spatquant

spatial quantization of photons
Edition 1.0.1, for version 1.0.1
3 August 2010

Diab Jerius, Dan Nguyen

Table of Contents

1 Copying

2 Introduction

`spatquant` quantizes the positions of a stream of photons according to a gridding method. It acts as a filter, either accumulating the input photons in bins and writing out a single photon per bin, or changing the position of each incoming photon according to the binning scheme and writing it out. It can also produce a two dimensional image of the binned photons which can be output in a variety of formats.

3 Overview

`spatquant` is a program which bins rays spatially. The rays can be accumulated (as in a detector) and the resultant weight of each bin written out as a ray, or they can be passed along, individually, with each ray's position changed to that of the appropriate bin. In either mode, it is possible to create a two dimensional image of the rays and store it.

3.1 Input Format

`spatquant` accepts as input rays encoded in the `bpipe` format.

3.2 Ray Filtering

Rays pass through `spatquant` in one of several paths.

- They may pass through completely unchanged. This is useful when one wishes to make a snapshot (i.e. an image) of the ray distribution at some point in the ray stream without disturbing the flow of the stream.
- Only those that would be binned are passed through (i.e. as in a mask operation). Masking is useful when the bins are really apertures, with spaces between bins which are opaque, and one wishes to process only those rays which pass through the non-opaque parts.
- They may be accumulated. Accumulating rays is appropriate when only the weight of the rays in each bin will be used. The rays which accumulate in a bin are represented in the output stream as a single ray with their combined weights; all other information is discarded. This greatly cuts down on the information flowing down the stream.
- Their positions may be changed to that of the center of the bin into which they would have been accumulated. This is useful when additional information (such as ray energy) is needed during down stream processing.
- They may be completely blocked. This mode is only useful if you're creating an image (see Section 3.5 [Images], page 7);

3.3 Binning

Rays are binned according to their positions in the xy plane. It is assumed that all of the rays have the same z coordinate; they are *not* projected there by `spatquant`. If rays are accumulated, the only information retained is the sum of the weights of the rays which fall within a bin; all other information is discarded. The weight of accumulated rays may optionally be divided by the area of their bins, providing their surface density (see Section 4.2 [Binning Parameters], page 9, `norm_area`).

3.3.1 Grid Schemes

`spatquant` has been designed to make it easy to implement different types of spatial gridding (rectilinear, polar, polar-log). While each of these schemes is two dimensional, each dimension is different depending upon the scheme. `spatquant` uses two dimensions, i and j, which are mapped by each of the schemes onto their particular grid. i and j have integral values, and represent the indices of the bins in their respective dimensions. j varies most

quickly in the data stored in memory, and is treated as a "dependent" variable. There may be different numbers of bins in the j dimension for any value of i. As an example, in a polar grid, i would be the index mapped onto radius and j that onto angle. The density of bins at small radii may be made different than at large radius by changing the extent of j.

3.3.1.1 Rectilinear Grid, Rectangular bins

Bins are rectangular in shape, with independent height and width dimensions, and are placed in a rectangular grid. There is no space between bins. i maps onto the rows (y), and j maps onto the columns (x). Bins limits are inclusive of their lower edges and exclusive of the higher edges; i.e., $n \leq x < n + \textit{binwidth}$.

When in sparse mode, the grid is positioned so that the center of the bin with $(i, j) = (0, 0)$ is at the user provided grid center coordinates (see Section 4.2 [Binning Parameters], page 9). The grid is positioned similarly if `spatquant` is in core mode and the numbers of requested rows and columns are odd. If the number of columns is even, the x coordinate is at `xcent` + `x_sz`/2. If the number of rows is even, the y coordinate is at `ycent` + `y_sz`/2.

3.3.1.2 Rectilinear Grid, Circular bins

Bins are circular in shape and are placed on a rectangular grid. The grid spacing is independent of the bin size, with the limitation that bins cannot overlap. i maps onto the rows (y), and j maps onto the columns (x). Rays which fall on bin edges are included in the bins.

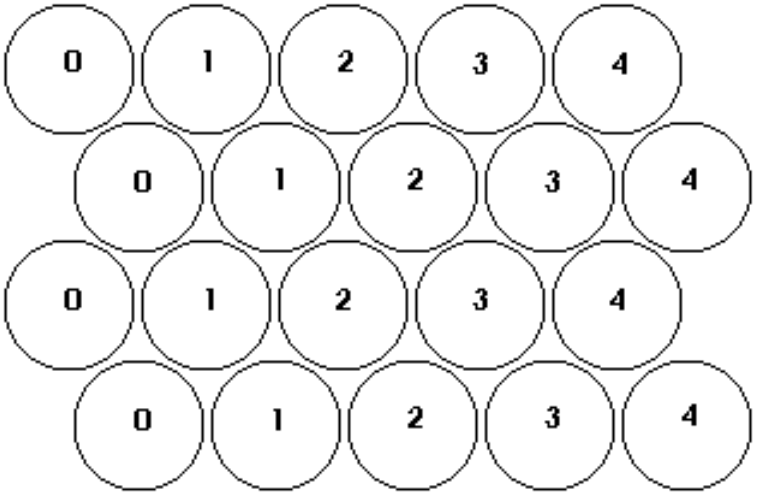
When in sparse mode, the grid is positioned so that the center of the bin with $(i, j) = (0, 0)$ is at the user provided grid center coordinates (see Section 4.2 [Binning Parameters], page 9). The grid is positioned similarly if `spatquant` is in core mode and the numbers of requested rows and columns are odd. If the number of columns is even, the x coordinate is at `xcent` + `x_sp`/2. If the number of rows is even, the y coordinate is at `ycent` + `y_sp`/2.

3.3.1.3 Circular Grid, Wedge bins

Bins are wedge in shape, and are placed in a circular grid. The index i, j respectively denote the radial, angular index of the grid. (see Section 4.2 [Binning Parameters], page 9).

3.3.1.4 Hexagonal Grid, Circular bins

Bins are circular in shape, and are placed in a hexagonal grid. The grid is set up like this (the numbers represent the indices of the "columns"):



Note that the "columns" follow the wiggles and that the "rows" are as in a rectilinear grid. The diameter of the bins can be no larger than the spacing between bin centers. The coordinates of the central bin are determined similarly to what is done for the rectilinear grid, rectangular bin scheme, with allowances for the extra width of the grid due to the staggering of the rows.

3.3.2 Grid Limits

It is often the case that the user is interested in *all* of the rays which arrive at a particular z-plane (regardless of their position on the xy plane). To accomodate this situation, as well as the ordinary case where the grid has edges (as in the simulation of a detector), `spatquant` bins data in two modes. If the user wishes to clip the rays outside a specified rectangle, the program operates in *core* mode, in which it creates a 2D matrix in memory (hence the term "core"). If no clipping is to be done, `spatquant` operates in *sparse* mode, where it keeps track only of those bins which contain rays. This mode relieves the user of needing to know how large an area the ray distribution will cover. In addition, if the amount of memory required for core mode is larger than a user specified limit, sparse mode is used automatically.

3.4 Ray Output Formats

The rays are written as bpipe formatted ray streams.

3.5 Images of the Ray distribution

An image of the photon distribution can be output, independent of the type of filtering of the rays. The image marix may be output as either ASCII text, an `/rdb` compatible database, or as a binary file. In all cases it is possible to prepend a header to the output describing the size of the image (the number of rows and columns) and its dimensions. The binary

image may be written in either single or double precision, and may be made compatible with either C or Fortran (unformatted records) style binary input. See Appendix A [Image Formats], page 15, for more details. `spatquant` can be instructed to output the size of the image to `stderr` in sparse mode (see Section 4.4 [Misc Parameters], page 13,`debug`). Normally the ray weights as accumulated are output, but they may be divided by the area of the bins, to provide their surface density (see Section 4.2 [Binning Parameters], page 9, `norm_area`).

4 Invoking spatquant

spatquant uses the IRAF compatible parameter interface to manage its parameters. A parameter file is required to run spatquant.

4.1 Input and Output specifications

input *filename*
: The ray stream to process. If it is 'stdin', input is taken from the standard input stream.

output *filename*
: Where to write the output ray stream to process. If it is 'stdout', output is written to the standard output stream.

4.2 Binning Parameters

4.2.1 Generic Filter and Grid

filter_mode *enumerated string*
('pass'|'quantpos'|'quantidx'|'quantize'|'bin'|'mask'|'negmask')
: Rays are filtered according to this parameter (see Section 3.2 [Filter], page 5).

  'pass' Rays are passed through untouched.

  'quantpos'
  : Rays which are not clipped and which would be accumulated are passed through, with their positions changed to the *position* of the bin into which they would have been accumulated

  'quantize'
  : A synonym for quantpos. Deprecated.

  'quantidx'
  : Rays which are not clipped and which would be accumulated are passed through, with their positions changed to the *index* of the bin into which they would have been accumulated

  'mask' Rays which would be accumulated are passed through unchanged.

  'negmask' Rays which would *not* be accumulated are passed through unchanged.

  'bin' Rays are accumulated into bins and the bins are written out as rays with the accumulated weights

  'none' No rays are output. Only useful if you just want to create an image.

norm_area *string list*
: This parameter determines if accumulated bin weights or image values are normalized by the areas of the bins. It takes a comma or white-space separated list of tokens which indicate the data which should be normalized. Possible tokens are:

'`bin`' the accumulated bin weights are normalized.

'`image`' the image pixel values are normalized

`xcent` *float*
`ycent` *float*
The coordinates of the center of the grid.

`quant_method` *enumerated string*
This parameter selects the grid and binning scheme.

'`rectrect`'
rectilinear grid and rectangular bins

'`rectcirc`'
rectilinear grid and circular bins

'`hexcirc`' hexagonal grid and circular bins

'`wedge`' circular grid and wedge bins

`clip` *boolean*
If this is true, the grid will be limited to specified dimensions. The parameters which determine those dimensions are specific to each grid scheme.

4.2.2 Rectilinear Grid, Rectangular Bin

`n_x` *integer*
The number of columns in the grid. This is used only if clipping is turned on.

`n_y` *integer*
The number of rows in the grid. This is used only if clipping is turned on.

`x_sz` *float* The width of a bin, also the distance in x between grid points.

`y_sz` *float* The height of a bin, also the distance in y between grid points.

4.2.3 Rectilinear Grid, Circular Bin

`n_x` *integer*
The number of columns in the grid. This is used only if clipping is turned on.

`n_y` *integer*
The number of rows in the grid. This is used only if clipping is turned on.

`x_sp` *float* The spacing in x between bin centers.

`y_sp` *float* The spacing in y between bin centers.

`diameter` *float*
The diameter of a bin. It must be less than or equal to the bin spacing.

4.2.4 Hexagonal Grid, Circular Bin

`n_x` *integer*
The number of columns in the grid. This is used only if clipping is turned on.

`n_y` *integer*
: The number of rows in the grid. This is used only if clipping is turned on.

`spacing` *float*
: The distance between centers of adjacent bins. It must be greater than or equal to the diameter of the bins.

`diameter` *float*
: The diameter of a bin. It must be less than or equal to the bin spacing.

4.2.5 Circular Grid, Wedge Bin

`rtype` *enumerated string* (‘`discrete`’|‘`exp`’|‘`file`’|‘`linear`’)
: How the wedges are to be entered.

  ‘`discrete`’
  : A finite number of wedge bins are to be entered by the user

  ‘`exp`’
  : The inner and outer radial dimension are defined as: a * b ^ (n - 1), a * b ^ n for n = 1, 2... , respectively. For n = 0, the inner and outer radial dimension are: 0 and a, respectively.

  ‘`file`’
  : An rdb file with columns Rmin, Rmax, Tmin and Tmax

  ‘`linear`’
  : The inner and outer radial dimemsion are defined as: deltaR * n and deltaR * (n+1), respectively.

`rspec` *string*
: `rspec`’s meaning depends upon the value of `rtype`:

  ‘`discrete`’
  : If the parameter `rtype` is set to ‘`discrete`’, then the parameter `rspec` must be entered as follows: Rmax,Nwedge,Tmin. Where Rmax denotes the outer radius of the wedge, Nwedge denotes the number of wedges within an annular ring. Tmin (in degrees) denotes theta min. A comma is used to delimit between an element of the list, a slash or colon can be used to delimit between different sets of lists.

  ‘`exp`’
  : If the parameter `rtype` is set to ‘`exp`’, the parameter `rspec` must be entered as follows: a,b,Nwedge,Tmin. Where the inner and outer radius isdefined as a * b^(n-1) and a * b^n respectively for n=1, 2, 3,... For n=0, the inner and outer radius is defined as 0 and a, respectively. Nwedge denotes the number of wedges within an annular ring. Tmin (in degrees) denotes theta min. A comma is used to delimit between an element of the list

  ‘`file`’
  : If the parameter `rtype` is set to ‘`file`’, the parameter `rspec` must be a name of an rdb file containing the following columns: Rmin, Rmax, Tmin and Tmax.

  ‘`linear`’
  : If the parameter `rtype` is set to ‘`linear`’, the parameter `rspec` must be entered as follows: deltaR,Nwedge,Tmin. Where the inner and outer radius isdefined as deltaR * n and deltaR * (n+1),

respectively. Nwedge denotes the number of wedges within an annular ring. Tmin (in degrees) denotes theta min. A comma is used to delimit between an element of the list

4.3 Image Output Parameters

4.3.1 Generic Image Output

`create_image` *boolean*
: If true, an image is created and output.

`image_file` *filename*
: The image is output to this file. If it is '`stdout`', the image is written to the standard output stream, if it is '`stderr`', it is written to the standard error stream.

`image_fmt` *enumerated string*
: The image may be written in any of the formats indicated in Section 3.5 [Images], page 7. This parameter selects which format to use. The available choices are: '`ascii-raw`', '`ascii-hdr`', '`binary-raw`', '`binary-hdr`', '`rdb-raw`', '`rdb-hdr`';

`image_lang` *enumerated string* ('`c`' | '`fortran`')
: The binary formats of an image may be written in either C or Fortran compatible records. This parameter selects which is used.

4.3.2 ASCII Image Output

ASCII images are printed in *i* major form; i.e., bins with a single value of *i* are printed most rapidly.

`i_prefix` *string*
: This string precedes the printing of a sequence of bins with the same *i* value. It is often left blank.

`i_prefix` *string*
: This string is printed after a sequence of bins with the same *i* value is printed, and before the printing of bins with the succeeding *i*. It is usually the string '`\n`', indicating that a new line is to begin.

`wt_format` *string*
: This is a `printf` style format string used to print the weight of a bin. It should *not* contain any padding to separate bin values.

`j_pad` *string*
: This parameter specifies what will be printed between bin values. It's usually '`\t`'.

4.3.3 '`/rdb`' Image Output

'`/rdb`' images are printed in *i* major form; i.e., bins with a single value of *i* are printed most rapidly.

`wt_format` *string*
: This is a `printf` style format string used to print the weight of a bin. It should *not* contain any extra padding.

`pos_format` *string*
: This is a `printf` style format string used to print the position of a bin. It is used once for each coordinate. It should *not* contain any padding to separate bin values.

4.4 Miscellaneous Parameters

`debug` *string list*
: This parameter accepts a comma or space delimited set of debugging flags, which causes `imageplot` to emit extra information to `stderr`. The available flags are:

  `time`
  : output the amount of time taken to read and process the input rays. This does not include time to parse the sparse tree and output the rays.

  `tree_dump`
  : If the sparse matrix method is used, the tree will be dumped to the standard error stream after it has been constructed and filled.

  `efficiency`
  : The sparse matrix representation may have unused bins as part of its space-saving strategy (see Section B.2.1 [Sparse], page 19). This flag causes the sparse matrix tree to be parsed and the bin usage efficiency to be determined.

  `pix_limits`
  : If the grid scheme supports the use of dynamic grid limits (see Section B.3 [Backend], page 19), this flag causes the indices of the grid limits to be written to the standard error if an image is created.

  `coord_limits`
  : If the grid scheme supports the use of dynamic grid limits (see Section B.3 [Backend], page 19), this flag causes the coordinates of the grid limits to be written to the standard error if an image is created.

  `dimen`
  : If the grid scheme supports the use of dynamic grid limits (see Section B.3 [Backend], page 19), this flag causes the number of bins in each dimension to be written to the standard error if an image is created.

`bufsz` *float* [Mb]
: `spatquant` uses this limit to determine if it has enough memory to create a core image. If the image would require more memory than `bufsz`, it uses sparse mode. Note that there is **no guarantee** that sparse mode will use less than `bufsz` Mb!!

`join_pad` *int*
`extend_pad` *int*
: These parameters are used to tune the efficiency of the sparse matrix implementation. Please don't touch! The default for `join_pad` is '`6`', for `extend_pad`, '`4`'.

`version` *boolean*
: If true, print the version of the program and exit.

Appendix A Output Image Formats

Output formats with the suffix '`-raw`' are written without a header; those with the suffix '`-hdr`' with one. The format of the headers depends upon the gridding scheme, that of the bin weights does not.

A.1 Image bin weight formats

The image bin weight output format is independent of the gridding scheme, but does depend upon whether the output is in ASCII or binary. In all cases empty bins are added so that the full complement of bins (as dictated by the grid limits) are present.

A.1.1 ASCII Image Format

Bins are written out in sets, one set per i value. The contents of the parameter `i_prefix` are written before each set, those of the parameter `i_suffix` after each set. Bin weights are printed using the format in the parameter `wt_format`. The contents of the parameter `j_pad` are written between each bin weight (but not after the last bin in a set).

A.1.2 '`/rdb`' Image Format

Each bin is written on a separate line. The data output for each bin is i, j, x, y, and the weight of the bin. They are separated by tabs. The x and y positions are written out with the format specified by the parameter `pos_format`, the weight is written with that specified by the parameter `wt_format`. A two line '`/rdb`' style header begins the file, with column names '`i`', '`j`', '`x`', '`y`', '`wt`'.

A.1.3 Binary Image Format

The output format depends upon the language compatibility mode. The data type (single or double precision) is determined by the `image_datum` parameter.

For C compatibility, the bins are dumped with no record information, with i increasing slower than j. In Fortran mode, each set of bins with the same i value is written as a record. (Fortran records are not written by a Fortran routine, but are simulated in C by writing a record length as a four byte integer before and after each record. This works on SGI's and Sun's.)

A.2 Image Header Formats

As in the bin weight data, the header format differs depending upon whether it is written in ASCII or binary. The actual information written will depend upon the gridding method, but the format is the same. ASCII headers are composed of a set of lines, one variable per line, which have the following template:

```
# <variable name> = <variable value>
```

C binary headers are written out as structures, with all padding intact. This is portable across 32bit Sparc and MIPS machines. Structure definitions are in the file '`spatquant.h`'. The structure is preceded by an integer which describes its type; the types are defined by the `enum BackEndType` in '`spatquant.h`'.

Fortran binary headers are written as a single record, preceded by a record containing an integer, which describes the backend. The variables are written in the same order as they are defined in the C structure. Again, see '`spatquant.h`' for more info. You will have to know the exact type of the variables being written and their order in order to read the header.

A.3 The Contents of Headers

Image header contents are specific to the grid scheme.

A.3.1 '`rectrect`'

The C structure definition of this header is found in '`spatquant.h`'. It is reproduced below:

```
typedef struct
{
  size_t ni, nj;
  long i_min, j_min;
  double x0, y0;
  double x_sz, y_sz;
} RectRectImage;
```

`ni` The number of rows

`nj` The number of cols

`i_min`
`j_min` the minimum i and j indices of the matrix

`x0`
`y0` the x and y values corresponding to `i = j = 0`.

`x_sz`

`y_sz` the extent of the bins in the x and y directions.

A.3.2 '`rectcirc`'

The C structure definition of this header is found in '`spatquant.h`'. It is reproduced below:

```
typedef struct
{
  size_t ni, nj;
  long i_min, j_min;
  double x0, y0;
  double diameter;
  double x_sp, y_sp;
} RectCircImage;
```

`ni` The number of rows

`nj` The number of cols

`i_min`
`j_min` the minimum i and j indices of the matrix

`x0`
`y0` the x and y values corresponding to `i = j = 0`.

`diameter` the diameter of the bins

`x_sp` the spacing between bins in the x direction.

`y_sp` the spacing between bins in the y direction.

A.3.3 'hexcirc'

The C structure definition of this header is found in '`spatquant.h`'. It is reproduced below:

```
typedef struct
{
  size_t ni, nj;
  long i_min, j_min;
  double x0, y0;
  double diameter, spacing;
  double x_sp, y_sp;
} HexCircImage;
```

`ni` The number of rows

`nj` The number of cols

`i_min`
`j_min` the minimum i and j indices of the matrix

`x0`
`y0` the x and y values corresponding to `i = j = 0`.

`diameter` the diameter of the bins

`spacing` the distance between centers of adjacent bins

`x_sp` the spacing between bins in the x direction. This is the same as the variable `spacing`.

`y_sp` the spacing between bins in the y direction. This is just $\frac{\sqrt{3}}{2}$`spacing`.

A.4 Sample Code

Here's some sample Fortran code which reads a header (which is similar to, if not exactly the same as, that used by the rectilinear gridding method), reads in the data, and prints it out. This code assumes that `image_datum` has the value '`float`'

Here's the header definition (in C).

```
typedef struct
{
  size_t ni, nj;
  long i_min, j_min;
  double x0, y0;
  double x_sz, y_sz;
} RectilinearImage;
```

The Fortran code:

```
real*8 x0, y0, x_sz, y_sz
real data(5)
integer ni, nj, i_min, j_min, hdr_type

open(unit=8, file='f.img', form='unformatted')

read(8) hdr_type
if (hdr_type .ne. 0) then
  write(*,*) "header type isn't 0, it's", hdr_type
  stop
end if

read(8) ni, nj, i_min, j_min, x0, y0, x_sz, y_sz
write (*,*)  ni, nj, i_min, j_min, x0, y0, x_sz, y_sz

if (nj .gt. 5) then
  write(*,*) "nj greater than built in limits"
  stop
end if

do i=1,ni
    read (8) (data(j), j=1,nj)
    write (*,*) data
end do

end
```

Appendix B The implementation of `spatquant`

This section describes the guts of the program, and how to incorporate new binning/grid schemes and output formats. It is not for the faint-hearted.

B.1 The structure of the beast

(under construction).

B.2 A look at the FrontEnd

(under construction).

B.2.1 The Sparse matrix implementation

(under construction).

B.3 A closer look at the Backend

(under construction).

Appendix C Verification and Validation

`spatquant` has undergone testing on several fronts to ensure that it works as advertised. Tests can be divided into those which test the front end (the bin manipulation routines) and those which deal with the back ends, which provide information about which bin a ray goes in. Some tests are applicable to both groups.

The following set of tests were performed with a set of rays which uniformly filled a square with sides of length 1.5. The rays were generated by `genphot`

1. This test checks edge effects for the '`rectrect`' grid and binning scheme. It checks that all rays are assigned to the correct bin, an operation which is sensitive to round-off error at bin edges. It also tests if the front end correctly deals with clipping in sparse and core modes. Because clipping is turned on, the backend makes decisions about which rays are to be accumulated.

   Clipping was turned on, and the grid was set to have bin widths of 0.5 and heights of 1.5. This exactly covers the input rays. The weights of the accumulated rays in each bin was compared against those expected from an independent summation. All input rays were properly accumulated (none were lost, and the correct number was found in each bin). A similar test, but switching the dimensions and number of bins between x and y, was performed, with the same results. These tests were performed with `spatquant` in both core and sparse mode (the latter turned on by setting `bufsz` to 0).
2. The same test as in the previous item was performed with clipping off. The backend no longer enforces limits in this setup. It results in the same output as the previous test (as it should).
3. When in sparse mode, the handling of bins is not symmetric in i and j. In order to ensure that the internal trees are being built correctly, a set of tests were run with the '`rectrect`' gridding and binning scheme. Each test involved comparing a set of rays filtered through `spatquant` in '`mask`' filter mode and a set with their x and y values swapped before and after the pass through `spatquant`. Both sets of rays were sorted in the order of increasing y and then x.

   In order to ensure that the '`rectrect`' scheme had no problems of its own, the test was run with clipping turned off.

   The remaining tests were run with clipping on. Sparse mode was turned on by setting `bufsz` to 0. Core mode was ensured by setting `bufsz` to a value larger than the size of the grid requirements. Tests were made for core mode, sparse mode, and a cross-comparison, to ensure that they both returned the same results. Comparison was done on a ray by ray basis for equivalence of position and weight.
4. The rays were passed through `spatquant` with `filter_mode` set to '`mask`' and '`bin`' with clipping on and off, for schemes '`rectrect`', '`hexcirc`', and '`rectcirc`'. For '`rectrect`' the bins were square, with sides of 0.1. For '`hexcirc`' and '`rectcirc`' the diameter was 0.01. When clipping, n_x and n_y were set to 40.

   The total weight and number of rays output for each permutation was tallied. An image was created with format '`binary-hdr`' and translated into fake rays with the same weight for each permutation. The weights of the images were compared to those

of the output rays. For the cases where accumulation took place, the number of nonzero pixels in the images were compared to the number of output rays.

5 History

19960708 added `quantidx` and `quantpos` functionality.

Concept Index

Variable and Parameter index

Table of Contents