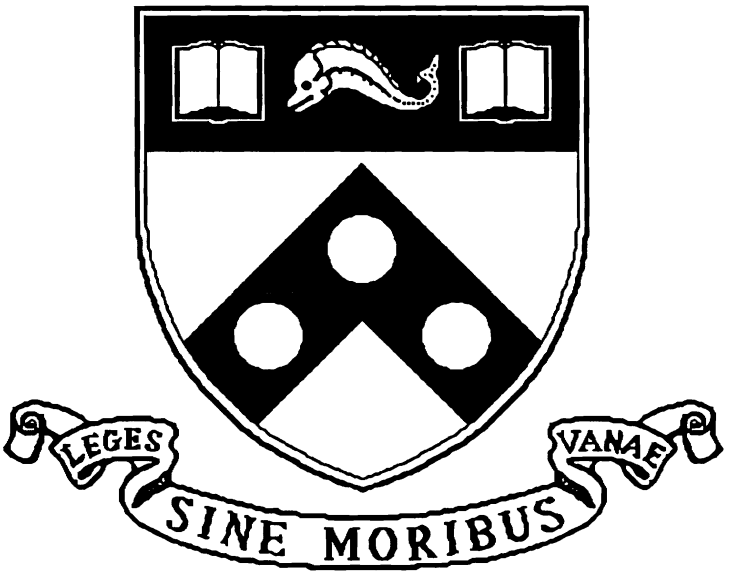
Superquadric Library, User Manual and Utility Programs

MS-CIS-92-11
GRASP LAB 300

Luca Bogoni



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

February 1992

Superquadric Library, User Manual, and Utility Programs

Luca Bogoni

GRASP Laboratory
Department of Computer and Information Science
University of Pennsylvania, Philadelphia, PA 19104

February 3, 1992

Abstract

Superquadrics are a family of parametric shapes that have been used as primitives for shape representation in computer vision and computer graphics. They can be used for modeling tapering and bending deformations and are recovered efficiently by a stable numerical procedure.

This document introduces the superquadric library, *SQ_lib*, developed at the GRASP Lab at the University of Pennsylvania.

The manual is organized into three parts. The first part provides the reader with a description of superquadrics models and deformations that can be performed. Furthermore, it introduces the coordinate systems conventions which are used in the library.

The second part presents some examples of applications on how one can use the functions defined in the library. It also lists utility programs which have been developed while conducting research. They provide a good source of examples for the application of the library.

Finally, the last part describes the datatypes and each of the functions which are supported in the library. The library itself is organized in two sets *Fundamental* and *Auxiliary* functions. A quick reference to all the functions and an index is provided.

Some of the functions and examples supplied perform data preprocessing and are connected to the PM image description also available from the GRASP Lab. These functions are provided in isolation from the remaining body of the library and can easily be excluded in the actual compilation of the library. Furthermore, routines for the visualization of the data, using X11, are also provided.

Contents

1 Introduction

This document introduces the Superquadric library, *SQ_lib*, developed at the GRASP Lab at the University of Pennsylvania. The initial effort of implementing the recovery process is due to the research of Frank Solina. Subsequent development and restructuring where carried out both by Alok Gupta and Bogoni Luca, who reorganized and enriched the initial routines and interface consolidating it into the present package.

Organization

- Section 2 provides the reader with a description of superquadrics models and deformations that can be performed. Furthermore, it introduces the coordinate systems conventions which are used in the library. At the end of this section, few references are provided.
- Section 3 presents some examples of applications on how to use the library. It also lists few utility programs which have been developed while conducting research. Some of them provide a good source of examples for the application of the library.
- Section 4 describes the datatypes and each of the functions which are supported in the library.

Some of the functions and the examples provided here are concerned with data preprocessing and are connected to the PM image description also developed at the GRASP Lab. These functions are provided in isolation from the remaining body of the library and can easily be excluded in the actual compilation of the library. The type of data which was preprocessed was laser range data but the actual application of the core routines performing the recovery process is entirely unrelated to the origin and the nature of the data.

The visualization of the data, using X11, is not too sophisticated but provides for some of the basic display operations. One might desire to modify it but that goes beyond the actual purpose of the package provided here.

Acknowledgements

The work collected in the Superquadric library was matured through the years and the many people who were involved in developing some of the functions now gathered in the library. I would like to thank Dr. Ruzena Bajcsy for the many discussions and for providing the resources in the GRASP lab to make it possible. Alok Gupta was a major player in the development and test of actual functions in this library and provided insight into the overall process. Frank Solina provided the initial recovery process and some of the preprocessing routines. Furthermore, I would like to thank

all my colleagues at the GRASP lab who have been supportive and made it a pleasant environment for the development of this library.

This research was supported by: Navy Grant N0014-88-K-0630, AFOSR Grants 88-0244, AFOSR 88-0296; Army/DAAL 03-89-C-0031PRI; NSF Grants CISE/CDA 88-22719, IRI 89-06770, and ASC 91 0813; and Du Pont Corporation.

2 Basic Definitions

2.1 A superquadric, what is it?

Superquadrics are a family of parametric shapes that have been used as primitives for shape representation in computer vision and computer graphics.

Definition : A superquadric surface is defined as the closed surface spanned by the vector **S** having x,y and z components specified as functions of the angles η and ω in the given intervals :

$$\mathbf{S}(\eta,\omega) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_1 \cos^{\varepsilon_1}(\eta)\cos^{\varepsilon_2}(\omega) \\ a_2 \cos^{\varepsilon_1}(\eta)\sin^{\varepsilon_2}(\omega) \\ a_3 \sin^{\varepsilon_1}(\eta) \end{bmatrix} \quad \begin{matrix} \frac{-\pi}{2} \le \eta \le \frac{\pi}{2} \\ -\pi \le \omega < \pi \end{matrix}$$

We identify components as $S_x(\eta, \omega), S_y(\eta, \omega)$, and $S_z(\eta, \omega)$[1].

The implicit superquadric equation can be derived from the above definition by eliminating η and ω :

$$\left(\left(\frac{x}{a_1}\right)^{\frac{2}{\varepsilon_2}} + \left(\frac{y}{a_2}\right)^{\frac{2}{\varepsilon_2}}\right)^{\frac{\varepsilon_2}{\varepsilon_1}} + \left(\frac{z}{a_3}\right)^{\frac{2}{\varepsilon_1}} = 1.$$

Thus, alternatively we can define the superquadric in terms of its implicit equation, as the locus of the points (x, y, z) satisfying the above equation.

The parameters a_1, a_2, and a_3 determine the size of the superquadric in the x,y and z directions (in object–centered coordinate system) respectively; while ε_1 and ε_2 represent the shape parameters in the latitude and in the longitude plane. Based on these parameters, superquadrics can model a large set of standard geometric primitives, such as spheres, cylinders, parallelopipeds as well as shapes in between.

If both ε_1 and ε_2 are equal to 1, the surface defines an ellipsoid. Cylindrical shapes are obtained for $\varepsilon_1 \ll 1$ and $\varepsilon_2 = 1$. Parallelopipeds are obtained for both ε_1 and $\varepsilon_2 \ll 1$. In our approach, the model recovery procedure allows ε_1 and ε_2 to assume values in the interval $[0 \ldots 1]$. For values of ε_1 and $\varepsilon_2 > 1$ the resulting parameterized shapes define objects which are not in the set of primitives we are interested in portraying. For instance, ε_1,$\varepsilon_2 = 2$ yield objects which are diamond–shaped bevels and as their value increases they become pinched.

2.2 Applying Deformations to Superquadrics

The representational power of superquadrics is augmented by the application of various *deformations* to the basic model. The deformations which we have included in our vocabulary are **tapering** and

[1] Actually the component in the z–direction is independent of ω but, we include it, at this point, only for symmetry. However, when deformations are applied ω becomes an effective component of the z–direction.

bending. For notation purposes we define $\mathbf{S}'$ as the model to which deformations have been applied and identify each of the components in the x,y, and z directions respectively by S_X, S_Y, S_Z and alternatively as (X, Y, Z), accordingly to the definition of the implicit equation.

2.2.1 Tapering

Linear tapering along the z–axis transforms the basic superquadric model from $\mathbf{S}$ to $\mathbf{S}'$, where (x, y, z) is transformed to (X, Y, Z). The transformed model is given by :

$$\begin{aligned} X &= f_x(z)\,x \quad \text{where} \quad f_x(z) = \tfrac{K_x}{a_3} z + 1 \\ Y &= f_y(z)\,y \quad \text{where} \quad f_y(z) = \tfrac{K_y}{a_3} z + 1 \\ Z &= z \end{aligned}$$

where K_x, K_y, $-1 \leq K_x, K_y \leq 1$, represent the tapering with respect to the x and y plane relative to the z direction.

2.2.2 Bending

Bending deformation of the superquadric surface vector is defined by the following transformation :

$$\begin{aligned} X &= x + \cos(\alpha)(R - r), \\ Y &= y + \sin(\alpha)(R - r), \\ Z &= \sin(\gamma)(\tfrac{1}{k} - r). \end{aligned}$$

Where k is the curvature and r is the projection of x and y components onto the bending plane $z - r$:

$$r = \cos(\alpha - \tan^{-1}(\frac{y}{x}))\sqrt{x^2 + y^2}$$

Bending transforms r into

$$R = k^{-1} - \cos(\gamma)(k^{-1} - r),$$

Where γ is the bending angle

$$\gamma = zk^{-1}$$

2.2.3 Combination of Tapering and Bending

The two independent deformations are applied by computing the corresponding homogeneous transformation matrices. It is possible to apply both transformations to a superquadric model sequentially. However, since matrix multiplication is not commutative, the order in which deformations are applied is important. The model recovery procedure has adopted the following structure to transform an object–centered superquadric model to a deformed superquadric in general position and orientation.

$$\mathbf{S}' = Translation(Rotation(Bending(Tapering(\mathbf{S}))))$$

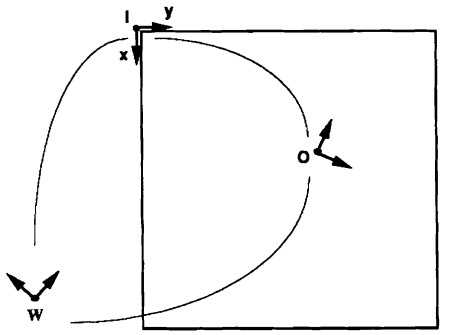


Figure 1: Coordinate Frames

Thus, bending and tapering introduce two parameters each in the final superquadric equation, bringing the total parameter count to 15. The minimization procedure is capable of recovering all 15 parameters simultaneously for a given data set. The above equation identifies the volumetric model used to describe parts in our system.

2.3 Super Ellipses: 2D SuperQuadrics

While superquadric primitives are exclusively 3D primitive, the library contains a recovery process which allows to recover 2D primitives in the shape of superquadrics. Effectively their equations do not involve the z component.

2.4 Coordinate systems

In order to understand the different references to the coordinate systems often discussed in this article and used in the routines we define the conventions adopted.

Three coordinate system frames are defined.

- **Image Coordinate System** (ICS) refers to the top left corner of the image. The x–axis correspond to the rows and the y–axis corresponds to the columns and the z–axis is pointing out of the plane. This right hand system was chosen in order to facilitate further interaction with the manipulator. The parameters defining this coordinate system are fixed.

- **Object Coordinate System** (OCS) located ate the centroid of the object. The parameters will be adjusted in the recovery process.

- **World Coordinate System** (WCS) located somewhere in the environment and it can be user specified.

We now describe the relation between the different coordinate frames and how one may apply transformations between the different systems.

Points are described as a default with respect to the Image Coordinate system. If the user would like to describe with respect to some other coordinate system (WCS) can easily do so by supplying his/her own transformation matrix T, describing rotation and translation, as one of the parameters. Given a transformation T, then a point, P_I, specified in ICS is expressed in the world coordinate system WCS by the expression:

$$P_W = T^{-1} P_I$$

Similarly we can express a point given in WCS by the expression:

$$P_I = T P_W$$

The figure 2.2.3 gives an idea of the relation between the different coordinate frames. The OCS and WCS are arbitrarily set to an angle, not necessarily perpendicular to the plane of the image. The only restriction is the orientation of the coordinate frame identifying the ICS. The arches amongst them represent the transformations between the different coordinate systems.

2.5 references

The references provided in this section are given as pointers to literature describing more in detail the mathematical derivations and reasoning which is behind the functions implemented in the SQ–library. Further applications of the library can be obtained by consulting the GRASP Newsletter.

For the details of the actual fitting procedure the user is referred to *Franc Solina*'s Ph.D. dissertation *Technical Report MS–CIS–87–111* also appeared in *IEEE PAMI, vol. 12, No. 2, Feb. 1990 Recovery of Parametric Models from Range Images: The Case for Superquadrics with Global Deformations", F. Solina and R. Bajcsy.*

Additional references describing the ideas behind some of the functions can also be found in *A.Gupta, L. Bogoni, and R. Bajcsy: Quantitative and qualitative measures for the evaluation of the superquadric models* in Proceedings of the IEEE Workshop on Interpretation of 3D Scenes, pp. 162–169, 1989. Further applications and procedures where developed in the context of *A. Gupta*'s Ph.D. dissertation, *Surface and Volumetric Segmentation of Complex 3–D Objects Using Parametric Shape Models, Technical Report MS–CIS–91–45.*

The actual minimization procedure *mrqmin()* is taken from *Numerical Recipes in C*, by Press et al.

3 User Manual

3.1 getting started: a tutorial

To best describe how to handle functions in the library, we have provided some examples. They illustrate some of the basic applications which can be performed using functions in this library. In the next section a description of each individual function will be given more in detail. After a list of the utility programs there is a list of topics pertaining compilation, inclusion of files, naming conventions etc.

3.2 Examples

In this section we consider some examples to illustrate the application of functions calls in the library. The description of the individual calls is only summary at this point. For further detail one should refer to the specific functions descriptions in section 4.4.

We consider three particular instances. The first two are instances, quite similar in structure, of functions calls required to set up the basic superquadric procedure both starting from laser-image, in PM–format (see section 3.4.2). The last one exhibit the application of other functions which allow the displaying the recovered parameters using X11.

3.2.1 From points to Fitting parameters

The function described in the example below illustrates the basic process required to obtain a superquadric fit of a set of 3D points. The user is supposed to provide a list of points in a WCS (world coordinate system) frame.

The function takes as parameters a list of points in three dimensions and a pointer which upon return will be set to the list of the structures describing the incremental recovery of the parameters with the last fit being the *best fit.*

Initially the option flags are initialized. The number of iterations to be performed in the recovery process is set to 5, and the deformations to be recovered are set to *Bend* and *Taper.*

Next the dimensions and orientation of the center of mass is recovered, *sq_estimate.*

The structure list is initialized and the number and the list of points is assigned to the structure. The transformation matrix defining the relation of the points to the world coordinate system is initialized to the identity matrix. Such initialization underlines that the points are already specified in the appropriate coordinate system.

The recovery routine is invoked. Since we are not interested in displaying the superquadric during the recovery process, we use the vanilla version without any simple graphics interface.

```
perform_recovery(n_points,
                 point_list,
                 rec_sq)

int       npoints;                    /* number of points to fit */
double    **point_list;               /* array with point coordinates */
SQ_STRUCT_LIST *rec_sq;               /* superquadric parameters */
{

  SQ_STRUCT_OPTS *flg;

  /* Recovery Options */

  flg = sq_make_opts();               /* initialize structure */
  flg->fit_type = _BEND_TAPER;        /* type of recovery desired */
  flg->n_iter   = 5;                  /* max # of  iterations allowed */


  sq_estimate(rec_sq,flg);            /* estimate the initial sq param.s */


  rec_sq = sq_make_struct_list();     /* initialize list of sq's to trace
                                         history of recovering process    */

  rec_sq->sq_points = point_list;
  rec_sq->sq_num_points = n_points;

  sq_identity_matrix(rec_sq->T);      /* no transformations  */


  sq_quadric_plain(rec_sq, flg);      /* recover the SQ to spec in flag */

}
```

3.2.2 From PM image to Fitting parameters

The following example is quite similar to the preceding one. The main difference is that it uses the PM format for the image. It is very similar to the source code for the standalone program, *sqsup*, one of the utility programs, for recovering the parameters for a superquadric from a PM–image.

The following program accepts at command line the specification of parameters which are then processed by the *sq_cmdparse()* routine. Additional functions for checking on the validity of parsing the arguments are provided. The image is then processed and the points are recovered. Dimensions and orientations are then determined, *sq_estimate*, determined. Finally, the parameters defining the superquadric are recovered by the minimization process.

The optional setting of the flags from within the program are exemplified in the next example.

```
#include <stdio.h>
#include <local/sq.h>

main(argc,argv)
int argc;
char **argv;
{
  SQ_OPTS_FLAGS *f;                    /* parsing and initialization flags */
  SQ_STRUCT_LIST *S;                   /* pointer to structure list */
  char *pm_file = (char *) NULL;       /* pm image to process */
  char *o_file[100];                   /* some name for output file */

  FILE *fp;
  pmpic *pm_img;                       /* pm format */
  int i;

  f = sq_make_opts();                  (a1)
  if (argc == 1)
    {
      sq_usage(argv[0]);
      exit(0);
    }

  sq_cmdparse(argv,argc,&f,&pm_file);  (b1)
  if (sq_parse_errcheck())             (b2)
    {
      if (sq_param_errmsg())           (b3)
       exit(0);
    }

  S = sq_make_struct_list();           (a2)

  if ((fp = fopen(pm_file,"r")) == NULL)
    {
      printf("file open error :%s \n",pm_file);
      exit(0);
    }
  if((pm_img = pm_read(fp,0)) == NULL)
    {
      printf("error in reading the pmfile %s",pm_file);
      exit(0);
    }
  fclose(fp);

  sq_pm_to_points(S,F,&pm_img);        (c1)
```

```
  sq_estimate(S,F);                    (c2)

  sq_quadric_plain(S,F);                /* run the sq routine */

  strcpy(o_file,infile);
  sq_print_to_file(S,strcat(o_file,".fitfile"));

}
```

The initialization with defaults for flag structures is performed,*a1, a2, a3* and parsing of the arguments from cmd line, *b1, b2, b3* constitutes the first phase of the program. Points are extracted from the pm image*c1*, and sequentially the major axis and basic transformation parameters are estimated *c2*. Finally the minimization process to recover the parameters is invoked. In the last step, all the recovered parameters are output to the *fit_file*.

3.2.3 Displaying using X11

Visualization of the data is important aspect of the recovery process for in that way one may be able to qualify the values which are returned by the fitting procedure. There are several utility programs which allow the plotting of the parameters' history through the recovery procedure, either while in progress or a posteriori. In this example, we show how one can use some of the displaying routines incorporated in the library.

This example allows the user to specify at cmd line a superquadric fitfile, followed by a scaling factor, and a time interval. The program will read the fitfile into a *SQ_STRUCT_LIST* structure and display each one on the opened window, waiting the time period specified as input parameter.

After having checked the input parameters, it initializes the display structure. Some of the parameters are set.(see section 4.2) The display is initialized by invoking *sq_set_display_dev*. Only one display window is supported in the library.

In this specific case we also allow the user to specify the actual length of time between the display of different superquadric wireframes.

The file is read in and then each of the superquadric is displayed. The lines defining the superquadric are computed and displayed in *sq_display*. Other routines just for computations of the wireframe are also provided. The lines are associated with one of the flags so that they might be erased later. Routines for writing text at given locations are also provided.

Note that also *<X11/Xlib.h>* file should be included.

```
#include <stdio.h>
#include <X11/Xlib.h>
#include <local/sq.h>

char *str1 ="Wait... Computing the lines";
```

```
char *str2 ="Superquadric # ";
char *str3 ="Number of lines #";

main (argc,argv)
char *argv[]; int argc;
{
  char fname[100];
  char buffer[100];
  double sc;

  SQ_STRUCT_LIST *Sq;                    /* list of recovered fitting */
  SQ_STRUCT *S;                          /* pointer for indexing */
  SQ_DISP_FLAGS *F;                      /* displaying flags */
  double T[4][4];
  int background;
  int i;
  int waiting_time;                      /* interim in seconds */
  int iter;                              /* counter for the iterations */

  if (argc != 4)
    {
      printf("%s: superquadric file, scaling factor, waiting time between each display\n", argv[0]);
      exit(0);
    }
  strcpy(fname,argv[1]);
  sscanf(argv[2],"%lf",&sc);
  waiting_time = atoi(argv[3]);

  if (waiting_time < 1)
    {
      printf("Pick a longer waiting time\n");
      exit(0);
    }
  if (sc < 0)
    {
      printf("Values between 0 and 1 will reduce the object, > 1 will expand it\n");
      exit(0);
    }

  F = sq_make_disp_flg();
  F->scale_sq = sc;
  F->xoff = -100;                      /* offset the position of coord */

  sq_set_display_dev("Superquadric fitfile display");

  F->bg_color = ((WHITE - 1)/2);       /* background color to ``gray'' */
  F->fg_color = WHITE;                 /* set the foreground  */
```

```
  F->i_matrix = 0; /* use identity viewing angle */
  F->coordsup = 1;                      /* show the coordinates system */

  sq_fill_xdisplay(F->bg_color);        /* color of display: Gray level */

  sq_read_file(&Sq,fname);

  S = Sq->first;                        /* initialize to first sq */

  iter = 0;
  while (S)
    {
      strcpy(buffer,str1);
      sq_draw_text(140,500,buffer, 0, 0, F->fg_color, F->bg_color);

      sq_display(S,F,Sq->def_type);

      sq_draw_text(140,500,buffer, 0, 0, F->bg_color, F->bg_color);
      sprintf(buffer,"%s %d %s %d",str2,(iter+1),str3,(F->num_lines));
      sq_draw_text(120,500,buffer, 0, 0, F->fg_color, F->bg_color);

      sleep((unsigned) waiting_time);

      /* here we could also fill the screen but since we are testing out
         the routines let's do that.
       */
      sq_draw_lines_ptr(F->lines,F->num_lines,F->bg_color,F->xoff,F->yoff);
      sq_draw_text(120,500,buffer, 0, 0, F->bg_color, F->bg_color);

      free(F->lines);                     /* ain't using them, return it */
      F->lines = (int ***) NULL;
      F->num_lines = 0;                   /* to the pool */

      iter++;
      S = S->next;
    }
}
```

Further examples can be obtained by looking at the source code for the utility programs.

3.3 Utility programs

In this section we describe some utility programs which were written using the superquadric library. They are currently available in */pkg/local/sq/util/bin*.

3.3.1 Post processing and Visualization of data

The programs which have been listed here are used for the visualization of recovered models. Some of the programs allow to generate postscript output of the models recovered.

sqsup This is actually the superquadric program as a stand alone. It allows the used to invoke the program from the cmd shell. It initially parses the cmd line. It does that by invoking the *sq_cmdparse()* routine. Upon successful parsing it invokes the *sq_super_file()* routine. It will print the fitfile and if the user has specified the verbose switch in the cmdline, it will generate the list of the points associated with the scanned image. The actual source code is labeled *sq_standalone*

Here are a list of the specifications that the user can set:

```
sqsup:
    -i (input file) in PM FORMAT
    -g (grid density)
    -n (numb iterations)
    -e (points to erode)
    -c (compensation) for the image
    -s (termination check) for the fitting procedure
    -V (verbose output)
    -k (calibration)
         0 : Non-uniform Gus image; [default]
         1 : Uniform Gus' image;
         file : User defined scanner parameters
    -f (fit type)
         1  : no deformation           (11 parameters); [default]
         2  : tapering                 (13 parameters);
         4  : bending                  (13 parameters);
         24 : tapering and bending     (15 parameters);
         42 : bending and tapering     (15 parameters);
    -F (coordinate frame)
         0 : Image Frame; [default]
         1 : Base Frame;
         2 : Object Frame (i.e. origin is at the centroid.)
         file: User defined reference frame
    -X (X display of sq's)
```

ndisp simple program meant as an example on the use of the display interface for superquadrics. It displays the sequence of superquadrics specified from a fitfile. It is essentially the core of the

displaying of the superquadrics when the program sqsup is run with the '-X' option.

mindist program to compute stats on the minimum euclidean distance of points to a superquadric model.

mintest designed originally to test the mindist program allows the user to visualize the position of point vis–a–vis the superquadric model and the initial guess in the estimate.

sqxtr_fit this program extracts a given fit from the fitfile generated by the superquadric program.

sqmkg will read a fit file and generate output format for xgraph. If no file should be specified then xgraph is automatically invoked. This program allows to trace the history of several parameters for a give fitting procedure.

sqmkg_objs much like the previous program only that it allows to trace the history of a parameter through a set of fitfiles. This is convenient when checking to see how a given parameter is varying in several fitting procedures.

3.3.2 DISP: a display program

disp Interactive program to display superquadric, pm pictures, histograms, place text, relocate, erase objects, etc. It maintains a small Data Base of the objects. It is meant for Visualization. Current implementation uses a *ccmd package* and interface with X11. (This is local. The package that it requires is used as an extremely user friendly interface.) It is extremely easy to use for by typing help on any command or question mark the user can obtain directions as to how to proceed.

It has effectively several types of commands. The first set is meant to input pictures, lines, points, superquadric wireframes. They are internally maintained as objects. The second group is meant to manipulate the objects. The third is meant to alter default setting of processing, such as background color, or the color the position of a given object. Then there are commands to save a portion of the screen, and general information access command. The following represent a very simple sample session to show how to interact with the program.

```
%
%
% disp
XDisplay program Version 1.0 of Wed June.11 90
CCMD Version 1.48 of Fri Jun 17 01:26:02 1988
Disp>
Disp>
Disp> ? Command, one of the following:
 3-dimesional   clear          coord-system   draw          edge
 erase          exit           help           histogram     histrow
 image-pts      line           move           nth-sup       path
 pic            pixel          points         refresh       relabel
 remove         rotate         set            show          superquadric
 write          version        save
Disp>
Disp>
Disp> help ? confirm with carriage return
  or Command, one of the following:
 3-dimesional   clear          coord-system   draw          edge
 erase          exit           help           histogram     histrow
 image-pts      line           move           nth-sup       path
 pic            pixel          points         refresh       relabel
 remove         rotate         set            show          superquadric
 write          version        save
Disp>
Disp> help superquadric
Displays all the superquadric models specified in a given fitfile.
The last model is saved as the object to be displayed. If you would like
a specific model then use the command 'nth-sup' to specify the fit number
It takes as input the fitfile.
Disp>
```

```
Disp>
Disp> set ? option, one of the following:
 bg              coord-sq        color           fg              mg
 real-draw       transformation   x-offset        y-offset        scale
Disp>
Disp> set fo
?Does not match keyword - "fo"
Disp>
Disp>
Disp> set fg (color display buffer) ? Decimal integer
Disp> set fg (color display buffer) 31
Setting the fg to 31.
Disp>
Disp> show (values) ? option, one of the following:
 objects      parameters
Disp> show (values) objects (list) ? confirm with carriage return
Disp> show (values) objects (list)
No objects in the list
Disp>
Disp> show (values) parameters (settings) ? confirm with carriage return
Disp> show (values) parameters (settings)

        position:               x offset 0,  y offset 0,
        color:                  bg 15 mg 31 fg 31
        euler angles:           phi 90 omega 45 psi -90
        scale for pictures:      1.00
        scale for sq:            1.00
        axis with sq            DISPLAYED
        transformation set      ON

Disp>
Disp> set ? option, one of the following:
 bg              coord-sq        color           fg              mg
 real-draw       transformation   x-offset        y-offset        scale
Disp>
Disp>
Disp> set coord-sq (of superquadric) off
Disp> sho parameters (settings)

        position:               x offset 0,  y offset 0,
        color:                  bg 15 mg 31 fg 31
        euler angles:           phi 90 omega 45 psi -90
        scale for pictures:      1.00
        scale for sq:            1.00
        axis with sq            NOT DISPLAYED
        transformation set      ON
Disp>
```

```
Disp> ? Command, one of the following:
 3-dimesional   clear          coord-system   draw          edge
 erase          exit           help           histogram     histrow
 image-pts      line           move           nth-sup       path
 pic            pixel          points         refresh       relabel
 remove         rotate         set            show          superquadric
 write          version        save

Disp> quit
%
%
```

3.4 Including, PM format, Compiling, etc.

3.4.1 What to include?

Each program using functions from the library should incorporate *<local/sq.h>* as one of the include file headers for the preprocessor. Furthermore, if X routines should be used, then <X11/Xlib.h> should also be included.

3.4.2 PM–format for images

PM is the the input for the files used in the routine which recovers the points from the image. The image is in the form of a $2\frac{1}{2}$D range image represented as a gray level image with points closer to the scanner having higher gray values. PM is the internal image representation used in the GRASP Lab.

We include in this section the relevant information for the PM format. The following structure declaration defines an area of memory to be used for the picture and some of the associate data types. It can be found in the include file for the pm library, *pm.h*.

```
#include <sys/types.h>
typedef struct {
        int   pm_id;          /* Magic number for pm format files.    */
        int   pm_np;          /* Number of planes. Normally 1.        */
        int   pm_nrow;        /* Number of rows. Typically 512.       */
        int   pm_ncol;        /* Number of columns. Typically 512.    */
        int   pm_nband;       /* Number of bands. Humans use only 1.  */
        int   pm_form;        /* Pixel format.                        */
        int   pm_cmtsize;     /* Number comment bytes. Includes NULL. */
        u_char  *pm_image;    /* The image itself.                    */
        char    *pm_cmt;      /* Description of operations performed. */
} pmpic;
```

The three other functions which have been used from pm are:

- *pm_isize*() – obtain the size of the picture.

```
/*
 * Size of the image in bytes.
 */
        #define pm_isize(p)        ((p)->pm_np * pm_psize(p))
```

- *pm_free*() – Free an allocated pmpic. All memory associated with pm is freed. This assumes that memory for pm, *pm-¿pm_image* and *pm-¿pm_cmt* have been mal- loc ed separately before. This means that the memory pointed to by pm is no longer valid.

```
        int  pm_free(pm);
        pmpic     *pm;
```

- *pm_alloc*() – *Pm_alloc* allocates memory for the header of a pmpic. *PM_MAGICNO* is assigned to *Pm_id*. *Pm_np* is set to 1. *Pm_nrow* and *Pm_ncol* are set to 512. *Pm_nband* is set to 1. *Pm_form* is set to *PM_C*. *Pm_cmtsize*, *pm_image* , and *pm_cmt* are set to 0.

```
pmpic *pm_alloc()
```

For further details the reader is referred to the actual manual pages.

3.4.3 Compiling

Compiling with this library is no different than any other library. One should include *-lsq* in the list of libraries to be searched at linking time. In the case that the routines chosen use X11 functions or functions dealing with the display, one should also link *-lX11 -ltermlib*. If you are also using *PM*, then you must also include *-lpmfb -lpm*.

Thus, for instance, if we were to compile our first example,(see section 3.2.2) we would issue the following cmd:

```
%cc -o eg1 eg1.c -lsq -lpmfb -lm -lpm -lX11 -ltermlib
```

3.4.4 The Labeling Convention

used in function names used in the library are all prefixed by the string *sq_*; while utility programs are prefixed by *sq*.

The Predefined Constants are restricted to very few and are listed at the beginning of section 4.2.

4 Library

In this section we describe the different functions. Initially we present the groups into which the functions for the library fall into. We then describe the data types which characterize the various parameters used and passed among the different functions. A detailed description of the usage of the functions follows.

4.1 Overview: functions descriptions by groups

The functions in the library are grouped in two categories. A Fundamental and an Auxiliary set can be distinguished. The latter set is provided for some of these functions turned out to be quite useful in manipulating the superquadrics or for computing specific functions often found useful.

Fundamental functions include the following groups:

- *Deformations*: direct and inverse for tapering, bending, cavity, and twist.
- *Inside Outside and Normal Angles.*
- *PM to Point computation.*
- *Estimation*: recovery of initial estimate for dimension and orientation.
- *Recovery*: minimization process to recover parameters either for a superquadric or a superellipse.
- *New Structures*: initialization of structures and default setting
- *I/O*: Command parsing routines and I/O functions for reading and writing to files.

Auxiliary functions include:

- *Top Level functions*: A list of functions provided for ease of programming to hide some of the specific calls.
- *Distance*: computation of the distance of a point from the superquadric surface. Iterative minimization is involved in this process.
- *Edge and curvature*: computes the edges and curvature
- *Additional points*: provides a line of support points located on the background.
- *Math functions*: set of routines to compute different mathematical functions.
- *Dynamic allocation*: few dynamic allocation routines for arrays.

- *Display functions*: to compute and display superquadrics as well as coord frames.
- *X11 routines*: basic X routines to draw lines, points, texts, etc..

4.2 Structures descriptions

The definitions and structures discussed below are available in the <*sq.h*> include file.

Constants predefined and used in the following declarations are mostly employed to define the deformations parameters.

- **_NUM_DEF** maximum number of deformation to be applied.[2]
- **_NO_DEF** no deformation.
- **_TAPER** tapering.
- **_TWIST** twisting.
- **_BEND** bending.
- **_CAVITY** cavity deformation.
- **_BEND_TAPER** first apply the bend transformation then taper.
- **_TAPER_BEND** first taper then bend.

The last two are provided mostly as a convenience since it is one of the combinations applied.

SQ_STRUCT – this structure contains the basic parameters describing the superquadric (sq). Deformations are described in the next struct.

```
typedef struct SQ_STRUCT {
  double a1, a2, a3,                    /* sq axes */
         a4, a5,                        /* scales for toroid */
         e1, e2,                        /* epsilon 1 and 2 */
         phi, omg, psi,                 /* Euler angles */
         x, y, z,                       /* translation in image coord */
         chisq, gof,                    /* Chi squared, goodness of fit */
         const[_NUM_DEF][5];            /* deformation constants */
  struct SQ_STRUCT *next;               /* pointer to a sq_struct */
} SQ_STRUCT;
```

[2] currently set to 3

SQ_STRUCT_LIST contains the basic information describing the list of superquadric fitting which the superquadric program will have recovered in the minimization routine.

```
typedef struct SQ_STRUCT_LIST {
  SQ_STRUCT *last;                    /* pointer to a sq_struct */
  SQ_STRUCT *first;                   /* pointer to first sq_struct */
  int num;                            /* Number of iterations */
  int err;                            /* error vector for fitting */
  int    def_type[_NUM_DEF];          /* Type of deformation processing */
  double T[4][4];                     /* homog. trans. to obj coord */
  double **sq_points;                 /* points describing the object */
  int    sq_num_points;               /* number of points */
} SQ_STRUCT_LIST;
```

SQ_OPTS_FLAGS contains flags defining the type of preprocessing for the image – subsampling, erosion, and scanner compensation – as well as directions for describing the type of fit vis–a–vis deformations, number of iterations. It allows the user to specify scanner calibration dependencies and, the reference frame. If the user provides the points, then the flags which can be altered are *num_iter, fit_type, good_on* and *verbose*.

```
 struct {
  char *file;                          /* file to get values */
  int  val;                            /* flag for option */
} SQ_FILE_VAL;

typedef struct SQ_OPTS_FLAGS {
  int grid,                            /* grid density */
      fit_type,                        /* type of fitting */
      num_iter,                        /* number of iterations */
      good_on,
      compen,
      erode_pt,                        /* number of points to erode */
      verbose,                         /* verbose output */
      X;                               /* display using X11 */
  SQ_FILE_VAL
      coord_frame,                     /* new coordinate frame */
      calib;                           /* calibration parameters */
} SQ_OPTS_FLAGS;
```

SQ_DISP_FLAGS these flags are available for the displaying the superquadric on the screen.

```
typedef struct SQ_DISP_FLAGS {
  int real_draw;                       /* drawn real time,else at once */
  int hidden_line;                     /* hidden line removal */
  int coordsup;                        /* to display the coordinates of sq */
  int xoff ;                           /* x & y offset in the disp. buffer */
```

```
  int yoff ;
  int fg_color;                    /* color for the foreground */
  int bg_color;
  int mg_color;
  int i_matrix;                    /* use the identity viewing angle */
  double phi,omega,psi;            /* rotation angles */
  double tmatrix[4][4];            /* matrix from above angles */
  double scale_pic;                /* scaling factor for picture */
  double scale_sq;                 /* scaling factor for sq */
  int ***lines;                    /* lines for the computed sq */
  int num_lines;                   /* total number of lines */
} SQ_DISP_FLAGS;
```

SQ_DIST_STRUCT This structure is employed in the computation of the distance of a point from a superquadric.

```
typedef struct SQ_DIST_STRUCT {
  double point[3];                 /* point to look for */
  double beta_dist;                /* beta guess */
  double dist;                     /* actual computed distance */
  double estim_angle[2];           /* estimated eta and omega */
  double guess_angle[2];           /* initial guess of eta and omega */
  int    iter;                     /* number of iterations */
} SQ_DIST_STRUCT;
```

4.3 Input and Output files

In this section we list the format of the fitfile, as well as the input specification of other files which can be used to specify parameter for the scanner, or a given transformation matrix defining the world coordinate frame with respect to which perform the recovery process.

4.3.1 Fitfile format

The name of the fitfile is is generated with the *fitfile* extension. Upon using *sqxtr_fit*, the utility program for unpacking the fitfile or to extract any of the specific intermediary fitting, the file is generated with the extension of the number corresponding to the iteration of the recovery process in which it was generated. Exception to this is the best fit, ie. the last one, which it is labeled with with suffix *fit_1000*.

The format for a given fitfile is as follows:

```
n                          --  number defining the fit
phi  omega   psi           --  Euler angles,  orientation of the OCS
a1    a2     a3            --  two shorter axis and major axis (z in OCS)
a4    a5                   --  twist and cavity deformations (ignore)
e1    e2                   --  epsilon1 and epsilon2 coefficients
d1    d2     d3            --  deformation ordering
tx    ty                   --  tapering with respect to x and y
cb    zmin  zmax  k  alpha --  center of bending, max extension in negative
                           --  positive z axis, radius of curvature,
                           --  angle of curvature
chisq                      --  goodness of fit (normalized chi squared error)
gof                        --  pure normalized chi sqrd error without the
                           --  the volume factor of (a1*a2*a3)
```

These mnemonics are used consistently throughout the utility programs and this paper.

The following is an example of a fitfile. It represents the last fit recovered in the fitting process, as the first line indicates.

```
1000

 4.043691 -2.171700 2.397726
 43.506426 -71.689937 1.984735
 22.568178 15.515298 48.532446
 0.000000 0.000000
 0.100000 0.140598
 2 4 1
 0.007059 -0.250017
 0.000000 -500.000000 500.000000 67.595095 4.702580
405.190681
0.128138
```

4.3.2 Input Format

Scanner dependent parameters, such as spatial and depth resolution vary from scanner to scanner. When specifying the format for a specific scanner, one can use the provided built in defaults or specify them by giving a file name. Two default formats are provided. The first one is mostly for use here at Penn, the second one present uniform ratio. The user can also specify his/her own by means of a file. (see above SQ_OPTS_FLAGS).

```
static SCAN_PARAM SCAN[2] = {

  {  0.67,                          /* V_RATIO */
     1.0,                           /* H_RATIO */
     1.5,                           /* Z_RATIO */
     15.0,                          /* Background depth */
     3.0 },                         /* Threshold for bg segmentation */

  {  1.0,
     1.0,
     1.500,
     15.0,
     3.0 }
};
```

Thus the form of the file should have contain the following five real numbers: small

```
double VERTICAL
double HORIZONTAL
double HEIGHT
double BACK_DEPTH
double THRESH
```

The file specification is assigned to the flag SQ_OPTS_FLAG as *F->calib.file = filename;*

4.4 Fundamental Functions

In the following subsections we describe the routines which the library supports. They are organized under headers which actually are the specific files comprising the library.

4.4.1 Deformations

In this subsection we have all the routines necessary to perform the deformation of a superquadric as well as most of the inverse deformations.

The functions provided performed forward, inverse transformation of a point as well as the the transformation for the normal on the superquadric at a given location.

In general *in_V* represent the input vector and *out_V* the resulting vector after the deformation has been applied. *constan* represent the array containing the type of deformations which need to be applied.

- **sq_func_taper** It performs the tapering transformation.

```
double SQ_FUNC_TAPER(z, a3, k)
double z,                    /* original point */
       a3,                   /* sq length along z */
       k;                    /* constant of deformation */
```

- **sq_taper_position** It transforms the position vector.

```
sq_taper_position(in_V, out_V, a3, constan)
double in_V[4],              /* original point */
       out_V[4],             /* transformed point */
       a3,                   /* sq length along z */
       constan[5];           /* deformation constants */
```

- **sq_taper_normal** changes the position of surface points and the surface normal vector after tapering along z axis, with a different linear function for x and y axis

```
sq_taper_normal(in_V, out_V, N, N_def, a3, constan)
double in_V[4],              /* original point */
       out_V[4],             /* transformed point */
       N[4],                 /* normal vector */
       N_def[4],             /* order of deformations */
       a3,                   /* sq length along z */
       constan[5];           /* deformation constants */
```

- **sq_inv_taper_position** performs inverse transformation for tapering.

```
sq_inv_taper_vector(in_V, out_V, a3, constan)
double in_V[4],                /* original point */
```

```
        out_V[4],                /* transformed point */
        a3,                      /* sq length along z */
        constan[5];              /* deformation constants */
```

- **sq_func_twist**

```
double SQ\_FUNC_TWIST(z, a3, k)
double z,                        /* original point */
       a3,                       /* sq length along z */
       k;                        /* constant of deformation */
```

- **sq_twist_postion**[3]

```
sq_twist_position(in_V, out_V, a3, constan)
double in_V[4],                  /* original point */
       out_V[4],                 /* transformed point */
       a3,                       /* sq length along z */
       constan[5];               /* deformation constants */
```

- **sq_twist_normal**

```
sq_twist_normal(in_V, out_V, N, N_def, a3, constan)
double in_V[4],                  /* original point */
       out_V[4],                 /* transformed point */
       N[4],                     /* normal vector */
       N_def[4],                 /* order of deformations */
       a3,                       /* sq length along z */
       constan[5];               /* deformation constants */
```

- **sq_bend_position** transforms bending position vector according to bending parameters.

```
sq_bend_position(in_V, out_V, constan)
double in_V[4],                  /* original point */
       out_V[4],                 /* transformed point */
       constan[5];               /* deformation constants */
```

- **sq_inv_bend_position** transforms surface position vector according to bending parameters.

```
sq_inv_bend_position(i1, I1, constan)
double in_V[4],                  /* original point */
       out_V[4],                 /* transformed point */
       constan[5];               /* deformation constants */
```

- **sq_bend_normal** transforms surface position vector according to bending parameters.

[3] **sq_inv_twist_position** This function is not currently supported by the library.

```
sq_bend_normal(v1, v2, N, N_def, constan)
double in_V[4],                 /* original point */
       out_V[4],                /* transformed point */
       N[4],                    /* normal vector */
       N_def[4],                /* order of deformations */
       constan[5];              /* deformation constants */
```

- **sq_cavity_position** applies the cavity deformation at a given position.[4]

```
sq_cavity_position(a1, a2, a3, i1, I1, constan)
double a1, a2, a3,              /* length of axis of sq */
       in_V[4],                 /* original point */
       out_V[4],                /* transformed point */
       constan[5];              /* deformation constants */
```

- **sq_cavity_normal**

```
sq_cavity_normal(a1, a2, a3, in_V, out_V, N, N_def, constan)
double a1, a2, a3,              /* length of axis of sq */
       in_V[4],                 /* original point */
       out_V[4],                /* transformed point */
       N[4],                    /* normal vector */
       N_def[4],                /* order of deformations */
       constan[5];              /* deformation constants */
```

- **sq_deform_position** Apply the deformation to a point

```
deform_point(point,def_type,S)
double point[3];                /* point to be deformed */
int def_type[_NUM_DEF];         /* deformation type */
SQ_STRUCT *S;
```

- **sq_inv_deform_position** Apply the deformation to a point

```
sq_inv_deform_position(point,def_type,S)
double point[3];
int def_type[_NUM_DEF];
SQ_STRUCT *S;
```

- **sq_deform_normal** It applies deformation rules to the given surface position vector p and surface normal *nn. Tapering and Bending allowed. (March 11,1989)*

```
sq_deform_normal(p,nn,def_type,S)
double p[3];
double nn[3];
int def_type[_NUM_DEF];
SQ_STRUCT *S;
```

[4] **sq_inv_cavity_position** This function is not currently supported by library.

4.4.2 Inside Outside, and Normal Angles

This set of functions and routines compute inside–outside function evaluation, computation of normal angles,

- **sq_compute_insideout** computes inside outside function for the given point and superquadric model. With no deformations are applied to the superquadric model.(see below for the inside–outside function with deformations). It returns:
  * if < 1 then point inside else
  * if $= 1$ then point on the superquadric else
  * if > 1 then point outside the superquadric.

```
double sq_compute_insideout(point,S)
double point[3];
SQ_STRUCT *S;
```

- **sq_compute_insideout_deformed** computes InsideOutside function with deformations: It first applies the deformation to the point and then it computes the inside–outside function (uses the above defined function). For a given point, we first apply the inverse deformations and then compute the inside outside function. Since the the equation for the inside outside function is expressed in term of X , Y, Z, (already transformed), and we now have a point, x,y,z we then need to take it to the non deformed superquadric, i.e. apply the inverse transformation. This way, the inside outside function can be computed. It returns:
  * if < 1 then point inside else
  * if $= 1$ then point on the superquadric else
  * if > 1 then point outside the superquadric.

```
double sq_computeinsideout_deformed(x,S,def_type)
double x[3];
SQ_STRUCT *S;
int def_type[_NUM_DEF];
```

- **sq_compute_normal_angles** Computes the unit normal vector and the normalizing constant at a point specified by eta and omega; deformations are taken into account.

```
sq_compute_normal_angles(ang,S,def_type,nn,norm)
double ang[2];
SQ_STRUCT *S;
int def_type[_NUM_DEF];
double nn[3];
double *norm;
```

- **sq_compute_normal_position** computes the normal at point have specified the cartesian coordinates Returns unit normal vector and the Normalizing constant. expects point expressed as (x,y,z).

```
sq_compute_normal_position(point,S,def_type,nn,norm)
double point[3];
SQ_STRUCT *S;
int def_type[3];
double nn[3];
double *norm;
```

- **sq_find_position** it returns the coordinates of the point on deformed superquadric given eta and omega angles.

```
sq_find_position(angles,p,def_type,S)
double angles[2];
double p[3];
double def_type[_NUM_DEF];
SQ_STRUCT *S;
```

- **sq_find_angle** computes eta and omega angles having specified a point on the sq.

```
sq_find_angle(point,angles,def_type,S)
double point[3];
double angle[2];
int def_type[_NUM_DEF];
SQ_STRUCT *S;
```

4.4.3 PM to Points Transformation

It is *PM_FORMAT* dependent. It performs the following tasks:

- It performs the compensation for the scanner, depending of the setting of the flag – defaults to no compensation. The user can specify one of the two defaults flags setting or providing its own by specifying a file in which these setting are given.
- It performs background segmentation for the image.
- erosion of the image is performed, again this depend of the setting of the flag. Defaults is set to no erosion.
- points are extracted from the image. The specified default sampling rate is 0, i.e. all the points are considered.

```
sq_pm_to_points(S, F, pm_image)
pmpic **pm_image;
SQ_OPTS_FLAGS *F;
SQ_STRUCT_LIST *S;
```

4.4.4 Estimation

Performs the estimation of the dimension of object and orientation of the axis based on the moment of inertia of the points. The z–direction is chosen to align with longest axis, followed by the x–axis and the y–axis.

- **sq_estimate**

```
sq_estimate(Sq_L,F)
SQ_STRUCT_LIST *Sq_L;
SQ_OPTS_FLAGS  *F;
```

4.4.5 Recovery: Quadric and SuperEllipse

- **sq_quadric** This is actually the superquadric routine which performs the minimization. Features are incorporated in the procedure so that if the user sets the X display flags, the superquadric will be displayed as the recovery process progresses.

```
sq_quadric(S_L, Flags, Fdisp)
SQ_STRUCT_LIST *S_L;
SQ_OPTS_FLAGS  *Flags;
SQ_DISP_FLAGS  *Fdisp;
```

- **sq_quadric_plain** This function provides a "vanilla" version of the superquadric recovery process. No output will be performed. Upon return the appropriate updates will have been accomplished and the history list of the recovered superquadrics will be contained in the *S_L* structure.

```
sq_quadric_plain (S_L,Flags)
SQ_STRUCT_LIST *S_L;
SQ_OPTS_FLAGS  *Flags;
```

- **sq_ellipse** It represents the two dimensional counter part of the *sq_quadric* routine. It derives the superellipse fitting a given set of points. Deformations may be performed just as in the regular *sq_quadric* function. *Alok Gupta. May 4, 1990*

```
sq_ellipse(S_L,Flags)
SQ_STRUCT_LIST *S_L;
SQ_OPTS_FLAGS  *Flags;
```

4.4.6 New Structures Allocation

The routines listed in this section provide allocation for structures, with default setting.

- each one of the following calls return clean structures, no special setting of any parameters.

```
SQ_STRUCT      *sq_make_struct();

SQ_STRUCT_LIST *sq_make_struct_list();

SQ_DIST_STRUCT *sq_make_dist_struct();
```

- returns a structure with the following default values

```
SQ_OPTS_FLAGS  *sq_make_opts();
```

  - grid density set: 1
  - calibration set to Uniform : 0
  - number of iterations to perform: 15
  - no deformation when recovering the model: _NO_DEF
  - no erosion on image around border :0
  - coordinate frame, set to Image :0
  - combined goodness of fit off: 0
  - compensation for range image off: 0
  - verbose mode off: 0
  - Xdisplay mode off: 0

- returns a structure with the following default values for the displaying

```
SQ_DISP_FLAGS  *sq_make_disp_flg();
```

  - real time drawing set off: 0
  - hidden line removal on: 1;
  - coordinate system displaying for the sq model off: 0
  - i_matrix on : 1, no transformations
  - x offset and y offset off: 0 (no translation)
  - fg ground color : WHITE
  - bg ground color : BLACK
  - middle ground color : BLACK (to differentiate an object from others)
  - number of lines :0
  - scale for the intensity: 1
  - scale for drawing of superquadric: 1 (eg. 2 to double in size)
  - transformation matrix: IDENTITY

4.4.7 Command Parsing

It contains routines to parse the flags in the *SQ_OPTS_FLGS* from the command line. It maintains a status of the error which occurred at parsing time which can also be accessed by function defined in this module.

- **sq_cmdparse** Parses the input from command line or an array of options specified in as *argv*. Upon success it returns the appropriate values set in the two structures and sets error flags in an error vector accessible through routines defined below.

```
sq_cmdparse(argv,argc,F,infile)
char **argv;
int argc;
SQ_OPTS_FLAGS *F;
char **infile;
```

The options which are considered valid are the following:

[i] *(input file)* in PM FORMAT . [string]

[g] *(grid density)* resolution for subsampling the image. This allows to consider relatively space data. [int > 0]

[n] *(number of iterations)* [int > 0]

[e] *(points to erode)* this is specific to the preprocessing. Given the noise present at the boundary of the object, it was resolved to provide an option to remove one, or more pixels around the object. [int > 0]

[c] *(compensation)* scanner dependent, allows to account for ratio of width and height parameters in the scanner resolution. [switch]

[s] *(termination check)* for the fitting procedure in terms of the quality of fit desired. [switch]

[V] *(verbose output)* during iteration shows the state of the recovery and some of the parameters currently recovered. [switch]

[k] *(calibration)* [int 0 or 1, string]

0 : Non-uniform Gus image; [default]

1 : Uniform Gus' image;

file : User defined scanner parameters

[f] *(fit type)* [numerical encoding of the deformations]

1 : no deformation ..... (11 parameters); [default]

2 : tapering ..... (13 parameters);

4 : bending ..... (13 parameters);

24 : tapering and bending ... (15 parameters);

42 : bending and tapering ... (15 parameters);

[F](*coordinate frame)* It provides the frame of reference respect to which to perform the actual recovery process. [numerical encoding of the deformations]

0 : Image Frame; [default]

1 : Base Frame; (a particular frame of choice used by Frank Solina

2 : Object Frame (i.e. origin is at the centroid.)

file : User defined reference frame [specified as a string]

[X] *(X display of sq's)* [switch] enable the display during the recovery process.

- **sq_parse_errcheck** will return the number of errors occurred.

```
sq_parse_errcheck();
```

- **sq_param_errmsg** given that an error has occurred it will print to "stdout" the relevant error messages. Also returns the number of errors encountered.

```
sq_param_errmsg();
```

- **sq_usage** prints out the usage message.

```
sq_usage();
```

4.4.8 I/O functions

These are routines to perform reading and writing to a file using the fitfile format. (Reading also from *stdin* or from a specified file.)

- **sq_read** reads the parameters for the superquadric structure from stdin. It expects the same format as fit_file

```
sq_read(S_L)
SQ_STRUCT_LIST *S_L;
```

- **sq_read_file** reads the parameters for the superquadric structure from specified file. The format of the file is same as fit_file.

```
sq_file(S,filename)
SQ_STRUCT_LIST *S_L;
char *filename
```

- **sq_print_to_file** it outputs the list of the superquadric structure to the specified file

```
sq_print_to_file(S,filename)
SQ_STRUCT_LIST *S_L;
char *filename
```

4.5 Auxiliary Functions

4.5.1 Top Level Funcs

Here are a group of function calls which allow the user to run the superquadric minimization routine just by specifying either a pm file or a list of points. Some of these functions and others providing the interface are collected to form the standalone program for the superquadric minimization process.

- **sq_super_file** It takes a pm file, flags (F) to qualify the type of processing to be performed on the input, flags (Fdisp) to specify, in case that X display flag is set, how to display the superquadric.(X needs to be set up, see *sq_standalone.c* for an example. It returns in S the linked list of the fittings and a pointer to a list of the points associated with the minimization process. It invokes *sq_pm_points*, *sq_estimate*, *sq_quadric*.

```
sq_super_file(S, F, Fdisp, pm_file)
SQ_STRUCT_LIST *S;
SQ_OPTS_FLAGS *F;
SQ_DISP_FLAGS *Fdisp;
char *pm_file;
```

- **sq_super_file_d** Same as above, uses the default settings for the flags.

```
sq_super_file_d(S,pm_file)
SQ_STRUCT_LIST *S;
char *pm_file ;
```

- **sq_super_points** It takes the points and returns the superquadric model recovered.

```
sq_super_points(S, F, Fdisp)
SQ_STRUCT_LIST *S;
SQ_OPTS_FLAGS *F;
SQ_DISP_FLAGS *Fdisp;
```

- **sq_super_points_d** Similar to above, default flags are used.

```
sq_super_points_d(S)
SQ_STRUCT_LIST *S;
```

4.5.2 Distance

A function to compute the Euclidean distance from a superquadric model to a point. Function tolerance may be specified. The returned structure D contains all the specifics for the distance.

```
sq_mindist(D,Sqptr,def_type,ftol)
SQ_DIST_STRUCT *D;
SQ_STRUCT *Sqptr;              /* superquadric structure associated */
int def_type[_NUM_DEF];        /* type of deformations */
double ftol;                   /* function tolerance in evaluation */
```

4.5.3 Edges Computation and Curvature

Edges Computation

- **sq_edges** computes the edges of the given superquadric model. The edge list return is a 2d array with the list of the angles associated with the location of the edges. It is computed by holding one of the angle fixed, setting it to zero, then deriving the position of the others. Thus, in the first array η is set to 0 and ω varies, and vice–versa in the second one. The first value in the arrays is a flag set either to 0 or 1. It is set to 0 if there are no edges with respect to the direction of η or ω. Consider for instance the case of a cylinder allined with the z–axis, that would have no edges on the xy–plane.

```
sq-edges(S,edge_list)
SQ_STRUCT *S;
double edge_list[2][5];
```

Sq_sample_angle Samples the surface in the specified interval and returns appropriate angle list.

- **sq_sample_angles** recursively dissects parameters η and ω to get parallel and meridians such that the dot product of unit normal is less that constant STEP

```
sq_sample_angles(nu1, nu2, om1, om2, ANs, e1, e2)
double nu1, nu2, om1, om2;
double ANs[MAXIMUM_LINES]
double e1, e2;
```

4.5.4 Additional Points

This file contains the function to supplement the original points on the object by returning a list of points on the supporting plane. The procedure uses a list of supporting points (not the original points), the transformation matrix from world coordinate system to the object coordinate system (as returned by sq_estimate() for the original points), the dimension of the object along z axis in object coordinate system (as returned by sq_estimate() for the original points), and three points on the background plane expressed in world coordinate system.

Supporting points are defined as the 3–D points immediately next to the border of the object. In the absence of any other information, the object is most likely to be supported (physically) by them. The correct way of using this procedure is to use it after getting initial estimates for the original points. This makes the transformation matrix and dimz available. The three points on the background are needed to define the background plane. Notice that the support points and the background points are in WCS.

For theory, results etc. see: [Gupta, Lea and Wohn] Proceedings of SPIE's *Intelligent Robots and Computer Vision*, VIII conference, Nov 1989 Philadelphia.

- **sq_addpoints**

```
double **  sq_addpoints(s_points,np,na,T,p1,p2,p3,dimz)

double s_points[][3];          /* list of support points */
int np;                        /* number of support points */
int na;                        /* number of points to add */
double T[4][4];                /* Trans. matrix from world to obj */
double p1[3],p2[3],p3[3];      /* 3 points on the bg in w.c.s */
double dimz;                   /* estimated dim. along major axis */
```

4.5.5 Math Functions

These are mathematical functions which are associated with the library.

- **sq_pow_c**, **sq_pow_s**. These two functions raise the *cos()* and *sin()* of the value x to the y power handling special cases of the signs of the *cos()* and *sin()* functions.

```
double sq_pow_c(x,y)
double x,y

double sq_pow_s(x,y)
double x,y
```

- **sq_sqr** standard square function

```
double sqr(x)
double x
```

- **sq_dist_3D** computes the distance between two points, A and B, in 3d.

```
double sq_dist_3D(A,B)
double A[3], B[3];
```

- **sq_matrix_mult** Multiplies a vector with a matrix. The vector in question has 3 component and in the actual matrix multiplication with a 4x4 matrix the fourth component is assumed to be 1.

```
sq_matrix_mult(vector, matrix, result)
double vector[4], matrix[4][4], result[4]
```

- **sq_invert_matrix** performs matrix inversion

```
invert_matrix(T, T_inv)
double T[4][4], T_inv[4][4]
```

- **sq_mult2matrix** performs two matrix multiplication $T1 \times T2 = T3$

```
mult2matrix(T1, T2, T3)
double T1[4][4], T2[4][4], T3[4][4]
```

- **sq_identity_matrix** returns an identity matrix

```
sq_identity_matrix(T)
double T[4][4]
```

- **sq_homog_trans** It returns the homogeneous transform (euler angles) [*Uses Z–Y–Z Euler angles.*]

  – Euler's angles:

  phi – rot. about the z–axis,

  omega – rot. about the y–axis,

  psi – rot. about the z-axis.

  – translation vector x,y,z.

  Start with the frame coincident with a known frame **A**. First rotate **B** about 'Zb' by an angle 'phi', then rotate about 'Yb' by an angle 'omega', and then rotate about 'Zb' by an angle 'psi'.

```
sq_homog_trans(phi, omega, psi, x, y, z, TR)
double phi, omega, psi, x, y, z, TR[4][4];
```

- **sq_fit_plane** through three given points, p1, p2, p3

```
sq_fit_plane(p1, p2, p3, a, b, c, d)
double p1[3], p2[3], p3[3];
double *a, *b, *c, *d;
```

- **sq_determinant** returns the determinant of a 3x3 matrix.

```
double sq_determinant(matrix)
double matrix[3][3];
```

4.5.6 Dynamic Allocation Routines

These routines perform dynamic allocation and reallocation of 2D and 3D arrays used in the sq library.

The functions will return:

- 0 if the memory can not be allocated – same as malloc()
- -1 on failure caused by one of the sizes being to small
- 1 upon success

"allocate" function will return a NULL pointer on failure due cases 0 or -1 above. "realloc" will not modify the original pointer. In both cases the appropriate flags will be returned

CAVEAT: dimension restrictions are imposed on d1 and d2 because, if a lower dimension is specified, then one might expect to use the indexing specified by the type of array allocation routine invoked. For example, allocating an array of size 1 would require indexing using a[i][0]. This would, in effect, be a two dimensional array with only one element along one direction.

- **sq_alloc_2arr_int**

```
sq_alloc_2arr_int(s,size,dim)
int ***s;
unsigned int size;
unsigned int dim;
```

- **sq_alloc_2arr_double**

```
sq_alloc_2arr_double(s,size,dim)
double ***s;
unsigned int size;
unsigned int dim;
```

- **sq_realloc_2arr_int** reallocate to 's' a two dimensional array with rows of dimension 'dim' resize the array. Also updates to the size of the smaller of the two.

```
sq_realloc_2arr_int(s,size,new_size,dim)
int ***s;
unsigned int size;
unsigned int new_size;
unsigned int dim;
```

- **sq_realloc_2arr_double** reallocate to 's' a two dimensional array with rows of dimension 'dim' resize the array. information is kept also updated to the size of the smaller of the two.

```
sq_realloc_2arr_double(s,size,new_size,dim)
double ***s;
unsigned int size;
unsigned int new_size;
unsigned int dim;
```

- **sq_alloc_3arr_int**

```
sq_alloc_3arr_int(s,size,d1,d2)
int ****s;
unsigned int size;
unsigned int d1,d2;
```

- **sq_realloc_3arr_int**

```
sq_realloc_3arr_int(s,size,new_size,d1,d2)
int ****s;
int size;
int new_size;
int d1,d2;
```

4.5.7 Display

- **sq_display** accepts a superquadric structure and some specifications regarding the modality of the display. It displays using X11 routines. The display should have been set up prior to invocation.

```
sq_display(S,Flags,def_type)
SQ_STRUCT *S;
SQ_DISP_FLAGS *Flags;
int def_type[_NUM_DEF];
```

- **sq_compute_wireframe** It computes the wireframe from the superquadric specification provided in *S*. The lines defining the superquadric are returned the *flg* structure.

```
sq_compute_wireframe(S,flg,def_type)
SQ_STRUCT *S;
SQ_DISP_FLAGS *flg;
int def_type[_NUM_DEF];
```

- **sq_compute_coord_frame** It computes the coordinate frame, returned in coords. The user can specify the length of the axis. Application of the transformation to the world coord system as provided in the transformation matrix in *flg*.

```
sq_compute_coord_frame(S,flg,coords,axis_len)
SQ_DISP_FLAGS *flg;
SQ_STRUCT *S;
int coords[3][2][2];                    /* coordinate for axis */
int axis_len;                           /* length of axis */
```

- **sq_compute_coord_frameM** Similar to the previous function, the user may already have both transformations instead of having the routine compute them from the superquadric parameters.

```
sq_compute_coord_frameM(tmatrix,T,coords,axis_len)
double tmatrix[4][4];
double T[4][4];
int coords[3][2][2];                    /* coordinate for axis */
int axis_len;                           /* length of axis */
```

- **sq_draw_coord_frame** draw the actual frame as specified in coord. Foreground and Background color may be specified. Axes are labeled.

```
sq_draw_coord_frame(coord,color_obj,bg_color,xoff,yoff)
int coord[3][2][2];
int color_obj;
int bg_color;
int xoff, yoff;
```

- **sq_transf_line** this function applies the transformation defined in T to the points A and B and returns the line associated with the transformation.

```
sq_transf_line(line,T,A,B)
int line[2][2];
double T[4][4];
double A[3];
double B[3];
```

4.5.8 X11 Routines

In most of these routines there are three variable which frequently appear. These are *color*, an integer 0–31, and *xoff, yoff*, offsets for the object to be drawn. This allows to have coordinates fixed and then modify the position of the object by varying this two components.

- **sq_InitX** sets up the display and the color table

```
int sq_InitX(name,w,h)
char *name;                    /* name to label it with */
int   w,h;                     /* height and width of widget */
```

- **sq_draw_1line** draw a single line specified starting and ending coordinates, color and possible offset.

```
sq_draw_1line(x1, y1, x2, y2, color, xoff, yoff)
int x1, y1,                    /* initial point */
    x2, y2;                    /* end point */
int color;
int xoff, yoff;
```

- **sq_draw_lines** draws a set of lines segments, specified by num_lines of the appropriate color.

```
sq_draw_lines(line, num_lines, color, xoff, yoff)
int line[][2][2];
int num_lines,
    color;
int xoff, yoff;
```

- **sq_draw_lines_ptr** similar to the previous one, only the lines can be specified by a different type of pointer

```
sq_draw_lines_ptr(line, number, color, xoff, yoff)
int ***line;
int number,
    color;
int xoff, yoff;
```

- **sq_draw_points** draws a list of points.

```
sq_draw_points(pt, number, color, xoff, yoff)
int **pt;
int number,
    color;
int xoff, yoff;
```

- **sq_draw_text** displays text at the specified x and y coords. User can specify color of foreground and background.

```
sq_draw_text(x, y, text, xoff, yoff, fg_color, bg_color)
int x, y;                    /* coordinate of text string */
char *text;                  /* pointer to text string */
int xoff, yoff;
int fg_color,                /* color used in the characters */
    bg_color;                /* background color */
```

- **sq_ClearScreen**

```
sq_ClearScreen();
```

- **sq_fill_xdisplay** Set the Xdisplay to a specified graylevel color.

```
sq_fill_xdisplay(grayvalue)
int grayvalue;
```

- **sq_fill_rect** this function fills a rectangle on the screen from the specified coordinates (x,y) by the extent specified by Dx and Dy

```
sq_fill_rect(x,y,Dx,Dy,grayvalue)
int x,y,
   Dx,Dy;                    /* size of rectangle */
int grayvalue;
```

- **sq_flush_xdisplay** flush the display one the buffer is set up.

```
sq_flush_xdisplay();
```

- **sq_set_display_dev** Initialize the display and sets the header to specified character string.

```
sq_set_display_dev(str)
char *str;
```

- **sq_draw_points_2arr** places points on the screen at the coordinates specified in the *pt* array.

```
sq_draw_points_2arr(pt, n, c,xoff,yoff)
int pt[][2];
int n, c;
int xoff, yoff;
```

- **sq_DrawImage8** display a pm picture.

```
sq_DrawImage8(p,x,y,w,h)
byte *p;
int x,y,w,h;
```

- **sq_RedrawImage** redraws the image at the given location from stored data instead of reloading from file.

```
sq_RedrawImage(p,x,y,w,h)
byte *p;                    /* pointer to the image */
int x,y,                    /* location to draw the image */
    w,h;
```

- **sq_SaveImage8** Save a pm picture from current display. User can specify starting and ending corners.

```
pmpic *sq_SaveImage8(s_row,s_col,e_row,e_col)
int s_row,s_col,
    e_row,e_col;
```

NOTE this routine alone is dependent on PM information.

Index