:U-MAX`
`CMSR_s_max`	`CMSR_s_max`	`:S-MAX`

DESCRIPTION

`CMSR_read_array_from_field_1` reads a subarray or the image in *field* and packs it into *array* on the front-end computer. The front-end array must be a 2D array but can be any front-end data type that provides an appropriate number of bits for the depth of the field.

The three parameters *array_width*, *array_height*, and *array_element_size* define the image array in the field from which the subarray is to be read. The *array_element_size* argument specifies the depth of the field. *array_width* is the length of axis 0 and *array_height* is the length of axis 1. The array is transferred so that axis 0 of *field* maps to the fastest varying dimension of the front-end array.

Note that *array_width*, *array_height*, and *array_element_size* refer to the image array to be transferred, not to the front-end array in which the image is stored. For example, a 128 by 128 1-bit image could be packed into a 16 x 16 front-end `char` or `CHARACTER` array, 8 image array elements to a front-end array element. When `CMSR_read_array_from_field` writes this image to the front-end array, the image field in 8 CM processors fills a byte of the front-end array. If *array_element_size* is 8, each CM processor fills a byte of the front-end array elements, and if *array_element_size* is 32, each CM processor fills a word of the front-end array elements.

The arguments *xoffset* and *yoffset* specify the location in the font-end *array* at which to begin reading in the data from the field. The subarray of the field to be read is defined by the arguments *xstart*, *ystart*, *width* and *height*. The portion of the field to be read is from (*xstart*, *ystart*) at the upper left corner, to ((*xstart* + *width*), (*ystart* + *height*) at the lower right. If the array is not large enough to hold the entire field subarray, the portions of the field on the right and bottom (+*x*, +*y*) beyond the array dimensions are clipped.

`CMSR_read_array_from_field_1` uses the byte ordering of the front-end computer. So if the front-end byte ordering is MSB-first, the most significant byte and bit are taken from the lowest processor address in CM memory.

SEE ALSO

`CMSR_read_array_from_field`

Chapter 3

Math Routines

This chapter documents the *Render Math routines. These routines provide utilities for performing common graphics math operations on vectors and matrices in front-end arrays or Connection Machine (CM) fields.

In addition, a set of routines is included for converting between the color spaces RGB, CMY, YIQ , HSV, and HSL.

The next section provides an overview of these routines. Following sections provide detailed descriptions of the individual routines.

3.1 Overview

In many applications it is necessary to manipulate the image's coordinate data for display. You must often scale, rotate, and translate objects in the image to position them properly in the display.

The standard method for applying geometric operations to the image coordinates is through transformation matrices. In this method the point coordinates are represented as vectors (e.g.,[x, y, z]) and a matrix is composed representing the transformation to be performed on the image. By applying the matrix to the set of point vectors, using the conventions of matrix algebra, we can generate a new set of coordinates defining the transformed position of the object in the display space.

*Render provides functions to allocate vector and matrix structures in either CM memory or on the front-end computer, and to perform the basic matrix operations. The routines that operate on front-end vectors and matrices operate on a single instance of these structures allocated as front-end arrays. The CM routines operate on a field of vectors or matrices in parallel.

If you would like more information on the use of matrix method, please see any basic text on computer graphics. Two particularly useful discussions are found in

- David F. Rogers, *Mathematical Elements for Computer Graphics* (New York: McGraw-Hill, 1990).
- James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, *Computer Graphics: Principles and Practice,* 2d ed. (Reading, Mass.: Addison-Wesley, 1990).

3.1.1 Vectors

Vectors in *Render are one-dimensional arrays of either two or three elements.

On the front-end computer, each vector is an array of double-precision floating-point values. On the CM, each vector is a single field of (dimension)*(*signif_len*+*exp_len*+1) bits, where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and the 1 is for a sign bit. Each element of the vector occupies a subfield of (*signif_len*+*exp_len*+1) bits, and these subfields are arranged so that element 0 is in the least significant bits.

In a position vector, representing the coordinates of a point, x occupies element 0, y occupies element 1, and z (if present) occupies element 2.

3.1.2 Matrices

Matrices in *Render are assumed to be square, homogeneous matrices. *Render supports matrices of dimension 2 or 3, for transforming two-dimensional or three-dimensional vectors.

The fact that homogeneous coordinates are used implies that an extra row and column are added to the matrix: a matrix of dimension n contains $(n+1)(n+1)$ elements. The additional row and column hold translation, perspective, and general scaling elements. The elements of a 3D transformation matrix are arranged as follows:

```
column =    0  1  2  3
```

Where

- RS = rotation, reflection, skew, and scaling elements

- P = perspective elements
- T = translation elements
- GS = global scale element

On the front end, a matrix is an appropriately sized array of double-precision floating-point values. On the CM, a matrix is a field of (dimension + 1) * (dimension + 1)(*signif_len* + *exp_len* +1) bits, where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and the 1 is for a sign bit. Each matrix element occupies one floating-point field of (*signif_len* + *exp_len* +1) bits.

3.1.3 Transformation Conventions

*Render uses the following conventions for transformations:

- Objects and operations are defined in a right-handed coordinate system.
- Screen space is right-handed, with the origin in the upper left corner of the screen: *x* increases to the left, *y* increases downwards, and *z* increases into the screen, away from the viewer.
- Rotations are clockwise about an axis as seen by an observer at the origin looking along the axis in the positive direction.

3.1.4 Color Spaces

*Render provides routines to convert between several widely used ways of representing color. Each way of representing color may be thought of as a color "space." For example, the RGB space can be pictured as a cube with three orthogonal axes for red, green, and blue.

Following this model, a specific color is a vector in the appropriate color space. For the color spaces that *Render currently supports, colors are 3-element vectors. Color vectors are organized as shown in the following chart:

Color Space	Element 0	Element 1	Element 2
RGB	red	green	blue
CMY	cyan	magenta	yellow
YIQ	Y(luminance)	I(chromaticity)	Q(chromaticity)
HSV	hue	saturation	value
HSL	hue	saturation	lightness

The supported color spaces are:

- RGB
  Additive color model using three primaries (red, green, and blue) in a Cartesian coordinate system. *Render uses floating-point values to represent the contributions of each primary. A value of 0.0 indicates no contribution, and 1.0 indicates full contribution. The main diagonal of this RGB "color cube" represents gray levels, with equal amounts of each primary. (0,0,0) is black, and (1,1,1) is white.
- CMY
  Subtractive color model using three primaries (cyan, magenta, and yellow) that is useful for hardcopy devices. This model uses the same Cartesian coordinate system as RGB, except that (0,0,0) is white and (1,1,1) is black.
- YIQ
  A re-coding of RGB that is used in commercial color television broadcast. The Y component is the luminance for a color. This term can therefore be used to display a color image as a grayscale image. The I and Q components encode chromaticity.
- HSV
  This model uses hue, saturation, and value to encode colors. The geometry of this space is a truncated hexcone. Hue is an angle from 0 to 2*pi radians. Red is at 0.0, green is at 2*pi/3 radians, and blue is at 4*pi/3 radians. Complementary colors are pi radians apart. Saturation is a fraction from 0.0 to 1.0. Value is a number from 0.0 to 1.0. When S is 0.0, H is irrelevant. When V is 0.0, H and S are irrelevant. (Also called HSB.)
- HSL
  Hue, lightness, and saturation. HSV deformed into a double hexcone. The hue origin at 0.0 radians is red.

Note that the saturation in HSV is not the same as that in HSL.

3.2 Front-End Vector Routines

This section documents the *Render routines that operate on vectors in front-end arrays. Vectors in *Render are one-dimensional arrays of either two or three elements allocated as an array of double-precision floating-point values.

The routines documented here are:

CMSR_fe_v_abs_2d
CMSR_fe_v_abs_3d

Returns the length of the specified front-end vector.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double
   CMSR_fe_v_abs_2d  (src_vector)
double    src_vector[2];

double
   CMSR_fe_v_abs_3d  (src_vector)
double    src_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
DOUBLE PRECISION FUNCTION CMSR_FE_V_ABS_2D  (src_vector)
DOUBLE PRECISION src_vector(2)

DOUBLE PRECISION FUNCTION CMSR_FE_V_ABS_3D  (src_vector)
DOUBLE PRECISION src_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-abs-2d  (src–vector)
CMSR:fe-v-abs-3d  (src–vector)
```

ARGUMENTS

src_vector	The vector for which the length is to be calculated. For **CMSR_fe_v_abs_2d** *src_vector* is a 1 x 2 front-end array of double-precision values. For **CMSR_fe_v_abs_3d** it is a 1 x 3 array.

DESCRIPTION

`CMSR_fe_v_abs_2d` and **`CMSR_fe_v_abs_3d`** return the length of the vector *src_vector.*

SEE ALSO

`CMSR_fe_v_abs_squared_2d`

`CMSR_fe_v_abs_squared_3d`

`CMSR_v_abs_2d`

`CMSR_v_abs_3d`

`CMSR_v_abs_squared_2d`

`CMSR_v_abs_squared_3d`

CMSR_fe_v_abs_squared_2d
CMSR_fe_v_abs_squared_3d

Returns the square of the length of a specified 2D (3D) front-end vector.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double
   CMSR_fe_v_abs_squared_2d  (src_vector)
double    src_vector[2];

double
   CMSR_fe_v_abs_squared_3d  (src_vector)
double    src_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
DOUBLE PRECISION FUNCTION CMSR_FE_V_ABS_SQUARED_2D(src_vector)
DOUBLE PRECISION src_vector(2)

DOUBLE PRECISION FUNCTION CMSR_FE_V_ABS_SQUARED_3D(src_vector)
DOUBLE PRECISION src_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-abs-squared-2d  (src-vector)
CMSR:fe-v-abs-squared-3d  (src-vector)
```

ARGUMENTS

src_vector	The vector for which the length squared is to be calculated. For **CMSR_fe_v_abs_squared_2d** *src_vector* is a 1 x 2 front-end array of double-precision values. For **CMSR_fe_v_abs_squared_3d** it is a 1 x 3 array.

DESCRIPTION

`CMSR_fe_v_abs_squared_2d` returns the square of the length of a 2D (1 x 2) *src_vector* in front-end memory.

`CMSR_fe_v_abs_squared_3d` returns the square of the length of a 3D (1 x 3) *src_vector* in front-end memory.

SEE ALSO

`CMSR_fe_v_abs_2d`

`CMSR_fe_v_abs_3d`

`CMSR_v_abs_2d`

`CMSR_v_abs_3d`

`CMSR_v_abs_squared_2d`

`CMSR_v_abs_squared_3d`

CMSR_fe_v_add_2d
CMSR_fe_v_add_3d

Performs element-wise addition of two vectors.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
   CMSR_fe_v_add_2d  (src1_vector, src2_vector, dest_vector)
double    src1_vector[2], src2_vector[2], dest_vector[2];

double *
   CMSR_fe_v_add_3d  (src1_vector, src2_vector, dest_vector)
double    src1_vector[3], src2_vector[3], dest_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_V_ADD_2D  (src1_vector, src2_vector, dest_vector)
DOUBLE PRECISION  src1_vector(2), src2_vector(2), dest_vector(2)

SUBROUTINE CMSR_FE_V_ADD_3D  (src1_vector, src2_vector, dest_vector)
DOUBLE PRECISION  src1_vector(3), src2_vector(3), dest_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-add-2d(src1-vector src2-vector &optional dest-vector)
CMSR:fe-v-add-3d(src1-vector src2-vector &optional dest-vector)
```

ARGUMENTS

src1_vector, src2_vector	One-dimensional arrays containing the vectors to be added.
dest_vector	A one-dimensional array containing the result of adding *src1_vector* and *src2_vector*. For the 2D routine these are 2-element arrays; for the 3D routine these are 3-element arrays.

DESCRIPTION

`CMSR_fe_v_add_2d` and `CMSR_fe_v_add_3d` do element-wise addition of the components of *src1_vector* and *src2_vector* and put the result in *dest_vector*. In C and Lisp this routine also returns a pointer to *dest_vector*.

dest_vector may be the same as either *src1_vector* or *src2_vector*.

If a vector is a position vector, *x* occupies the first element, *y* occupies the second element, and *z* (if present) occupies the third element.

CMSR_fe_v_copy_2d
CMSR_fe_v_copy_3d

Copies one vector to another.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
   CMSR_fe_v_copy_2d  (src_vector, dest_vector)
double    src_vector[2], dest_vector[2];

double *
   CMSR_fe_v_copy_3d  (src_vector, dest_vector)
double    src_vector[3], dest_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_V_COPY_2D  (src_vector, dest_vector)
DOUBLE PRECISION  src_vector(2), dest_vector(2)

SUBROUTINE CMSR_FE_V_COPY_3D  (src_vector, dest_vector)
DOUBLE PRECISION  src_vector(3), dest_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-copy-2d(src-vector &optional dest-vector)
CMSR:fe-v-copy-3d(src-vector &optional dest-vector)
```

ARGUMENTS

src_vector	A one-dimensional array containing the vector to be copied.
dest_vector	A one-dimensional array to which *src_vector* is to be copied.

For the 2D routine these are 2-element arrays; for the 3D routine these are 3-element arrays.

DESCRIPTION

`CMSR_fe_v_copy_2d` and **`CMSR_fe_v_copy_3d`** copy *src_vector* to *dest_vector*. In C and Lisp a pointer is also returned to *dest_vector*.

If a vector is a position vector, *x* occupies the first element, *y* occupies the second element, and *z* (if present) occupies the third element.

CMSR_fe_v_cos_between_2d
CMSR_fe_v_cos_between_3d

Computes cosine of angle between two (three) vectors.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double
   CMSR_fe_v_cos_between_2d (src1_vector, src2_vector)
double    src1_vector[2], src2_vector[2];

double
   CMSR_fe_v_cos_between_3d (src1_vector, src2_vector)
double    src1_vector[3], src2_vector[3];
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-math-fort.h'
    DOUBLE PRECISION FUNCTION CMSR_FE_V_COS_BETWEEN_2D
&                                              (src1_vector, src2_vector)
    DOUBLE  PRECISION  src1_vector(2), src2_vector(2)

    DOUBLE PRECISION FUNCTION CMSR_FE_V_COS_BETWEEN_3D
&                                              (src1_vector, src2_vector)
    DOUBLE  PRECISION  src1_vector(3), src2_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-cos-between-2d(src-vector1 src-vector2)
CMSR:fe-v-cos-between-3d(src-vector1 src-vector2)
```

ARGUMENTS

src1_vector, src2_vector

One-dimensional arrays containing the vectors.

For the 2D routine these are 2-element arrays; for the 3D routine these are 3-element arrays. *x* occupies element 0, *y* occupies element 1, and *z* (if present) occupies element 2.

DESCRIPTION

`CMSR_fe_v_cos_between_2d` and **`CMSR_fe_v_cos_between_3d`** return the cosine of the angle between two vectors. This is the dot-product of the normalized vectors. The source vectors, *src1_vector* and *src2_vector*, need not be unit length.

Neither vector should be 0 length.

ERRORS

If either vector is of length 0, the result of this routine is undefined.

CMSR_fe_v_cross_product_3d

Calculates the cross-product of two 3D vectors.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_v_cross_product_3d (src1_vector, src2_vector, to_vector)
double    src1_vector[3];
double    src2_vector[3];
double    to_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_V_CROSS_PRODUCT_3D(vector1, vector2, to_vector)
DOUBLE PRECISION src1_vector(3)
DOUBLE PRECISION src2_vector(3)
DOUBLE PRECISION to_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-cross-product-3d
       (src1-vector src2-vector &optional to-vector)
```

ARGUMENTS

src1_vector, src2_vector	1 x 3 arrays of double-precision numbers containing the vectors to be operated on.
to_vector	A 1 x 3 array in which the cross-product of *src1_vector* and *src2_vector* is returned.

DESCRIPTION

`CMSR_fe_v_cross_product_3d` calculates the cross-product between the 3-dimensional vectors *src1_vector* and *src2_vector* and stores the result in *to_vector.*

In C and Lisp, **`CMSR_fe_v_cross_product_3d`** also returns a pointer to *to_vector.*

SEE ALSO

`CMSR_cross_product_3d`

CMSR_fe_v_dot_product_2d
CMSR_fe_v_dot_product_3d

Returns the dot product of two 2D (3D) vectors.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double
  CMSR_fe_v_dot_product_2d (src1_vector, src2_vector)
double    src1_vector[2];
double    src2_vector[2];

double
  CMSR_fe_v_dot_product_3d (src1_vector, src2_vector)
double    src1_vector[3];
double    src2_vector[3];
```

Fortran Syntax

```
      INCLUDE '/usr/include/cm/cmsr-math-fort.h'
      DOUBLE PRECISION FUNCTION CMSR_FE_V_DOT_PRODUCT_2D
     &                                          (src1_vector, src2_vector)
      DOUBLE PRECISION src1_vector(2)
      DOUBLE PRECISION src2_vector(2)

      DOUBLE PRECISION FUNCTION CMSR_FE_V_DOT_PRODUCT_3D
     &                                          (src1_vector, src2_vector)
      DOUBLE PRECISION src1_vector(3)
      DOUBLE PRECISION src2_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-dot-product-2d (src1-vector src2-vector)
CMSR:fe-v-dot-product-3d (src1-vector src2-vector)
```

ARGUMENTS

src1_vector, src2_vector

1 x 2 arrays of double-precision numbers containing the vectors to be operated on.

DESCRIPTION

`CMSR_fe_v_dot_product_2d` and **`CMSR_fe_v_dot_product_3d`** return the dot product of the two front-end vectors *src1_vector* and *src2_vector*.

SEE ALSO

`CMSR_v_dot_product_2d`

`CMSR_v_dot_product_3d`

CMSR_fe_v_is_zero_2d
CMSR_fe_v_is_zero_3d

Tests whether a vector is zero length.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
int
  CMSR_fe_v_is_zero_2d (vector)
double vector[2];

int
  CMSR_fe_v_is_zero_3d (vector)
double vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
LOGICAL FUNCTION CMSR_FE_V_IS_ZERO_2D (vector)
DOUBLE PRECISION vector(2)

LOGICAL FUNCTION CMSR_FE_V_IS_ZERO_3D (vector)
DOUBLE PRECISION vector(3)
```

Lisp Syntax

```
CMSR:fe-v-is-zero-2d (vector)
CMSR:fe-v-is-zero-3d (vector)
```

ARGUMENTS

vector	A one-dimensional array containing the vector to be tested. For the 2D routine these are 2-element arrays; for the 3D routine these are 3-element arrays.

DESCRIPTION

`CMSR_fe_v_is_zero_2d` and **`CMSR_fe_v_is_zero_3d`** test whether *vector* is zero length. If the given vector is of length 0, these routines return true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp). If the vector has length, these routines return false (.FALSE. in Fortran, NULL in C, nil in Lisp).

The *x* coordinate occupies the first element, *y* occupies the second element, and *z* (if present) occupies the third element.

CMSR_fe_v_negate_2d
CMSR_fe_v_negate_3d

Multiplies each vector element by –1.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

double *
   CMSR_fe_v_negate_2d  (src_vector, dest_vector)
double    src_vector[2], dest_vector[2];

double *
   CMSR_fe_v_negate_3d  (src_vector, dest_vector)
double    src_vector[3], dest_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_V_NEGATE_2D  (src_vector, dest_vector)
DOUBLE PRECISION src_vector(2), dest_vector(2)

SUBROUTINE CMSR_FE_V_NEGATE_3D  (src_vector, dest_vector)
DOUBLE PRECISION src_vector(3), dest_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-negate-2d(src–vector &optional dest–vector)
CMSR:fe-v-negate-3d(src–vector &optional dest–vector)
```

ARGUMENTS

src_vector	A one-dimensional array containing the vector to be negated.
dest_vector	A one-dimensional array containing the result of negating *src_vector.* For the 2D routine these are 2-element arrays; for the 3D routine these are 3-element arrays.

DESCRIPTION

`CMSR_fe_v_negate_2d` and `CMSR_fe_v_negate_3d` multiply each vector element by –1 and put the result in *dest_vector*. In C and Lisp these routines also return a pointer to *dest_vector.*

If a vector is a position vector, *x* occupies the first element, *y* occupies the second element, and *z* (if present) occupies the third element.

CMSR_fe_v_normalize_2d
CMSR_fe_v_normalize_3d

Normalizes a vector to a unit vector.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
   CMSR_fe_v_normalize_2d  (src_vector, dest_vector)
double    src_vector[2], dest_vector[2];

double *
   CMSR_fe_v_normalize_3d  (src_vector, dest_vector)
double    src_vector[3], dest_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_V_NORMALIZE_2D  (src_vector, dest_vector)
DOUBLE PRECISION src_vector(2), dest_vector(2)

SUBROUTINE CMSR_FE_V_NORMALIZE_3D  (src_vector, dest_vector)
DOUBLE PRECISION src_vector(3), dest_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-normalize-2d(src–vector &optional dest–vector)
CMSR:fe-v-normalize-3d(src–vector &optional dest–vector)
```

ARGUMENTS

src_vector	A one-dimensional array containing the vector to be normalized.
dest_vector	A one-dimensional array containing the result of normalizing *src_vector.*

For the 2D routine these are 2-element arrays; for the 3D routine these are 3-element arrays. x occupies element 0, y occupies element 1, and z (if present) occupies element 2.

DESCRIPTION

`CMSR_fe_v_normalize_2d` and **`CMSR_fe_v_normalize_3d`** compute a unit vector pointing in the same direction as *src_vector* and put the result in *dest_vector*. In C and Lisp these routines also return a pointer to *dest_vector*.

The source vector should not be zero length.

ERRORS

If the *src_vector* is zero length, the behavior of this routine is undefined.

CMSR_fe_v_perpendicular_2d
CMSR_fe_v_perpendicular_3d

Constructs a unit vector perpendicular to one 2D or to two 3D vectors.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_v_perpendicular_2d (src_vector, dest_vector)
double    src_vector[2], dest_vector[2];

double *
  CMSR_fe_v_perpendicular_3d (src1_vector, src2_vector, dest_vector)
double    src1_vector[3], src2_vector[3], dest_vector[3];
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_FE_V_PERPENDICULAR_2D (src_vector, dest_vector)
   DOUBLE PRECISION src_vector(2), dest_vector(2)

   SUBROUTINE CMSR_FE_V_PERPENDICULAR_3D
&                                   (src1_vector, src2_vector, dest_vector)
   DOUBLE PRECISION src1_vector(3), src2_vector(3), dest_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-perpendicular-2d(src1-vector &optional dest-vector)
CMSR:fe-v-perpendicular-3d
                        (src1-vector src2-vector &optional dest-vector)
```

ARGUMENTS

src–vector, src–vector1, src–vector2	One-dimensional arrays containing the vectors to be operated on.
dest_vector	A one-dimensional array containing the result of the routine.

For the 2D routine these are 2-element arrays; for the 3D routine these are 3-element arrays. x occupies element 0, y occupies element 1, and z (if present) occupies element 2.

DESCRIPTION

`CMSR_fe_v_perpendicular_2d` constructs a unit vector perpendicular to *src_vector*, and puts the result in *dest_vector.* In C and Lisp this routine also returns a pointer to *dest_vector.*

The source vector need not be unit length, but *src_vector* should not be zero length.

`CMSR_fe_v_perpendicular_3d` constructs a unit vector perpendicular to *src1_vector* and *src2_vector* and puts the result in *dest_vector.* In C and Lisp this routine also returns a pointer to *dest_vector.* The source vectors need not be unit length.

The cross-product of the source vectors should not be zero length.

CMSR_fe_v_print_2d
CMSR_fe_v_print_3d

Prints the vector on `stdout`.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_v_print_2d  (src_vector)
double    src_vector[2];

double *
  CMSR_fe_v_print_3d  (src_vector)
double    src_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_V_PRINT_2D  (src_vector)
DOUBLE PRECISION  src_vector(2)

SUBROUTINE CMSR_FE_V_PRINT_3D  (src_vector)
DOUBLE PRECISION  src_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-print-2d(src-vector)
CMSR:fe-v-print-3d(src-vector)
```

ARGUMENTS

src_vector	A one-dimensional array containing the vector to be printed. For the 2D routine this is a 2-element array; for the 3D routine this is a 3-element array. x occupies element 0, y occupies element 1, and z (if present) occupies element 2.

DESCRIPTION

`CMSR_fe_v_print_2d` and `CMSR_fe_v_print_3d` print the *src_vector* on `stdout`. In C and Lisp these routines also return a pointer to *src_vector*.

The elements of the vector are printed on one line, separated by spaces, and followed by a carriage return.

CMSR_fe_v_reflect_2d
CMSR_fe_v_reflect_3d

Calculates a reflectance vector for specified incident and normal vectors.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
   CMSR_fe_v_reflect_2d (incident_vector, normal_vector, dest_vector);
double    incident_vector[2], normal_vector[2], dest_vector[2];

double *
   CMSR_fe_v_reflect_3d(incident_vector, normal_vector, dest_vector);
double    incident_vector[3], normal_vector[3], dest_vector[3];
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_FE_V_REFLECT_2D
&                                  (incident_vector, normal_vector, dest_vector)
   DOUBLE PRECISION incident_vector(2), normal_vector(2), dest_vector(2)

   SUBROUTINE CMSR_FE_V_REFLECT_3D
&                                  (incident_vector, normal_vector, dest_vector)
   DOUBLE PRECISION incident_vector(3), normal_vector(3), dest_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-reflect-2d(incident-vector normal-vector
                     &optional dest-vector)
CMSR:fe-v-reflect-3d(incident-vector normal-vector
                     &optional dest-vector)
```

ARGUMENTS

incident_vector	A one-dimensional array containing the vector indicating the direction of the incident light.
normal_vector	A one-dimensional array containing the vector indicating the normal vector of the surface from which the light is to reflect.
dest_vector	A one-dimensional array containing the vector indicating the direction of the reflected light.

For the 2D routine these are 2-element arrays; for the 3D routine these are 3-element arrays. x occupies element 0, y occupies element 1, and z (if present) occupies element 2.

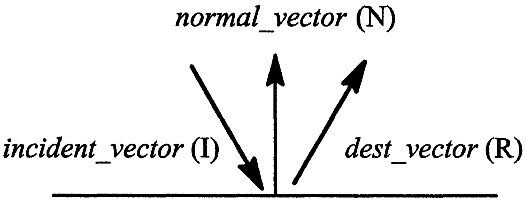
DESCRIPTION

`CMSR_fe_v_reflect_2d` and `CMSR_fe_v_reflect_3d` determine the vector resulting from reflecting *incident_vector* around *normal_vector* and put the result in *dest_vector*. In C and Lisp these routines also return a pointer to *dest_vector*. The incident and normal vectors need not be unit length, but the reflected vector will be.

Neither input vector should be zero length.

To build the destination vector, the incident and normal vectors are first normalized. The reflected vector (R), is then constructed from the unit-length incident vector (I) and unit-length normal (N) vector:

$$R = I-2*(N \text{ dot } I)*N$$



CMSR_fe_v_scale_2d
CMSR_fe_v_scale_3d

Multiplies a vector by a constant scale value.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_v_scale_2d (src_vector, scale_value, dest_vector)
double   src_vector[2], scale_value, dest_vector[2];

double *
  CMSR_fe_v_scale_3d (src_vector, scale_value, dest_vector)
double   src_vector[3], scale_value, dest_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_V_SCALE_2D(src_vector, scale_value, dest_vector)
DOUBLE PRECISION src_vector(2), scale_value, dest_vector(2)

SUBROUTINE CMSR_FE_V_SCALE_3D(src_vector, scale_value, dest_vector)
DOUBLE PRECISION src_vector(3), scale_value, dest_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-scale-2d(src–vector scale–value &optional dest–vector)
CMSR:fe-v-scale-3d(src–vector scale–value &optional dest–vector)
```

ARGUMENTS

src_vector	A one-dimensional array containing the vector to be scaled.
scale_value	The scaling factor to be applied to *src_vector.*
dest_vector	A one-dimensional array containing the result of scaling *src_vector* by the *scale_value.*

For the 2D routine the arrays contain 2 elements; for the 3D routine the arrays contain 3 elements. *x* occupies element 0, *y* occupies element 1, and *z* (if present) occupies element 2.

DESCRIPTION

`CMSR_fe_v_scale_2d` and **`CMSR_fe_v_scale_3d`** multiply each element of *src_vector* by the constant *scale_value* and put the result in *dest_vector*. In C and Lisp, these routines also return a pointer to *dest_vector*.

dest_vector may be the same as *src_vector.*

CMSR_fe_v_subtract_2d
CMSR_fe_v_subtract_3d

Subtracts each element of one vector from another.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_v_subtract_2d (src1_vector, src2_vector, dest_vector)
double    src1_vector[2], src2_vector[2], dest_vector[2];

double *
  CMSR_fe_v_subtract_3d (src1_vector, src2_vector, dest_vector)
double    src1_vector[3], src2_vector[3], dest_vector[3];
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_FE_V_SUBTRACT_2D
&                              (src1_vector, src2_vector, dest_vector)
   DOUBLE PRECISION src1_vector(2), src2_vector(2), dest_vector(2)

   SUBROUTINE CMSR_FE_V_SUBTRACT_3D
&                              (src1_vector, src2_vector, dest_vector)
   DOUBLE PRECISION src1_vector(3), src2_vector(3), dest_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-subtract-2d (src1-vector src2-vector &optional dest-vector)
CMSR:fe-v-subtract-3d (src1-vector src2-vector &optional dest-vector)
```

ARGUMENTS

src1_vector, src2_vector	One-dimensional arrays containing the vectors to be subtracted.
dest_vector	A one-dimensional array containing the result of subtracting *src1_vector* and *src2_vector*.

For the 2D routine the arrays contain 2 elements; for the 3D routine the arrays contain 3 elements. *x* occupies element 0, *y* occupies element 1, and *z* (if present) occupies element 2.

DESCRIPTION

`CMSR_fe_v_subtract_2d` and **`CMSR_fe_v_subtract_3d`** subtract each element of *src2_vector* from *src1_vector* (*src1_vector* – *src2_vector*) and put the result in *dest_vector*. In C and Lisp these routines also return a pointer to *dest_vector*.

dest_vector may be the same as *src1_vector* or *src2_vector*.

CMSR_fe_v_transform_2d CMSR_fe_v_transform_3d

Returns vector transformed by transformation matrix.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

double *
  CMSR_fe_v_transform_2d (src_vector, src_matrix, dest_vector)

double    src_vector[2];
double    src_matrix[3][3];
double    dest_vector[2];

double *
  CMSR_fe_v_transform_3d (src_vector, src_matrix, dest_vector)

double    src_vector[3];
double    src_matrix[4][4];
double    dest_vector[3];
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'

   SUBROUTINE CMSR_FE_V_TRANSFORM_2D
&                                      (src_vector, src_matrix, dest_vector)

   INTEGER src_vector(2)
   INTEGER src_matrix(3,3)
   INTEGER dest_vector(2)

   SUBROUTINE CMSR_FE_V_TRANSFORM_3D
&                                      (src_vector, src_matrix, dest_vector)

   INTEGER src_vector(3)
   INTEGER src_matrix(4,4)
   INTEGER dest_vector(3)
```

Lisp Syntax

`CMSR:fe-v-transform-vector-2d`
(*src–vector src–matrix* `&optional` *dest–vector*)

`CMSR:fe-v-transform-vector-3d`
(*src–vector src–matrix* `&optional` *dest–vector*)

ARGUMENTS

src_vector	For **`CMSR_fe_v_transform_2d`** a 1 x 2 array, and for **`CMSR_fe_v_transform_3d`** a 1 x 3 array containing the vector to be transformed.
src_matrix	For **`CMSR_fe_v_transform_2d`** a 3 x 3 array, and for **`CMSR_fe_v_transform_3d`** a 4 x 4 array containing the homogeneous transformation matrix to be applied to *src_vector.* The matrix elements are stored in row- major order.
dest_vector	For **`CMSR_fe_v_transform_2d`** a 1 x 2 array, and for **`CMSR_fe_v_transform_3d`** a 1 x 3 array containing the transformed vector.

DESCRIPTION

`CMSR_fe_v_transform_2d` and **`CMSR_fe_v_transform_3d`** calculate the result of transforming the vector *src_vector* by the homogeneous transformation matrix *src_matrix* and store the result in *dest_vector. dest_vector* may be the same as *src_vector.*

In C and Lisp, these routines also return a pointer to *dest_vector.* In Lisp *dest_vector* is optional; space for the vector is allocated if it is not specified.

ERRORS

If the homogeneous coordinate of the transformed vector goes to zero, the result of this routine is undefined.

SEE ALSO

`CMSR_v_transform_2d`

`CMSR_v_transform_3d`

`CMSR_v_transform_const_2d`

`CMSR_v_transform_const_3d`

CMSR_fe_v_transmit_3d

Creates a transmittance vector for light refracted through two materials.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_v_transmit_3d
      (incident_vector, normal_vector, index1, index2, transmitted_vector)
double   incident_vector[3], normal_vector[3];
double   index1, index2;
double   transmitted_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'

   SUBROUTINE CMSR_FE_V_TRANSMIT_3d
&           (incident_vector, normal_vector, index1, index2, transmitted_vector)
DOUBLE PRECISION incident_vector(3), normal_vector(3)
DOUBLE PRECISION index1, index2
DOUBLE PRECISION transmitted_vector(3)
```

Lisp Syntax

```
CMSR:fe-v-transmit-3d (incident-vector normal-vector index1 index2
                          &optional transmitted-vector)
```

ARGUMENTS

incident_vector A one-dimensional array of 3 elements containing the vector indicating the direction of the incident light. *x* occupies element 0, *y* occupies element 1, and *z* (if present) occupies element 2.

For the 2D routine these are 2-element arrays; for the 3D routine these are 3-element arrays.

normal_vector	A one-dimensional array of 3 elements containing the vector indicating the surface normal of the material through which the light is to pass. *x* occupies element 0, *y* occupies element 1, and *z* (if present) occupies element 2.
index1, *index2*	The index of refraction of the medium containing *incident_vector* and *transmitted_vector*, respectively.
transmitted_vector	A one-dimensional array of 3 elements containing the vector indicating the direction of the transmitted light. *x* occupies element 0, *y* occupies element 1, and *z* (if present) occupies element 2.

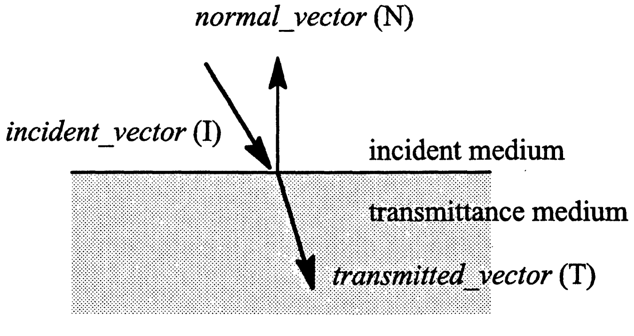
DESCRIPTION

`CMSR_fe_v_transmit_3d`, given an incident vector, surface normal, and indices of refraction for the two materials, constructs a transmitted light vector and puts the vector in *transmitted_vector*. In C and Lisp this routine also returns a pointer to *transmitted_vector*.

The unit length *transmitted_vector* T is given by:

$$T = \frac{n_1}{n_2} I + N\left(\frac{n_1}{n_2}(N \cdot -I) - \sqrt{1 + \left(\frac{n_1}{n_2}\right)^2 \left((N \cdot -I)^2 - 1\right)}\right)$$

N is the unit vector in the direction of *normal_vector*. I is the unit vector in the direction of *incident_vector*. n1 is *index1*, the index of refraction of the medium containing the incident vector. n2 is *index2*, the index of refraction of the medium which contains the transmitted vector.



If the expression under the square root becomes negative, the result is total internal reflection. If total internal reflection occurs, `CMSR_fe_v_transmit_3d` returns a NULL pointer.

Neither the incident vector nor the normal vector should be length 0. The second index of refraction should not be 0.

3.3 Front-end Matrix Routines

This section documents the *Render routines that operate on matrices in front-end arrays. Matrices in *Render are assumed to be square, homogeneous matrices. *Render supports matrices of dimension 2 or 3, for transforming two-dimensional or three-dimensional vectors. On the front end, a matrix is an appropriately sized array of double-precision floating-point values.

The routines documented here are:

CMSR_fe_identity_matrix_2d CMSR_fe_identity_matrix_3d

Creates 2D (3D) homogeneous transformation identity matrix in front-end array.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_identity_matrix_2d  (dest_matrix)
double    dest_matrix[3][3];

double *
  CMSR_fe_identity_matrix_3d  (dest_matrix)
double dest_matrix[4][4];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_IDENTITY_MATRIX_2D  (dest_matrix)
DOUBLE PRECISION dest_matrix(3,3)

SUBROUTINE CMSR_FE_IDENTITY_MATRIX_3D  (dest_matrix)
DOUBLE PRECISION dest_matrix(4,4)
```

Lisp Syntax

```
CMSR:fe-identity-matrix-2d (&optional dest-matrix)
CMSR:fe-identity-matrix-3d (&optional dest-matrix)
```

ARGUMENTS

dest_matrix	A 3 x 3 array for **`CMSR_fe_identity_matrix_2d`** or a 4 x 4 array for **`CMSR_fe_identity_matrix_3d`** into which a homogeneous identity matrix is stored. The matrix elements are stored in row-major order.

DESCRIPTION

`CMSR_fe_identity_matrix_2d` and **`CMSR_fe_identity_matrix_3d`** set the front-end matrix *dest_matrix* to an identity matrix for 2D or 3D homogeneous transformations.

In C and Lisp, the routines also return a pointer to *dest_matrix*. In Lisp, *dest–matrix* is optional; space is allocated if the matrix is not specified.

The identity matrix is the identity element for matrix multiplication. It is an array in which all elements are set to 0 except for the diagonal elements, which are set to 1.

SEE ALSO

`CMSR_identity_matrix_2d`

`CMSR_identity_matrix_3d`

CMSR_fe_m_copy_2d
CMSR_fe_m_copy_3d

Copies 2D (3D) transformation matrix between front-end arrays.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
   CMSR_fe_m_copy_2d (from_matrix, to_matrix)
double    from_matrix[3][3];
double    to_matrix[3][3];

double *
   CMSR_fe_m_copy_3d (from_matrix, to_matrix)
double    from_matrix[4][4];
double    to_matrix[4][4];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_M_COPY_2D (from_matrix, to_matrix)
DOUBLE PRECISION from_matrix(3,3)
DOUBLE PRECISION to_matrix(3,3)

SUBROUTINE CMSR_FE_M_COPY_3D (from_matrix, to_matrix)
DOUBLE PRECISION from_matrix(4,4)
DOUBLE PRECISION to_matrix(4,4)
```

Lisp Syntax

```
CMSR:fe-m-copy-2d (from-matrix &optional to-matrix)
CMSR:fe-m-copy-3d (from-matrix &optional to-matrix)
```

ARGUMENTS

from_matrix

For `CMSR_fe_m_copy_2d` a 3 x 3 array, or for `CMSR_fe_m_copy_3d` a 4 x 4 array containing the homogeneous transformation matrix to be copied. The matrix elements are stored in row-major order.

to_matrix

For `CMSR_fe_m_copy_2d` a 3 x 3 array, or for `CMSR_fe_m_copy_3d` a 4 x 4 array into which *from_matrix* is to be copied. The matrix elements are stored in row-major order.

DESCRIPTION

`CMSR_fe_m_copy_2d` and `CMSR_fe_m_copy_3d` copy the front-end matrix *from_matrix* to the front-end matrix *to_matrix.*

In C and Lisp, `CMSR_fe_m_copy_2d` and `CMSR_fe_m_copy_3d` also return a pointer to *to_matrix*. In Lisp, *from–matrix* is optional; space is allocated for the matrix if it is not specified.

CMSR_fe_m_determinant_2d
CMSR_fe_m_determinant_3d

Returns the determinant of a matrix.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double
  CMSR_fe_m_determinant_2d (matrix)
double    matrix[3][3];

double
  CMSR_fe_m_determinant_3d (matrix)
double    matrix[4][4];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
DOUBLE PRECISION FUNCTION CMSR_FE_M_DETERMINANT_2D (matrix)
DOUBLE PRECISION matrix(3,3)

DOUBLE PRECISION FUNCTION CMSR_FE_M_DETERMINANT_3D (matrix)
DOUBLE PRECISION matrix(4,4)
```

Lisp Syntax

```
CMSR:fe-m-determinant-2d (matrix)
CMSR:fe-m-determinant-3d (matrix)
```

ARGUMENTS

matrix	For **CMSR_fe_m_determinant_2d** a 3 x 3 array, or for **CMSR_fe_m_determinant_3d** a 4 x 4 array, containing a homogeneous transformation matrix. The matrix elements are stored in row-major order.

DESCRIPTION

`CMSR_fe_m_determinant_2d` and **`CMSR_fe_m_determinant_3d`** return the determinant of the matrix specified in *matrix*.

CMSR_fe_m_invert_2d
CMSR_fe_m_invert_3d

Creates the inverse of a 2D (3D) matrix using arrays on the front-end computer.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_m_invert_2d (from_matrix, to_matrix))
double    from_matrix[3][3], to_matrix[3][3];

double *
  CMSR_fe_m_invert_3d (from_matrix, to_matrix)
double    from_matrix[4][4], to_matrix[4][4];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_M_INVERT_2D (from_matrix, to_matrix)
DOUBLE PRECISION from_matrix(3, 3), to_matrix(3, 3)

SUBROUTINE CMSR_FE_M_INVERT_3D (from_matrix, to_matrix)
DOUBLE PRECISION from_matrix(4, 4), to_matrix(4, 4)
```

Lisp Syntax

```
CMSR:fe-m-invert-2d(from–matrix &optional to–matrix)
CMSR:fe-m-invert-3d(from–matrix &optional to–matrix)
```

ARGUMENTS

from_matrix	An array containing the transformation matrix to be inverted. For the 2D routine this is a 3 x 3 array of homogeneous coordinates; for the 3D routine this is a 4 x 4 array. The matrix elements are stored in row-major order.
to_matrix	An array into which the inverted *from_matrix* is to be copied. For the 2D routine this is a 3 x 3 array of homogeneous coordinates; for the 3D routine this is a 4 x 4 array. The matrix elements are stored in row-major order.

DESCRIPTION

`CMSR_fe_m_invert_2d` and **`CMSR_fe_m_invert_3d`** place the inverse of *from_matrix* in *to_matrix*. The destination matrix, *to_matrix*, may be identical to the source matrix *from_matrix*. If the matrix is singular (that is, if its determinant is zero), a fatal error occurs.

In C and Lisp, these routines also return a pointer to *to_matrix*. In Lisp, *to–matrix* is optional; space is allocated if the matrix is not specified.

ERRORS

If the matrix is singular (that is, if its determinant is zero), a fatal error occurs.

SEE ALSO

`CMSR_m_invert_2d`

`CMSR_m_invert_3d`

CMSR_fe_m_multiply_2d
CMSR_fe_m_multiply_3d

Multiplies two 2D (3D) transformation matrices.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

double *
   CMSR_fe_m_multiply_2d (src1_matrix, src2_matrix, dest_matrix)

double   src1_matrix[3][3];
double   src2_matrix[3][3];
double   dest_matrix[3][3];

double *
   CMSR_fe_m_multiply_3d (src1_matrix, src2_matrix, dest_matrix)

double   src1_matrix[4][4];
double   src2_matrix[4][4];
double   dest_matrix[4][4];
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-math-fort.h'

    SUBROUTINE CMSR_FE_M_MULTIPLY_2D
&                                (src_matrix1  src_matrix2  dest_matrix)
    DOUBLE PRECISION src1_matrix (3, 3)
    DOUBLE PRECISION src2_matrix (3, 3)
    DOUBLE PRECISION dest_matrix (3, 3)

    SUBROUTINE CMSR_FE_M_MULTIPLY_3D
&                                   (src_matrix1 src_matrix2 dest_matrix)
    DOUBLE PRECISION src1_matrix (4, 4)
    DOUBLE PRECISION src2_matrix (4, 4)
    DOUBLE PRECISION dest_matrix (4, 4)
```

Lisp Syntax

```
CMSR:fe-m-multiply-2d (src1-matrix src2-matrix &optional dest-matrix)
CMSR:fe-m-multiply-3d (src1-matrix src2-matrix &optional dest-matrix)
```

ARGUMENTS

src1_matrix, src2_matrix	For `CMSR_fe_m_multiply_2d` 3 x 3 arrays, and for `CMSR_fe_m_multiply_3d` 4 x 4 arrays, containing the homogeneous transformation matrices to be multiplied. The matrix elements are stored in row-major order.
dest_matrix	For `CMSR_fe_m_multiply_2d` a 3 x 3 array, and for `CMSR_fe_m_multiply_3d` a 4 x 4 array, in which the product of *src1_matrix* and *src2_matrix* is returned. The matrix elements are stored in row-major order.

DESCRIPTION

`CMSR_fe_m_multiply_2d` and `CMSR_fe_m_multiply_3d` calculate the product of two homogeneous transformation matrices, (*src–matrix1* src–matrix2*) and store the result in *dest_matrix*. The destination matrix may be the same as either source matrix.

In C and Lisp, these routines also return a pointer to *dest_matrix*. In Lisp *dest–matrix* is optional; space is allocated for the matrix if it is not specified.

SEE ALSO

`CMSR_m_multiply_2d`

`CMSR_m_multiply-3d`

`CMSR_m_multiply_const_2d`

`CMSR_m_multiply_const_3d`

CMSR_fe_m_print_2d
CMSR_fe_m_print_3d

Prints the contents of a matrix on `stdout`.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

double *
  CMSR_fe_m_print_2d (src_matrix)
double    src_matrix[3][3];

double *
  CMSR_fe_m_print_3d (src_matrix)
double    src_matrix[4][4];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'

SUBROUTINE CMSR_FE_M_PRINT_2D (src_matrix)

DOUBLE PRECISION src_matrix(3, 3)

SUBROUTINE CMSR_FE_M_PRINT_3D (src_matrix)

DOUBLE PRECISION src_matrix(4, 4)
```

Lisp Syntax

```
CMSR:fe-m-print-2d(src-matrix)

CMSR:fe-m-print-3d(src-matrix)
```

ARGUMENTS

src_matrix	The front-end array to be printed. For **`CMSR_fe_m_print_2d`** *src–matrix* is a 3 x 3 array. For **`CMSR_fe_m_print_3d`** *src–matrix* is a 4 x 4 array.

DESCRIPTION

`CMSR_fe_m_print_2d` and **`CMSR_fe_m_print_3d`** print the double-precision floating-point contents of the *src_matrix* array on **`stdout`**. The matrix elements are printed one row per line, separated by spaces, and followed by a carriage return.

In C and Lisp these routines also return a pointer to *src_matrix*.

SEE ALSO

`CMSR_m_print_2d`

`CMSR_m_print_3d`

CMSR_fe_oblique_proj_matrix

Creates an oblique projection matrix for a specified angle and foreshortening.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_oblique_proj_matrix  (foreshortenting, angle, dest_matrix)
double   foreshortening;
double   angle;
double   dest_matrix[4][4];
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_FE_OBLIQUE_PROJ_MATRIX
&                                           (foreshortenting, angle, dest_matrix)
   DOUBLE PRECISION foreshortening;
   DOUBLE PRECISION angle;
   DOUBLE PRECISION dest_matrix(4, 4);
```

Lisp Syntax

```
CMSR:fe-oblique-proj-matrix
                      (foreshortenting angle  &optional dest-matrix)
```

ARGUMENTS

foreshortenting	The ratio of the projected length of a line in z, to its true length.The length of a projected z-axis unit vector.
angle	The angle between the projected z-axis and the true horizontal of the object.

dest_matrix	A 4 x 4 front-end array containing a homogeneous transformation matrix that expresses the oblique projection defined by *foreshortenting* and *angle.*

DESCRIPTION

`CMSR_fe_oblique_proj_matrix` builds an oblique projection matrix and stores it in *dest_matrix.* In C and Lisp, this routine also returns a pointer to *dest_matrix.*

An oblique projection is one in which the parallel projectors intersect with the projection plane at an oblique angle. In this routine, *angle* is the angle that the projected z-axis makes with the horizontal. The projection is onto the plane $z = 0$. *Foreshortening* is the length of a projected z-axis unit vector. When *foreshortening* is 0, an orthographic projection results.

CMSR_fe_ortho_proj_matrix

Creates an orthographic projection matrix perpendicular to a specified plane.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_ortho_proj_matrix (axis, dest_matrix)
int       axis:
double    dest_matrix[4][4];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_ORTHO_PROJ_MATRIX (axis, dest_matrix)
INTEGER axis
DOUBLE PRECISION dest_matrix(4, 4)
```

Lisp Syntax

```
CMSR:fe-ortho-proj-matrix (axis &optional dest–matrix)
```

ARGUMENTS

axis	The axis that will be the plane of projection. *axis* may be specified either symbolically or with an integer: ▪ *x* axis = **CMSR_x** or 0 ▪ *y* axis = **CMSR_y** or 1 ▪ *z* axis = **CMSR_z** or 2
dest_matrix	A 4 x 4 front-end array containing a homogeneous transformation matrix that expresses an orthographic projection onto *axis*.

DESCRIPTION

`CMSR_fe_ortho_proj_matrix` builds an orthographic projection matrix and stores it in *dest_matrix*. In C and Lisp this routine also returns a pointer to *dest_matrix*.

An orthographic projection is a perpendicular projection onto one of the coordinate planes. This projection is commonly used for engineering drawings.

In this routine, the *axis* parameter determines the plane of projection:

If *axis* is `CMSR_x` or 0, then the plane of projection is $x = 0$.

If *axis* is `CMSR_y` or 1, then the plane of projection is $y = 0$.

If *axis* is `CMSR_z` or 2, then the plane of projection is $z = 0$.

If *axis* is not one of the above, a fatal error results.

ERRORS

If axis is not `CMSR_x` (0) , `CMSR_y` (1), or `CMSR_z` (2), a fatal error results.

CMSR_fe_perspective_matrix

Creates a perspective transformation matrix.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

double *
  CMSR_fe_perspective_matrix  (axis, center_of_proj, dest_matrix)

int       axis;
double    center_of_proj;
double    dest_matrix[4][4];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'

SUBROUTINE CMSR_FE_PERSPECTIVE_MATRIX
&                                     (axis, center_of_proj, dest_matrix)

INTEGER  axis
DOUBLE  PRECISION  center_of_proj
DOUBLE  PRECISION  dest_matrix(4, 4)
```

Lisp Syntax

```
CMSR:fe-perspective-matrix
                              (axis center-of-proj  &optional dest-matrix)
```

ARGUMENTS

axis	The axis that will be the plane of projection. *axis* may be specified either symbolically or with an integer: ▪ *x* axis = `CMSR_x` or 0 ▪ *y* axis = `CMSR_y` or 1 ▪ *z* axis = `CMSR_z` or 2 If *axis* is not one of the above, a fatal error results.

center_of_proj	Specifies the point on *axis* on which the projection is to be centered. If *center_of_proj* is zero, a fatal error results.
dest_matrix	A 4 x 4 front-end array containing the homogeneous transformation matrix created by the routine.

DESCRIPTION

`CMSR_fe_perspective_matrix` builds a perspective transformation matrix and stores it in *dest_matrix*. In C and Lisp, this routine also returns a pointer to *dest_matrix*. *axis* specifies the axis of projection. *center_position* is the center of projection along *axis*.

When the completed perspective transformation is applied to object coordinates, object size is reduced with increasing distance from the center of the projection.

Note that **`CMSR_fe_perspective_matrix`** creates a matrix that maps from one 3D space into another 3D space. To transform an object for drawing into a two-dimensional image buffer, you must concatenate a projection matrix to this perspective matrix.

ERRORS

A fatal error results if *axis* is not **`CMSR_x`** (0) , **`CMSR_y`** (1), or **`CMSR_z`** (2) or if *center_of_proj* is zero.

CMSR_fe_perspective_proj_matrix

Creates a transformation matrix composed of a perspective transformation and an orthogonal projection.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_perspective_proj_matrix(axis, center_of_proj, dest_matrix)
int       axis;
double    center_of_proj;
double    dest_matrix[4][4];
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-math-fort.h'
    SUBROUTINE CMSR_FE_PERSPECTIVE_PROJ_MATRIX
&                                    (axis, center_of_proj, dest_matrix)
    DOUBLE PRECISION axis
    DOUBLE PRECISION center_of_proj
    DOUBLE PRECISION dest_matrix(4, 4)
```

Lisp Syntax

```
CMSR:fe-perspective-proj-matrix
                           (axis center-of-proj &optional dest-matrix)
```

ARGUMENTS

axis	The axis that will be the plane of projection. *axis* may be specified either symbolically or with an integer: ▪ *x* axis = `CMSR_x` or 0 ▪ *y* axis = `CMSR_y` or 1 ▪ *z* axis = `CMSR_z` or 2 If *axis* is not one of the above, a fatal error results.

center_of_proj	Specifies the point on *axis* on which the projection is to be centered. If *center_of_proj* is zero, a fatal error results.
dest_matrix	A 4 x 4 front-end array containing the homogeneous transformation matrix created by the routine.

DESCRIPTION

`CMSR_fe_perspective_proj_matrix` builds a transformation matrix that is a composition of a perspective transformation centered on *center_of_proj* and an orthogonal projection along the axis specified by *axis.* This transformation matrix is placed in *dest_matrix.* In C and Lisp, this routine also returns a pointer to *dest_matrix.*

The *axis* parameter determines the plane of projection.

If *axis* is **`CMSR_x`** or 0, then the plane of projection is $x = 0$.

If *axis* is **`CMSR_y`** or 1, then the plane of projection is $y = 0$.

If *axis* is **`CMSR_z`** or 2, then the plane of projection is $z = 0$.

ERRORS

A fatal error results if *axis* is not **`CMSR_x`** (0) , **`CMSR_y`** (1), or **`CMSR_z`** (2) or if *center_of_proj* is zero.

CMSR_fe_rotation_matrix_2d

Creates a 2D transformation matrix with a specified rotation in a front-end array.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_rotation_matrix_2d (theta, dest_matrix)
double    theta;
double    dest_matrix[3][3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_ROTATION_MATRIX_2D (theta, dest_matrix)
DOUBLE PRECISION theta
DOUBLE PRECISION dest_matrix(3,3)
```

Lisp Syntax

```
CMSR:fe-rotation-matrix-2d (theta &optional dest-matrix)
```

ARGUMENTS

theta	A double-precision value specifying the the angle of rotation, in radians, to be incorporated into the transformation matrix in *dest_matrix*.
dest_matrix	A 3 x 3 array in which the resulting 2D transformation matrix is returned. The matrix elements are stored in row-major order.

DESCRIPTION

`CMSR_fe_rotation_matrix_2d` calculates a two-dimensional homogeneous transformation matrix with a rotation of *theta* radians and places the result in *dest_matrix*.

In C and Lisp, **`CMSR_fe_rotation_matrix_2d`** also returns a pointer to *dest_matrix*. In Lisp *dest_matrix* is optional; space is allocated if the matrix is not specified.

The rotation is about the origin of the image. Positive rotations are counter-clockwise.

SEE ALSO

`CMSR_fe_x_rotation_matrix_3d`

`CMSR_fe_y_rotation_matrix_3d`

`CMSR_fe_z_rotation_matrix_3d`

`CMSR_rotation_matrix_2d`

`CMSR_rotation_const_matrix_2d`

`CMSR_x_rotation_const_matrix_3d`

`CMSR_x_rotation_matrix_3d`

`CMSR_y_rotation_const_matrix_3d`

`CMSR_y_rotation_matrix_3d`

`CMSR_z_rotation_const_matrix_3d`

`CMSR_z_rotation_matrix_3d`

CMSR_fe_scale_matrix_2d
CMSR_fe_scale_matrix_3d

Creates a 2D (3D) transformation matrix with specified scaling values in a front-end array.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
   CMSR_fe_scale_matrix_2d (sx, sy, dest_matrix)
double    sx, sy;
double    dest_matrix[3][3];

double *
   CMSR_fe_scale_matrix_3d (sx, sy, sz, dest_matrix)
double    sx, sy, sz;
double    dest_matrix[4][4];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_SCALE_MATRIX_2D (sx, sy, dest_matrix)
DOUBLE PRECISION sx, sy
DOUBLE PRECISION dest_matrix(3,3)

SUBROUTINE CMSR_FE_SCALE_MATRIX_3D (sx, sy, sz, dest_matrix)
DOUBLE PRECISION sx, sy. sz
DOUBLE PRECISION dest_matrix(4,4)
```

Lisp Syntax

```
CMSR:fe-scale-matrix-2d (sx sy &optional dest-matrix)
CMSR:fe-scale-matrix-3d (sx sy sz &optional dest-matrix)
```

ARGUMENTS

sx	A double-precision value specifying the *x* coordinate scaling value to be incorporated into the transformation matrix in *dest_matrix.*
sy	A double-precision value specifying the *y* coordinate scaling value to be incorporated into the transformation matrix in *dest_matrix.*
sy	A double-precision value specifying the *z* coordinate scaling value to be incorporated into the transformation matrix in *dest_matrix.*
dest_matrix	A 3 x 3 array for **`CMSR_fe_scale_matrix_2d`**, or a 4 x 4 array for **`CMSR_fe_scale_matrix_3d`**, in which the resulting transformation matrix is returned. The matrix elements are stored in row-major order.

DESCRIPTION

`CMSR_fe_scale_matrix_2d` calculates a two-dimensional homogeneous transformation matrix with the scaling terms *sx* and *sy*. **`CMSR_fe_scale_matrix_3d`** calculates a three-dimensional transformation matrix with the scaling terms *sx*, *sy*, and *sz*. The scaling matrix is stored in *dest_matrix.*

In C and Lisp these routines also return a pointer to *dest_matrix*. In Lisp *dest–matrix* is optional; space is allocated if the matrix is not specified.

SEE ALSO

`CMSR_scale_const_matrix_2d`

`CMSR_scale_const_matrix_3d`

`CMSR_scale_matrix_2d`

`CMSR_scale_matrix_3d`

CMSR_fe_translation_matrix_2d
CMSR_fe_translation_matrix_3d

Creates a 2D (3D) transformation matrix in a front-end array with specified translation values.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_translation_matrix_2d (tx, ty, dest_matrix)
double    tx, ty;
double    dest_matrix[3][3];

double *
  CMSR_fe_translation_matrix_3d (tx, ty, tz, dest_matrix)
double    tx, ty, tz;
double    dest_matrix[4][4];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_TRANSLATION_MATRIX_2D (tx, ty, dest_matrix)
DOUBLE PRECISION tx, ty
DOUBLE PRECISION dest_matrix(3,3)

SUBROUTINE CMSR_FE_TRANSLATION_MATRIX_3D (tx, ty, tz, dest_matrix)
DOUBLE PRECISION tx, ty, tz
DOUBLE PRECISION dest_matrix(4,4)
```

Lisp Syntax

```
CMSR:fe-translation-matrix-2d (tx ty &optional dest-matrix)
CMSR:fe-translation-matrix-3d (tx ty tz &optional dest-matrix)
```

ARGUMENTS

tx	A double-precision value specifying the *x* translation value to be incorporated into the transformation matrix in *dest_matrix*.
ty	A double-precision value specifying the *y* translation value to be incorporated into the transformation matrix in *dest_matrix*.
tz	A double-precision value specifying the *z* translation value to be incorporated into the transformation matrix in *dest_matrix*.
dest_matrix	A 3 x 3 array for **`CMSR_fe_translation_matrix_2d`** or a 4 x 4 array for **`CMSR_fe_translation_matrix_3d`**, in which the resulting transformation matrix is returned. The matrix elements are stored in row-major order.

DESCRIPTION

`CMSR_fe_translation_matrix_2d` and **`CMSR_fe_translation_matrix_3d`** calculate a 2D or 3D transformation matrix, respectively, which translates homogeneous coordinate values by *tx, ty,* and, in the 3D case, *tz*. The matrix is stored in the front-end matrix *dest_matrix*.

In C and Lisp, these routines also return a pointer to *dest_matrix*. In Lisp *dest–matrix* is optional; space is allocated for the matrix if it is not specified.

SEE ALSO

`CMSR_translation_const_matrix_2d`

`CMSR_translation_const_matrix_3d`

`CMSR_translation_matrix_2d`

`CMSR_translation-matrix-3d`

CMSR_fe_view_matrix

Creates a viewing transformation matrix.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

double *
  CMSR_fe_view_matrix (eye_vector, look_at_vector, roll, view_matrix)

double    eye_vector[3];
double    look_at_vector[3];
double    roll;
double    view_matrix[4][4];
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-math-fort.h'

    INTEGER FUNCTION CMSR_FE_VIEW_MATRIX
&                               (eye_vector, look_at_vector, roll, view_matrix)

    DOUBLE PRECISION eye_vector(3)
    DOUBLE PRECISION look_at_vector(3)
    DOUBLE PRECISION roll
    DOUBLE PRECISION view_matrix(4, 4)
```

Lisp Syntax

```
CMSR:fe-view-matrix
            (eye-vector look-at-vector &optional (roll 0) view-matrix)
```

ARGUMENTS

eye_vector	A one-dimensional, 3-element, position vector: x occupies element 0, y occupies element 1, and z occupies element 2. The *view_matrix* created transforms this point to the negative z axis.

look_at_vector	A one-dimensional, 3-element, position vector: *x* occupies element 0, *y* occupies element 1, and *z* occupies element 2. The *view_matrix* created transforms this point to the origin of the viewing space.
roll	The amount of rotation about the *z* axis to be included in the *view_matrix* transformation.
view_matrix	An array containing the completed view transformation matrix. For the 2D routine this is a 3 x 3 array of homogeneous coordinates; for the 3D routine this is a 4 x 4 array. The matrix elements are stored in row-major order.

DESCRIPTION

`CMSR_fe_view_matrix` builds a viewing transformation matrix, puts it in *view_matrix*, and returns a pointer to *view_matrix*. This matrix transforms the coordinate space defined by *eye_vector, look_at_vector*, and *roll* so that the *look_at_vector* position is at the origin and the *eye_vector* position is on the negative *z* axis.

The *roll* argument allows you to specify how much this coordinate space should be rotated around the *z* axis. Each of the vectors must be of dimension 3 and the *view_matrix* must be 4 x 4.

ERRORS

If *eye_vector* is identical to *look_at_vector*, a fatal error results because the view can not be determined.

SEE ALSO

`CMSR_fe_view_proj_matrix`

CMSR_fe_view_proj_matrix

Creates a viewing transformation matrix.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

double *
   CMSR_fe_view_proj_matrix
                (center_of_proj, eye_vector, look_at_vector, roll, dest_matrix)

double     center_of_proj;
double     eye_vector[3];
double     look_at_vector[3];
double     roll;
double     dest_matrix[4][4];
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-math-fort.h'

    INTEGER FUNCTION CMSR_FE_VIEW_PROJ_MATRIX
&                  (center_of_proj, eye_vector, look_at_vector, roll, dest_matrix)

    DOUBLE PRECISION  center_of_proj
    DOUBLE PRECISION  eye_vector(3)
    DOUBLE PRECISION  look_at_vector(3)
    DOUBLE PRECISION  roll
    DOUBLE PRECISION  dest_matrix(4, 4)
```

Lisp Syntax

```
CMSR:fe-view-proj-matrix
             (eye-vector look-at-vector &optional (roll 0) dest-matrix)
```

ARGUMENTS

center_of_proj	Specifies the point on the *z* axis on which the projection is to be centered. If *center_of_proj* is zero, a fatal error results.
eye_vector	A one-dimensional, 3-element, position vector: *x* occupies element 0, *y* occupies element 1, and *z* occupies element 2. The *view_matrix* created transforms this point to the negative *z* axis.
look_at_vector	A one-dimensional, 3-element, position vector: *x* occupies element 0, *y* occupies element 1, and *z* occupies element 2. The *view_matrix* created transforms this point to the origin of the viewing space.
roll	The amount of rotation about the *z* axis to be included in the *view_matrix* transformation.
dest_matrix	A 4 x 4 front-end array containing the homogeneous transformation matrix created by the routine.

DESCRIPTION

`CMSR_fe_view_proj_matrix` returns a pointer to a matrix that is a composition of a viewing transformation defined by *eye_vector*, *look_at_vector*, and *roll,* with a perspective projection along the *z* axis centered on *center_of_proj.*

This matrix transforms the coordinate space defined by *eye_vector, look_at_vector*, and *roll* so that the *look_at_vector* position is at the origin and the *eye_vector* position is on the negative *z* axis and then projects this transformation onto the *z* axis at the plane located at *center_of_proj.*

ERRORS

A fatal error results if *eye_vector* is identical to *look_at_vector* or if *center_of_proj* is zero.

CMSR_fe_x_rotation_matrix_3d
CMSR_fe_y_rotation_matrix_3d
CMSR_fe_z_rotation_matrix_3d

Creates, in a front-end array, a 3D transformation matrix with a specified rotation about the *x* (*y*, *z*) axis.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_x_rotation_matrix_3d (theta, dest_matrix)
double *
  CMSR_y_rotation_matrix_3d (theta, dest_matrix)
double *
  CMSR_z_rotation_matrix_3d (theta, dest_matrix)
double    theta;
double    dest_matrix[4][4];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_X_ROTATION_MATRIX_3D (theta, dest_matrix)
SUBROUTINE CMSR_Y_ROTATION_MATRIX_3D (theta, dest_matrix)
SUBROUTINE CMSR_Z_ROTATION_MATRIX_3D (theta, dest_matrix)
DOUBLE PRECISION theta
DOUBLE PRECISION dest_matrix(4,4)
```

Lisp Syntax

```
CMSR:x-rotation-matrix-3d (theta &optional dest-matrix)
CMSR:y-rotation-matrix-3d (theta &optional dest-matrix)
CMSR:z-rotation-matrix-3d (theta &optional dest-matrix)
```

ARGUMENTS

theta	A double-precision value specifying the the angle of rotation about the axis, in radians, to be incorporated into the transformation matrix in *dest_matrix*.
dest_matrix	A 4 x 4 array in which the resulting 3D homogeneous transformation matrix is returned. The matrix elements are stored in row-major order.

DESCRIPTION

`CMSR_fe_x_rotation_matrix_3d`, **`CMSR_fe_y_rotation_matrix_3d`**, and **`CMSR_fe_z_rotation_matrix_3d`** calculate a three-dimensional transformation matrix with a rotation of *theta* radians around the *x*, *y*, or *z* axis and store it in *dest_ matrix*. Positive rotations are counter-clockwise as you look down the axis from the origin.

In C and Lisp, these routines both return a pointer to the resulting matrix and also place the result in *dest_matrix*. In Lisp *dest–matrix* is optional; space is allocated if the matrix is not specified.

SEE ALSO

`CMSR_fe_rotation_matrix_2d`

`CMSR_rotation_matrix_2d`

`CMSR_rotation_const_matrix_2d`

`CMSR_x_rotation_const_matrix_3d`

`CMSR_x_rotation_matrix_3d`

`CMSR_y_rotation_const_matrix_3d`

`CMSR_y_rotation_matrix_3d`

`CMSR_z_rotation_const_matrix_3d`

`CMSR_z_rotation_matrix_`

3.4 Front-End Color Conversion

This section documents the new *Render routines that convert color vectors between color spaces.

CMSR_fe_rgb_to_cmy
CMSR_fe_cmy_to_rgb

Converts color vector from RGB to CMY (CMY to RGB) color models.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_rgb_to_cmy (rgb_vector, cmy_vector);
double   rgb_vector[3], cmy_vector[3];

double *
  CMSR_fe_cmy_to_rgb (cmy_vector, rgb_vector);
double   cmy_vector[3], rgb_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'

INTEGER FUNCTION CMSR_FE_RGB_TO_CMY (rgb_vector, cmy_vector)
DOUBLE PRECISION rgb_vector(3), cmy_vector(3)

INTEGER FUNCTION CMSR_FE_CMY_TO_RGB (cmy_vector, rgb_vector)
DOUBLE PRECISION cmy_vector(3), rgb_vector(3)
```

Lisp Syntax

```
CMSR:fe-rgb-to-cmy (rgb-vector &optional cmy-vector)
CMSR:fe-cmy-to-rgb (cmy-vector &optional rgb-vector)
```

ARGUMENTS

rgb_vector	A one-dimensional array of 3 elements containing an RGB color triplet. The red intensity is in the first element, the green intensity is in the second element, and the blue intensity is in the third element. Each of the RGB color components should be in the range of [0,1].
cmy_vector	A one-dimensional array of 3 elements containing a CMY color triplet. The cyan intensity is in the first element, the magenta intensity is in the second element, and the yellow intensity is in the third element. Each of the CMY color components should be in the range of [0,1].

DESCRIPTION

`CMSR_fe_rgb_to_cmy` converts the RGB triplet in *rgb_vector* to CMY triplet, places the result in *cmy_vector*, and returns a pointer to *cmy_vector*. The relationship is

```
(c,m,y) = (1,1,1) - (r,g,b)
```

`CMSR_fe_cmy_to_rgb` converts the CMY triplet in *cmy_vector* to RGB, places the result in *rgb_vector* and returns a pointer to *rgb_vector*. The relationship is

```
(r,g,b) = (1,1,1) - (c,m,y)
```

CMSR_fe_rgb_to_yiq
CMSR_fe_yiq_to_rgb

Converts color vector from RGB to YIQ (YIQ to RGB) color models.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
   CMSR_fe_rgb_to_yiq (rgb_vector, yiq_vector)
double    rgb_vector[3], yiq_vector[3];

double *
   CMSR_fe_yiq_to_rgb (yiq_vector, rgb_vector)
double    yiq_vector[3], rgb_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'

INTEGER FUNCTION CMSR_FE_RGB_TO_YIQ (rgb_vector, yiq_vector)
DOUBLE PRECISION rgb_vector(3), yiq_vector(3)

INTEGER FUNCTION CMSR_FE_YIQ_TO_RGB (yiq_vector, rgb_vector)
DOUBLE PRECISION yiq_vector(3), rgb_vector(3)
```

Lisp Syntax

```
CMSR:fe-rgb-to-yiq (rgb-vector &optional yiq-vector)
CMSR:fe-yiq-to-rgb (yiq-vector &optional rgb-vector)
```

ARGUMENTS

rgb_vector	A one-dimensional array of 3 elements containing an RGB color triplet. The red intensity is in the first element, the green intensity is in the second element, and the blue intensity is in the third element. Each of the RGB color components should be in the range of [0,1].
yiq_vector	A one-dimensional array of 3 elements containing a YIQ color triplet.

DESCRIPTION

`CMSR_fe_rgb_to_yiq` converts an RGB triplet in *rgb_vector* to a YIQ triplet, places the result in *yiq_vector*, and returns a pointer to *yiq_vector*. Each of the RGB color components should be in the range [0,1].

`CMSR_fe_yiq_to_rgb` converts a YIQ triplet in *yiq_vector* to an RGB triplet, places the result in *rgb_vector,* and returns a pointer to *rgb_vector*. Each of the YIQ color components should be in the range [0,1] but this restriction is not enforced.

CMSR_fe_rgb_to_hsv
CMSR_fe_hsv_to_rgb

Converts a color vector from RGB to HSV (HSV to RGB) color models.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_rgb_to_hsv (rgb_vector, hsv_vector)
double    rgb_vector[3], hsv_vector[3];

double *
  CMSR_fe_hsv_to_rgb (hsv_vector, rgb_vector)
double    hsv_vector[3], rgb_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'

SUBROUTINE CMSR_fe_rgb_to_hsv (rgb_vector, hsv_vector)
DOUBLE PRECISION rgb_vector(3), hsv_vector(3)

SUBROUTINE CMSR_fe_hsv_to_rgb (hsv_vector, rgb_vector)
DOUBLE PRECISION hsv_vector(3), rgb_vector(3)
```

Lisp Syntax

```
CMSR:fe-rgb-to-hsv (rgb-vector &optional hsv-vector)
CMSR:fe-hsv-to-rgb (hsv-vector &optional rgb-vector)
```

ARGUMENTS

rgb_vector	A one-dimensional array of 3 elements containing an RGB color triplet. The red intensity is in the first element, the green intensity is in the second element, and the blue intensity is in the third element. Each of the RGB color components should be in the range of [0,1].
hsv_vector	A one-dimensional array of 3 elements containing an HSV color triplet. The hue of the color is in the first element, the saturation is in the second element, and the value is in the third element. Hue should be in the range [0, 2*pi], and saturation and value should be in the range [0,1].

DESCRIPTION

`CMSR_fe_rgb_to_hsv` converts the RGB triplet in *rgb_vector* to HSV, places the result in *hsv_vector*, and returns a pointer to *hsv_vector*. Hue will be between 0.0 and 2*pi. If *s* is zero, *h* is irrelevant and is set to zero. If *v* is zero, *h* and *s* are irrelevant and are also set to zero.

`CMSR_fe_hsv_to_rgb` converts the HSV triplet in *hsv_vector* to an RGB triplet, places the result in *rgb_vector* and returns a pointer to *rgb_vector*. Hue is taken modulo 2*pi. If *s* is zero, *h* is irrelevant. If *v* is zero, *s* and *v* are irrelevant.

CMSR_fe_rgb_to_hsl
CMSR_fe_hsl_to_rgb

Converts a color vector from RGB to HSL (HSL to RGB) color models.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_rgb_to_hsl (rgb_vector, hsl_vector)
double rgb_vector[3], hsl_vector[3];

double *
  CMSR_fe_hsl_to_rgb (hsl_vector, rgb_vector)
double hsl_vector[3], rgb_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_FE_RGB_TO_HSL (rgb_vector, hsl_vector)
DOUBLE PRECISION rgb_vector(3), hsv_vector(3)

SUBROUTINE CMSR_FE_HSL_TO_RGB (hsv_vector, rgb_vector)
DOUBLE PRECISION hsv_vector(3), rgb_vector(3)
```

Lisp Syntax

```
CMSR:fe-rgb-to-hsl (rgb–vector &optional hsv–vector)
CMSR:fe-hsl-to-rgb (hsv–vector &optional rgb–vector)
```

ARGUMENTS

rgb_vector	A one-dimensional array of 3 elements containing an RGB color triplet. The red intensity is in the first element, the green intensity is in the second element, and the blue intensity is in the third element. Each of the RGB color components should be in the range of [0,1].
hsl_vector	A one-dimensional array of 3 elements containing an HSL color triplet. The hue of the color is in the first element, the saturation is in the second element, and the lightness is in the third element. Hue should be in the range [0, 2*pi], and saturation and lightness should be in the range [0,1].

DESCRIPTION

`CMSR_fe_rgb_to_hsl` converts the RGB triplet in *rgb_vector* to an HSL triplet and places the result in *hsl_vector*. **`CMSR_fe_hsl_to_rgb`** converts the HSL triplet in *hsl_vector* to an RGB triplet and places the result in *rgb_vector*.

In C and Lisp these routines also return a pointer to the result vector.

If saturation is zero, the resulting color is a gray shade. In this case hue is irrelevant and is set to zero. If lightness is zero, the color is black. In this case both hue and saturation are irrelevant and are set to zero.

3.5 Front-End Miscellaneous Routines

This section contains utility routines to convert between degrees and radians:

CMSR_fe_deg_to_rad
CMSR_fe_rad_to_deg

Converts degrees to radians (radians to degrees).

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double
   CMSR_fe_deg_to_rad (value)
double value;

double
   CMSR_fe_rad_to_deg (value)
double value;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
DOUBLE PRECISION FUNCTION CMSR_FE_DEG_TO_RAD (value)
DOUBLE PRECISION value

DOUBLE PRECISION FUNCTION FSR_FE_RAD_TO_DEG (value)
DOUBLE PRECISION value
```

Lisp Syntax

```
CMSR:fe-deg-to-rad (value)
CMSR:fe-rad-to-deg (value)
```

ARGUMENTS

value	The value to be converted.

DESCRIPTION

`CMSR_fe_deg_to_rad` accepts the argument *value*, in degrees, and returns it expressed in radians.

`CMSR_fe_rad_to_deg` accepts the argument *value*, in radians, and returns it expressed in degrees.

3.4 Front-End Color Conversion

This section documents the new *Render routines that convert color vectors between color spaces.

CMSR_fe_rgb_to_cmy CMSR_fe_cmy_to_rgb

Converts color vector from RGB to CMY (CMY to RGB) color models.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
double *
  CMSR_fe_rgb_to_cmy (rgb_vector, cmy_vector);
double    rgb_vector[3], cmy_vector[3];

double *
  CMSR_fe_cmy_to_rgb (cmy_vector, rgb_vector);
double    cmy_vector[3], rgb_vector[3];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'

INTEGER FUNCTION CMSR_FE_RGB_TO_CMY (rgb_vector, cmy_vector)
DOUBLE PRECISION rgb_vector(3), cmy_vector(3)

INTEGER FUNCTION CMSR_FE_CMY_TO_RGB (cmy_vector, rgb_vector)
DOUBLE PRECISION cmy_vector(3), rgb_vector(3)
```

Lisp Syntax

```
CMSR:fe-rgb-to-cmy (rgb-vector &optional cmy-vector)
CMSR:fe-cmy-to-rgb (cmy-vector &optional rgb-vector)
```

3.6 CM Vector Routines

This section documents the *Render routines that operate on vectors in Paris fields in CM memory. Vectors in *Render are one-dimensional arrays of either two or three elements.

On the CM, each vector is a single field of (dimension)*(*signif_len*+*exp_len*+1) bits, where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and the 1 is for a sign bit. Each element of the vector occupies a subfield of (*signif_len*+*exp_len*+1) bits, and these subfields are arranged so that element 0 is in the least significant bits.

The routines documented here are:

CMSR_v_abs_2d
CMSR_v_abs_3d

Calculates the length of a 2D (3D) vector.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
   CMSR_v_abs_2d  (dest_field, src_vector_field, signif_len, exp_len)
void
   CMSR_v_abs_3d  (dest_field, src_vector_field, signif_len, exp_len field)
CM_field_id_t   dest_field;
CM_field_id_t   src_vector_field;
unsigned int    signif_len;
unsigned int    exp_len;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
CMSR_V_ABS_2D  (dest_field, src_vector_field, signif_len, exp_len)
CMSR_V_ABS_3D  (dest_field, src_vector_field, signif_len, exp_len field)
INTEGER  dest_field
INTEGER  src_vector_field
INTEGER  signif_len
INTEGER  exp_len
```

Lisp Syntax

```
CMSR:v-abs-2d   (dest-field  src-vector-field
                       &optional  (signif-len 23)  (exp-len 8))
CMSR:v-abs-3d   (dest-field  src-vector-field
                       &optional  (signif-len 23)  (exp-len 8))
```

ARGUMENTS

dest_field	A Paris field identifier specifying the field in CM memory to which the result is written. *dest_field* must be in the same VP set as *src_vector_field.*
src_vector_field	A Paris field indentifier specifying the field containing the vector. Each element has a length of (*signif_len* + *exp_len* + 1) bits. For **`CMSR_v_abs_2d`** the vector contains two floating-point values organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1) For **`CMSR_v_abs_3d`** the vector contains three floating-point values organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1)
signif_len	The length, in bits, of the significand of the floating-point values in *dest_field* and *src_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_field* and *src_vector_field.*

DESCRIPTION

For each active processor, **`CMSR_v_abs_2d`** and **`CMSR_v_abs_3d`** calculate the length of the vector specified in *src_vector_field* and write the result to *dest_field.* The fields *src_vector_field* and *dest_field* must both be in the current VP set.

SEE ALSO

`CMSR_v_abs_squared_2d`

`CMSR_v_abs_squared_3d`

`CMSR_fe_v_abs_2d`

`CMSR_fe_v_abs_3d`

`CMSR_fe_v_abs_squared_2d`

`CMSR_fe_v_abs_squared_3d`

CMSR_v_abs_squared_2d
CMSR_v_abs_squared_3d

Calculates the length squared of a 2D (3D) vector.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_v_abs_squared_2d
                        (dest_field, src_vector_field, signif_len, exp_len field)
void
  CMSR_v_abs_squared_3d
                        (dest_field, src_vector_field, signif_len, exp_len field)
CM_field_id_t  dest_field;
CM_field_id_t  src_vector_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_V_ABS_SQUARED_2D
&                       (dest_field, src_vector_field, signif_len, exp_len field)
   SUBROUTINE CMSR_V_ABS_SQUARED_3D
&                       (dest_field, src_vector_field, signif_len, exp_len field)
   INTEGER  dest_field
   INTEGER  src_vector_field
   INTEGER  signif_len, exp_len
```

Lisp Syntax

```
CMSR:v-abs-squared-2d(dest-field  src-vector-field
                           &optional (signif-len 23) (exp-len 8))
CMSR:v-abs-squared-3d(dest-field  src-vector-field
                           &optional '(signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_field	A Paris field identifier specifying the field in CM memory to which the result is written. *dest_field* must be in the same VP set as *src_vector_field.*
src_vector_field	A Paris field indentifier specifying the field containing the vector. Each element has a length of (*signif_len* + *exp_len* + 1) bits. For **`CMSR_v_abs_squared_2d`** the vector contains two floating-point values organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1) For **`CMSR_v_abs_squared_3d`** the vector contains three floating-point values organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in *dest_field* and *src_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_field* and *src_vector_field.*

DESCRIPTION

For each active processor, **`CMSR_v_abs_squared_2d`** and **`CMSR_v_abs_squared_3d`** calculate the the length squared of the vector specified in *src_vector_field* and write the result to *dest_field. src_vector_field* and *dest_field* must both be in the current VP set.

SEE ALSO

`CMSR_v_abs_3d`, **`CMSR_v_abs_3d`**

`CMSR_fe_v_abs_2d`, **`CMSR_fe_v_abs_3d`**

`CMSR_fe_v_abs_squared_2d`, **`CMSR_fe_v_abs_squared_3d`**

CMSR_v_add_2d
CMSR_v_add_3d

Adds 2D (3D) vectors element by element.

SYNTAX

C Syntax

#include <cm/cmsr.h>

void
  CMSR_v_add_2d
  (*dest_vector_field, src1_vector_field, src2_vector_field, signif_len, exp_len*)

void
  CMSR_v_add_3d
  (*dest_vector_field, src1_vector_field, src2_vector_field, signif_len, exp_len*)

CM_field_id_t *dest_vector_field;*
CM_field_id_t *src1_vector_field;*
CM_field_id_t *src2_vector_field;*
unsigned int *signif_len;*
unsigned int *exp_len;*

Fortran Syntax

INCLUDE '/usr/include/cm/cmsr-math-fort.h'

SUBROUTINE CMSR_V_ADD_2D
& (*dest_vector_field, src1_vector_field, src2_vector_field, signif_len, exp_len*)

SUBROUTINE CMSR_V_ADD_3D
& (*dest_vector_field, src1_vector_field, src2_vector_field, signif_len, exp_len*)

INTEGER *dest_vector_field;*
INTEGER *src1_vector_field;*
INTEGER *src2_vector_field;*
INTEGER *signif_len;*
INTEGER *exp_len;*

Lisp Syntax

```
CMSR:v-add-2d  (dest-vector-field src1-vector-field src2-vector-field
                    &optional (signif-len 23) (exp-len 8))
CMSR:v-add-3d  (dest-vector-field src1-vector-field src2-vector-field
                    &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_vector_field	A Paris field identifier specifying the vector field in CM memory to which the result is written. *dest_vector_field* must be in the same VP set as *src1_vector_field* or *src2_vector_field.*
src1_vector_field, src2_vector_field	Paris field indentifiers specifying the fields containing the vectors that are to be added.

For `CMSR_v_add_2d` the vector fields contain two floating-point values organized so that *x* occupies the least significant bits and *y* the most significant bits. Each element has a length of (*signif_len* + *exp_len* + 1) bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).

For `CMSR_v_add_3d` the vector contains three floating-point values organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. Each element has a length of (*signif_len* + *exp_len* + 1) bits. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).

signif_len	The length, in bits, of the significand of the floating-point values in *dest_vector_field, src1_vector_field,* and *src2_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_vector_field, src1_vector_field,* and *src2_vector_field.*

DESCRIPTION

For each active processor, `CMSR_v_add_2d` and `CMSR_v_add_3d` add each element of the vector specified in *src1_vector_field* to the corresponding element of *src2_vector_field* and write the result to *dest_vector_field.* The fields *src1_vector_field, src2_vector_field,* and *dest_vector_field* must be in the current VP set.

CMSR_v_alloc_heap_field_2d
CMSR_v_alloc_heap_field_3d

Allocates a vector field organized as a 2-element (3-element) array on the heap and return its Paris field ID.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
CM_field_id_t
  CMSR_v_alloc_heap_field_2d (signif_len, exp_len)
unsigned int  signif_len, exp_len;

CM_field_id_t
  CMSR_v_alloc_heap_field_3d (signif_len, exp_len)
unsigned int  signif_len, exp_len;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
FUNCTION INTEGER CMSR_V_ALLOC_HEAP_FIELD_2D(signif_len, exp_len)
INTEGER  signif_len, exp_len;

FUNCTION INTEGER CMSR_V_ALLOC_HEAP_FIELD_3D(signif_len, exp_len)
INTEGER  signif_len, exp_len;
```

Lisp Syntax

```
CMSR:v-alloc-heap-field-2d (signif-len exp-len)
CMSR:v-alloc-heap-field-3d (signif-len exp-len)
```

ARGUMENTS

signif_len	The length, in bits, of the significand of the floating-point values to be stored in the returned field.
exp_len	The length, in bits, of the exponent of the floating-point values to be stored in the returned field.

DESCRIPTION

`CMSR_v_alloc_heap_field_2d` and **`CMSR_v_alloc_heap_field_3d`** allocate a vector field on the heap and return its Paris field ID.

A vector field is organized to contain an array of floating-point values. Each element of the vector has a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.

`CMSR_v_alloc_heap_field_2d` allocates a field organized as a 2-element array. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).

`CMSR_v_alloc_heap_field_3d` allocates a field organized as a 3-element array. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).

If the vector is a position vector, the field is organized so that *x* occupies the first element, *y* occupies the second element, and *z* (if present) occupies the third element.

CMSR_v_alloc_stack_field_2d
CMSR_v_alloc_stack_field_3d

Allocates a vector field organized as a 2-element (3-element) array on the stack and return its Paris field ID.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

CM_field_id_t
  CMSR_v_alloc_stack_field_2d (signif_len, exp_len)
unsigned int  signif_len, exp_len;

CM_field_id_t
  CMSR_v_alloc_stack_field_3d (signif_len, exp_len)
unsigned int  signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   FUNCTION INTEGER CMSR_V_ALLOC_STACK_FIELD_2D
&                                              (signif_len, exp_len)
   INTEGER signif_len, exp_len;

   FUNCTION INTEGER CMSR_V_ALLOC_STACK_FIELD_3D
&                                              (signif_len, exp_len)
   INTEGER signif_len, exp_len;
```

Lisp Syntax

```
CMSR:v-alloc-stack-field-2d (signif-len exp-len)
CMSR:v-alloc-stack-field-3d (signif-len exp-len)
```

ARGUMENTS

signif_len	The length, in bits, of the significand of the floating-point values to be stored in the returned field.
exp_len	The length, in bits, of the exponent of the floating-point values to be stored in the returned field.

DESCRIPTION

`CMSR_v_alloc_stack_field_2d` and **`CMSR_v_alloc_stack_field_3d`** allocate a vector field on the stack and return its Paris field ID.

A vector field is organized to contain an array of floating-point values. Each element of the vector has a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.

`CMSR_v_alloc_stack_field_2d` allocates a field organized as a 2-element array. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).

`CMSR_v_alloc_stack_field_3d` allocates a field organized as a 3-element array. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).

If the vector is a position vector, the field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits.

CMSR_v_copy_2d
CMSR_v_copy_3d

Copies the two (three)values in one vector field to another.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_v_copy_2d (dest_vector_field, src_vector_field, signif_len, exp_len)
CM_field_id_t  dest_vector_field, src_vector_field;
unsigned int   signif_len, exp_len;

void
  CMSR_v_copy_3d (dest_vector_field, src_vector_field, signif_len,
exp_len)
CM_field_id_t  dest_vector_field, src_vector_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-math-fort.h'
    SUBROUTINE CMSR_V_copy_2D
&                         (dest_vector_field, src_vector_field, signif_len, exp_len)
    INTEGER                  dest_vector_field, src_vector_field
    INTEGER                  signif_len, exp_len

    SUBROUTINE CMSR_V_copy_3D
&                         (dest_vector_field, src_vector_field, signif_len, exp_len)
    INTEGER                  dest_vector_field, src_vector_field
    INTEGER                  signif_len, exp_len
```

Lisp Syntax

`CMSR:v-copy-2d`
(dest–vector–field src–vector–field **`&optional`** *signif–len exp–len)*
`CMSR:v-copy-3d`
(dest–vector–field src–vector–field **`&optional`** *signif–len exp–len)*

ARGUMENTS

dest_vector_field	A Paris field identifier specifying the vector field in CM memory to which *src_vector_field* is to be copied.
src_vector_field	A Paris field identifier specifying the vector field in CM memory from which the vectors are to be copied. For **`CMSR_v_copy_2d`** this vector field contains two floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. The vector field is organized so that x occupies the least significant bits and y the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1). For **`CMSR_v_copy_3d`** this vector field contains three floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. The vector field is organized so that x occupies the least significant bits, y the following bits, and z the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in *dest_vector_field* and *src_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_vector_field* and *src_vector_field.*

DESCRIPTION

For each active processor in the current VP set, **`CMSR_v_copy_2d`** and **`CMSR_v_copy_3d`** copy the value in *src_vector_field* to *dest_vector_field.*

All fields must be in the current VP set. *dest_vector_field* may be the same field as *src_vector_field,* or the fields may be totally disjoint. However, partially overlapping fields cause unpredictable results.

CMSR_v_copy_const_2d
CMSR_v_copy_const_3d

Broadcasts a specified 2D (3D) vector to a vector field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_v_copy_const_2d(dest_vector_field, src_vector, signif_len,
exp_len)
CM_field_id_t  dest_vector_field;
double         src_vector[2];
unsigned int   signif_len, exp_len;

void
  CMSR_v_copy_const_3d (dest_vector_field vector signif_len exp_len)
CM_field_id_t  dest_vector_field;
double         src_vector[3];
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_V_COPY_CONST_2D
&                        (dest_vector_field, src_vector, signif_len, exp_len)
   INTEGER            dest_vector_field
   DOUBLE PRECISION   src_vector(2)
   INTEGER            signif_len, exp_len

   SUBROUTINE CMSR_V_COPY_CONST_3D
&                           (dest_vector_field vector signif_len exp_len)
   INTEGER            dest_vector_field
   DOUBLE PRECISION   src_vector(3)
   INTEGER            signif_len, exp_len
```

Lisp Syntax

```
CMSR:v-copy-const-2d (vector-field src-vector
                      &optional (signif-len 23) (exp-len 8))
CMSR:v-copy-const-3d (vector-field src-vector
                      &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_vector_field	A Paris field indentifier specifying the field in which the 2D vector is to be stored. Each element of the vector has a length of (*signif_len* + *exp_len* + 1) bits. For `CMSR_v_copy_const_2d` the vector field contains two floating-point values organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1). For `CMSR_v_copy_const_3d` the vector field contains three floating-point values organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).
src_vector	A 1 x 2 array for `CMSR_v_copy_const_2d` or a 1 x 3 array for `CMSR_v_copy_const_3d`, containing the vector to be loaded into the vector field.
signif_len	The length, in bits, of the significand of the floating-point values in *dest_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_vector_field.*

DESCRIPTION

For each active processor, `CMSR_v_copy_const_2d` and `CMSR_v_copy_const_3d` place the contents of *src_vector* in *dest_vector_field.*

SEE ALSO

`CMSR_v_write_to_processor_2d`

`CMSR_v_write_to_processor_3d`

`CMSR_v_read_from_processor_2d`

`CMSR_v_read_from_processor_3d`

CMSR_v_cos_between_2d
CMSR_v_cos_between_3d

Finds the cosine of the angle between two 2D (3D) vectors.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
   CMSR_v_cos_between_2d
    (dest_field, src1_vector_field, src2_vector_field, signif_len, exp_len)
CM_field_id_t  dest_field, src1_vector_field, src2_vector_field;
unsigned int   signif_len, exp_len;

void
   CMSR_v_cos_between_3d
    (dest_field, src1_vector_field, src2_vector_field, signif_len, exp_len)
CM_field_id_t  dest_field, src1_vector_field, src2_vector_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-math-fort.h'
    SUBROUTINE CMSR_V_COS_BETWEEN_2D
&      (dest_field, src1_vector_field, src2_vector_field, signif_len, exp_len)
    INTEGER            dest_field, src1_vector_field, src2_vector_field
    INTEGER            signif_len, exp_len

    SUBROUTINE CMSR_V_COS_BETWEEN_3D
&      (dest_field, src1_vector_field, src2_vector_field, signif_len, exp_len)
    INTEGER            dest_field, src1_vector_field, src2_vector_field
    INTEGER            signif_len, exp_len
```

Lisp Syntax

```
CMSR:v-cos-between-2d (dest-field src1-vector-field src2-vector-field
                       &optional signif-len exp-len)
CMSR:v-cos-between-3d (dest-field src1-vector-field src2-vector-field
                       &optional signif-len exp-len)
```

ARGUMENTS

dest_field

A Paris field identifier specifying the field in CM memory in which the cosine calculated by the routine is stored. The length of this field is (*signif_len* + *exp_len* + 1) bits.

src1_vector_field, src2_vector_field

Paris field identifiers specifying the vector field in CM memory containing the vectors to be operated on.

For **`CMSR_v_cos_between_2d`** this vector field contains two floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. The vector field is organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).

For **`CMSR_v_cos_between_3d`** this vector field contains three floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. The vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).

signif_len

The length, in bits, of the significand of the floating-point values in the vector fields.

exp_len

The length, in bits, of the exponent of the floating-point values in the vector fields.

DESCRIPTION

For each active processor in the current VP set, **`CMSR_v_cos_between_2d`** and **`CMSR_v_cos_between_3d`** find the cosine of the angle between the vectors in the two source fields and place the result in the destination field. This is the dot-product of the normalized vectors, however, the source field vectors need not be unit length. All fields must be in the current VP set.

Note that the *dest_field* is a standard floating-point field and not a vector.

CMSR_v_cross_product_3d

Calculates the cross-product of two 3D vectors.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
   CMSR_v_cross_product_3d
    (dest_vector_field, src1_vector_field, src2_vector_field, signif_len, exp_len)
CM_field_id_t  dest_vector_field;
CM_field_id_t  src1_vector_field;
CM_field_id_t  src2_vector_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_V_CROSS_PRODUCT_3D
&  (dest_vector_field, src1_vector_field, src2_vector_field, signif_len, exp_len)
   INTEGER  dest_vector_field
   INTEGER  src1_vector_field
   INTEGER  src2_vector_field
   INTEGER  signif_len, exp_len
```

Lisp Syntax

```
CMSR:v-cross-product-3d
                    (dest-vector-field src1-vector-field  src-vector-field-2
                        &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_field	A Paris field identifier specifying the field in CM memory to which the result is written. *dest_field* must be in the same VP set as *src1_vector_field* and *src2_vector_field.*

src1_vector_field, src2_vector_field	Paris field indentifiers specifying the fields in CM memory containing the 3D coordinates of the vectors. *src1_vector_field* and *src2_vector_field* must be in the same VP set as *dest_field.* The coordinates are floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. The vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in *dest_field*, *src_field_1*, and *src_field_2.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_field*, *src_field_1*, and *src_field_2.*

DESCRIPTION

For each active processor, `CMSR_v_cross_product_3d` calculates the cross-product between *src1_vector_field* and *src2_vector_field* and puts the result in the *dest_field.*

All fields must be in the current VP set. The *dest* field may be the same field as *src1_vector_field* or *src2_vector_field,* or the fields may be totally disjoint. However, partially overlapping fields cause unpredictable results.

SEE ALSO

`CMSR_fe_v_cross_product_3d`

CMSR_v_dot_product_2d
CMSR_v_dot_product_3d

Calculates the dot product of two 2D (3D) vectors.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
   CMSR_v_dot_product_2d
        (product_field, src1_vector_field, src2_vector_field, signif_len, exp_len)

void
   CMSR_v_dot_product_3d
        (product_field, src1_vector_field, src2_vector_field, signif_len, exp_len)

CM_field_id_t  product_field;
CM_field_id_t  src1_vector_field;
CM_field_id_t  src2_vector_field;
unsigned int   signif_len;
unsigned int   exp_len;
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-math-fort.h'

    SUBROUTINE CMSR_V_DOT_PRODUCT_2D
&           (product_field, src1_vector_field, src2_vector_field, signif_len, exp_len)

    SUBROUTINE CMSR_V_DOT_PRODUCT_3D
&           (product_field, src1_vector_field, src2_vector_field, signif_len, exp_len)

    INTEGER  product_field
    INTEGER  src1_vector_field
    INTEGER  src2_vector_field
    INTEGER  signif_len
    INTEGER  exp_len
```

Lisp Syntax

`CMSR:v-dot-product-2d``(`*product–field src1–vector–field src2–vector–field*
`&optional` `(`*signif–len* `23)` `(`*exp–len* `8))`

`CMSR:v-dot-product-3d``(`*product–field src1–vector–field src2–vector–field*
`&optional` `(`*signif–len* `23)` `(`*exp–len* `8))`

ARGUMENTS

product_field	A Paris field identifier specifying the field in CM memory to which the product of *src1_vector_field* and *src2_vector_field* is written.
src1_vector_field, src2_vector_field	Paris field indentifiers specifying the fields in CM memory containing the vectors to be multiplied. *src1_vector_field* and *src2_vector_field* must be in the same VP set as *product_field.*
signif_len	The length, in bits, of the significand of the floating-point values in *product_field*, *src_field_1*, and *src_field_2.*
exp_len	The length, in bits, of the exponent of the floating-point values in *product_field*, *src_field_1*, and *src_field_2.*

DESCRIPTION

For each active processor, **`CMSR_v_dot_product_2d`** and **`CMSR_v_dot_product_3d`** calculate the dot product of *src1_vector_field* and *src2_vector_field* and put the result in the *product–field.*

For **`CMSR_v_dot_product_2d`**, *product_field*, *src1_vector_field*, and *src2_vector_field* contain two floating-point values organized so that *x* occupies the least significant bits and *y* the most significant bits. Each element has a length of (*signif_len* + *exp_len* + 1) bits where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).

For **`CMSR_v_dot_product_3d`**, *product_field*, *src1_vector_field*, and *src2_vector_field* contain three floating-point values organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. Each element has a length of (*signif_len* + *exp_len* + 1) bits. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).

All fields must be in the current VP set. The *product* field may be the same field as *src1_vector_field* or *src2_vector_field,* or the fields may be totally disjoint. However, partially overlapping fields cause unpredictable results.

SEE ALSO

`CMSR_v_dot_product_3d`

`CMSR_fe_v_dot_product_2d`

`CMSR_fe_v_dot_product_3d`

CMSR_v_field_length

Returns the length of a vector field of a specified rank.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
int
  CMSR_v_field_length (rank, signif_len, exp_len);
int rank;
unsigned int signif_len;
unsigned int exp_len;
```

Fortran Syntax

```
INTEGER FUNCTION CMSR_V_FIELD_LENGTH (rank, signif_len, exp_len)
INTEGER rank
INTEGER signif_len
INTEGER exp_len
```

Lisp Syntax

```
CMSR:v-field-length(rank &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

rank	The number of dimensions in the vector for which the field is allocated.
signif_len	The length, in bits, of the significand of the floating-point values in the field.
exp_len	The length, in bits, of the exponent of the floating-point values in the field.

DESCRIPTION

`CMSR_v_field_length` returns the length, in bits, of the Paris field that must be allocated to hold a vector of *rank* elements.

Each element of the vector is assumed to be a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.

SEE ALSO

`CMSR_m_field_length`

CMSR_v_is_zero_2d
CMSR_v_is_zero_3d

Tests whether each 2D (3D) vector in a vector field is of length 0.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
   CMSR_v_is_zero_2d (bit_field, vector_field, signif_len, exp_len)
CM_field_id_t bit_field, vector_field;
unsigned int  signif_len, exp_len;

void
   CMSR_v_is_zero_3d (bit_field, vector_field, signif_len, exp_len)
CM_field_id_t bit_field, vector_field;
unsigned int  signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_V_IS_ZERO_2D
&                               (bit_field, vector_field, signif_len, exp_len)
   INTEGER bit_field, vector_field;
   INTEGER signif_len, exp_len;

   SUBROUTINE CMSR_V_IS_ZERO_3D
&                               (bit_field, vector_field, signif_len, exp_len)
   INTEGER bit_field, vector_field;
   INTEGER signif_len, exp_len;
```

Lisp Syntax

```
CMSR:v-is-zero-2d(bit-field vector-field,
                       &optional (signif-len 23) (exp-len 8))
CMSR:v-is-zero-3d (bit-field vector-field
                       &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

bit_field	A Paris field identifier specifying the field in CM memory in which the results of testing *vector_field* are to be returned.
vector_field	A Paris field identifier specifying the field in CM memory containing the vectors to be tested. For the 2D routine these are 2-element arrays; for the 3D routine these are 3-element arrays. Each element is assumed to be a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.
signif_len	The length, in bits, of the significand of the floating-point values in *vector_field*.
exp_len	The length, in bits, of the exponent of the floating-point values in *vector_field*.

DESCRIPTION

For each active processor in the current VP set, `CMSR_v_is_zero_2d` and `CMSR_v_is_zero_3d` test whether the vector contained in *vector_field* is zero length. If, in a particular processor, the vector in *vector_field* is of length zero, the corresponding bit in *bit_field* is set to 1. If the vector has non-zero length, the bit is set to 0.

CMSR_v_negate_2d
CMSR_v_negate_3d

Multiplies each vector element in the 2D (3D) vector field by –1.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_v_negate_2d
                (dest_vector_field, src_vector_field, signif_len, exp_len)
CM_field_id_t  dest_vector_field, src_vector_field;
unsigned int   signif_len, exp_len;

void
  CMSR_v_negate_3d
                (dest_vector_field, src_vector_field, signif_len, exp_len)
CM_field_id_t  dest_vector_field, src_vector_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_V_NEGATE_2D
&                    (dest_vector_field, src_vector_field, signif_len, exp_len)
   INTEGER  dest_vector_field, src_vector_field
   INTEGER  signif_len, exp_len

   SUBROUTINE CMSR_V_NEGATE_3D
&                    (dest_vector_field, src_vector_field, signif_len, exp_len)
   INTEGER  dest_vector_field, src_vector_field
   INTEGER  signif_len, exp_len
```

Lisp Syntax

```
CMSR:v-negate-2d (dest-vector-field src-vector-field
                  &optional (signif-len 23) (exp-len 8))
CMSR:v-negate-3d (dest-vector-field src-vector-field
                  &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_vector_field	A Paris field identifier specifying the vector field in CM memory in which the inverted vectors are to be stored.
src_vector_field	A Paris field identifier specifying the vector field in CM memory from which to read the vectors to be inverted. For `CMSR_v_negate_2d` this vector field contains two floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. The vector field is organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1). For `CMSR_v_negate_3d` this vector field contains three floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. The vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in *dest_vector_field* and *src_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_vector_field* and *src_vector_field.*

DESCRIPTION

For each active processor in the current VP set, `CMSR_v_negate_2d` and `CMSR_v_negate_3d` multiply each vector element in *src_vector_field* by –1 and put the result in *dest_vector_field.*

Both fields must be in the current VP set. *dest_vector_field* may be the same field as *src_vector_field,* or the fields may be totally disjoint. However, partially overlapping fields cause unpredictable results.

CMSR_v_normalize_2d
CMSR_v_normalize_3d

Normalizes each 2D (3D) vector in a vector field to a unit vector.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
  CMSR_v_normalize_2d
               (dest_vector_field, src_vector_field, signif_len, exp_len)
CM_field_id_t  dest_vector_field, src_vector_field;
unsigned int   signif_len, exp_len;

void
  CMSR_v_normalize_3d
               (dest_vector_field, src_vector_field, signif_len, exp_len)
CM_field_id_t  dest_vector_field, src_vector_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_V_NORMALIZE_2D
&                     (dest_vector_field, src_vector_field, signif_len, exp_len)
   INTEGER dest_vector_field, src_vector_field
   INTEGER signif_len, exp_len

   SUBROUTINE CMSR_V_NORMALIZE_3D
&                     (dest_vector_field, src_vector_field, signif_len, exp_len)
   INTEGER dest_vector_field, src_vector_field
   INTEGER signif_len, exp_len
```

Lisp Syntax

```
CMSR:v-normalize-2d (dest–vector–field src–vector–field
                     &optional (signif–len 23) (exp–len 8))
CMSR:v-normalize-3d (dest–vector–field src–vector–field
                     &optional (signif–len 23) (exp–len 8))
```

ARGUMENTS

dest_vector_field	A Paris field identifier specifying the vector field in CM memory in which the normalized vectors are to be stored.
src_vector_field	A Paris field identifier specifying the vector field in CM memory from which to read the vectors to be normalized. For **`CMSR_v_normalize_2d`** this vector field contains two floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. The vector field is organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1). For **`CMSR_v_normalize_3d`** this vector field contains three floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. The vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in *dest_vector_field* and *src_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_vector_field* and *src_vector_field.*

DESCRIPTION

For each active processor in the current VP set, **`CMSR_v_normalize_2d`** and **`CMSR_v_normalize_3d`** compute a unit vector pointing in the same direction as *src_vector_field* and puts the result in *dest_vector_field.*

Both fields must be in the current VP set. *dest_vector_field* may be the same field as *src_vector_field,* or the fields may be totally disjoint. However, partially overlapping fields cause unpredictable results.

CMSR_v_perpendicular_2d
CMSR_v_perpendicular_3d

Creates a unit vector perpendicular to one 2D vector (or two 3D vectors).

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_v_perpendicular_2d
                (dest_vector_field, src_vector_field, signif_len, exp_len)
CM_field_id_t  dest_vector_field, src_vector_field;
unsigned int   signif_len, exp_len;

void
  CMSR_v_perpendicular_3d
  (dest_vector_field, src1_vector_field, src2_vector_field, signif_len,
exp_len)
CM_field_id_t  dest_vector_field, src1_vector_field, src2_vector_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
      INCLUDE '/usr/include/cm/cmsr-math-fort.h'
      SUBROUTINE CMSR_V_PERPENDICULAR_2D
&                   (dest_vector_field, src_vector_field, signif_len, exp_len)
      INTEGER             dest_vector_field, src_vector_field
      INTEGER             signif_len, exp_len

      SUBROUTINE CMSR_V_PERPENDICULAR_3D
&         (dest_vector_field, src1_vector_field, src2_vector_field, signif_len, exp_len)
      INTEGER  dest_vector_field, src1_vector_field, src2_vector_field
      INTEGER  signif_len, exp_len
```

Lisp Syntax

```
CMSR:v-perpendicular-2d (dest-vector-field src-vector-field
                         &optional (signif-len 23) (exp-len 8))
CMSR:v-perpendicular-3d
                    (dest-vector-field src1-vector-field src2-vector-field
                      &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_vector_field	A Paris field identifier specifying the vector field in CM memory in which the results of the routine are stored.

src_vector_field, src1_vector_field src2_vector_field

Paris field identifiers specifying the vector field in CM memory containing the vectors to be operated on.

For **CMSR_v_perpendicular_2d**, *src_vector_field* and *dest_vector_field* contain two floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. A 2D vector field is organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).

For **CMSR_v_perpendicular_3d** *dest_vector_field, src1_ vector_ field,* and *src2_vector_field* each contain three floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. A 3D vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).

signif_len	The length, in bits, of the significand of the floating-point values in the vector fields.
exp_len	The length, in bits, of the exponent of the floating-point values in the vector fields.

DESCRIPTION

For each active processor in the current VP set, **`CMSR_v_perpendicular_2d`** constructs a unit vector perpendicular to *src_vector_field* and puts the result in *dest_vector_field.* The source vector need not be unit-length, but *src–vector* should not be zero length.

For each active processor in the current VP set, **`CMSR_v_perpendicular_3d`** constructs a unit vector perpendicular to *src1_vector_field* and *src2_vector_field* and puts the result in *dest_vector_field.* The source vectors need not be unit-length.

All fields must be in the current VP set. *dest_vector_field* may be the same field as *src_vector_field, src1_vector_field,* or *src2_vector_field,* or the fields may be totally disjoint. However, partially overlapping fields cause unpredictable results.

CMSR_v_print_2d
CMSR_v_print_3d

Prints on `stdout` the contents of a specified processor's 2D (3D) vector field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
   CMSR_v_print_2d  (processor, src_vector_field, signif_len, exp_len)
void
   CMSR_v_print_3d  (processor, src_vector_field, signif_len, exp_len)
unsigned int   processor;
CM_field_id_t  src_vector_field;
unsigned int   signif_len;
unsigned int   exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_V_PRINT_2D
&                               (processor, src_vector_field, signif_len, exp_len)
   SUBROUTINE CMSR_V_PRINT_3D
&                               (processor, src_vector_field, signif_len, exp_len)
   INTEGER  processor;
   INTEGER  src_vector_field;
   INTEGER  signif_len;
   INTEGER  exp_len;
```

Lisp Syntax

```
CMSR:v-print-2d  (processor src-vector-field
                    &optional  (signif-len 23)  (exp-len 8))
CMSR:v-print-3d  (processor src-vector-field
                    &optional  (signif-len 23)  (exp-len 8))
```

ARGUMENTS

processor	The send address of the processor from which you wish to print the vector.
src_vector_field	Paris field identifiers specifying the vector field in CM memory containing the vector to be printed. For `CMSR_v_print_2d` the vector field contains two floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. The vector field is organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1). For `CMSR_v_print_3d` the vector field contains three floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. The vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in the vector fields.
exp_len	The length, in bits, of the exponent of the floating-point values in the vector fields.

DESCRIPTION

`CMSR_v_print_2d` and `CMSR_v_print_3d` print on `stdout` the double-precision floating-point contents of *src_vector_field* in the given *processor*.

The vector elements are printed on a single line, separated by spaces, and followed by a carriage return.

CMSR_v_read_from_processor_2d
CMSR_v_read_from_processor_3d

Reads a 2D (3D) vector field from a specified processor into a front-end array.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

double *
   CMSR_v_read_from_processor_2d
               (processor, src_vector_field, dest_vector, signif_len, exp_len)

unsigned int   processor;
CM_field_id_t  src_vector_field;
double         dest_vector[2];
unsigned int   signif_len, exp_len;

double *
   CMSR_v_read_from_processor_3d
               (processor, src_vector_field, dest_vector, signif_len, exp_len)

unsigned int   processor;
CM_field_id_t  src_vector_field;
double         dest_vector[3];
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
      INCLUDE '/usr/include/cm/cmsr-math-fort.h'

      SUBROUTINE CMSR_V_READ_FROM_PROCESSOR_2D
&                (processor, src_vector_field, dest_vector, signif_len, exp_len)

      INTEGER            processor
      INTEGER            src_vector_field
      DOUBLE PRECISION   dest_vector(2)
      INTEGER            signif_len, exp_len
```

```
   SUBROUTINE CMSR_V_READ_FROM_PROCESSOR_3D
&                  (processor, src_vector_field, dest_vector, signif_len, exp_len)
   INTEGER            processor
   INTEGER            src_vector_field
   DOUBLE PRECISION   dest_vector(3)
   INTEGER            signif_len,  exp_len
```

Lisp Syntax

```
CMSR:v-read-from-processor-2d
                  (processor  vector-field
                   &optional  dest-vector  (signif-len  23)  (exp-len  8))
CMSR:v-read-from-processor-3d
                  (processor  vector-field
                   &optional  dest-vector  (signif-len  23)  (exp-len  8))
```

ARGUMENTS

processor	The processor send address from which the vector is to be read.
src_vector_field	A Paris field indentifier specifying the field in *processor* from which the 2D vector is to be read. Each element of the vector has a length of (*signif_len* + *exp_len* + 1) bits. A 2D vector field contains two floating-point values organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1). A 3D vector field contains three floating-point values organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).
dest_vector	For `CMSR_v_read_from_processor_2d` a 1 x 2 array, or for `CMSR_v_read_from_processor_3d` a 1 x 3 array, in which the vector is to be returned.
signif_len	The length, in bits, of the significand of the floating-point values in *src_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *src_vector_field.*

DESCRIPTION

`CMSR_v_read_from_processor_2d` and **`CMSR_v_read_from_processor_3d`** read and return the contents of *vector_field* from the given processor, *processor,* to *dest_vector.*

In C and Lisp these routines also return a pointer to *dest_vector.*

If the *dest_vector* parameter on the front-end is NULL, a new vector is allocated.

SEE ALSO

`CMSR_v_write_to_processor_2d`

`CMSR_v_write_to_processor_3d`

CMSR_v_reflect_2d
CMSR_v_reflect_3d

Calculates a 2D (3D) reflectance vector for specified incident and normal vectors.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
   CMSR_v_reflect_2d
             (dest_vector_field, src_vector_field, normal_vector_field,
              signif_len, exp_len)
CM_field_id_t  dest_vector_field, src_vector_field, normal_vector_field;
unsigned int   signif_len, exp_len;

void
   CMSR_v_reflect_3d
             (dest_vector_field, src_vector_field, normal_vector_field,
              signif_len, exp_len)
CM_field_id_t  dest_vector_field, src_vector_field, normal_vector_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_V_REFLECT_2D
&            (dest_vector_field, src_vector_field, normal_vector_field,
&             signif_len, exp_len)
   INTEGER dest_vector_field, src_vector_field, normal_vector_field
   INTEGER signif_len, exp_len

   SUBROUTINE CMSR_V_REFLECT_3D
&            (dest_vector_field, src_vector_field, normal_vector_field,
&             signif_len, exp_len)
   INTEGER dest_vector_field, src_vector_field, normal_vector_field
   INTEGER signif_len, exp_len
```

Lisp Syntax

```
CMSR:v-reflect-2d
              (dest-vector-field src-vector-field normal-vector-field
               &optional (signif-len 23) (exp-len 8))
CMSR:v-reflect-3d
              (dest-vector-field src-vector-field normal-vector-field
               &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_vector_field	A Paris field identifier specifying the vector field in CM memory in which the reflectance vector is stored.
src_vector_field	A Paris field identifier specifying the vector field in CM memory containing the vector indicating the direction of the incident light.
normal_vector_field	A Paris field identifier specifying the vector field in CM memory containing the vector indicating the surface normal of the surface from which the light is to reflect. For `CMSR_v_reflect_2d` this vector field contains two floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. A 2D vector field is organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1). For `CMSR_v_reflect_3d` this vector field contains three floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. A 3D vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in *dest_vector_field, src_vector_field*, and *normal_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_vector_field, src_vector_field*, and *normal_vector_field.*

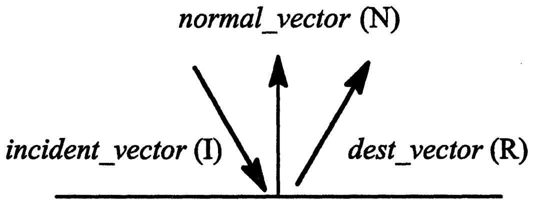
DESCRIPTION

For each active processor in the current VP set, **`CMSR_v_reflect_2d`** and **`CMSR_v_reflect_3d`** determine the vector resulting from reflecting *src_vector_field* around *normal_vector_field* and put the result in *dest_vector_field.* The vectors in *src_vector_field* and *normal_vector_field* need not be unit-length, but the reflected vectors in *dest_vector_field* will be.

All fields must be in the current VP set. *dest_vector_field* may be the same field as *src_vector_field,* or the fields may be totally disjoint. However, partially overlapping fields cause unpredictable results.

To build the destination vector in each processor, the incident and normal vectors are first normalized. The reflected vector (R), is then constructed from the unit-length incident vector (I) and unit-length normal vectors (N):

$$R = I - 2*(N \text{ dot } I)*N$$



CMSR_v_ref_x
CMSR_v_ref_y
CMSR_v_ref_z

Returns the field ID of an element of a 2D (3D) vector.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
CM_field_id_t
  CMSR_v_ref_x (vector_field, signif_len, exp_len)
CM_field_id_t
  CMSR_v_ref_y (vector_field, signif_len, exp_len)
CM_field_id_t
  CMSR_v_ref_z (vector_field, signif_len, exp_len)
CM_field_id_t  vector_field;
unsigned int   signif_len;
unsigned int   exp_len;
```

Fortran Syntax

```
INTEGER FUNCTION CMSR_V_REF_X (vector_field, signif_len, exp_len)
INTEGER FUNCTION CMSR_V_REF_Y (vector_field, signif_len, exp_len)
INTEGER FUNCTION CMSR_V_REF_Z (vector_field, signif_len, exp_len)
INTEGER  vector_field
INTEGER  signif_len
INTEGER  exp_len
```

Lisp Syntax

```
CMSR:v-ref-x (vector-field &optional (signif-len 23) (exp-len 8))
CMSR:v-ref-y (vector-field &optional (signif-len 23) (exp-len 8))
CMSR:v-ref-z (vector-field &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

vector_field	The Paris field ID of the vector from which the element is to be read.
signif_len	The length, in bits, of the significand of the floating-point values in the field.
exp_len	The length, in bits, of the exponent of the floating-point values in the field.

DESCRIPTION

`CMSR_v_ref_x`, **`CMSR_v_ref_y`** and **`CMSR_v_ref_z`** return the field ID of the subfield of *vector_field* containing, respectively, the *x*, *y*, or *z* element of the vector.

The length of each element is (*signif_len* + *exp_len* + 1) bits, where *signif_len* is the length of the coordinate significand, *exp_len* is the length of the coordinate exponent, and 1 is the sign bit. The vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits.

CMSR_v_scale_2d
CMSR_v_scale_3d

Multiplies a 2D (3D) vector field by a scaling factor field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_v_scale_2d
   (dest_vector_field, src_vector_field, scale_field, signif_len, exp_len)
CM_field_id_t  dest_vector_field, src_vector_field, scale_field;
unsigned int   signif_len, exp_len;

void
  CMSR_v_scale_3d
   (dest_vector_field, src_vector_field, scale_field, signif_len, exp_len)
CM_field_id_t  dest_vector_field, src_vector_field, scale_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-math-fort.h'
    SUBROUTINE CMSR_V_SCALE_2D
&      (dest_vector_field, src_vector_field, scale_field, signif_len, exp_len)
    INTEGER            dest_vector_field, src_vector_field, scale_field;
    INTEGER            signif_len, exp_len;

    SUBROUTINE CMSR_V_SCALE_3D
&      (dest_vector_field, src_vector_field, scale_field, signif_len, exp_len)
    INTEGER            dest_vector_field, src_vector_field, scale_field;
    INTEGER            signif_len, exp_len;
```

Lisp Syntax

```
CMSR:v-scale-2d (dest-vector-field src-vector-field scale-field
                &optional (signif-len 23) (exp-len 8))
CMSR:v-scale-3d (dest-vector-field src-vector-field
                &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_vector_field	A Paris field identifier specifying the vector field in CM memory in which the scaled vectors are to be stored.
src_vector_field	A Paris field identifier specifying the vector field in CM memory in which the vectors to be scaled are stored. For **`CMSR_v_scale_2d`** this vector field contains two floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. A 2D vector field is organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1). For **`CMSR_v_scale_3d`** this vector field contains three floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. A 3D vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).
scale_field	A Paris field identifier specifying the factors by which the vectors in *src_vector_field* are to be scaled.
signif_len	The length, in bits, of the significand of the floating-point values in *dest_vector_field* and *src_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_vector_field* and *src_vector_field.*

DESCRIPTION

For each active processor in the current VP set, **`CMSR_v_scale_2d`** and **`CMSR_v_scale_3d`** multiply each component of *src_vector_field* by *scale_field*, and place the result in *dest_vector_field.*

All fields must be in the current VP set. *dest_vector_field* may be the same field as *src_vector_field,* or the fields may be totally disjoint. However, partially overlapping fields cause unpredictable results.

CMSR_v_scale_const_2d
CMSR_v_scale_const_3d

Multiplies a 2D (3D) vector field by a scaling factor constant.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
  CMSR_v_scale_const_2d
      (dest_vector_field, src_vector_field, scale, signif_len, exp_len)
CM_field_id_t  dest_vector_field, src_vector_field;
double         scale;
unsigned int   signif_len, exp_len;

void
  CMSR_v_scale_const_3d
      (dest_vector_field, src_vector_field, scale, signif_len, exp_len)
CM_field_id_t  dest_vector_field, src_vector_field;
double         scale;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_V_SCALE_CONST_2D
&        (dest_vector_field, src_vector_field, scale, signif_len, exp_len)
   INTEGER            dest_vector_field, src_vector_field;
   DOUBLE PRECISION   scale;
   INTEGER            signif_len, exp_len;

   SUBROUTINE CMSR_V_SCALE_CONST_3D
&        (dest_vector_field, src_vector_field, scale, signif_len, exp_len)
   INTEGER            dest_vector_field, src_vector_field;
   DOUBLE PRECISION   scale;
   INTEGER            signif_len, exp_len;
```

Lisp Syntax

```
CMSR:v-scale-const-2d (dest-vector-field src-vector-field scale
                       &optional (signif-len 23) (exp-len 8))
CMSR:v-scale-const-3d (dest-vector-field src-vector-field scale
                       &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_vector_field	A Paris field identifier specifying the vector field in CM memory in which the scaled vectors are to be stored.
src_vector_field	A Paris field identifier specifying the vector field in CM memory in which the vectors to be scaled are stored. For `CMSR_v_scale_const_2d` this vector field contains two floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. A 2D vector field is organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1). For `CMSR_v_scale_const_3d` this vector field contains three floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. A 3D vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).
scale	The factor by which the vectors in *src_vector_field* are to be stored.
signif_len	The length, in bits, of the significand of the floating-point values in *dest_vector_field* and *src_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_vector_field* and *src_vector_field.*

DESCRIPTION

For each active processor in the current VP set, `CMSR_v_scale_const_2d` and `CMSR_v_scale_const_3d` multiply each component of *src_vector_field* by the constant value *scale*, and place the result in *dest_vector_field.*

Both fields must be in the current VP set. *dest_vector_field* may be the same field as *src_vector_field,* or the fields may be totally disjoint. However, partially overlapping fields cause unpredictable results.

CMSR_v_subtract_2d
CMSR_v_subtract_3d

Subtracts one 2D (3D) vector from another.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
  CMSR_v_subtract_2d
   (dest_vector_field, src1_vector_field, src2_vector_field, signif_len, exp_len)

CM_field_id_t  dest_vector_field, src1_vector_field, src2_vector_field;
unsigned int   signif_len, exp_len;

void
  CMSR_v_subtract_3d
   (dest_vector_field, src1_vector_field, src2_vector_field, signif_len, exp_len)

CM_field_id_t  dest_vector_field, src1_vector_field, src2_vector_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-math-fort.h'
    SUBROUTINE CMSR_V_SUBTRACT_2D
&       (dest_vector_field, src1_vector_field, src2_vector_field, signif_len, exp_len)
    INTEGER dest_vector_field, src1_vector_field, src2_vector_field
    INTEGER signif_len, exp_len

    SUBROUTINE CMSR_V_SUBTRACT_3D
&       (dest_vector_field, src1_vector_field, src2_vector_field, signif_len, exp_len)
    INTEGER dest_vector_field, src1_vector_field, src2_vector_field
    INTEGER signif_len, exp_len
```

Lisp Syntax

`CMSR:v-subtract-2d` (*dest–vector–field src1–vector–field src2–vector–field* **`&optional`** (*signif–len* `23`) (*exp–len* `8`))

`CMSR:v-subtract-3d` (*dest–vector–field src1–vector–field src2–vector–field* **`&optional`** (*signif–len* `23`) (*exp–len* `8`))

ARGUMENTS

dest_vector_field A Paris field identifier specifying the vector field in CM memory in which the results of the subtraction are to be stored.

src1_vector_field, src2_vector_field,

Paris field identifiers specifying the vector field in CM memory in which the vectors to be subtracted are stored.

For **`CMSR_v_subtract_2d`** this vector field contains two floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. A 2D vector field is organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).

For **`CMSR_v_subtract_3d`** this vector field contains three floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. A 3D vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).

signif_len The length, in bits, of the significand of the floating-point values in *dest_vector_field* and *src_vector_field.*

exp_len The length, in bits, of the exponent of the floating-point values in *dest_vector_field* and *src_vector_field.*

DESCRIPTION

For each active processor in the current VP set, **`CMSR_v_subtract_2d`** and **`CMSR_v_subtract_3d`** subtract each element of *src2_vector_field* from *src1_vector_field* (*src1_vector_field – src2_vector_field*) and put the result in *dest_vector.*

Both fields must be in the current VP set. *dest_vector_field* may be the same field as *src_vector_field,* or the fields may be totally disjoint. However, partially overlapping fields cause unpredictable results.

CMSR_v_transform_2d
CMSR_v_transform_3d

Applies a 2D (3D) transformation matrix field to a vector field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
   CMSR_v_transform_2d
    (dest_vector_field, src_vector_field, src_matrix_field, signif_len, exp_len)

void
   CMSR_v_transform_3d
    (dest_vector_field, src_vector_field, src_matrix_field, signif_len, exp_len)

CM_field_id_t  dest_vector_field;
CM_field_id_t  src_vector_field;
CM_field_id_t  src_matrix_field;
unsigned int   signif_len;
unsigned int   exp_len;
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-math-fort.h'

    SUBROUTINE CMSR_V_TRANSFORM_2d
&       (dest_vector_field, src_vector_field, src_matrix_field, signif_len, exp_len)

    SUBROUTINE CMSR_V_TRANSFORM_3d
&       (dest_vector_field, src_vector_field, src_matrix_field, signif_len, exp_len)

    INTEGER     dest_vector_field
    INTEGER     src_vector_field
    INTEGER     src_matrix_field
    INTEGER     signif_len
    INTEGER     exp_len
```

Lisp Syntax

```
CMSR:v-transform-2d  (dest-vector-field src-vector-field src-matrix-field
                         &optional (signif-len 23) (exp-len 8))
CMSR:v-transform-3d  (dest-vector-field src-vector-field matrix-field
                         &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_vector_field	A Paris field identifier specifying the field in CM memory to which the transformed vector is written. *dest_vector_field* must be in the same VP set as *src_vector_field* and *src_matrix_field.*
src_vector_field	A Paris field identifier specifying the field in CM memory from which the vector to be transformed is taken. *src_vector_field* must be in the same VP set as *dest_vector_field* and *src_matrix_field.* The vector elements are floating-point values each having a length of (*signif_len* + *exp_len* + 1) bits, where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. For **CMSR_v_transform_2d** the vector field is organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1). For **CMSR_v_transform_3d** the vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).
src_matrix_field	A Paris field identifier specifying the field in CM memory containing the homogeneous transformation matrix to be applied to the source vector. *src_matrix_field* must be in the same VP set as *src_vector_field* and *dest_vector_field.* Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. For **CMSR_v_transform_2d** the matrix field contains a 3 x 3 homogeneous transformation matrix stored in row-major order. The length of the entire field is 9 * (*signif_len* + *exp_len* + 1).

For `CMSR_v_transform_3d` the matrix field contains a 4 x 4 homogeneous transformation matrix stored in row-major order. The length of the entire field is 16 * (*signif_len* + *exp_len* + 1).

signif_len	The length, in bits, of the significand of the floating-point values in *src_vector_field, dest_vector_field*, and *src_matrix_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *src_vector_field, dest_vector_field*, and *src_matrix_field.*

DESCRIPTION

For each active processor, `CMSR_v_transform_2d` and `CMSR_v_transform_3d` apply the transformation defined by the matrix in *src_matrix_field* to the vector in *src_vector_field* and write the transformed vector into *dest_vector_field.*

Note that the matrix is a homogeneous transformation, (3 x 3 for 2D and 4 x 4 for 3D), but the vectors do not support homogeneous coordinates. That is, the 2D vector has only two elements ([*xy*] instead of [*xyw*]), and the 3D vector has only 3 elements ([*xyz*] instead of [*xyzw*]). This means that the resulting vector is actually of the form *x*/*w*, *y*/*w*, and, if 3D *z*/*w*.

All fields must be in the current VP set. The *dest_vector_field* may be the same field as *src_vector_field,* or the two fields may be totally disjoint. However, partially overlapping fields cause unpredictable results.

SEE ALSO

`CMSR_v_transform_const_2d`

`CMSR_v_transform_const_3d`

`CMSR_fe_v_transform_2d`

`CMSR_fe_v_transform_3d`

CMSR_v_transform_const_2d
CMSR_v_transform_const_3d

Applies a single 2D (3D) transformation matrix to a 2D vector field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
  CMSR_v_transform_const_2d
          (dest_vector_field, src_vector_field, src_matrix, signif_len, exp_len)

CM_field_id_t  dest_vector_field;
CM_field_id_t  src_vector_field;
double         src_matrix[3][3];
unsigned int   signif_len, exp_len;

void
  CMSR_v_transform_const_3d
          (dest_vector_field, src_vector_field, src_matrix, signif_len, exp_len)

CM_field_id_t  dest_vector_field;
CM_field_id_t  src_vector_field;
double         src_matrix[4][4];
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'

   SUBROUTINE CMSR_V_TRANSFORM_CONST_2D
&             (dest_vector_field, src_vector_field, src_matrix, signif_len, exp_len)

   INTEGER             dest_vector_field
   INTEGER             src_vector_field
   DOUBLE PRECISION    src_matrix(3,3)
   INTEGER             signif_len
   INTEGER             exp_len

   SUBROUTINE CMSR_V_TRANSFORM_CONST_3D
&          (dest_vector_field, src_vector_field, src_matrix, signif_len, exp_len)
```

```
INTEGER            dest_vector_field
INTEGER            src_vector_field
DOUBLE PRECISION   src_matrix(4,4)
INTEGER            signif_len
INTEGER            exp_len
```

Lisp Syntax

```
CMSR:v-transform-const-2d
                         (dest-vector-field src-vector-field src-matrix
                              &optional (signif-len 23) (exp-len 8))
CMSR:v-transform-const-3d
                         (dest-vector-field src-vector-field src-matrix
                              &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_vector_field	A Paris field identifier specifying the field in CM memory to which the transformed vector is written. *dest_vector_field* must be in the same VP set as *src_vector_field.*
src_vector_field	A Paris field identifier specifying the field in CM memory from which the vector to be transformed is taken. *src_vector_field* must be in the same VP set as *dest_vector_field.* The vector elements are floating-point values each having a length of (*signif_len* + *exp_len* + 1) bits, where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. For **`CMSR_v_transform_const_2d`** the vector field is organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1). For **`CMSR_v_transform_const_3d`** the vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).
src_matrix	For **`CMSR_v_transform_const_2d`** a 3 x 3 homogeneous transformation matrix, or for **`CMSR_v_transform_const_3d`** a 4 x 4

	matrix, stored on the front-end computer in row-major order. Each element of the matrix is a double-precision floating-point value.
signif_len	The length, in bits, of the significand of the floating-point values in *src_vector_field,* and *dest_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *src_vector_field,* and *dest_vector_field.*

DESCRIPTION

`CMSR_v_transform_const_2d` applies the transformation defined by *src_matrix* to the vector in *src_vector_field* in each active processor and writes the transformed vector into *dest_vector_field.*

Note that the matrix is a homogeneous transformation, (3 x 3 for 2D and 4 x 4 for 3D), but the vectors do not support homogeneous coordinates. That is, the 2D vector has only two elements ([*xy*] instead of [*xyw*]), and the 3D vector has only 3 elements ([*xyz*] instead of [*xyzw*]). This means that the resulting vector is actually of the form *x/w*, *y/w*, and, if 3D *z/w*.

All fields must be in the current VP set. The *dest_vector_field* may be the same field as *src_vector_field,* or the two fields may be totally disjoint. However, partially overlapping fields cause unpredictable results.

SEE ALSO

`CMSR_v_transform_2d`

`CMSR_v_transform_3d`

`CMSR_fe_v_transform_2d`

`CMSR_fe_v_transform_3d`

CMSR_v_transmit_3d

Creates a transmittance vector for light refracted through two materials.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
  CMSR_v_transmit_3d
      (transmitted_vector_field, incident_vector_field, normal_vector_field,
      index1, index2, signif_len, exp_len)

CM_field_id_t  transmitted_vector_field, incident_vector_field;
CM_field_id_t  normal_vector_field;
CM_field_id_t  index1, index2;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'

   SUBROUTINE CMSR_V_TRANSMIT_3D
&      (transmitted_vector_field, incident_vector_field, normal_vector_field,
&        index1, index2, signif_len, exp_len)

   INTEGER transmitted_vector_field, incident_vector_field, normal_vector_field
   INTEGER index1, index2
   INTEGER signif_len, exp_len
```

Lisp Syntax

```
CMSR:v-transmit-3d
      (transmitted-vector-field incident-vector-field normal-vector-field
      index1 index2 &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

transmitted_vector_field	A Paris field identifier specifying the vector field in CM memory in which vectors indicating the direction of the transmitted light is stored.
incident_vector_field	A Paris field identifier specifying the vector field in CM memory containing the vectors indicating the direction of the incident light.
normal_vector_field	A Paris field identifier specifying the vector field in CM memory containing the vectors indicating the surface normal of the surface through which the light is to pass. For `CMSR_v_transmit_2d` this vector field contains two floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. A 2D vector field is organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1). For `CMSR_v_transmit_3d` this vector field contains three floating-point values, each having a length of (*signif_len* + *exp_len* + 1) bits. A 3D vector field is organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1).
index1, *index2*	The index of refraction of the medium containing *incident_vector* and *transmitted_vector*, respectively.
signif_len	The length, in bits, of the significand of the floating-point values in *transmitted_vector_field, incident_vector_field*, and *normal_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *transmitted_vector_field, incident_vector_field,* and *normal_vector_field.*

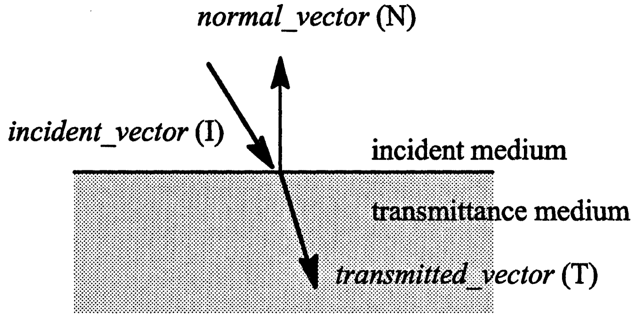
DESCRIPTION

For each active processor in the current VP set, **`CMSR_v_transmit_2d`** and **`CMSR_v_transmit_3d`** calculate the vector of the transmitted light based on the *incident_vector_field*, *normal_vector_field*, and indices of refraction for the two materials, *index1*, and *index2*. The result of this calculation in each processor is stored in the *transmitted_vector_field*. All fields must be in the current VP set.

The unit length *transmitted_vector* T is given by:

T = (n1/n2)*I+((n1/n2)*(N dot –I)–sqrt(1+(n1/n2)^2*((N dot –I)^2)–1))*N

N is the unit vector in the direction of *normal_vector*. I is the unit vector in the direction of *incident_vector*. n1 is *index1*, the index of refraction of the medium containing the incident vector. n2 is *index2*, the index of refraction of the medium which contains the transmitted vector.



If the expression under the square root becomes negative, the result is total internal reflection. If total internal reflection occurs, **`CMSR_v_transmit_3d`** returns a NULL pointer.

Neither the incident vector nor the normal vector should be length 0. The second index of refraction should not be 0.

CMSR_v_write_to_processor_2d
CMSR_v_write_to_processor_3d

Writes a 2D (3D) vector from a front-end array into a specified processor's vector field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
  CMSR_v_write_to_processor_2d
                (processor, dest_vector_field, src_vector, signif_len, exp_len)

unsigned int  processor;
CM_field_id_t dest_vector_field;
double        src_vector[2];
unsigned int  signif_len, exp_len;

void
  CMSR_v_write_to_processor_3d
                (processor, dest_vector_field, src_vector, signif_len, exp_len)

unsigned int  processor;
CM_field_id_t dest_vector_field;
double        src_vector[3];
unsigned int  signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'

   SUBROUTINE CMSR_V_WRITE_TO_PROCESSOR_2D
&               (processor, dest_vector_field, src_vector, signif_len, exp_len)

   INTEGER          processor
   INTEGER          dest_vector_field
   DOUBLE PRECISION src_vector(2)
   INTEGER          signif_len, exp–len
```

```
   SUBROUTINE CMSR_V_WRITE_TO_PROCESSOR_3D
&                  (processor, dest_vector_field, src_vector, signif_len, exp_len)
   INTEGER            processor
   INTEGER            dest_vector_field
   DOUBLE PRECISION   src_vector(3)
   INTEGER            signif_len
   INTEGER            exp-len
```

Lisp Syntax

```
CMSR:v-write-to-processor-2d
                          (processor vector-field src-vector
                           &optional (signif-len 23)(exp-len 8))
CMSR:v-write-to-processor-3d
                          (processor vector-field src-vector
                           &optional (signif-len 23)(exp-len 8))
```

ARGUMENTS

processor	The processor send address to which the vector is to be sent.
dest_vector_field	A Paris field indentifier specifying the field in *processor* in which the 2D vector is to be stored. Each element has a length of (*signif_len* + *exp_len* + 1) bits. For **`CMSR_v-write-to-processor-2d`** the vector field contains two floating-point values organized so that *x* occupies the least significant bits and *y* the most significant bits. The length of the entire field is 2 * (*signif_len* + *exp_len* + 1). For **`CMSR_v_write_to_processor_3d`** the vector field contains three floating-point values organized so that *x* occupies the least significant bits, *y* the following bits, and *z* the most significant bits. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).
src_vector	For **`CMSR_v-write-to-processor-2d`** a 1 x 2 array, or for **`CMSR_v_write_to_processor_3d`** a 1 x 3 array, containing the vector to be loaded into the vector field.
signif_len	The length, in bits, of the significand of the floating-point values in *dest_vector_field.*

exp_len	The length, in bits, of the exponent of the floating-point values in *dest_vector_field*.

DESCRIPTION

`CMSR_v-write-to-processor-2d` and **`CMSR_v_write_to_processor_3d`** place the contents of *src_vector* in *dest_vector_field* in *processor*.

SEE ALSO

`CMSR_v_read_from_processor_2d`

`CMSR_v_read_from_processor_3d`

3.7 CM Matrix Routines

This section documents the *Render routines that operate on matrices in Paris fields in CM memory. Matrices in *Render are assumed to be square, homogeneous matrices. *Render supports matrices of dimension 2 or 3, for transforming two-dimensional or three-dimensional vectors.

On the CM, a matrix is a field of (dimension + 1) * (dimension + 1)(*signif_len* + *exp_len* +1) bits, where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and the 1 is for a sign bit. Each matrix element occupies one floating-point field of (*signif_len* + *exp_len* +1) bits.

The routines documented here are:

CMSR_identity_matrix_2d
CMSR_identity_matrix_3d

Loads a 2D (3D) identity matrix into a matrix field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_identity_matrix_2d (dest_matrix_field, signif_len, exp_len)
void
  CMSR_identity_matrix_3d (dest_matrix_field, signif_len, exp_len)
CM_field_id_t  dest_matrix_field;
unsigned int   signif_len;
unsigned int   exp_len;
```

Fortran Syntax

```
      INCLUDE '/usr/include/cm/cmsr-math-fort.h'
      SUBROUTINE CMSR_IDENTITY_MATRIX_2D
     &                           (dest_matrix_field, signif_len, exp_len)
      SUBROUTINE CMSR_IDENTITY_MATRIX_3D
     &                           (dest_matrix_field, signif_len, exp_len)
      INTEGER dest_matrix_field
      INTEGER signif_len
      INTEGER exp_len
```

Lisp Syntax

```
CMSR:identity-matrix-2d (dest-matrix-field
                           &optional (signif-len 23) (exp-len 8))
CMSR:identity-matrix-3d (dest-matrix-field
                           &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_matrix_field	A Paris field identifier specifying the field in CM memory into which the identity matrix is to be written. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. For **`CMSR_identity_matrix_2d`**, *dest_matrix_field* accepts a 3 x 3 homogeneous transformation matrix stored in row-major order. The length of the entire field is 9 * (*signif_len* + *exp_len* + 1). For **`CMSR_identity_matrix_3d`** the matrix field accepts a 4 x 4 homogeneous transformation matrix stored in row-major order. The length of the entire field is 16 * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in *dest_matrix_field*.
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_matrix_field*.

DESCRIPTION

`CMSR_identity_matrix_2d` and **`CMSR_identity_matrix_3d`** write a 2D or 3D identity matrix, respectively, into *dest_matrix_field* in each active processor in the current VP set.

SEE ALSO

`CMSR_fe_identity_matrix_2d`

`CMSR_fe_identity_matrix_3d`

CMSR_m_alloc_heap_field_2d
CMSR_m_alloc_heap_field_3d

Allocate a 2D (3D) matrix field in the heap memory space and return its Paris field ID.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
CM_field_id_t
  CMSR_m_alloc_heap_field_2d (signif_len, exp_len)
unsigned int  signif_len, exp_len;

CM_field_id_t
  CMSR_m_alloc_heap_field_3d (signif_len, exp_len)
unsigned int  signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   INTEGER FUNCTION CMSR_M_ALLOC_HEAP_FIELD_2D
&                                      (signif_len, exp_len)
   INTEGER  signif_len, exp_len

   INTEGER FUNCTION CMSR_M_ALLOC_HEAP_FIELD_3D
&                                      (signif_len, exp_len)
   INTEGER  signif_len, exp_len
```

Lisp Syntax

```
CMSR:m-alloc-heap-field-2d (signif-len exp-len)
CMSR:m-alloc-heap-field-3d (signif-len exp-len)
```

ARGUMENTS

signif_len	The length, in bits, of the significand of the floating-point values to be stored in the returned field.
exp_len	The length, in bits, of the exponent of the floating-point values to be stored in the returned field.

DESCRIPTION

`CMSR_m_alloc_heap_field_2d` and **`CMSR_m_alloc_heap_field_3d`** allocate a matrix in the heap memory space and return its Paris field ID.

A matrix field is organized to contain a square homogeneous transformation matrix stored in row-major order. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.

`CMSR_m_alloc_heap_field_2d` allocates a field for a 3 x 3 matrix. The length of the entire field is (3 * 3) * (*signif_len* + *exp_len* + 1).

`CMSR_m_alloc_heap_field_3d` allocates a field for a 4 x 4 matrix. The length of the entire field is (4 * 4) * (*signif_len* + *exp_len* + 1).

CMSR_m_alloc_stack_field_2d
CMSR_m_alloc_stack_field_3d

Allocates a 2D (3D) matrix field on the stack memory space and return its Paris field ID.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
CM_field_id_t
  CMSR_m_alloc_stack_field_2d (signif_len, exp_len)
unsigned int  signif_len, exp_len;

CM_field_id_t
  CMSR_m_alloc_stack_field_3d (signif_len, exp_len)
unsigned int  signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   INTEGER FUNCTION CMSR_M_ALLOC_STACK_FIELD_2D
&                                   (signif_len, exp_len)
   INTEGER  signif_len, exp_len

   INTEGER FUNCTION CMSR_M_ALLOC_STACK_FIELD_3D
&                                   (signif_len, exp_len)
   INTEGER  signif_len, exp_len
```

Lisp Syntax

```
CMSR:m-alloc-stack-field-2d (signif-len exp-len)
CMSR:m-alloc-stack-field-3d (signif-len exp-len)
```

ARGUMENTS

signif_len	The length, in bits, of the significand of the floating-point values to be stored in the returned field.
exp_len	The length, in bits, of the exponent of the floating-point values to be stored in the returned field.

DESCRIPTION

`CMSR_m_alloc_stack_field_2d` and **`CMSR_m_alloc_stack_field_3d`** allocate a matrix on the stack and return its Paris field ID.

A matrix field is organized to contain a square homogeneous transformation matrix stored in row-major order. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.

`CMSR_m_alloc_stack_field_2d` allocates a field for a 3 x 3 matrix. The length of the entire field is (3 * 3) * (*signif_len* + *exp_len* + 1).

`CMSR_m_alloc_stack_field_3d` allocates a field for a 4 x 4 matrix. The length of the entire field is (4 * 4) * (*signif_len* + *exp_len* + 1).

CMSR_m_copy_2d
CMSR_m_copy_3d

Copies one 2D (3D) matrix field to another.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
   CMSR_m_copy_2d(dest_matrix_field, src_matrix_field, signif_len, exp_len)
CM_field_id_t  dest_matrix_field, src_matrix_field;
unsigned int   signif_len, exp_len;

void
   CMSR_m_copy_3d(dest_matrix_field, src_matrix_field, signif_len, exp_len)
CM_field_id_t  dest_matrix_field, src_matrix_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'

   SUBROUTINE CMSR_M_COPY_2D
&                    (dest_matrix_field, src_matrix_field, signif_len, exp_len)
   INTEGER  dest_matrix_field, src_matrix_field, signif_len, exp_len

   SUBROUTINE CMSR_M_COPY_3D
&                    (dest_matrix_field, src_matrix_field, signif_len, exp_len)
   INTEGER  dest_matrix_field, src_matrix_field, signif_len, exp_len
```

Lisp Syntax

```
CMSR:m-copy-2d  (dest-matrix-field src-matrix-field
                 &optional (signif-len 23) (exp-len 8))
CMSR:m-copy-3d  (dest-matrix-field src-matrix-field
                 &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_matrix_field	A Paris field identifier specifying the matrix field in CM memory into which the *src_matrix_field* is to be copied. *dest_matrix_field* must be in the same VP set as *src_matrix_field.*
src_matrix_field	A Paris field indentifier specifying the matrix field in CM memory to be copied into *dest_matrix_field. src_matrix_field* must be in the same VP set as *dest_matrix_field.* For **`CMSR_m_copy_2d`** these matrix fields contain a 3 x 3 homogeneous transformation matrix stored in row-major order. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The length of the entire field is (3 * 3) * (*signif_len* + *exp_len* + 1). For **`CMSR_m_copy_3d`** these matrix fields contain a 4 x 4 homogeneous transformation matrix stored in row-major order. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The length of the entire field is (4 * 4) * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in *dest_matrix_field* and *src_matrix_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_matrix_field* and *src_matrix_field.*

DESCRIPTION

For each active processor in the current VP set, **`CMSR_m_copy_2d`** and **`CMSR_m_copy_3d`** copy *src_matrix_field* to *dest_matrix_field.* Both fields must be in the current VP set.

CMSR_m_copy_const_2d
CMSR_m_copy_const_3d

Broadcasts a front-end 2D (3D) matrix into a CM matrix field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_m_copy_const_2d
                        (dest_matrix_field, src_matrix, signif_len, exp_len)
CM_field_id_t  dest_matrix_field;
double         src_matrix[3][3];
unsigned int   signif_len, exp_len;

void
  CMSR_m_copy_const_3d
                        (dest_matrix_field, src_matrix, signif_len, exp_len)
CM_field_id_t  dest_matrix_field;
double         src_matrix[4][4];
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_M_COPY_CONST_2D
&                      (dest_matrix_field, src_matrix, signif_len, exp_len)
   INTEGER            dest_matrix_field
   DOUBLE PRECISION   src_matrix(3,3)
   INTEGER            signif_len, exp_len

   SUBROUTINE CMSR_M_COPY_CONST_3D
&                      (dest_matrix_field, src_matrix, signif_len, exp_len)
   INTEGER            dest_matrix_field
   DOUBLE PRECISION   src_matrix(4,4)
   INTEGER            signif_len, exp_len
```

Lisp Syntax

```
CMSR:m-copy-const-2d (dest-matrix-field src-matrix
                      &optional (signif-len 23) (exp-len 8))

CMSR:m-copy-const-3d (dest-matrix-field src-matrix
                      &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_matrix_field	A Paris field identifier specifying the matrix field in CM memory into which the *src_matrix* is to be copied. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. For **CMSR_m_copy_2d** this matrix field contains a 3 x 3 homogeneous transformation matrix stored in row-major order. The length of the entire field is (3 * 3) * (*signif_len* + *exp_len* + 1). For **CMSR_m_copy_3d** this matrix field contains a 4 x 4 homogeneous transformation matrix stored in row-major order. The length of the entire field is (4 * 4) * (*signif_len* + *exp_len* + 1).
src_matrix	A two-dimensional front-end array containing the matrix of floating-point values to be broadcast to *dest_matrix_field.* For **CMSR_m_copy_2d**, *src_matrix* is a 3 x 3 array. For **CMSR_m_copy_3d**, *src_matrix* is a 4 x 4 array.
signif_len	The length, in bits, of the significand of the floating-point values in *dest_matrix_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_matrix_field.*

DESCRIPTION

CMSR_m_copy_const_2d and **CMSR_m_copy_const_3d** copy the front-end matrix *src_matrix* into *dest_matrix_field* in each active processor in the current VP set.

CMSR_m_determinant_2d
CMSR_m_determinant_3d

Calculates the determinant of each 2D (3D) matrix stored in a matrix field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
   CMSR_m_determinant_2d  (determ_field, matrix_field, signif_len, exp_len)
void
   CMSR_m_determinant_3d  (determ_field, matrix_field, signif_len, exp_len)
CM_field_id_t determ_field,  matrix_field;
unsigned int signif_len,  exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_M_DETERMINANT_2D
&                                (determ_field, matrix_field, signif_len, exp_len)
   SUBROUTINE CMSR_M_DETERMINANT_3D
&                                (determ_field, matrix_field, signif_len, exp_len)
   INTEGER  determ_field,  matrix_field
   INTEGER  signif_len,  exp_len
```

Lisp Syntax

```
CMSR:m-determinant-2d  (determ-field, matrix-field
                        &optional  (signif-len 23)  (exp-len 8))
CMSR:m-determinant-3d  (determ-field, matrix-field
                        &optional  (signif-len 23)  (exp-len 8))
```

ARGUMENTS

determ_field	A Paris field identifier specifying the field in CM memory into which the matrix determinant is to be stored. The length of the field is (*signif_len* + *exp_len* + 1) bits, where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.
matrix_field	A Paris field identifier specifying the matrix field in CM memory containing the matrices for which the determinant is to be calculated. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. For **`CMSR_m_determinant_2d`** this matrix field contains a 3 x 3 homogeneous transformation matrix stored in row-major order. The length of the entire field is 9 * (*signif_len* + *exp_len* + 1). For **`CMSR_m_determinant_3d`** this matrix field contains a 4 x 4 homogeneous transformation matrix stored in row-major order. The length of the entire field is 16 * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in *determ_field* and *matrix_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *determ_field* and *matrix_field.*

DESCRIPTION

For each active processor in the current VP set, **`CMSR_m_determinant_2d`** and **`CMSR_m_determinant_3d`** calculate the determinant of the matrix stored in *matrix_field* and place it in *determ_field.*

CMSR_m_field_length

Returns the length of a matrix field of a specified rank.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
int
CMSR_m_field_length (rank, signif_len, exp_len)
int rank;
unsigned int signif_len;
unsigned int exp_len;
```

Fortran Syntax

```
INTEGER FUNCTION CMSR_M_FIELD_LENGTH (rank signif_len exp_len)
INTEGER rank;
INTEGER signif_len;
INTEGER exp_len;
```

Lisp Syntax

```
CMSR:m-field-length (rank &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

rank	The number of dimensions in the matrix for which the field is allocated.
signif_len	The length, in bits, of the significand of the floating-point values in the field.
exp_len	The length, in bits, of the exponent of the floating-point values in the field.

DESCRIPTION

`CMSR_m_field_length` returns the length, in bits, of the Paris field that must be allocated to hold a matrix of (1 + *rank*) elements square.

SEE ALSO

`CMSR_v_field_length`

CMSR_m_invert_2d
CMSR_m_invert_3d

Calculates the inverse of each 2D (3D) matrix in the matrix field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_m_invert_2d
                  (dest_matrix_field, src_matrix_field, signif_len, exp_len)
void
  CMSR_m_invert_3d
                  (dest_matrix_field, src_matrix_field, signif_len, exp_len)

CM_field_id_t  dest_matrix_field, src_matrix_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_M_INVERT_2D
&                   (dest_matrix_field, src_matrix_field, signif_len, exp_len)
   SUBROUTINE CMSR_M_INVERT_3D
&                   (dest_matrix_field, src_matrix_field, signif_len, exp_len)

   INTEGER  dest_matrix_field, src_matrix_field;
   INTEGER  signif_len, exp_len;
```

Lisp Syntax

```
CMSR:m-invert-2d (dest-matrix-field src-matrix-field
                  &optional (signif-len 23) (exp-len 8))
CMSR:m-invert-3d (dest-matrix-field src-matrix-field
                  &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_matrix_field	A Paris field identifier specifying the matrix field in CM memory in which the inverted matrices are to be stored. *dest_matrix_field* must be in the same VP set as *src_matrix_field.*
src_matrix_field	A Paris field indentifier specifying the matrix field in CM memory containing the matrices to be inverted. *src_matrix_field* must be in the same VP set as *dest_matrix_field.* Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The length of the entire field is (4 * 4) * (*signif_len* + *exp_len* + 1). For **`CMSR_m_copy_2d`** these matrix fields contain a 3 x 3 homogeneous transformation matrix stored in row-major order. For **`CMSR_m_copy_3d`** these matrix fields contain a 4 x 4 homogeneous transformation matrix stored in row-major order.
signif_len	The length, in bits, of the significand of the floating-point values in *dest_matrix_field* and *src_matrix_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_matrix_field* and *src_matrix_field.*

DESCRIPTION

For each active processor in the current VP set, **`CMSR_m_invert_2d`** and **`CMSR_m_invert_3d`** calculate the inverse of *src_matrix_field* and store the result in *dest_matrix_field*. Both fields must be in the current VP set. *dest_matrix_field* may be the same field as *src_matrix_field,* or the fields may be totally disjoint. However, partially overlapping fields cause unpredictable results.

NOTE: If *src_matrix_field* in some processor is singular (that is, if the matrix determinant is zero) the contents of *dest_matrix_field* for that processor are undefined.

CMSR_m_multiply_2d
CMSR_m_multiply_3d

Multiplies two 2D (3D) matrices.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
  CMSR_m_multiply_2d
                    (dest_matrix_field, src1_matrix_field, src2_matrix_field,
                     signif_len., exp_len)

void
  CMSR_m_multiply_3d
                    (dest_matrix_field, src1_matrix_field, src2_matrix_field,
                    signif_len, exp_len)

CM_field_id_t  dest_matrix_field;
CM_field_id_t  src1_matrix_field;
CM_field_id_t  src2_matrix_field;
unsigned int   signif_len;
unsigned int   exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'

   SUBROUTINE CMSR_M_MULTIPLY_2D
&  (dest_matrix_field, src1_matrix_field, src2_matrix_field, signif_len, exp_len)

   SUBROUTINE CMSR_M_MULTIPLY_3D
&  (dest_matrix_field, src1_matrix_field, src2_matrix_field, signif_len, exp_len)

   INTEGER  dest_matrix_field
   INTEGER  src1_matrix_field
   INTEGER  src2_matrix_field
   INTEGER  signif_len
   INTEGER  exp_len
```

Lisp Syntax

```
CMSR:m-multiply-2d(dest-matrix-field src1-matrix-field src2-matrix-field
                   &optional (signif-len 23) (exp-len 8))
CMSR:m-multiply-3d(dest-matrix-field src1-matrix-field src2-matrix-field
                   &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_matrix_field A Paris field identifier specifying the field in CM memory into which the resulting 2D matrix is to be written. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.

The 2D matrix field contains a 3 x 3 homogeneous transformation matrix stored in row-major order. The length of the entire field is 9 * (*signif_len* + *exp_len* + 1).

The 3D matrix field contains a 4 x 4 homogeneous transformation matrix stored in row-major order. The length of the entire field is 16 * (*signif_len* + *exp_len* + 1).

dest_matrix_field must be in the same VP set as *src1_matrix_field* and *src2_matrix_field.*

src1_matrix_field, src2_matrix_field

Paris field indentifiers specifying the fields in CM memory containing the 2D matrices to be multiplied. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.

The 2D matrix field contains a 3 x 3 homogeneous transformation matrix stored in row-major order. The length of the entire field is 9 * (*signif_len* + *exp_len* + 1).

The 3D matrix field contains a 4 x 4 homogeneous transformation matrix stored in row-major order. The length of the entire field is 16 * (*signif_len* + *exp_len* + 1).

src1_matrix_field and *src2_matrix_field* must be in the same VP set as *dest_matrix_field.*

signif_len	The length, in bits, of the significand of the floating-point values in *dest_matrix_field*, *src1_matrix_field*, and *src2_matrix_field*.
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_matrix_field*, *src1_matrix_field*, and *src2_matrix_field*.

DESCRIPTION

For each active processor, **`CMSR_m_multiply_2d`** and **`CMSR_m_multiply_3d`** calculate the product of the two matrices (*src_matrix_field_1* * *src_matrix_field_2*) and write the result in *dest_matrix_field*.

All fields must be in the current VP set. The *dest_matrix_field* and *src_matrix_field_1* may be the same field or totally disjoint fields, but the fields must not overlap. Partial overlap may cause unpredictable results.

SEE ALSO

`CMSR_m_multiply_const_2d`

`CMSR_m_multiply_const_3d`

`CMSR_fe_m_multiply_2d`

`CMSR_fe_m_multiply_3d`

CMSR_m_multiply_const_2d
CMSR_m_multiply_const_3d

Multiplies a single 2D (3D) matrix with a 2D (3D) matrix field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
   CMSR_m_multiply_const_2d
            (dest_matrix_field, src_matrix_field, matrix, signif_len exp_len)

CM_field_id_t  dest_matrix_field, src_matrix_field;
double         matrix[3][3];
unsigned int   signif_len, exp_len;


void
   CMSR_m_multiply_const_3d
            (dest_matrix_field, src_matrix_field, matrix, signif_len exp_len)

CM_field_id_t  dest_matrix_field, src_matrix_field;
double         matrix[4][4];
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'

CMSR_M_MULTIPLY_CONST_2D
            (dest_matrix_field, src_matrix_field, matrix, signif_len, exp_len)

INTEGER            dest_matrix_field, src_matrix_field
DOUBLE PRECISION   matrix (3)(3)
INTEGER            signif_len, exp_len


CMSR_M_MULTIPLY_CONST_3D
            (dest_matrix_field, src_matrix_field, matrix, signif_len exp_len)

INTEGER            dest_matrix_field, src_matrix_field
DOUBLE PRECISION   matrix (4)(4)
INTEGER            signif_len, exp_len
```

Lisp Syntax

```
CMSR:m-multiply-const-2d(dest-matrix-field src-matrix-field matrix
                          &optional (signif-len 23) (exp-len 8))
CMSR:m-multiply-const-3d(dest-matrix-field src-matrix-field matrix
                          &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_matrix_field	A Paris field identifier specifying the field in CM memory to which the result is written. *dest_matrix_field* must be in the same VP set as *src_matrix_field.* Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The 2D matrix field contains a 3 x 3 homogeneous transformation matrix stored in row-major order. The length of the entire field is 9 * (*signif_len* + *exp_len* + 1). The 3D matrix field contains a 4 x 4 homogeneous transformation matrix stored in row-major order. The length of the entire field is 16 * (*signif_len* + *exp_len* + 1).
src_matrix_field	A Paris field identifier specifying the field in CM memory from which the matrix to be transformed is taken. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The 2D matrix field contains a 3 x 3 homogeneous transformation matrix stored in row-major order. The length of the entire field is 9 * (*signif_len* + *exp_len* + 1). The 3D matrix field contains a 4x4 homogeneous transformation matrix stored in row-major order. The length of the entire field is 16 * (*signif_len* + *exp_len* + 1).
matrix	A 3 x 3 homogeneous transformation matrix stored on the front-end computer in row-major order. Each element of the matrix is a double-precision floating-point value.
signif_len	The length, in bits, of the significand of the floating-point values in *src_matrix_field* and *dest_matrix_field.*

exp_len	The length, in bits, of the exponent of the floating-point values in *src_matrix_field* and *dest_matrix_field.*

DESCRIPTION

`CMSR_m_multiply_const_2d` and **`CMSR_m_multiply_const_3d`** multiply the transformation matrix *matrix* with the matrix in each active processor in *src_matrix_field* and store the result in *dest_matrix_field.*

All fields must be in the current VP set. The *dest_matrix_field* may be the same field as *src_matrix_field,* or the fields may be totally disjoint. However, partially overlapping fields cause unpredictable results.

SEE ALSO

`CMSR_m_multiply_2d`

`CMSR_m_multiply-3d`

`CMSR_fe_m_multiply_2d`

`CMSR_fe_m_multiply_3d`

CMSR_m_print_2d
CMSR_m_print_3d

Prints on `stdout` the contents of 2D (3D) matrix in a given processor.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_m_print_2d (processor, src_matrix_field, signif_len, exp_len)
void
  CMSR_m_print_3d (processor, src_matrix_field, signif_len, exp_len)

unsigned int   processor;
CM_field_id_t  src_matrix_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_M_PRINT_2D
&                         (processor, src_matrix_field, signif_len, exp_len)
   SUBROUTINE CMSR_M_PRINT_3D
&                         (processor, src_matrix_field, signif_len, exp_len)
   INTEGER  processor
   INTEGER  src_matrix_field
   INTEGER  signif_len, exp_len
```

Lisp Syntax

```
CMSR:m-print-2d (processor src-matrix-field
                  &optional (signif-len 23) (exp-len 8))
CMSR:m-print-3d (processor src-matrix-field
                  &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

processor	The send address of the processor from which you wish to print the matrix.
src_matrix_field	A Paris field identifier specifying the matrix field in CM memory containing the matrix to be copied. For **`CMSR_m_copy_2d`** this matrix field contains a 3 x 3 homogeneous transformation matrix stored in row-major order. For **`CMSR_m_copy_3d`** this matrix field contains a 4 x 4 homogeneous transformation matrix stored in row-major order. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The length of the entire field is (4 * 4) * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in *src_matrix_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *src_matrix_field.*

DESCRIPTION

`CMSR_m_print_2d` and **`CMSR_m_print_3d`** print on **`stdout`** the double-precision floating-point contents of *src_matrix_field* in the given *processor.*

The matrix elements are printed one row per line, separated by spaces, and followed by a carriage return.

CMSR_m_read_from_processor_2d
CMSR_m_read_from_processor_3d

Reads a 2D (3D) matrix field from a specified processor into a front-end array.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
   CMSR_m_read_from_processor_2d
                (processor, src_matrix_field, dest_matrix, signif_len , exp_len)

unsigned int   processor;
CM_field_id_t  src_matrix_field;
double         dest_matrix[3][3];
unsigned int   signif_len,  exp_len;

void
   CMSR_m_read_from_processor_3d
                (processor, src_matrix_field, dest_matrix, signif_len . exp_len)

unsigned int   processor;
CM_field_id_t  src_matrix_field;
double         dest_matrix[4][4];
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'

   SUBROUTINE CMSR_M_READ_FROM_PROCESSOR_2D
&               (processor, src_matrix_field, dest_matrix, signif_len, exp_len)

   INTEGER          processor
   INTEGER          src_matrix_field
   DOUBLE PRECISION dest_matrix(3,3)
   INTEGER          signif_len
   INTEGER          exp_len
```

```
   SUBROUTINE CMSR_M_READ_FROM_PROCESSOR_3D
&               (processor, src_matrix_field, dest_matrix, signif_len, exp_len)
   INTEGER            processor
   INTEGER            src_matrix_field
   DOUBLE PRECISION   dest_matrix(4,4)
   INTEGER            signif_len
   INTEGER            exp_len
```

Lisp Syntax

```
   CMSR:m-read-from-processor-2d
                        (processor src-matrix-field
                         &optional matrix (signif-len 23) (exp-len 8))
   CMSR:m-read-from-processor-3d
                        (processor src-matrix-field
                         &optional matrix (signif-len 23) (exp-len 8))
```

ARGUMENTS

processor	The send address of the processor from which the matrix is to be read.
src_matrix_field	A Paris field indentifier specifying the field in *processor* from which the 3D matrix is to be read. The 2D matrix field contains a 3 x 3 homogeneous transformation matrix, and the 3D matrix field contains a 4 x 4 homogeneous transformation matrix. The elements of the matrix are stored in row-major order. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The length of the entire field is 16 * (*signif_len* + *exp_len* + 1).
dest_matrix	A 3 x 3 array for `CMSR_m_read_from_processor_2d` and a 4 x 4 array for `CMSR_m_read_from_processor_3d` in which the matrix is to be returned.
signif_len	The length, in bits, of the significand of the floating-point values in *src_matrix_field*.

exp_len	The length, in bits, of the exponent of the floating-point values in *src_matrix_field.*

DESCRIPTION

`CMSR_m_read_from_processor_2d` and `CMSR_m_read_from_processor_3d` read the contents of *src_matrix_field* from the given processor and store the matrix elements in the front-end array *dest_matrix.*

SEE ALSO

`CMSR_m_write_to_processor_2d`

`CMSR_m_write_to_processor_3d`

CMSR_m_ref_2d
CMSR_m_ref_3d

Returns the field ID of a specified element of a 2D (3D) matrix.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

CM_field_id_t
  CMSR_m_ref_2d (matrix_field, row column, signif_len, exp_len)

CM_field_id_t
  CMSR_m_ref_3d (matrix_field, row, column, signif_len, exp_len)

CM_field_id_t matrix_field;
unsigned int  row;
unsigned int  column;
unsigned int  signif_len;
unsigned int  exp_len;
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-math-fort.h'

    INTEGER FUNCTION CMSR_M_REF_2D
&                                   (matrix_field, row, column, signif_len, exp_len)

    INTEGER FUNCTION CMSR_M_REF_3D
&                                   (matrix_field, row, column, signif_len, exp_len)

    INTEGER  matrix_field
    INTEGER  row
    INTEGER  column
    INTEGER  signif_len
    INTEGER  exp_len
```

Lisp Syntax

```
CMSR:m-ref-2d (matrix-field row column
              &optional (signif-len 23) (exp-len 8))
CMSR:m-ref-3d (matrix-field row column
              &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

matrix_field	The Paris field ID of the matrix from which the matrix element specified by (*row*, *column*) is to be referenced. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. **`CMSR_m_ref_2d`** references a 2D matrix field containing a 3 x 3 homogeneous transformation matrix stored in row-major order. The length of the entire field is 9 * (*signif_len* + *exp_len* + 1). **`CMSR_m_ref_3d`** references a 3D matrix field containing a 4x4 homogeneous transformation matrix stored in row-major order. The length of the entire field is 16 * (*signif_len* + *elen* + 1).
row	The row position of the element to be returned. Matrices are stored in row-major order.
column	The column position of the element to be returned.
signif_len	The length, in bits, of the significand of the floating-point values in the field.
exp_len	The length, in bits, of the exponent of the floating-point values in the field.

DESCRIPTION

`CMSR_m_ref_2d` and **`CMSR_m_ref_3d`** return the field ID for the subfield of *matrix_field* in which the element (*row*, *column*) is stored.

CMSR_m_write_to_processor_2d
CMSR_m_write_to_processor_3d

Writes a 2D (3D) matrix from a front-end array into a matrix field on a specified processor.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_m_write_to_processor_2d
             (processor, dest_matrix_field, src_matrix, signif_len, exp_len)
unsigned int  processor;
CM_field_id_t dest_matrix_field;
double        src_matrix[3][3];
unsigned int  signif_len, exp_len;

void
  CMSR_m_write_to_processor_3d
             (processor, dest_matrix_field, src_matrix, signif_len, exp_len)
unsigned int  processor;
CM_field_id_t dest_matrix_field;
double        src_matrix[4][4];
unsigned int  signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_M_WRITE_TO_PROCESSOR_2D
&                 (processor, dest_matrix_field, src_matrix, signif_len, exp_len)
   INTEGER          processor
   INTEGER          dest_matrix_field
   DOUBLE PRECISION src_matrix(3,3)
   INTEGER          signif_len, exp_len
```

```
    SUBROUTINE CMSR_M_WRITE_TO_PROCESSOR_3D
&                (processor dest_matrix_field src_matrix signif_len exp_len)
    INTEGER           processor
    INTEGER           dest_matrix_field
    DOUBLE PRECISION  src_matrix(4,4)
    INTEGER           signif_len, exp_len
```

Lisp Syntax

```
CMSR:m-write-to-processor-2d (processor dest-matrix-field src-matrix
                              (signif-len 23) (exp-len 8))
CMSR:m-write-to-processor-3d (processor dest-matrix-field src-matrix
                              (signif-len 23) (exp-len 8))
```

ARGUMENTS

processor	The processor send address to which the matrix is to be sent.
dest_matrix_field	A Paris field indentifier specifying the field in *processor* in which the *src_matrix* is to be stored. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The elements are stored in row-major order. The 2D matrix field must be large enough for a 3 x3 matrix; the length of the field must be at least 9 * (*signif_len* + *exp_len* + 1). The 3D matrix field must be large enough for a 4 x4 matrix; the length of the field must be at least 16 * (*signif_len* + *exp_len* + 1).
src_matrix	For `CMSR_m_read_from_processor_2d` a 3 x 3 array and for `CMSR_m_ read_from_processor_3d` a 4 x 4 array, that is to be broadcast to *dest_matrix_field.*
signif_len	The length, in bits, of the significand of the floating-point values in *matrix_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *matrix_field.*

DESCRIPTION

`CMSR_m_read_from_processor_2d` and **`CMSR_m_write_to_processor_3d`** place the contents of *src_matrix* in *dest_matrix–field* on the specified processor.

SEE ALSO

`CMSR_m_read_from_processor_2d`

`CMSR_m_read_from_processor_3d`

CMSR_rotation_const_matrix_2d

Inserts specified rotation into a 2D transformation matrix field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_rotation_const_matrix_2d
                        (dest_matrix_field, theta, signif_len, exp_len)
CM_field_id_t  dest_matrix_field;
double         theta;
unsigned int   signif_len;
unsigned int   exp_len;
```

Fortran Syntax

```
      INCLUDE '/usr/include/cm/cmsr-math-fort.h'
      SUBROUTINE CMSR_ROTATION_CONST_MATRIX_2D
&                       (dest_matrix_field, theta, signif_len, exp_len)
      INTEGER dest_matrix_field
      DOUBLE PRECISION theta
      INTEGER signif_len
      INTEGER exp_len
```

Lisp Syntax

```
CMSR:rotation-const-matrix-2d
                 (dest-matrix-field theta
                  &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_matrix_field	A Paris field identifier specifying the field in CM memory to which the 2D transformation matrix is to be returned.
	The 2D matrix field contains a 3 x 3 homogeneous transformation matrix stored in row-major order. Each element of the matrix is a

	floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The length of the entire field is 9 * (*signif_len* + *exp_len* + 1).
theta	A double-precision value on the front-end computer. *theta* is the rotation in radians to be incorporated into the transformation matrix in *dest_matrix_field.*
signif_len	The length, in bits, of the significand of the floating-point values in *dest_matrix_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_matrix_field.*

DESCRIPTION

For each active processor in the current VP set, **`CMSR_rotation_const_matrix_2d`** creates a two-dimensional rotation matrix in *dest_matrix_field* by setting the field to an identity matrix and then inserting a rotation of *theta* radians. All fields must be in the current VP set.

SEE ALSO

`CMSR_rotation_matrix_2d`

`CMSR_x_rotation_const_matrix_3d`

`CMSR_x_rotation_matrix_3d`

`CMSR_y_rotation_const_matrix_3d`

`CMSR_y_rotation_matrix_3d`

`CMSR_z_rotation_const_matrix_3d`

`CMSR_z_rotation_matrix_3d`

`CMSR_fe_rotation_matrix_2d`

`CMSR_fe_x_rotation_matrix_3d`

`CMSR_fe_y_rotation_matrix_3d`

`CMSR_fe_z_rotation_matrix_3d`

CMSR_rotation_matrix_2d

Inserts field of 2D rotation values into a transformation matrix field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_rotation_matrix_2d
                      (dest_matrix_field, theta_field, signif_len, exp_len)
CM_field_id_t  dest_matrix_field;
CM_field_id_t  theta_field;
unsigned int   signif_len;
unsigned int   exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_ROTATION_MATRIX_2D
&                        (dest_matrix_field, theta_field, signif_len, exp_len)
   INTEGER  dest_matrix_field
   INTEGER  theta_field
   INTEGER  signif_len
   INTEGER  exp_len
```

Lisp Syntax

```
CMSR:rotation-matrix-2d (dest-matrix-field theta-field
                    &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_matrix_field A Paris field identifier specifying the field in CM memory to which the 2D transformation matrix is to be returned.

The 2D matrix field contains a 3 x 3 homogeneous transformation matrix stored in row-major order. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the

	length of the exponent, and 1 is the sign bit. The length of the entire field is 9 * (*signif_len* + *exp_len* + 1).
theta_field	A Paris field identifier specifying the field in CM memory containing the the rotation angle, in radians, to be inserted into *dest_matrix_field*. *theta_field* must be in the same VP set as *dest_matrix_field*. *theta_field* is a floating-point value of length (*signif_len* + *exp_len* + 1 + *color_len*), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.
signif_len	The length, in bits, of the significand of the floating-point values in *theta_field* and *dest_matrix_field*.
exp_len	The length, in bits, of the exponent of the floating-point values in *theta_field* and *dest_matrix_field*.

DESCRIPTION

For each active processor, **`CMSR_rotation_matrix_2d`** creates a two-dimensional rotation matrix by setting the transformation matrix in *dest_matrix_field* to an identity matrix and then inserting a rotation of *theta_field* radians. All fields must be in the current VP set.

SEE ALSO

`CMSR_rotation_const_matrix_2d`

`CMSR_x_rotation_const_matrix_3d`

`CMSR_x_rotation_matrix_3d`

`CMSR_y_rotation_const_matrix_3d`

`CMSR_y_rotation_matrix_3d`

`CMSR_z_rotation_const_matrix_3d`

`CMSR_z_rotation_matrix_3d`

`CMSR_fe_rotation_matrix_2d`

`CMSR_fe_x_rotation_matrix_3d`

`CMSR_fe_y_rotation_matrix_3d`

`CMSR_fe_z_rotation_matrix_3d`

CMSR_scale_const_matrix_2d
CMSR_scale_const_matrix_3d

Inserts specified 2D (3D) scaling terms into a transformation matrix field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
   CMSR_scale_const_matrix_2d
                        (dest_matrix_field, sx, sy, signif_len, exp_len)

CM_field_id_t  dest_matrix_field;
double         sx, sy;
unsigned int   signif_len;
unsigned int   exp_len;

void
   CMSR_scale_const_matrix_3d
                      (dest_matrix_field, sx, sy, sz, signif_len, exp_len)

CM_field_id_t  dest_matrix_field;
double         sx, sy, sz;
unsigned int   signif_len;
unsigned int   exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_SCALE_CONST_MATRIX_2D
&                           (dest_matrix_field, sx, sy, signif_len, exp_len)
   INTEGER           dest_matrix_field
   DOUBLE PRECISION  sx, sy
   INTEGER           signif_len
   INTEGER           exp_len
```

```
    SUBROUTINE CMSR_SCALE_CONST_MATRIX_3D
&                           (dest_matrix_field, sx, sy, sz, signif_len, exp_len)
    INTEGER            dest_matrix_field
    DOUBLE PRECISION   sx, sy, sz
    INTEGER            signif_len
    INTEGER            exp_len
```

Lisp Syntax

```
CMSR:scale-const-matrix-2d(dest-matrix-field sx sy
                                  &optional (signif-len 23) (exp-len 8))
CMSR:scale-const-matrix-3d(dest-matrix-field sx sy sz
                                  &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_matrix_field	A Paris field identifier specifying the field in CM memory to which the transformation matrix is to be written. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The 2D matrix field contains a 3 x 3 homogeneous transformation matrix stored in row-major order. The length of the entire field is 9 * (*signif_len* + *exp_len* + 1). The 3D matrix field contains a 4 x 4 homogeneous transformation matrix stored in row-major order. The length of the entire field is 16 * (*signif_len* + *exp_len* + 1).
sx	A double-precision value on the front-end computer. *sx* is the scaling value for the *x* axis to be incorporated into the transformation matrix in *dest_matrix_field.*
sy	A double-precision value on the front-end computer. *sy* is the scaling value for the *x* axis to be incorporated into the transformation matrix in *dest_matrix_field.*
sz	A double-precision value on the front-end computer. For `CMSR_scale_const_matrix_3d`, *sz* is the scaling value for the *z* axis to be incorporated into the transformation matrix in *dest_matrix_field.*

signif_len	The length, in bits, of the significand of the floating-point values in *dest_matrix_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_matrix_field.*

DESCRIPTION

`CMSR_scale_const_matrix_2d` cretates a two-dimensional scaling matrix by setting the transformation matrix in *dest_matrix_field* to an identity matrix and then inserting the scaling terms *sx* and *sy*.

`CMSR_scale_const_matrix_3d` creates a three-dimensional scaling matrix by setting the transformation matrix in *dest_matrix_field* to an identity matrix and then inserting the scaling terms *sx*, *sy*, and *sz*.

All fields must be in the current VP set.

SEE ALSO

`CMSR_scale_const_matrix_3d`

`CMSR_scale_matrix_2d`

`CMSR_scale_matrix_3d`

`CMSR_fe_scale_matrix_2d`

`CMSR_fe_scale_matrix_3d`

CMSR_scale_matrix_2d
CMSR_scale_matrix_3d

Inserts fields of 2D (3D) scaling terms into a transformation matrix field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
  CMSR_scale_matrix_2d
               (dest_matrix_field, sx_field, sy_field, signif_len, exp_len)

CM_field_id_t  dest_matrix_field;
CM_field_id_t  sx_field, sy_field;
unsigned int   signif_len;
unsigned int   exp_len;

void
  CMSR_scale_matrix_3d
          (dest_matrix_field, sx_field, sy_field, sz_field, signif_len, exp_len)

CM_field_id_t  dest_matrix_field;
CM_field_id_t  sx_field, sy_field, sz_field;
unsigned int   signif_len;
unsigned int   exp_len;
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-math-fort.h'

    SUBROUTINE CMSR_SCALE_MATRIX_2D
&                   (dest_matrix_field, sx_field, sy_field, signif_len, exp_len)

    INTEGER  dest_matrix_field
    INTEGER  sx_field, sy_field
    INTEGER  signif_len
    INTEGER  exp_len
```

```
   CMSR_SCALE_MATRIX_3D
&             (dest_matrix_field, sx_field, sy_field, sz_field, signif_len, exp_len)
   INTEGER  dest_matrix_field
   INTEGER  sx_field, sy_field, sz_field
   INTEGER  signif_len
   INTEGER  exp_len
```

Lisp Syntax

```
CMSR:scale-matrix-2d (dest-matrix-field sx-field sy-field
                           &optional (signif-len 23) (exp-len 8))
CMSR:scale-matrix-3d (dest-matrix-field sx-field sy-field sz-field
                           &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_matrix_field	A Paris field identifier specifying the field in CM memory containing a 2D transformation matrix. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The 2D matrix field contains a 3 x 3 homogeneous transformation matrix stored in row-major order. The length of the entire field is 9 * (*signif_len* + *exp_len* + 1). The 3D matrix field contains a 4 x 4 homogeneous transformation matrix stored in row-major order. The length of the entire field is 16 * (*signif_len* + *exp_len* + 1).
sx_field	A Paris field identifier specifying the field in CM memory containing the *x* coordinate scaling value to be inserted into *dest_matrix_field*. *sx_field* must be in the same VP set as *dest_matrix_field*. *sx_field* is a floating-point value of length (*signif_len* + *exp_len* + 1 +*color_len*), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.
sy_field	A Paris field identifier specifying the field in CM memory containing the *y* coordinate scaling value to be inserted into

	dest_matrix_field. sy_field must be in the same VP set as *dest_matrix_field.* *sy_field* is a floating-point value of length (*signif_len* + *exp_len* + 1 + *color_len*), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent and 1 is the sign bit.
sz_field	For **`CMSR_scale_matrix_3d`**, a Paris field identifier specifying the field in CM memory containing the *z* coordinate scaling value to be inserted into *dest_matrix_field. sz_field* must be in the same VP set as *dest_matrix_field.* *sz_field* is a floating-point value of length (*signif_len* + *exp_len* + 1 + *color_len*), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.
signif_len	The length, in bits, of the significand of the floating-point values in *sx_field, sy_field*, and *dest_matrix_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *sx_field, sy_field*, and *dest_matrix_field.*

DESCRIPTION

For each active processor, **`CMSR_scale_matrix_2d`** creates a two-dimensional scaling matrix by setting the transformation matrix in *dest_matrix_field* to an identity matrix and then inserting the scaling terms from *sx_field* and *sy_field.*

For each active processor, **`CMSR_scale_matrix_3d`** creates a three-dimensional scaling matrix by setting the transformation matrix in *dest_matrix_field* to an identity matrix and then inserting the scaling terms from *sx_field, sy_field,* and *sz_field.*

All fields must be in the current VP set.

SEE ALSO

`CMSR_scale_const_matrix_2d`

`CMSR_scale_const_matrix_3d`

`CMSR_fe_scale_matrix_2d`

`CMSR_fe_scale_matrix_3d`

CMSR_trans_const_matrix_2d
CMSR_trans_const_matrix_3d

Inserts 2D (3D) translation terms into a transformation matrix field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
   CMSR_trans_const_matrix_2d
                                  (dest_matrix_field, tx, ty signif_len, exp_len)

CM_field_id_t  dest_matrix_field;
double         tx, ty;
unsigned int   signif_len;
unsigned int   exp_len;

void
   CMSR_trans_const_matrix_3d
                              (dest_matrix_field, tx, ty,  tz, signif_len, exp_len)

CM_field_id_t  dest_matrix_field;
double         tx, ty, tz;
unsigned int   signif_len;
unsigned int   exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_TRANS_CONST_MATRIX_2D
&                                  (dest_matrix_field, tx, ty, signif_len, exp_len)
   INTEGER            dest_matrix_field
   DOUBLE PRECISION   tx, ty
   INTEGER            signif_len
   INTEGER            exp_len
```

```
      SUBROUTINE CMSR_TRANS_CONST_MATRIX_3D
&                             (dest_matrix_field, tx, ty, tz, signif_len, exp_len)
      INTEGER            dest_matrix_field
      DOUBLE PRECISION   tx
      DOUBLE PRECISION   ty
      DOUBLE PRECISION   tz
      INTEGER            signif_len
      INTEGER            exp_len
```

Lisp Syntax

```
CMSR:trans-const-matrix-2d(dest-matrix-field tx ty
                                   &optional (signif-len 23) (exp-len 8))
CMSR:trans-const-matrix-3d(dest-matrix-field tx ty tz
                                   &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_matrix_field	A Paris field identifier specifying the field in CM memory to which the transformation matrix is to be written. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The 2D matrix field contains a 3 x 3 homogeneous transformation matrix stored in row-major order. The length of the entire field is 9 * (*signif_len* + *exp_len* + 1). The 3D matrix field contains a 4 x 4 homogeneous transformation matrix stored in row-major order. The length of the entire field is 16 * (*signif_len* + *exp_len* + 1).
tx	A double-precision value on the front-end computer. *tx* is the *x* translation value to be incorporated into the transformation matrix in *dest_matrix_field*.
ty	A double-precision value on the front-end computer. *ty* is the *y* translation value to be incorporated into the transformation matrix in *dest_matrix_field*.

tz	For **`CMSR_trans_const_matrix_3d`**, a double-precision value on the front-end computer. *tz* is the *z* translation value to be incorporated into the transformation matrix in *dest_matrix_field.*
signif_len	The length, in bits, of the significand of the floating-point values in *dest_matrix_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_matrix_field.*

DESCRIPTION

`CMSR_trans_const_matrix_2d` inserts the translation terms *tx* and *ty* into the matrix in each active processor in *dest_matrix_field.*

`CMSR_trans_const_matrix_3d` inserts the translation terms *tx*, *ty*, and *tz* into the matrix in each active processor in *dest_matrix_field.*

SEE ALSO

`CMSR_translation_matrix_2d`

`CMSR_translation-matrix-3d`

`CMSR_fe_translation_matrix_2d`

`CMSR_fe_translation_matrix_3d`

CMSR_translation_matrix_2d
CMSR_translation_matrix_3d

Inserts fields of 2D (3D) translation terms into a transformation matrix field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
  CMSR_translation_matrix_2d
             (dest_matrix_field, tx_field, ty_field, signif_len, exp_len)

CM_field_id_t  dest_matrix_field;
CM_field_id_t  tx_field, ty_field;
unsigned int   signif_len;
unsigned int   exp_len;

void
  CMSR_translation_matrix_3d
        (dest_matrix_field, tx_field, ty_field, tz_field, signif_len, exp_len)

CM_field_id_t  dest_matrix_field;
CM_field_id_t  tx_field, ty_field, tz_field;
unsigned int   signif_len;
unsigned int   exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'

   SUBROUTINE CMSR_TRANSLATION_MATRIX_2D
&                    (dest_matrix_field, tx_field, ty_field, signif_len, exp_len)

   INTEGER  dest_matrix_field
   INTEGER  tx_field, ty_field
   INTEGER  signif_len
   INTEGER  exp_len
```

```
   CMSR_TRANSLATION_MATRIX_3D
&              (dest_matrix_field, tx_field, ty_field, tz_field, ,signif_len, exp_len)
   INTEGER  dest_matrix_field
   INTEGER  tx_field, ty_field, tz_field
   INTEGER  signif_len
   INTEGER  exp_len
```

Lisp Syntax

```
   CMSR:translation-matrix-2d(dest-matrix-field tx-field ty-field
                              &optional (signif-len 23) (exp-len 8))
   CMSR:translation-matrix-3d(dest-matrix-field tx-field ty-field tz-field
                              &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_matrix_field	A Paris field identifier specifying the field in CM memory containing a transformation matrix. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The 2D matrix field contains a 3 x 3 homogeneous transformation matrix stored in row-major order. The length of the entire field is 9 * (*signif_len* + *exp_len* + 1). The 3D matrix field contains a 4 x 4 homogeneous transformation matrix stored in row-major order. The length of the entire field is 16 * (*signif_len* + *exp_len* + 1).
tx_field	A Paris field identifier specifying the field in CM memory containing the *x* coordinate translation value to be inserted into *dest_matrix_field*. *tx_field* must be in the same VP set as *dest_matrix_field*. *tx_field* is a floating-point value of length (*signif_len* + *exp_len* + 1 + *color_len*), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.
ty_field	A Paris field identifier specifying the field in CM memory containing the *y* coordinate translation value to be inserted into

	dest_matrix_field. ty_field must be in the same VP set as *dest_matrix_field.* *ty_field* is a floating-point value of length (*signif_len* + *exp_len* + 1 + *color_len*), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.
tz_field	For **`CMSR_translation-matrix-3d`**, a Paris field identifier specifying the field in CM memory containing the *z* coordinate translation value to be inserted into *dest_matrix_field. tz_field* must be in the same VP set as *dest_matrix_field.* *tz_field* is a floating-point value of length (*signif_len* + *exp_len* + 1 + *color_len*), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.
signif_len	The length, in bits, of the significand of the floating-point values in *tx_field, ty_field*, and *dest_matrix_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *tx_field, ty_field*, and *dest_matrix_field.*

DESCRIPTION

For each active processor, **`CMSR_translation_matrix_2d`** creates a two-dimensional translation matrix by setting the transformation matrix in *dest_matrix_field* to an identity matrix and then inserting the translation terms *tx_field* and *ty_field.*

For each active processor, **`CMSR_translation_matrix_3d`** creates a three-dimensional translation matrix by setting the transformation matrix in *dest_matrix_field* to an identity matrix and then inserting the translation terms *tx_field*, *ty_field,* and *tz_field.*

All fields must be in the current VP set.

SEE ALSO

`CMSR_translation_const_matrix_2d`

`CMSR_translation_const_matrix_3d`

`CMSR_fe_translation_matrix_2d`

`CMSR_fe_translation_matrix_3d`

CMSR_x_rotation_const_matrix_3d
CMSR_y_rotation_const_matrix_3d
CMSR_z_rotation_const_matrix_3d

Inserts a specified rotation around *x* (*y*, *z*) into a 3D transformation matrix field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
  CMSR_x_rotation_const_matrix_3d
                          (dest_matrix_field, theta, signif_len, exp_len)
void
  CMSR_y_rotation_const_matrix_3d
                          (dest_matrix_field, theta, signif_len, exp_len)
void
  CMSR_z_rotation_const_matrix_3d
                          (dest_matrix_field, theta, signif_len, exp_len)

CM_field_id_t  dest_matrix_field;
double         theta;
unsigned int   signif_len;
unsigned int   exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'

   SUBROUTINE CMSR_X_ROTATION_CONST_MATRIX_3D
&                          (dest_matrix_field, theta, signif_len, exp_len)
   SUBROUTINE CMSR_Y_ROTATION_CONST_MATRIX_3D
&                          (dest_matrix_field, theta, signif_len, exp_len)
   SUBROUTINE CMSR_Z_ROTATION_CONST_MATRIX_3D
&                          (dest_matrix_field, theta, signif_len, exp_len)

   INTEGER           dest_matrix_field
   DOUBLE PRECISION  theta
   INTEGER           signif_len
   INTEGER           exp_len
```

Lisp Syntax

```
CMSR:x-rotation-const-matrix-3d
                    (dest-matrix-field theta
                        &optional (signif-len 23) (exp-len 8))
CMSR:y-rotation-const-matrix-3d
                    (dest-matrix-field theta
                        &optional (signif-len 23) (exp-len 8))
CMSR:z-rotation-const-matrix-3d
                    (dest-matrix-field theta
                        &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_matrix_field	A Paris field identifier specifying the field in CM memory to which the 3D transformation matrix is to be returned. The 3D matrix field contains a 4x4 homogeneous transformation matrix stored in row-major order. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The length of the entire field is 16 * (*signif_len* + *exp_len* + 1).
theta	A double-precision value on the front-end computer. *theta* is the rotation, in radians, to be incorporated into the transformation matrix in *dest_matrix_field.*
signif_len	The length, in bits, of the significand of the floating-point values in *dest_matrix_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_matrix_field.*

DESCRIPTION

`CMSR_x_rotation_const_matrix_3d`, **`CMSR_y_rotation_const_matrix_3d`**, and **`CMSR_z_rotation_const_matrix_3d`** create a three-dimensional rotation matrix in *dest_matrix_field* by setting the field to an identity matrix and then inserting a rotation of *theta* radians around the *x*, *y*, or *z* axis, respectively.

All fields must be in the current VP set.

SEE ALSO

`CMSR_x_rotation_matrix_3d`

`CMSR_y_rotation_matrix_3d`

`CMSR_z_rotation_matrix_3d`

`CMSR_rotation_matrix_2d`

`CMSR_rotation_const_matrix_2d`

`CMSR_fe_rotation_matrix_2d`

`CMSR_fe_x_rotation_matrix_3d`

`CMSR_fe_y_rotation_matrix_3d`

`CMSR_fe_z_rotation_matrix_3d`

CMSR_x_rotation_matrix_3d
CMSR_y_rotation_matrix_3d
CMSR_z_rotation_matrix_3d

Inserts field of rotation values around *x* (*y*, *z*) into a 3D transformation matrix field.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
  CMSR_x_rotation_matrix_3d
                    (dest_matrix_field, theta_field, signif_len, exp_len)
void
  CMSR_y_rotation_matrix_3d
                    (dest_matrix_field, theta_field, signif_len, exp_len)
void
  CMSR_z_rotation_matrix_3d
                    (dest_matrix_field, theta_field, signif_len, exp_len)

CM_field_id_t  dest_matrix_field;
CM_field_id_t  theta_field;
unsigned int   signif_len;
unsigned int   exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'

   SUBROUTINE CMSR_X_ROTATION_MATRIX_3D
&                       (dest_matrix_field, theta_field, signif_len, exp_len)
   SUBROUTINE CMSR_Y_ROTATION_MATRIX_3D
&                       (dest_matrix_field, theta_field, signif_len, exp_len)
   SUBROUTINE CMSR_Z_ROTATION_MATRIX_3D
&                       (dest_matrix_field, theta_field, signif_len, exp_len)

   INTEGER  dest_matrix_field
   INTEGER  theta_field
   INTEGER  signif_len
   INTEGER  exp_len
```

Lisp Syntax

```
CMSR:x-rotation-matrix-3d (dest-matrix-field theta-field
                            &optional (signif-len 23) (exp-len 8))
CMSR:y-rotation-matrix-3d (dest-matrix-field theta-field
                            &optional (signif-len 23) (exp-len 8))
CMSR:z-rotation-matrix-3d (dest-matrix-field theta-field
                            &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

dest_matrix_field	A Paris field identifier specifying the field in CM memory to which the 3D transformation matrix is to be returned. The 3D matrix field contains a 4x4 homogeneous transformation matrix stored in row-major order. Each element of the matrix is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The length of the entire field is 16 * (*signif_len* + *exp_len* + 1).
theta_field	A Paris field identifier specifying the field in CM memory containing the rotation angle, in radians, to be inserted into *dest_matrix_field*. *theta_field* must be in the same VP set as *dest_matrix_field*. *theta_field* is a floating-point value of length (*signif_len* + *exp_len* + 1 + *color_len*), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit.
signif_len	The length, in bits, of the significand of the floating-point values in *theta_field* and *dest_matrix_field*.
exp_len	The length, in bits, of the exponent of the floating-point values in *theta_field* and *dest_matrix_field*.

DESCRIPTION

For each active processor, **`CMSR_x_rotation_matrix_3d`**, **`CMSR_y_rotation_matrix_3d`**, and **`CMSR_z_rotation_matrix_3d`** create a three-dimensional rotation matrix by setting the transformation matrix in *dest_matrix_field* to an identity matrix and then inserting a rotation of *theta_field* radians around the *x*, *y*, or *z* axis, respectively.

All fields must be in the current VP set.

SEE ALSO

`CMSR_x_rotation_const_matrix_3d`

`CMSR_y_rotation_const_matrix_3d`

`CMSR_z_rotation_const_matrix_3d`

`CMSR_rotation_matrix_2d`

`CMSR_rotation_const_matrix_2d`

`CMSR_fe_rotation_matrix_2d`

`CMSR_fe_x_rotation_matrix_3d`

`CMSR_fe_y_rotation_matrix_3d`

`CMSR_fe_z_rotation_matrix_3d`

3.8 CM Color Conversion Routines

This section documents the *Render routines that convert color vectors between color spaces.

CMSR_rgb_to_cmy
CMSR_cmy_to_rgb

Converts color vector fields RGB to CMY (CMY to RGB) color models.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_rgb_to_cmy(cmy_vector_field, rgb_vector_field, signif_len, exp_len)
CM_field_id_t  cmy_vector_field, rgb_vector_field;
unsigned int   signif_len, exp_len;

void
  CMSR_cmy_to_rgb(rgb_vector_field, cmy_vector_field, signif_len, exp_len)
CM_field_id_t  rgb_vector_field, cmy_vector_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
      INCLUDE '/usr/include/cm/cmsr-math-fort.h'
      SUBROUTINE CMSR_RGB_TO_CMY
     &          (cmy_vector_field, rgb_vector_field, signif_len, exp_len)
      INTEGER cmy_vector_field, rgb_vector_field
      INTEGER signif_len, exp_len

      SUBROUTINE CMSR_CMY_TO_RGB
     &          (cmy_vector_field, rgb_vector_field, signif_len, exp_len)
      INTEGER rgb_vector_field, cmy_vector_field
      INTEGER signif_len, exp_len
```

Lisp Syntax

```
CMSR:rgb-to-cmy (cmy-vector-field rgb-vector-field
                  &optional (signif-len 23) (exp-len 8))
CMSR:cmy-to-rgb ( rgb-vector-field cmy-vector-field
                  &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

rgb_vector_field	A Paris field identifier specifying the field in CM memory containing the RGB color triplet. The red intensity is in the first element, the green intensity is in the second element, and the blue intensity is in the third element. Each intensity should be in the range of [0,1].
cmy_vector_field	A Paris field identifier specifying the field in CM memory containing the CMY color triplet. The cyan intensity is in the first element, the magenta intensity is in the second element, and the yellow intensity is in the third element. Each intensity should be in the range of [0,1]. Each element in these fields is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in *rgb_vector_field* and *cmy_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *rgb_vector_field* and *cmy_vector_field.*

DESCRIPTION

For each active processor in the current VP set, **`CMSR_rgb_to_cmy`** converts the RGB triplet in *rgb_vector_field* to a CMY triplet and places the result in *cmy_vector_field.* The relationship is

```
(c,m,y) = (1,1,1) - (r,g,b)
```

Similarly, for each active processor in the current VP set, **`CMSR_cmy_to_rgb`** converts the CMY triplet in *cmy_vector_field* to RGB and places the result in *rgb_vector_field.* The relationship is

```
(r,g,b) = (1,1,1) - (c,m,y)
```

CMSR_rgb_to_yiq
CMSR_yiq_to_rgb

Converts color vector fields from RGB to YIQ (YIQ to RGB) color models.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
   CMSR_rgb_to_yiq  (yiq_vector_field, rgb_vector_field, signif_len exp_len)
CM_field_id_t yiq_vector_field, rgb_vector_field;
unsigned int signif_len,  exp_len;

void
   CMSR_yiq_to_rgb  (rgb_vector_field, yiq_vector_field,  signif_len exp_len)
CM_field_id_t rgb_vector_field, yiq_vector_field;
unsigned int signif_len,  exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-math-fort.h'
   SUBROUTINE CMSR_RGB_TO_YIQ
&                         (yiq_vector_field, rgb_vector_field, signif_len, exp_len)
   INTEGER  yiq_vector_field, rgb_vector_field
   INTEGER  signif_len,  exp_len;

   SUBROUTINE CMSR_YIQ_TO_RGB
&                         (rgb_vector_field, yiq_vector_field, signif_len exp_len)
   INTEGER  rgb_vector_field, yiq_vector_field;
   INTEGER  signif_len,  exp_len;
```

Lisp Syntax

```
CMSR:rgb-to-yiq  (yiq-vector-field  rgb-vector-field
                  &optional  (signif-len 23)  (exp-len 8))
CMSR:yiq-to-rgb  (rgb-vector-field  yiq-vector-field
                  &optional  (signif-len 23)  (exp-len 8))
```

ARGUMENTS

rgb_vector_field	A Paris field identifier specifying the field in CM memory containing the RGB color triplet. The red intensity is in the first element, the green intensity is in the second element, and the blue intensity is in the third element. Each of the RGB color components should be in the range of [0,1].
yiq_vector_field	A Paris field identifier specifying the field in CM memory containing the YIQ color triplet. The Y (luminance) intensity is in the first element, the I (orange-cyan chromaticity) intensity is in the second element, and the Q (green-magenta chromaticity) intensity is in the third element. Each intensity should be in the range of [0,1]. Each element in these fields is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in *rgb_vector_field* and *yiq_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *rgb_vector_field* and *yiq_vector_field.*

DESCRIPTION

For each active processor in the current VP set, `CMSR_rgb_to_yiq` converts an RGB triplet in *rgb_vector_field* to a YIQ triplet and places the result in *yiq_vector_field.*

For each active processor in the current VP set, `CMSR_yiq_to_rgb` converts a YIQ triplet in *yiq_vector_field* to an RGB triplet and places the result in *rgb_vector_field.*

CMSR_rgb_to_hsv
CMSR_hsv_to_rgb

Converts color vector fields from RGB to HSV (HSV to RGB) color models.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
   CMSR_rgb_to_hsv (hsv_vector_field, rgb_vector_field, signif_len, exp_len)
CM_field_id_t  hsv_vector_field, rgb_vector_field;
unsigned int   signif_len, exp_len;

void
   CMSR_hsv_to_rgb (rgb_vector_field, hsv_vector_field, signif_len, exp_len)
CM_field_id_t  rgb_vector_field, hsv_vector_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmsr-math-fort.h'
    SUBROUTINE CMSR_RGB_TO_HSV
&                        (hsv_vector_field, rgb_vector_field, signif_len, exp_len)
    INTEGER  hsv_vector_field, rgb_vector_field;
    INTEGER  signif_len, exp_len;

    SUBROUTINE CMSR_HSV_TO_RGB
&                        (rgb_vector_field, hsv_vector_field, signif_len, exp_len)
    INTEGER  rgb_vector_field, hsv_vector_field;
    INTEGER  signif_len, exp_len;
```

Lisp Syntax

```
CMSR:rgb-to-hsv  (rgb-vector-field hsv-vector-field
                   &optional (signif-len 23) (exp-len 8))
CMSR:hsv-to-rgb  (hsv-vector-field rgb-vector-field
                   &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

rgb_vector_field	A Paris field identifier specifying the field in CM memory containing the RGB color triplet. The red intensity is in the first element, the green intensity is in the second element, and the blue intensity is in the third element. Each of intensity should be in the range of [0,1].
hsv_vector_field	A Paris field identifier specifying the field in CM memory containing the HSV color triplet. The hue of the color is in the first element, the saturation is in the second element, and the value is in the third element. Hue should be in the range [0, 2*pi], and saturation and value should be in the range [0,1]. Each element in these fields is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in *rgb_vector_field* and *hsv_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *rgb_vector_field* and *hsv_vector_field.*

DESCRIPTION

For each active processor in the current VP set, `CMSR_rgb_to_hsv` converts the RGB triplet in *rgb_vector_field* to an HSV triplet and places the result in *hsv_vector_field.* Hue will be between 0.0 and 2*pi. If *s* is zero, *h* is irrelevant and is set to zero. If *v* is zero, *h* and *s* are irrelevant and both are set to zero.

For each active processor in the current VP set, `CMSR_hsv_to_rgb` converts the HSV triplet in *hsv_vector_field* to an RGB triplet and places the result in *rgb_vector_field.* Hue is taken modulo 2*pi. If *s* is zero, *h* is irrelevant and is set to zero. If *v* is zero, *s* and *v* are irrelevant and both are set to zero.

CMSR_rgb_to_hsl
CMSR_hsl_to_rgb

Converts color vector fields from RGB to HSL (HSL to RGB) color models.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
  CMSR_rgb_to_hsl  (hsl_vector_field, rgb_vector_field, signif_len, exp_len)
CM_field_id_t  hsl_vector_field, rgb_vector_field;
unsigned int   signif_len, exp_len;
void
  CMSR_hsl_to_rgb  (rgb_vector_field, hsl_vector_field, signif_len, exp_len)
CM_field_id_t  rgb_vector_field, hsl_vector_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
      INCLUDE '/usr/include/cm/cmsr-math-fort.h'
      SUBROUTINE CMSR_RGB_TO_HSL
&                          (hsl_vector_field, rgb_vector_field, signif_len, exp_len)
      INTEGER  hsl_vector_field, rgb_vector_field;
      INTEGER  signif_len, exp_len;

      SUBROUTINE CMSR_HSL_TO_RGB
&                          (rgb_vector_field, hsl_vector_field, signif_len, exp_len)
      INTEGER  rgb_vector_field, hsl_vector_field;
      INTEGER  signif_len, exp_len;
```

Lisp Syntax

```
CMSR:rgb-to-hsl  (rgb-vector-field hsl-vector-field
                  &optional (signif-len 23) (exp-len 8))
CMSR:hsl-to-rgb  (hsl-vector-field rgb-vector-field
                  &optional (signif-len 23) (exp-len 8))
```

ARGUMENTS

rgb_vector_field	A Paris field identifier specifying the field in CM memory containing the RGB color triplet. The red intensity is in the first element, the green intensity is in the second element, and the blue intensity is in the third element. Each intensity should be in the range of [0,1].
hsl_vector_field	A Paris field identifier specifying the field in CM memory containing the HSL color triplet. The hue of the color is in the first element, the saturation is in the second element, and the lightness is in the third element. Hue should be in the range [0, 2*pi], and saturation and lightness should be in the range [0,1].
	Each element in these fields is a floating-point value having a length of (*signif_len* + *exp_len* + 1), where *signif_len* is the length of the significand, *exp_len* is the length of the exponent, and 1 is the sign bit. The length of the entire field is 3 * (*signif_len* + *exp_len* + 1).
signif_len	The length, in bits, of the significand of the floating-point values in *rgb_vector_field* and *hsl_vector_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *rgb_vector_field* and *hsl_vector_field.*

DESCRIPTION

For each active processor in the current VP set, `CMSR_rgb_to_hsl` converts the RGB triplet in *rgb_vector_field* to an HSL triplet and places the result in *hsl_vector_field.*

For each active processor in the current VP set, `CMSR_hsl_to_rgb` converts the HSL triplet in *hsl_vector_field* to an RGB triplet and places the result in *rgb_vector_field.*

If saturation is zero, the resulting color is a gray shade. In this case hue is irrelevant and is set to zero. If lightness is zero, the color is black. In this case both hue and saturation are irrelevant and are set to zero.

3.9 CM Miscellaneous Routines

This section contains utility routines that convert between degrees and radians.

CMSR_deg_to_rad
CMSR_rad_to_deg

Converts degrees to radians (radians to degrees) for CM fields.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>
void
   CMSR_deg_to_rad  (dest_field, src_field, signif_len, exp_len)
void
   CMSR_rad_to_deg  (dest_field, src_field, signif_len, exp_len)
CM_field_id_t  dest_field;
CM_field_id_t  src_field;
unsigned int   signif_len;
unsigned       exp_len;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmsr-math-fort.h'
SUBROUTINE CMSR_DEG_TO_RAD  (dest_field, src_field, signif_len, exp_len)
SUBROUTINE CMSR_RAD_TO_DEG  (dest_field, src_field, signif_len, exp_len)
INTEGER  dest_field
INTEGER  src_field
INTEGER  signif_len
INTEGER  exp_len
```

Lisp Syntax

```
CMSR:deg-to-rad(dest-field  src-field
                &optional  (signif-len 23)  (exp-len 8))
CMSR:rad-to-deg(dest-field  src-field
                &optional  (signif-len 23)  (exp-len 8))
```

ARGUMENTS

dest_field	A Paris field identifier specifying the field in CM memory to which the result is written. *dest_field* must be in the same VP set as *src_field.*
src_field	A Paris field identifier specifying the field in CM memory from which the value to be converted is taken. *src_field* must be in the same VP set as *dest_field.*
signif_len	The length, in bits, of the significand of the floating-point values in *dest_field* and *src_field.*
exp_len	The length, in bits, of the exponent of the floating-point values in *dest_field* and *src_field.*

DESCRIPTION

For each active processor, `CMSR_rad_to_deg` calculates the degrees equivalent to the number of radians specified in *src_field* and writes the result to *dest_field.*

Conversely, for each active processor, `CMSR_deg_to_rad` calculates the radians equivalent to the number of degrees specified in *src_field* and writes the result to *dest_field.*

SEE ALSO

`CMSR_fe_deg_to_rad`

`CMSR_fe_deg_to_rad`

`CMSR_fe_rad_to_deg`

Chapter 4

Dithering Routines

A workstation that has only a 1-bit (black/white) display, cannot display grayscale continuous tone images unless they are dithered or *halftoned.* *Render contains several functions to support the halftoning of grayscale images.

*Render's halftone routines convert a grayscale image of floating-point or double floating-point values to a 1-bit-per-pixel image suitable for displaying on a black and white monitor. These routines include:

- **`CMSR_u_halftone`**
- **`CMSR_f_halftone`**
- **`CMSR_u_halftone_dot_diffusion`**
- **`CMSR_f_halftone_dot_diffusion`**
- **`CMSR_u_halftone_error_propagation`**
- **`CMSR_f_halftone_error_propagation`**

In addition, two routines are provided that convert color RGB images to grayscale:

- **`CMSR_u_rgb_to_gray`**
- **`CMSR_f_rgb_to_gray`**

Each pixel in a grayscale image represents an intensity level of gray from black to white. For example, an 8-bit grayscale image can display one of 256 intensity levels at each pixel. However, in 1-bit, black and white, displays, each pixel can only be either on or off. Halftoning allows you to transfer grayscale images to 1-bit displays by using varying densities of black and white pixels to approximate the grayscale intensities of the original image.

`CMSR_u_halftone` and **`CMSR_f_halftone`** convert a grayscale image of floating-point or double floating-point values, respectively, to a 1-bit-per-pixel image suitable for displaying on a black and white monitor. These routines are the easiest interface to the

*Renders dithering routines. They use a 5th-order dot-diffusion algorithm to halftone the images.

`CMSR_u_halftone_dot_diffusion` and `CMSR_f_halftone_dot_diffusion` allow you to select the *order* of the dither that is to be applied to your image. The number of shades of gray that can be represented on the 1-bit image is determined by the order of the dithering. A higher-order dither increases the number of intensities but loses some image detail, thus exchanging geometric information for color information.

`CMSR_u_halftone_error_propagation` and `CMSR_f_halftone_error_propagation` allow you to choose an error propagation dither instead of dot diffusion. Error propagation compares the grayscale value to a threshold value to determine whether the corresponding 1-bit pixel should be on or off. But the algorithm then also distributes the "error" of that decision to neighboring pixels. That is, if a pixel is turned off because its color value was just below the threshold, error propagation increases the likelihood that the neighboring pixels are turned on. The effect is to produce patterns of black and white pixels that approximate the grayscale intensities of the original image.

Detailed descriptions of each of these routines follow.

CMSR_u_halftone
CMSR_f_halftone

Converts a grayscale unsigned integer (floating-point) image to a 1-bit, black and white, image.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
   CMSR_u_halftone  (dithered_picture_field, picture_field, len)

CMSR_field_id_t   dithered_picture_field,  picture_field;
unsigned int      len;

void
   CMSR_f_halftone
               (dithered_picture_field, picture_field, signif_len, exp_len)

CM_field_id_t dithered_picture_field,  picture_field;
unsigned int signif_len,  exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-fort.h'

   SUBROUTINE  CMSR_U_HALFTONE (dithered_picture_field, picture_field, len)

   INTEGER  dithered_picture_field,  picture_field
   INTEGER  len

   SUBROUTINE CMSR_F_HALFTONE
&             (dithered_picture_field, picture_field, signif_len, exp_len)

   INTEGER dithered_picture_field, picture_field
   INTEGER signif_len, exp_len
```

Lisp Syntax

```
CMSR:u-halftone  (dithered-picture-field picture-field len)

CMSR:f-halftone  (dithered-picture-field, picture-field, signif-len, exp-len)
```

ARGUMENTS

dithered_picture_field
A 1-bit field. The halftoned image is written to this field.

picture_field A field containing the image to be halftoned.

For **`CMSR_u_halftone`**, the first 8 bits of this field are used to compute *dithered_picture_field.*

For **`CMSR_f_halftone`** the color values must be floating-point values in the range of 0 to 1. Negative values or values greater than 1 are clipped to this range.

len For **`CMSR_u_halftone`**, the length of *picture_field.*

signif_len, exp_len
For **`CMSR_f_halftone`**, the length of the significand and exponent, respectively, of the floating-point values in *picture_field.*

NOTE: *dithered_picture_field* and *picture_field* must be in the same two-dimensional VP set.

DESCRIPTION

`CMSR_f_halftone` and **`CMSR_u_halftone`** convert a grayscale image in *picture_field* to a 1-bit per pixel image in *dithered_picture_field* that you can display on a one-plane black and white monitor.

These functions convert the color values in *picture_field* into patterns of black and white pixels in the destination field that approximate the gray shadings of the original image; darker areas have a greater density of black pixels, and lighter areas include more white pixels. This converted image may then be displayed on a 1-bit display.

Note, however, that some spatial resolution is lost. The boundaries between objects in the converted image are less well defined than in the original, and some image detail may be lost.

Specifically, **`CMSR_u_halftone`** and **`CMSR_f_halftone`** use a 5th-order dot-diffusion algorithm to simulate 64 levels of gray intensity. See **`CMSR_f_halftone_dot_diffusion`** for more detail.

*Render also include two functions, **`CMSR_u_rgb_to_gray`** and **`CMSR_f_rgb_to_gray`**, which convert color images expressed as RGB values to grayscale images.

CMSR_u_halftone_dot_diff CMSR_f_halftone_dot_diff

Converts a grayscale unsigned integer (floating-point) image to a 1-bit, black and white image using a selected dot-diffusion algorithm.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
   CMSR_u_halftone_dot_diff
                       (dithered_picture_field, picture_field, order, len)

CM_field_id_t  dithered_picture_field, picture_field;
int            order;
unsigned int   len;


void
   CMSR_f_halftone_dot_diff  (dithered_picture_field, picture_field, order,
                              signif_len, exp_len)

CM_field_id_t  dithered_picture_field, picture_field;
int            order;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-fort.h'

   SUBROUTINE CMSR_U_HALFTONE_DOT_DIFF
&                (dithered_picture_field, picture_field, order, len)

   INTEGER  dithered_picture_field, picture_field
   INTEGER  order
   INTEGER  len

   SUBROUTINE CMSR_F_HALFTONE_DOT_DIFF
&                (dithered_picture_field, picture_field, order, signif_len, exp_len)

   INTEGER  dithered_picture_field, picture_field
   INTEGER  order
   INTEGER  signif_len, exp_len
```

Lisp Syntax

`CMSR:u-halftone-dot-diff`
(*dithered–picture–field, picture–field, order, len*)

`CMSR:d-halftone-dot-diff`
(*dithered–picture–field, picture–field, order, signif–len, exp–len*)

ARGUMENTS

dithered_picture_field
A 1-bit field. The halftoned image is written to this field.

picture_field A field containing the image to be halftoned.

For **`CMSR_u_halftone_dot_diff`**, the first 8 bits of this field are used to compute *dithered_picture_field.*

For **`CMSR_f_halftone_dot_diff`**, the color values must be floating-point values in the range of 0 to 1. Negative values or values greater than 1 are clipped to this range.

Note: *dithered_picture_field* and *picture_field* must be in the same two-dimensional VP set.

order The desired order of the dither matrix used to halftone *picture_field.* Orders 0 to 5 are supported as follows:

- order = 0: "rounds" the *picture_field* color value to nearest black or white
- order = 1: *picture_field* color values reduced to 4 intensities
- order = 2 or 3: *picture_field* color values reduced to 16 intensities
- order = 4 or 5: *picture_field* color values reduced to 64 intensities

len For **`CMSR_u_halftone_dot_diff`**, the length of *picture_field.*

signif_len, exp_len
For **`CMSR_f_halftone_dot_diff`**, the length of the significand and exponent, respectively, of the floating-point values in *picture_field.*

DESCRIPTION

`CMSR_u_halftone_dot_diff` and **`CMSR_f_halftone_dot_diff`** convert a grayscale image in *picture_field* to a 1-bit-per-pixel image in *dithered_picture_field* using a dither matrix of a specified order. The converted image is suitable for display on a one-plane, black and white monitor.

Grayscale images supply a single color value per pixel that maps to a color look-up table of gray intensity levels. For example, an 8-bit grayscale display provides 256 intensity levels with which to render the object. Each pixel may be set to one of those intensity levels, supporting a continuous range of shading.

Single-bit, "monotone," displays support only two settings for each pixel, on or off, white or black. In order to display a grayscale image on a 1-bit display, the grayscale intensity at each pixel must be resolved to either white or black.

The simplest way to do this is to "round" the gray intensity to the nearest 1-bit value, either white or black. Pixels whose intensity is less than .5 are rounded to black, pixels greater than .5 are rounded to white. This produces an image with sharp contrast and with a considerable loss of image detail, since entire ranges of gray shadings are lost. This is the method used by **`CMSR_u_halftone_dot_diff`** or **`CMSR_f_halftone_dot_diff`** with an order of 0.

Dot-diffusion methods retain more of the visual detail of the image by replacing the grayscale intensities in the original image with a pattern of black and white pixels; the value of a single grayscale pixel is combined with neighboring pixels and "diffused" over an area of the single-bit image. Darker areas have a greater density of black pixels, and lighter areas include more white pixels. The viewer perceives these areas as grayscale intensities.

Lower dither orders retain spatial resolution in the converted image at the expense of visual resolution. That is, they produce simple, high-contrast images in which the location of the objects is clearly defined but details of shading are lost. Higher-order dithers retain visual resolution at the expense of spatial resolution. That is, they retain more of the levels of shading in the original image, giving the impression of a more detailed image; but individual points from the original image are diffused over a larger area, reducing the image contrast and making the boundaries of the original objects less clear.

*Render also include two functions, **`CMSR_f_rgb_to_gray`** and **`CMSR_u_rgb_to_gray`**, that convert color images expressed as RGB values to grayscale images.

CMSR_u_halftone_err_prop
CMSR_f_halftone_err_prop

Converts a grayscale unsigned integer (floating-point) image to a 1-bit image using error propagation.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
   CMSR_u_halftone_err_prop (dithered_picture_field, picture_field, len) ;

CMSR_field_id_t   dithered_picture_field, picture_field;
unsigned int      len;

void
   CMSR_f_halftone_err_prop
               (dithered_picture_field, picture_field, signif_len, exp_len) ;

CM_field_id_t  dithered_picture_field, picture_field;
unsigned int   signif_len, exp_len;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmsr-fort.h'

   SUBROUTINE CMSR_U_HALFTONE_ERR_PROP
&                    (dithered_picture_field, picture_field, len)

   INTEGER  dithered_picture_field, picture_field
   INTEGER  len

   SUBROUTINE CMSR_F_HALFTONE_ERR_PROP
&                    (dithered_picture_field, picture_field, signif_len, exp_len)

   INTEGER  dithered_picture_field, picture_field
   INTEGER  signif_len, exp_len
```

Lisp Syntax

`CMSR:f-halftone-err-prop` (*dithered–picture–field*, *picture–field*, *len*)

`CMSR:d-halftone-error-prop`
(*dithered–picture–field*, *picture–field*, *signif–len*, *exp–len*)

ARGUMENTS

dithered_picture_field	A 1-bit field. The halftoned image is written to this field.
picture_field	A field containing the image to be halftoned. For **`CMSR_u_halftone_err_prop`**, the first 8 bits of this field are used to compute *dithered_picture_field*. For **`CMSR_f_halftone_err_prop`**, the color values must be floating-point values in the range of 0 to 1. Negative values or values greater than 1 are clipped to this range.
len	For **`CMSR_u_halftone_err_prop`**, the length of *picture_field*.
signif_len, *exp_len*	For **`CMSR_f_halftone_err_prop`**, the length of the significand and exponent, respectively, of the floating-point values in *picture_field*.

NOTE: *dithered_picture_field* and *picture_field* must be in the same two-dimensional VP set.

DESCRIPTION

`CMSR_u_halftone_err_prop` and **`CMSR_f_halftone_err_prop`** convert a grayscale image in *picture_field* to a 1-bit-per-pixel image in *dithered_picture_field* and apply *error propagation* to improve the visual quality of the image. The converted image is suitable for display on a one-plane, black and white monitor.

Grayscale images supply a single color value per pixel that maps to a color look-up table of gray intensity levels. For example, an 8-bit grayscale display provides 256 intensity levels with which to render the object. Each pixel may be set to one of those intensity levels, supporting a continuous range of shading.

Single bit, black and white or monotone, displays support only two settings for each pixel, on or off, white or black. In order to display a grayscale image on a 1-bit display, the grayscale intensity at each pixel must be resolved to either white or black.

Error propagation compares the grayscale value to a threshold value to determine whether the corresponding 1-bit pixel should be on or off. But the algorithm then also distributes the "error" of that decision to neighboring pixels. That is, if a pixel is turned off because its color value is just below the threshold, error propagation increases the likelihood that the neighboring pixels are turned on. The effect is to produce patterns of black and white pixels that approximate the grayscale intensities of the original image.

The error propagation applied by these functions produces visual resolution comparable to the 5th-order dot diffusion applied by `CMSR_u_halftone_dot_diffusion` or `CMSR_f_halftone_dot_diffusion`.

CMSR_f_rgb_to_gray
CMSR_u_rgb_to_gray

Converts RGB floating-point (unsigned integer) data to grayscale.

SYNTAX

C Syntax

```
#include <cm/cmsr.h>

void
  CMSR_f_rgb_to_gray
      (gray_field, red_field, green_field, blue_field, signif_len, exp_len)

CM_field_id_t gray_field;
CM_field_id_t red_field, green_field, blue_field;
unsigned int signif_len, exp_len;

void
  CMSR_u_rgb_to_gray(gray_field, red_field, green_field, blue_field, len)

CM_field_id_t gray_field;
CM_field_id_t red_field, green_field, blue_field;
unsigned int len;
```

Fortran Syntax

```
   INCLUDE'/usr/include/cm/cmsr-fort.h'

   SUBROUTINE CMSR_F_RGB_TO_GRAY
&             (gray_field, red_field, green_field, blue_field, signif_len, exp_len)

   INTEGER gray_field
   INTEGER red_field, green_field, blue_field
   INTEGER signif_len, exp_len

   SUBROUTINE CMSR_U_RGB_TO_GRAY
&             (gray_field, red_field, green_field, blue_field, len)

   INTEGER gray_field
   INTEGER red_field, green_field, blue_field
   INTEGER len
```

Lisp Syntax

`CMSR:f-rgb-to-gray`
(*gray-field, red-field, green-field, blue-field, signif-len, exp-len*)

`CMSR:u-rgb-to-gray` (*gray-field, red-field, green-field, blue-field, len*)

ARGUMENTS

gray_field	The grayscale image is written to this field.
red_field, green_field, blue_field	The fields containing the red, green, and blue color values to be converted. For **`CMSR_u_rgb_to_gray`**, the first 8 bits of each field are used to compute *gray_field.* For **`CMSR_f_rgb_to_gray`**, the color values must be floating-point values in the range of 0 to 1. Negative values or values greater than 1 are clipped to this range.
len	For **`CMSR_u_rgb_to_gray`**, the length of *red_field, green_field,* and *blue_field.*
signif_len, exp_len	For **`CMSR_f_halftone_err_prop`**, the length of the significand and exponent, respectively, of the floating-point values in *red_field, green_field,* and *blue_field.*

Note: The fields *gray_field, red_field, green_field,* and *blue_field.* must all be in the same two-dimensional VP set.

DESCRIPTION

For each processor in the current VP set, **`CMSR_f_rgb_to_gray`** and **`CMSR_u_rgb_to_gray`** convert the triplet of red, green, and blue color values in the fields *gray_field, red_field, green_field,* and *blue_field* to a single grayscale value in the field *gray_field.* The converted image is suitable for display on a grayscale monitor or for conversion to a 1-bit monotone image using the *Render dot-diffusion or error propagation routines.

`CMSR_f_rgb_to_gray` and **`CMSR_u_rgb_to_gray`** compute the 'Y' term in the YIQ color model in the same way a black and white TV set turns a color signal into a brightness signal. Specifically, these functions compute a linear combination of RGB as follows: Y = .3 * R + .59 * G + .11 * B

Alphabetical Index of Routines

This index lists the *Render routines alphabetically.

Keyword Index of Routines

This index lists the Image File Interface routines sorted by the key words that appear in their names.
