Geometry Toolbox Manual

Chapter 2

Operator Descriptions

Chapter 2 - Operator Descriptions

A. Introduction

This chapter contains the man1 pages for all programs in the Geometry toolbox, organized alphabetically by object name. Any man1 page can be accessed by typing the following command.

```
% kman <program_name>
```

This information can also be accessed by selecting the *Help* button on the program's graphical user interface, which is displayed when running the program from within cantata, or when running it from the command line using the *-gui* option.

The keywords (-k) option in kman allows the user to provide a keyword string for searching. When invoked with the *-k* option, kman will perform the search and provide a list of the names and short descriptions of programs found by the search.

```
% kman -k "keyword string"
```

If you wish to display only the command line usage information for a program, enter the program name, followed by -U.

```
% <program_name> -U
```

B. Geometry Operators

B.1. color_legend — *Construct Geometry Representing a Color Legend*

Object Information

Category:	*Geometry*
Subcategory:	*Annotations*
Operator:	*Color Legend*
Object Type:	*kroutine*
In Cantata:	*Yes*

Description

color_legend is used to contruct geometry representing a color legend showing the colors contained in a vis colormap. The length and thickness of the legend can be controlled, along with its 2d position in space. The legend can be oriented either vertically or horizontally. An outline around the distinct colors in the legend may be optionally included. The colors in the legend can be reversed.

See Also

kmakecmap

Command Line Arguments

```
Usage for color_legend: Construct Geometry Representing a Color Legend
% color_legend
  -i           (infile)  input vis colormap
  -o           (outfile) output geometry object

  [-name]      (string)  color legend
               (default = color legend)
  [-orient]    (boolean) legend orientation
          1  (vertical),
          or 2  (horizontal)
[default: 1]
  [-reverse]   (boolean) reverse colors?
               (default = false)
  [-outline]   (boolean) make outline?
               (default = true)
  [-length]    (float)   length of legend
               (0 <= length <= 2, default = 1)
  [-thickness] (float)   thickness of legend
               (0 <= thickness <= 0.25, default = 0.05)
  [-xposition] (float)   x position
               (-1 <= xposition <= 1, default = -0.85)
  [-yposition] (float)   y position
               (-1 <= yposition <= 1, default = -0.5)
```

B.2. extents_box — *Generate Geometry Representing Extents Box Around Data*

Object Information

Category: *Geometry*
Subcategory: *Annotations*
Operator: *Extents Box*
Object Type: *kroutine*
In Cantata: *Yes*

Description

The extents box module is used to create a geometry object depicting the bounds of the space occupied in R3 of the input data.

If present, the location segment of the input data must be in R3 (i.e., have X, Y and Z components), while the value portion of the input data may be of an arbitrary dimension. In the instance when input data is provided which has no location segment, the coordinates used are the indeces into the (w,h,d) portion of the value segment. In other words, if the input data set has a value segment of size width=32, height=48, depth=24, elements=3, and time=N, and no location data is present, the resulting box would have it's two corners positioned at (0,0,0) and (31,47,23).

Two different box types may be selected by the user. The first type always results in a rectangular shaped box, regardless of the location data. This box type is called a "min/max" box. The maximum extents of the location (either from the explicit location data or from the scheme outlined above) of the data are computed, and a box is constructed with two corners at those locations. The other box type may result in a non-rectangular box, depending upon the behavior of the explicit location data.

The user may elect to have a "colored" box or a white box. If we consider the three dimensions of the data to correspond to the logical variables (u,v,w), then the box will be colored so that red lines indicate constant (v,w); green lines indicate constant (u,w) and cyan lines indicate constant (u,v).

Additionally, the user may provide an explicit name (text string) for the resulting geometry object. This name may be available to assist in object identification and manipulation in a downstream renderer, depending upon which one is being used.

Command Line Arguments

```
Usage for extents_box: Generate Geometry Representing Extents Box Around Data
% extents_box
  -i           (infile)  input data object
  -o           (outfile) output geometry object

  [-box_style] (boolean) choose box style
          1  (white box),
          or 2  (colored box)
[default: 2]
  [-box_type]  (boolean) choose box type
          1  (min/max 3D box),
          or 2  (shrink-wrap bounding hull)
[default: 1]
  [-obj_name]  (string)  object name
               (default = extents_box)
```

B.3. isosurface — *Create Geometry Representing a 3D Contour within Data*

Object Information

Category:	*Geometry*
Subcategory:	*Data to Geometry*
Operator:	*Isosurface*
Object Type:	*kroutine*
In Cantata:	*Yes*

Description

isosurface accepts as input a 3 dimensional dataset, or a volume of data, and produces a geometry object which represents a contour surface in three dimensions.

The input dataset may or may not have associated location data. If no location data is present, locations are computed as a function of the grid index. For example, if the input data has no location data, and the dimensions of the value segement are: width=32; height=48; depth=24; elements=1; time=1, then the coordinates for the output geometry will be in the range [0..31] in X; [0..47] in Y; and [0..23] in Z.

This module has been written so that it processes data in "slices" of constant W (where the three logical dimensions of data can be indexed using (u,v,w)). The data for a given slice is not "read in" (from the input transport mechanism) until it is actually needed. Output is buffered, and the result is that it is possible to compute isosurfaces for very large datasets.

This module is an implementation of the "marching cubes" algorithm for isosurface generation. The "fixes" identified in later Siggraph papers have not been implemented.

The user may provide an explicit name (text string) for the resulting geometry object. This name may be available to assist in object identification and manipulation in a downstream renderer, depending upon which one is being used.

The geometry produced by this module consists of one or more primitives consisting of disjoint triangles, along with normals (if the user has elected to have them created, which is the default and will result in much better images than when the renderer computes the normals), but no color. In later releases of this module, we will support coloring of the surface according a function of a second variable (dataset).

Command Line Arguments

```
Usage for isosurface: Create Geometry Representing a 3D Contour within Data
% isosurface
  -i              (infile)  input 3d data object
  -level          (float)   contouring level
                   unbounded
  -o              (outfile) output geometry object

  [-do_normals]   (boolean) compute normals?
                  (default = true)
  [-flip_normals] (boolean) reverse normals?
                  (default = true)
  [-name]         (string)  object name
                  (default = isosurface)
```

B.4. kapplytexture — *Assign a Texture to a Geometry Object*

Object Information

Category: *Geometry*
Subcategory: *Texture Operators*
Operator: *Apply Texture*
Object Type: *kroutine*
In Cantata: *Yes*

Description

kapplytexture This routine associates an object containing a texture with a geometry object containing geometry with texture coordinate information. This texture can then be used by a downstream renderer.

Command Line Arguments

```
Usage for kapplytexture: Assign a Texture to a Geometry Object
% kapplytexture
  -i         (infile)  input geometry object apply texture to
  -i2        (infile)  texture object to apply
  -o         (outfile) output geometry object with associated texture
```

B.5. kgmapdata — *Map Data: Map Value Data Through a Visualization Colormap*

Object Information

Category:	*Geometry*
Subcategory:	*Visualization Colormap*
Operator:	*Map Vismap*
Object Type:	*kroutine*
In Cantata:	*Yes*

Description

kgmapdata is used to map value data from an input data object through a vis colormap. Each value in the input value segment will be used as an index into the vis colormap. The output value element vectors will be replaced with the indexed map vectors from the vis colormap.

See Also

kmakecmap

Command Line Arguments

```
Usage for kgmapdata: Map Data:  Map Value Data Through a Visualization Colormap
% kgmapdata
  -i         (infile)  input data object
  -m         (infile)  input vis colormap
  -o         (outfile) output data object
```

B.6. kgridder_2d — *Compute z=f(x,y) over a user-defined grid given scatter data.*

Object Information

Category:	*Geometry*
Subcategory:	*Region Operators*
Operator:	*Grid 2D Data*
Object Type:	*kroutine*
In Cantata:	*Yes*

Description

kgridder_2d is used to fit a surface through scatter data. Input to this module consists of a set of points, and output consists of a 2D array of function values. A choice of several numerical methods are provided, each yielding a possibly different surface.

At present, the methods available are as follows.

Fourier Analysis (default)

The input data is assumed to be an sampled set of points from a function which can be reconstructed using 2D fourier analysis. Given the input values, the frequency components of the estimated function are computed. The values at each output grid point are computed using this function.

Inverse Distance Method

At each output grid point, all points that lie within a user-defined search radius are averaged together using an inverse distance formula. The user is provided with an exponent which is applied to the denomimator of this forumula. Increasing the value of the exponent will tend to cause points farther away to have less influence. Decreasing the value of the exponent will tend to make the surface a bit smoother.

Common problems:

Search Radius Too Small

The search radius must be large enough so as to have some overlap with adjacent output grid points, and large enough to include input data. In areas of sparse input data, often the output grid will have it's value set to "UNDEFINED", as no input points are close enough to influence this grid point. The two ways to work around this are to use a larger search radius (the radius is in the same space as the input data, by the way), or to use a different method which works better with sparse data, such as Fourier Analysis.

Results of Fourier Analysis too Extreme

On occasion, this method will produce values which vary wildly from the input, resulting in surfaces which have marked peaks or valleys, seemingly going off into infinity. This is produced by extrapolation into a region where there is no data, based upon a partial derivative computed at two data points close together. The only way to work around this is to include more points in the surface computation, or to use a method which doesn't use partial derivatives, such as the Inverse Distance Method.

COMING SOON: Bivariate Analysis

(Fall 1994)

The extents of the output grid, by default, are computed for you. If this isn't acceptable, then set the min/max values for the grid yourself.

Command Line Arguments

```
Usage for kgridder_2d: Compute z=f(x,y) over a user-defined grid given scatter data.
% kgridder_2d
  -i             (infile)  input data object
  -o             (outfile) output data object

  [-grid_size_x] (integer) value indicating x-resolution of output grid
                 (grid_size_x > 0.0, default = 20)
  [-grid_size_y] (integer) value indicating y-resolution of output grid
                 (grid_size_y > 0.0, default = 20)

  [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]

    [Optional Mutually Inclusive Group - Specify all or none of the 4 options:]
      [-inverse_distance_method]          (flag)   use inverse-distance method
      [-inverse_distance_radius]          (double) user-defined radius, inside which points will be averaged
                                          (inverse_distance_radius > 0.0, default = 1)
      [-inverse_distance_exponent]        (double) exponent to use in denominator of inversion equation
                                          (inverse_distance_exponent > 0.0, default = 1)
      [-inverse_distance_undefined_value] (double) the value to use when the output grid point is not defined
                                          (inverse_distance_undefined_value > 0.0, default = 1)

    [Optional Mutually Inclusive Group - Specify all or none of the 2 options:]
      [-idsfft_method]  (flag)   use 2d fourier analysis flag
      [-idsfft_npoints] (double) number of points to use in computing partial derivatives
                        (idsfft_npoints > 0.0, default = 5)

  [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
    [-compute_grid_min_max] (flag) Compute output grid min/max from input location data

    [Optional Mutually Inclusive Group - Specify all or none of the 5 options:]
      [-user_grid_min_max] (flag)   user defined grid min/max vertices, in 2D
      [-user_grid_x_min]   (double) X min coordinate for interpolation grid
                           (unbounded, default = 0)
      [-user_grid_y_min]   (double) Y min coordinate for interpolation grid
                           (unbounded, default = 0)
      [-user_grid_x_max]   (double) X max coordinate for interpolation grid
                           (unbounded, default = 1)
      [-user_grid_y_max]   (double) Y max coordinate for interpolation grid
                           (unbounded, default = 1)
```

B.7. kimportfacet — *Import FACET data*

Object Information

Category: *Geometry*
Subcategory: *Data to Geometry*
Operator: *Import Facet Data*
Object Type: *kroutine*
In Cantata: *Yes*

Description

kimportfacet will input a file containing facet information into a data object using geometry services.

The file will contain one or more "big parts", with each big part consisting of a list of XYZ points. For each big part, there will be one or more "small parts", with each small part consisting of a number of facets. Each facet is simply a triangle specified by three index numbers into the vertices list contained in the big part.

The resulting geometry will consist of a series of disjoint triangle primitives, with one primitive for every small part. Note that each disjoint triangle primitive contains several triangles.

All triangles in the format are specified to be RH. RH normals will be generated for each triangle.

The following example illustrates the format of a facet file. The explanations, separated by '->' should not appear in the facet file.

```
information line
        1               <- 1 big part
square with fin             <- name of big part
0, 0.00 1.00 0.00 0.00        <- mirror, ax+by+cz-d = 0
        5               <- #nodes in big part
0.00  0.00  0.00            <- xyz of node 1
5.00  0.00  0.00              separated by at least 1 space
5.00  5.00  0.00
5.00  0.00  0.00
5.00  5.00  4.00
        2               <- number of small parts
square                   <- name of 1st small part
        2               <- number of facets in 1st small part
1 2 3   1 0011 1            <- facet 1 has nodes 1,2,3 followed by
1 3 4   1 0011 2              material index, group #, facet #
fin                    <- name of 2nd small part
        2               <- number of facets in 2nd small part
1 3 5   1 0011 1
```

The material index, group #, and facet # are consumed but not used. In future releases, the material index may be used to add color to the geometry.

The facet file format is also known as the General Dynamics ACAD format.

Command Line Arguments

```
Usage for kimportfacet: Import FACET data
% kimportfacet
  -i          (infile)  input facet file
  -o          (outfile) output geometry object

  [-n]        (string)  object name
              (default = facets)
```

B.8. kmakecmap — *Generate Colormap: Create a Colormap for Mapping Float Data*

Object Information

Category: *Geometry*
Subcategory: *Visualization Colormap*
Operator: *Create Vismap*
Object Type: *kroutine*
In Cantata: *Yes*

Description

The kmakecmap module is used to create a "visualization colormap." A "visualization colormap" is different from a garden variety colormap in one important way. The visualization colormap uses a part of the location segment to indicate how colors are extracted from the map. In normal colormaps, a "pixel" value is used as an index into the colormap, producing some new value(s). Using this scheme, it is not possible to have perform mapping functions on floating point or large integer data.

The visualization colormap uses values in the location segment which effectively "bracket" entries in the map. In other words, the user may define a range of values which map to a particular map entry, thus producing non-linear and discontinuous mapping functions.

At present, this module supports only a linear range which gets spread evenly over the entire colormap. The range is implemented using floating point values. There is one location segment entry per map entry, which defines the minimum value corresponding to that entry. All values greater than or equal to a particular bracket value, but less than the next one, will define the range for a given map entry.

The color information in the colormap may be specified in either RGB or HSV space, each of which has it's own advantages and disadvantages. The alpha, or opacity, channel is optionally specified seperately from the color information.

Many of the visualization modules in the Geometry toolbox make use of the visualization colormap. A magic cookie string is placed into the visualization colormap, and is detected by these modules. These modules will not work (at present) if any other type of map is presented to them.

Command Line Arguments

```
Usage for kmakecmap: Generate Colormap:  Create a Colormap for Mapping Float Data
% kmakecmap
  -outf      (outfile) output vis colormap

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:

    [Optional Mutually Inclusive Group - Specify all or none of the 7 options:]
      [-use_hsv] (flag)  color parameters in HSV space
      [-hs]      (float) hue start value
                 (0 <= hs <= 1, default = 0)
      [-he]      (float) hue end value
                 (0 <= he <= 1, default = 1)
```

```
    [-ss]      (float) saturation start value
               (0 <= ss <= 1, default = 1)
    [-se]      (float) saturation end value
               (0 <= se <= 1, default = 1)
    [-vs]      (float) value start value
               (0 <= vs <= 1, default = 1)
    [-ve]      (float) value end value
               (0 <= ve <= 1, default = 1)

  [Optional Mutually Inclusive Group - Specify all or none of the 7 options:]
    [-use_rgb] (flag)  color parameters in RGB space
    [-rs]      (float) red start value
               (0 <= rs <= 1, default = 0)
    [-re]      (float) red end value
               (0 <= re <= 1, default = 1)
    [-gs]      (float) green start value
               (0 <= gs <= 1, default = 1)
    [-ge]      (float) green end value
               (0 <= ge <= 1, default = 1)
    [-bs]      (float) blue start value
               (0 <= bs <= 1, default = 1)
    [-be]      (float) blue end value
               (0 <= be <= 1, default = 1)

[-n]       (integer) number of entries in the vis colormap.
           (2 <= n <= 512, default = 256)
[-minval]  (float)   value used for minimum bracket value in vis colormap
           (unbounded, default = 0)
[-maxval]  (float)   value assioated with maximum bracket value in map.
           (unbounded, default = 0)

[Optional Mutually Inclusive Group - Specify all or none of the 3 options:]
  [-use_alpha] (flag)  include opacity info in alpha channel of colormap
  [-as]        (float) alpha start value
               (0 <= as <= 1, default = 0)
  [-ae]        (float) alpha end value
               (0 <= ae <= 1, default = 1)
```

B.9. kprgeom — *Print ASCII Report of Geometry Input*

Object Information

Category: *Geometry*
Subcategory: *Information*
Operator: *Print Geometry*
Object Type: *kroutine*
In Cantata: *Yes*

Description

Kprgeom is a tool used for printing the contents of a file created by geometry services. It is analagous to the "kprdata" module. Using the command line flags, the user may choose to have only the location data, the normals (or radii if the input file is spheres), or the colors, or any combination of those printed as output.

At present, only the following primitive types are supported: disjoint and connected line segments, disjoint and connected triangles, spheres, quadmeshes, and octmeshes. All other primitives will not be printed, although it some of the global object characteristics (name, etc) will be printed.

In future releases, the full gamut of geometry primitives will be supported.

Command Line Arguments

```
Usage for kprgeom: Print ASCII Report of Geometry Input
% kprgeom
  -i            (infile)  input geometry object

  [-loc]        (boolean) print locatons?
                (default = true)
  [-norm_radii] (boolean) print normals or radii?
                (default = true)
  [-color]      (boolean) print colors?
                (default = true)
  [-att]        (boolean) print attributes?
                (default = true)
  [-tc]         (boolean) print texture coords?
                (default = true)
  [-o]          (outfile) output ASCII text file
                (default = null)
```

B.10. kslicer — *Extract (N-1)D Region from (N)D Object*

Object Information

Category:	*Geometry*
Subcategory:	*Region Operators*
Operator:	*Ortho Slice*
Object Type:	*kroutine*
In Cantata:	*Yes*

Description

This routine will generate an (N-1)D data set from an N-D data set. This is done by selecting a plane to slice along and a slice number into that plane. Planes are specified as being orthogonal to either the width, height, depth, time, or element dimenstions of the input data object. The slice plane can either be specified as a specific plane number, or a percentage depth into the data along the specified slice plane.

The user also has the option of reorienting the slice to be aligned with the width-height plane. This will make it suitable for direct viewing with any of the image viewing programs. If the original data set contains no location data, then explicit corner markers may be added to position the data in space.

Examples

kslicer -i volume:ytterbium -o sliced.ytterbium -reo -p 2 -ps 0.4 will slice the ytterbium volume along height at a plane that is 40% deep into the height dimension. The resulting plane will be reoriented to face the width-height plane.

Restrictions

Rectilinear data will not be presereved when slicing. When slicing data with uniform location data, the reorientation feature will be ignored.

Command Line Arguments

```
Usage for kslicer: Extract (N-1)D Region from (N)D Object
% kslicer
  -i         (infile)  input file to slice
  -o         (outfile) output file containing slice

  [-p]       (boolean) choose orthogonal slice plane
          1  (slice along constant width),
          2  (slice along constant height),
          3  (slice along constant depth),
          4  (slice along constant time),
          or 5  (slice along constant element)
[default: 1]
  [-reo]     (flag)    reorient slice to face image plane
  [-loc]     (flag)    force explicit location data

  [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
    [-ps] (float) slice at percentage depth on slice plane
          (0 <= ps <= 1, default = 0.5)
    [-s]  (float) slice at position on slice plane
          (s >= 0.0, default = 0)
```

B.11. octmesh — *Create an Octmesh Geometry Object from 3D Data*

Object Information

Category: *Geometry*
Subcategory: *Data to Geometry*
Operator: *Octmesh*
Object Type: *kroutine*
In Cantata: *Yes*

Description

octmesh is used to create a "mesh" out of three-dimensional value data. The value data must either have an element size of 4 and contain RGBalpha values, or an element size of 1, in which case it will be used as map indicies into a vis colormap. Location data must also be present in the input file. Location data is required to exists in R3 (i.e., each location vector must have and X, Y and Z component).

A vis colormap may be provided for coloring the octmesh vectors. The generated mesh will then contain the color vectors determined by mapping the original value points through the vis colormap.

See Also

quadmesh, kmakecmap

Command Line Arguments

```
Usage for octmesh: Create an Octmesh Geometry Object from 3D Data
% octmesh
  -i         (infile)  input 3d data for octmesh
  -o         (outfile) output geometry octmesh

  [-cmap]    (infile)  input vis colormap
             (default = null)
  [-name]    (string)  object name
             (default = octmesh)
```

B.12. plot_color — *Create Polylines from ASCII XYZ data and a Vis Cmap*

Object Information

Category:	*Geometry*
Subcategory:	*Data to Geometry*
Operator:	*Plot Color*
Object Type:	*kroutine*
In Cantata:	*Yes*

Description

plot_color will read in an ASCII file of coordinate information and color indices and produce a series of connected polyline primitives.

The ASCII input should contain blocks of information, where each block consists of a number indicating how many vertices will be contained in the polyline and then a series of XYZ points and color indices, with one for each vertex. The format of the input ASCII file is illustrated below :

```
number of vertices in first connected polyline
X value   Y value   Z value   color index
X value   Y value   Z value   color index
X value   Y value   Z value   color index
   .         .         .         .
   .         .         .         .
   .         .         .         .
X value   Y value   Z value   color index
number of vertices in second connected polyline
X value   Y value   Z value   color index
   .         .         .         .
   .         .         .         .
   .         .         .         .
```

A single connected polyline primitive will be created for each successive block of information. A color will be stored for each vertex in the line. The color is determined from the provided vis colormap; the ASCII color indicies are used to index into the vis colormap.

See Also

plot_xyz

Command Line Arguments

```
Usage for plot_color: Create Polylines from ASCII XYZ data and a Vis Cmap
% plot_color
  -i         (infile)  input ASCII data containing polyline points
  -cmap      (infile)  Input vis colormap
  -o         (outfile) output geometry object

  [-name]    (string)  object name
             (default = plot_color)
```

B.13. plot_xyz — *Create Polylines from ASCII XYZ Data*

Object Information

Category:	*Geometry*
Subcategory:	*Data to Geometry*
Operator:	*Plot XYZ*
Object Type:	*kroutine*
In Cantata:	*Yes*

Description

plot_xyz will read in an ASCII file of coordinate information and produce a series of connected polyline primitives.

The ASCII input should contain blocks of information, where each block consists of a number indicating how many vertices will be contained in the polyline and then a series of XYZ points, with one for each vertex. The format of the input ASCII file is illustrated below :

```
number of vertices in first connected polyline
X value   Y value   Z value
X value   Y value   Z value
X value   Y value   Z value
   .         .         .
   .         .         .
   .         .         .
X value   Y value   Z value
number of vertices in second connected polyline
X value   Y value   Z value
   .         .         .
   .         .         .
```

. . .

A single connected polyline primitive will be created for each successive block of information. The resulting geometry will not contain any color data.

See Also

plot_color

Command Line Arguments

```
Usage for plot_xyz: Create Polylines from ASCII XYZ Data
% plot_xyz
  -i          (infile)  input ASCII data containing polyline points
  -o          (outfile) output geometry object

  [-name]     (string)  object name
              (default = plot_xyz)
```

B.14. quadmesh — *Create a Quadmesh Geometry Object from 2D Data*

Object Information

Category:	*Geometry*
Subcategory:	*Data to Geometry*
Operator:	*Quadmesh*
Object Type:	*kroutine*
In Cantata:	*Yes*

Description

quadmesh is used to create a "mesh" out of two-dimensional value data. The value data must either have an element size of 4 and contain RGBalpha values, or an element size of 1, in which case it will be used as map indicies into a vis colormap. Location data must also be present in the input file. Location data is required to exists in R3 (i.e., each location vector must have and X, Y and Z component).

A vis colormap may be provided for coloring the quadmesh vectors. The generated mesh will then contain the color vectors determined by mapping the original value points through the vis colormap.

The resulting quadmesh will contain location data, color data, and normal data. Texture coordinates may be optionally generated which can be used to map vertex points on the quadmesh into colors on a texture map. A texture map can be associated later using the kapplytexture operator. Note that the use of a vis colormap and texture coordinates are mutually exclusive.

See Also

octmesh, kmakecmap, kapplytexture

Command Line Arguments

```
Usage for quadmesh: Create a Quadmesh Geometry Object from 2D Data
% quadmesh
  -i         (infile)  input 2d data for quadmesh
  -o         (outfile) ouput geometry quadmesh

  [-name]    (string)  object name
             (default = quadmesh)

  [Optional Mutually Exclusive Group - Specify one or none of the 3 options:]
    [-cmap] (infile) input vis colormap
            (default = null)

    [Optional Mutually Inclusive Group - Specify all or none of the 5 options:]
      [-tc2d]   (flag)   generate 2D texture coords
      [-umin2d] (double) umin
                (0 <= umin2d <= 1, default = 0)
      [-umax2d] (double) umax
                (0 <= umax2d <= 1, default = 1)
      [-vmin2d] (double) vmin
                (0 <= vmin2d <= 1, default = 0)
      [-vmax2d] (double) vmax
                (0 <= vmax2d <= 1, default = 1)

    [Optional Mutually Inclusive Group - Specify all or none of the 7 options:]
      [-tc3d]   (flag)   generate 3D texture coords
      [-umin3d] (double) umin
                (0 <= umin3d <= 1, default = 0)
      [-umax3d] (double) umax
                (0 <= umax3d <= 1, default = 1)
      [-vmin3d] (double) vmin
                (0 <= vmin3d <= 1, default = 0)
      [-vmax3d] (double) vmax
                (0 <= vmax3d <= 1, default = 1)
      [-wmin3d] (double) wmin
                (0 <= wmin3d <= 1, default = 0)
      [-wmax3d] (double) wmax
                (0 <= wmax3d <= 1, default = 1)
```

B.15. rmonster — *Interactive Geometry Visualization*

Object Information

Category: *Geometry*
Subcategory: *Visualization*
Operator: *RenderMonster (TM)*
Object Type: *xvroutine*
In Cantata: *Yes*

Description

The RenderMonster is the Khoros software geometry renderer. Implemented as an Xvroutine, it may be included as part of a Cantata workspace or run from the command line.

The RenderMonster accepts input in the form of geometry created by the visualization modules in the Geometry Toolbox, and produces images, displaying them, and interacting with the user.

To use RenderMonster, first compute some geometry with one of the modules which produce geometry (extents_box, isosurface, etc). The input parameters to RenderMonster should use the name of a file (stream, shared memory, etc.) that contains geometry data. Note that if you are working on a display capabale of true color, then you should start RenderMonster in 24-bit mode. When RenderMonster comes up, it will pop up two windows. One window is the control panel for the RenderMonster, the second is the image display window (unless image display has been supressed from the command line). The image display window will initially be the "background color", that is, no geometry is initially rendered. Press on the "render" button on the control panel and any input geometry will be rendererd.

The default position for the camera is roughly (0,0,40) looking in the -Z direction towards the XY plane.

Direct interaction with the rendered image is available with the mouse. Pressing different mouse buttons on the image and moving the mouse will perform different transformations. When a transformation is initiated by pressing on a mouse button, you will see a bounding box representing the extents of the geometry. Moving the mouse will transform this bounding box. When the desired transformation has been performed, releasing the mouse button will initiate a rendering at the new transformation.

Rotations

Pressing the middle mouse button on the rendered image and moving the mouse will allow you to rotate the geometry. When you press down on the middle mouse button, you will see a circle drawn on the image. Moving the mouse up and down within this circle will perform an X rotation. Moving the mouse left and right within this circle will perform a Y rotation. Moving the mouse around the outside of the circle will perform a Z rotation.

Scaling

Pressing the middle mouse button while holding down the shift key on the rendered image and moving the mouse will allow you to scale the geometry. Moving the mouse to the left will scale downward and moving the mouse up and to the left will scale upward.

Translation

Pressing the right mouse button the rendered image and moving the mouse will allow you to translate the geometry in the X and Y directions. Translations in the Z direction is accomplished by pressing the right mouse button and the shift key.

RenderMonster Control Panes

A number of different control panes are present on the interface for RenderMonster. These are described below.

Objects Pane

The Objects pane provides for control over different object-level properties; selecting an object from the pulldown gives you control over that particular object. Selecting the root object implies that property changes apply globally to all objects. Different object shaders are selected from the pull-down menu here. Note that selecting objects here also indicates which object you wish to transform. You can directly control object transformations, either from the Transformations Pane or directly from the image.

Cameras Pane

The Cameras pane provides control over the camera parameters, such as the eye point, the look-at point, the up vector, the frustrom of vision (which defines the angle frustrom in degrees), as well as the front and back clipping planes.

Lights Pane

The Lights pane provides control to the array of lights which are provided for use in illuminating the scene. Within the light array, the user is free to set the light type be directional, bidirectional or point-light. There is one global ambient light which is separate from this array of lights. The initial configuration of the 3-element light array is 3 directional light sources, only one of which is turned on.

Imaging Pane

The Imaging pane contains controls for changing the image size, the image display, and the background color. Currently the only controls which are operational on this pane are the controls for changing the background color.

Transformations Pane

The Transformations pane provides control over the transformation of the geometry data, the viewpoint, the focal point, or the lights. Controls for rotation, translation, and scaling are present here. The transformations occur around the specified center point.

Input Pane

The Input pane contains four input lines. these can be used to specify files containing geometry data to be rendered. If input files were specified on startup, the files specified here will replace them.

Output Pane

The Output pane contains a selection for saving the currently displayed image. This feature is currently unimplemented.

Shaders Available

A number of different shaders are supported in the RenderMonster. Typical surface shaders are no-light (where the color of the surface doesn't depend upon interactions with the lights in the scene), flat (where the polygons are assumed to be of a constant shade), gouroud (where colors are interpolated between vertices across the face of the polygon) and phong (where the surface normal is interpolated across the face of the polygon). At present, only no-light, flat and gouroud shaders are supported.

For volume rendering, three shader types are supported; these are "voxel-dot" (where a single point, possibly transparent, is produced for each voxel), "splat homogeneous" (where the 2d voxel footprint is approximated with a rectangle of constant shade and opacity) and "splat homogeneous gauss" (where the 2d voxel footprint is approximated with a rectangle of constant shade, but the opacity in the rectangle is approximated with a 2D gaussian function.

One note about triangle shading : Most shading models assume per-vertex normals, and all shading calculations are performed at each vertex, with the triangle's shade interpolated across the face of the triangle. Geometry Services, in contrast, provides a wide range of combinations of per-vertex/per-facet data; color and normals may be specified either per-vertex or per-facet. By necessity, some of the potential speed increases are lost when per-face normals or colors are used as these specifications imply that shared vertices must have shading calculations performed on a per-facet basis. The portions of the rendering which is shared, regardless of the per-facet or per-vertex specification, is the geometric transformation (this calculation is less expensive than the cost of shading a vertex).

Supported Geometry

There are a number of primitives which can be rendered directly by the RenderMonster.

Disjoint Triangles

Disjoint triangles, where every triangle is represented by three vertices, can be rendered by RenderMonster. Triangles with no colors, per-vertex colors, and per-face colors are supported. If no colors are provided, the global object color is used. If no normals are provided, then normals are generated on input. Texture coordinates are not interpreted for triangles.

Connected Triangles

Connected triangles, also known as triangle strips or triangle meshes, have the benefit of reducing vertex redundancy, achieving up to a threefold reduction in space over disjoint triangles by employing vertices which are shared between adjacent triangles. Up to a threefold performance increase may be realized due to the fact that three times fewer per-vertex operations are performed. For the September 1 1994 release of Khoros 2.0, connected triangle primitives are silently converted, within the renderer at read-in time, to disjoint triangle primitives if the triangles have per-face colors or normals or has no normals.

Spheres

Spheres, where each sphere is represented by a vertex are supported. The spheres are actually tesselated into triangles for rendering. If the provided spheres data has no colors or radii, the global object color and a default radius will be used.

Disjoint and Connected Lines

Both disjoint and connected lines are understood by RenderMonster.

Quadmeshes

Quadmeshes are supported. Currently, provided colors must be per-vertex and consist of RGBA vectors. If no normals are provided, then normals are generated on input. Texture coordinate information is not yet utilized.

Octmeshes

Octmeshes are supported. Currently, provided colors must be per-vertex and consist of RGBA vectors. Texture coordinate information is not yet utilized.

STAY TUNED

In future releases, texture mapping will be supported, along support for rendering text, directed points, and cylinder primitives.

KNOWN BUGS

The following list (probably incomplete) enumerates the known bugs in the RenderMonster.

1. The output mechanism is not fully implemented. Entering a filename using the -o option will result in a 3-band byte image being created. Ultimately, the resulting image (what you get when you specify something with the -o option) will be a 4-band, floating point image containing RGBA in the range 0..1.

2. Clipping of the interactive transformation bounding box is not occurring, causing strange things when the bounding box is rotated or scaled past the eye position.

3. Manipulating controls on the Transformations Pane is not well synchronized with the interactive transformations.

Other bugs are present, but lack of time prohibits listing them all.

Examples

rmonster -i geometry:ytter_iso2

See Also

geometry

Command Line Arguments

```
Usage for rmonster: Interactive Geometry Visualization
% rmonster
  [-i]                    (infile)  Input Object
                          (default = null)
  [-i2]                   (infile)  Input Object2
                          (default = null)
  [-i3]                   (infile)  Input Object2
                          (default = null)
  [-i4]                   (infile)  Input Object2
                          (default = null)
  [-image_display_choice] (boolean) Logical toggle selection
          1  (3-3-2 image),
          2  (24bit image),
          or 3  (No Display)
[default: 1]
  [-width]                (integer) width
```

```
                        (32 <= width <= 4096, default = 256)
[-height]               (integer) height
                        (32 <= height <= 4096, default = 256)
[-o]                    (outfile) Resulting Image
                        (default = null)
```

B.16. sphere_maker — *Produce Spheres Whose Radii are Determined from Data*

Object Information

Category:	*Geometry*
Subcategory:	*Data to Geometry*
Operator:	*Spheres*
Object Type:	*kroutine*
In Cantata:	*Yes*

Description

The sphere maker takes as input a data object containing both value and location data and produces a set of spheres where each location vector in the location data determines the center of a sphere. The value in the value data corresponding to the location vector will be used to determine the sphere's radius. Location data is required to exists in R3 (i.e., each location vector must have and X, Y and Z component).

Command Line Arguments

```
Usage for sphere_maker: Produce Spheres Whose Radii are Determined from Data
% sphere_maker
  -i              (infile)  input data object
  -o              (outfile) output geometry object

  [-cmap]         (infile)  input vis colormap
                  (default = null)
  [-name]         (string)  object name
                  (default = spheres)
  [-radius_scale] (float)   scale factor for radii
                  (radius_scale >= 0.0, default = 1)
```

This page left intentionally blank

Table of Contents

This page left intentionally blank