

GLSurf 2.3 Documentation

Christophe Raffalli

March 22, 2005

Contents

1 Introduction.	1
2 Installation.	2
2.1 Compilation from source	2
2.2 Binary installation	2
2.3 Issues concerning shared library	2
3 Commands	3
4 Syntax.	5
5 Global parameters.	6
6 Surface and curve parameters.	6
7 Drawing parameters.	8
8 Light parameters.	8
9 Key bindings.	8
10 Mouse control	9
11 Compilation	10
12 Interface with POVRay	10
13 An example.	10

1 Introduction.

GLSurf is a program (similar to Surf: <http://surf.sourceforge.net>) to draw 3D surfaces or 3D curves from their implicit equations (that is drawing the set of points (x, y, z) such that $f(x, y, z) = 0$).

It offers an intuitive and simple syntax to construct your functions, it can draw multiple surfaces simultaneously and it can use all the power of OpenGL to animate the surface, use transparency, etc ...

GLSurf is a “command interpreter”: you type some commands and it executes them ! This document is mainly a list of commands and a description of the different syntactic items.

A good way to use GLSurf is to write a sequence of commands in a file and then type `glsurf file` to execute all the commands or to drag and drop this file on the `glsurf` executable.

2 Installation.

2.1 Compilation from source

You need first to install Objective Caml (recommended version: 3.07), lablGL (version 1.0 or later) and camlimages (version 2.1 or later). For camlimages, it may be better to first install a few library (jpeg, gif, ungif, ...) for Glsurf to support more file format.

First, you need edit the first lines of the Makefile (you may need to play with the library depending on the way OpenGL is installed, but normaly it should work with no modification).

Finally you can type `make opt` (or `make` if your platform does not support ocamlopt) and `make install`. The name of the program is `glsurf`.

2.2 Binary installation

The binary distributions are provided with an executable file (`glsurf.exe` under Windows, `glsurf` under linux) and some examples of script files. You can use it by typing the name of the executable followed by the name of a script file. You can also drag and drop a script file on the executable.

2.3 Issues concerning shared library

Under Windows, `glsurf.exe` needs `opengl32.dll`, `glu32.dll` and `glut32.dll`. The binary distribution contains a copy of this libraries, but you should preferably use the one distributed with your system if you want to use your accelerated 3D graphic cards (note: on my Windows XP, `glut32.dll` was absent, but it could come with the driver of your graphic card).

Remark: under Windows, if you have more than one version of a dll file and you want to be sure to know the one you are using, the best is to put the dll file in the same directory than the executable.

Under Linux, the binary distribution was compiled using Mandrake 9.1 and uses the following library:

```
libXpm.so.4 => /usr/X11R6/lib/libXpm.so.4 (0x40014000)
libpng.so.3 => /usr/lib/libpng.so.3 (0x40032000)
libz.so.1 => /lib/libz.so.1 (0x40057000)
libjpeg.so.62 => /usr/lib/libjpeg.so.62 (0x40065000)
libungif.so.4 => /usr/lib/libungif.so.4 (0x40087000)
libGL.so.1 => /usr/X11R6/lib/libGL.so.1 (0x40090000)
libGLU.so.1 => /usr/X11R6/lib/libGLU.so.1 (0x400bd000)
libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x40156000)
libXmu.so.6 => /usr/X11R6/lib/libXmu.so.6 (0x40164000)
libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x4017a000)
libglut.so.3 => /usr/X11R6/lib/libglut.so.3 (0x4025a000)
libpthread.so.0 => /lib/i686/libpthread.so.0 (0x40291000)
libm.so.6 => /lib/i686/libm.so.6 (0x402e1000)
libdl.so.2 => /lib/libdl.so.2 (0x40303000)
libc.so.6 => /lib/i686/libc.so.6 (0x40306000)
libgcc_s.so.1 => /lib/libgcc_s.so.1 (0x40439000)
libXt.so.6 => /usr/X11R6/lib/libXt.so.6 (0x40443000)
libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x40495000)
libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x4049e000)
libXi.so.6 => /usr/X11R6/lib/libXi.so.6 (0x404b5000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

3 Commands

(* ... *) is a comment. nested comments are not yet supported.

area *ident*; computes the area of a surface previously build using the **surface** command. Note: you compute the surface of the triangulation which is only an approximation.

begin ... end the modification of parameters or values between “begin” and “end” are not visible outside except for the curves and surfaces. This allows to change some parameters to compute some surfaces and curves and then restores the previous values.

color *ident* = (*float*, ...); defines a new color (may have 3 of 4 components: rgb or rgba, the fourth component if for transparency). In fact this command is identical to **vector** and **point**!

curve [**on** *ident1* **refine** *ident2*]*ident* = *expression*; takes an expression *f* using the variables *x*, *y*, *z* and builds the intersection of the surface *ident1*, the surface $f(x,y,z) = 0$ and the bounding cube (see the description of the **origin** and **size** variables). The surface *ident1* should be computed previously using the “surface” command.

If “on *ident1*” if not given, the plane $z = 0$ is used.

If “refine *ident2*” is given, a refined version of the surface *ident1* is computed: to draw a curve, the triangle composing a surface may be divided in smaller triangles. If you draw the curve and the surface *ident1*, the curve may not appear exactly on the surface. If *ident1* and *ident2* are identical, then the refined surface replaces the old one.

The global and curve parameters such as line color, ... are evaluated when the **curve** command is called and the curve will always be draw with these values. Further modification of parameters will not affect previously defined curve (see the sections “global parameters” and “surface and curve parameters”).

delete *ident*, ...; deletes the given surfaces or curves. Surfaces can take quite a lot of memory, so it may be a good idea to delete the unused ones.

draw *string*, ...; draw the surfaces or curves whose name are given (they must have been computed before using the surface or the curve commands). See the section “key bindings” to see how to animate the image.

input *string*; load the file whose name is given.

let *ident* = *expression*; defines a new value for an identifier. Examples:

```
let x = 2 * pi;
```

There are predefined identifiers: **pi**, **true**, **false**, **fastest**, **nicest**, **dont_care**, **off**.

let *ident*(*ident*, ...) = *expression*; defines a new function. You can use as many variables as you want, and you can use this function in future expressions.

```
let f(x,y,z) = x^2 + y^2 + z^2 - sin(x*y*z);
```

note: the only predefined functions are **exp**, **log**, **sin**, **cos**, **sqrt**. Other functions may be added in future version is someone asks for them !

light *int* (*float*, ...); defines the light number *i* (where *i* is the given integer between 0 and 7) at the given position. The position may use affine coordinate (3 components) or projective coordinates (4 components). This light uses the current value of the light color parameters. By default there are two lights at position (0.8, 0.8, 1.0, 0.0) and (-0.8, -0.8, 1.0, 0.0), using the default light color parameters. The last component being zero, this means that the light are at the infinite in the direction given by the three first components.

See the section “light parameters” to set the light color.

light *int* **off**; turn off the light whose number is given.

light *int* **on**; turn on the light whose number is given.

object *ident = objdef*; allow to draw polytopes, lines or points. *objdef* is a list of comment choosed among the following:

```
vertices x1,y1,z1 x2,y2,z2 ...  
triangles a1,b1,c1 a2,b2,c2 ...  
line a1,a2,...  
points a1,a2,...
```

- The vertices commands define an array A of points in space.
- The triangles commands define a list of triangles that belong to the object. The vertices of the triangles are defined as indices in the array A .
- Each line command defines a broken line starting at the first point and ending at the last point. The vertices of the triangles are defined as indices in the array A .
- The points command define a list of points by their indice in the array A .

When the object is build, the parameters **line_...** affect all lines in the object, the parameters **point_...** affect all points and the other parameters affect the triangles.

Here is an example (a cube will all its faces, edges and vertices (See the file `examples/Other/cube`):

```
object cube =  
  vertices 0, 0, 0 0, 0, 1 0, 1, 0 0, 1, 1  
           1, 0, 0 1, 0, 1 1, 1, 0 1, 1, 1  
  triangles 0, 2, 1 1, 2, 3 4, 5, 6 5, 7, 6  
           0, 1, 4 1, 5, 4 2, 6, 3 3, 6, 7  
           0, 4, 2 2, 4, 6 1, 3, 5 3, 7, 5  
  line 0,1,3,2,0  
  line 4,5,7,6,4  
  line 0,4  
  line 1,5  
  line 2,6  
  line 3,7  
  points 0,1,2,3,4,5,6,7;
```

point *ident = (float, ...)*; defines a new point. In fact this command is identical to **vector** and **color**!

print *expression*; prints an expression on the terminal.

quality *ident*; gives the “quality” of the surface whose name is given. The quality of a triangle is the length of biggest side divided by the length of the smallest side. The average and the worst quality are given.

If a surface is not singular or too complex (the warning “`minsize reached`” is not printed), then the worst quality should be smaller that $\sqrt{3} \times 2^{1+\text{max_division_diff}}$

string *ident = string*; Set the value of the given string parameters. Remark: you can not create new variables of type string using this command (as you can with **let**. The only string parameters are: `file_format`, `file_prefix`, `pov_line_texture`, `pov_point_texture`, `pov_surface_texture` and `pov_preamble`).

surface *ident* = *expression*; takes an expression f using the variables x, y, z and builds the intersection of the surface $f(x, y, z) = 0$ with the bounding cube (see the description of the **origin** and **size** variables).

The program builds a triangulation of the surface. The global and surface parameters such as `front_ambient`, ... are evaluated the first time the **surface** command is called. Further modification of these parameters are ignored (see the sections “global parameters” and “surface and curve parameters”).

The color of the surface is controlled by the variables described in the section “surface parameters”.

topology *ident*; computes the Euler characteristics of the surface. It’s main purpose is to check the correctness of the triangulation. The algorithm seems now good enough to compute the correct Euler characteristics in most cases.

More precisely, your surface needs to be smooth (if someone can tell me how to build a correct triangulation of a singular surface ...) and the program should not say “Minsize reached xxx times”. To achieve this you can lower the value of the surface parameter **min_size**, but the computation will take more time and memory.

Note: it is very easy to construct equation of smooth surfaces were some components will never be discovered by GISurf ... But you can always change the surface parameters to try to get the correct answer.

Note bis: it is not so easy to find algebraic smooth (or even analytic) surfaces that are not triangulated correctly and that do not print the “minsize reached” warning !

vector *ident* = (*float*, ...); defines a new vector.

4 Syntax.

int: the usual syntax, examples: 4, 3, -2.

float: the usual syntax, examples: 4, 4.2, -4e12, 0.005.

ident: sequence of characters using [a-zA-Z0-9_] and starting with [a-zA-Z]. Examples: toto, ti-ti, s1.

string: string with usual C syntax. Examples "this is a \"string\"".

expressions: usual syntax, with usual priorities. Examples:

```
let pt1(x,y,z) = z - 1;
let pt2(x,y,z) = -1*cy*y + cz*z + 1;
let pt3(x,y,z) = sin(2*pi/3)*cy*x -cos(2*pi/3)*cy*y + cz*z + 1;
let pt4(x,y,z) = sin(4*pi/3)*cy*x -cos(4*pi/3)*cy*y + cz*z + 1;
let q0(x,y,z) = pt1(x,y,z)*pt2(x,y,z)*pt3(x,y,z)*pt4(x,y,z);
```

There are seven predefined identifiers: **pi**, **true**, **false**, **fastest**, **nicest**, **dont_care**, **off** and five functions: **exp**, **log**, **sin**, **cos**, **sqrt**. Other functions may be added in future version if someone asks for them !

Note: function composition is allowed ! You can write $f(g(x, y, z), h(y))$ if the function have the correct number of arguments.

You can also perform formal transformation of expressions:

derive(expression,variable) means the derivative of an expression (example: $derive(x^2, x)$ means $2 * x$).

simplify(expression) simplifies an expression.

develop(expression) develops an expression.

5 Global parameters.

The current value of the following parameters are used both when you use the **object**, **surface**, **curve** and the **draw** commands:

file_format (type: string, default: "png") the format used to save images (see the binding of the "s" and "r" keys).

file_prefix (type: string, default: "image") the prefix used to generate file names when saving images (see the binding of the "s" key and "r" keys).

origin (type: vector3, default: (0,0,0)) center of the initial cube (the bounding cube).

pov_preamble (type: string, default: "") A preamble added to any POVray file generated. (a default preamble is build from the OpenGL parameters if the value is ""). See section ?? about the POVray interface.

size (type: float, default: 2) diameter of the initial cube (length of the edges).

6 Surface and curve parameters.

The current value of the following parameters are attached to the build object when you call the **surface**, **curve** or **object** commands:

adjust_angle (type: float, default: $\pi/2.0$) to minimize the number of triangles, the cube vertexes are moved if near to the surface. But they are not moved if the angle between the normal of the surface at the new position and the normal at any point on the surface in the same cube is greater than **adjust_angle**.

The value of **adjust_angle** should be greater than the value of **max_angle** otherwise some vertexes can not be moved and a lot of small triangles are not eliminated.

adjust_factor (type float, default 2.0) if l is the size of the smallest cube edge at vertex v , then v will not be moved further than l divided by **adjust_factor**. This parameter must be greater than 2.

adjust_try_newton (type bool, default true) If this parameter is true, a newton algorithm is used to try to move the cube vertexes.

back_ambient (type: vector4, default: (0.2, 0.2, 0.2, 1.0)) ambient color of the back faces of the surface.

back_diffuse (type: vector4, default: (0.3, 0.7, 0.2, 1.0)) diffused color of the back faces of the surface.

back_shininess (type: float, default: 60.0) shininess of the back faces of the surface.

back_specular (type: vector4, default: (0.7, 0.7, 0.7, 1.0)) specular color of the back faces of the surface.

compile (type: bool, default: false) if true, the component of the function are compiled (a C file is produced, compiled and linked with the program). This allows to increase speed for complex function but you need a C compiler installed (see the compilation section).

epsilon (type: float, default: 1e-10) used for numerical computation.

flip_diagonals (type: bool, default: true) When computing surfaces, try to flip edges between two triangles to get a Delaunay (approximatively) triangulation of the surface.

front_ambient (type: vector4, default: (0.2,0.2,0.2,1.0)) ambient color of the front faces of the surface.

front_diffuse (type: vector4, default: (0.7,0.3,0.2,1.0)) diffused color of the front faces of the surface.

front_shininess (type: float, default: 60.0) shininess of the front faces of the surface.

front_specular (type: vector4, default: (0.7,0.7,0.7,1.0)) specular color of the front faces of the surface.

line_color (type: vector4, default: (1.0,1.0,1.0,1.0)) line color (use to display curves).

line_width (type: float, default: 2) line width (use to display curves). This parameter may often be ignored by graphic cards and this may depend upon the value of **line_smooth**. If the graphic card really obeys the OpenGL specification, this parameter should not be ignored when **line_smooth** is **off**. However, this parameter is correctly used in POVRay files.

min_size (type: float, default: **size** \times 0.01, **size** is a global parameter) minimal size of a cube. No cube smaller than this value are considered.

out_max_angle (type: float, default: $\pi/4$) maximum angle between \vec{df} at two vertexes of the same cube. If it is not the case, and if the size of the cube is greater than **min_size**, then the cube is divided in height cubes.

point_color (type: vector4, default: (1.0,1.0,1.0,1.0)) point color (use to display points).

point_size (type: float, default: 2). This parameter may often be ignored by graphic cards and this may depend upon the value of **point_smooth**. If the graphic card really obeys the OpenGL specification (which is not always the case), this parameter should not be ignored when **point_smooth** is **off**. However, this parameter is correctly used in POVRay files.

pov_isosurface (type: bool, default: false) when sending a surface to POVRay, if a polynomial (see the **pov_poly** parameter) is not used then, if the parameter is true GIsurf will output an isosurface (this may result in quite slow rendering). If it is false, GIsurf will output all the triangles in the POVRay file (resulting in a quite large file).

pov_line_texture (type: string, default: "") the texture applied to curves and lines when using POVRay (a texture is build from the OpenGL parameters if the value is ""). See section ?? about the POVRay interface.

pov_point_texture (type: string, default: "") the texture applied to points when using POVRay (a default texture is build from the OpenGL parameters if the value is ""). See section ?? about the POVRay interface.

pov_poly (type: bool, default: true) try to use polynomial (poly) in POVRay files. It works if the function is a polynomial of degree less or equal to seven.

pov_surface_texture (type: string, default: "") the texture applied to surfaces and triangles when using POVRay (a default texture is build from the OpenGL parameters if the value is ""). See section ?? about the POVRay interface.

max_angle (type: float, default: $\pi/8$) maximum angle between the normal of two points on the surface in the same cube. If it is not the case, and if the size of the cube is greater than **min_size**, then the cube is divided in height cubes.

max_division_diff (type: int, default 2) maximum difference between the number of division of adjacent cubes.

min_depth (type: int, default: 2) minimum number of divisions of the initial cube. $8^{\text{min_depth}}$ is the minimum number of considered cubes. Do not increase this parameter to much!

transparent (type: bool, default: false) is this surface transparent ? If the surface is transparent, it is better to have the same front and back colors.

7 Drawing parameters.

The current value of the following parameters are used every time you use the **draw** command:

background (type: vector3, defaults (0.1,0.1,0.3)) the background color.

lmodel_ambient (type: vector4, default (0.2,0.2,0.2,1.0)) ambient color of the light model.

lmodel_twoside (type: bool, default: true) if false, front colors are used for both sides of all surfaces.

fog (type: off|fastest|nicest|dont_care, default: off).

perspective_correction (type: fastest|nicest|dont_care, default: fastest).

line_smooth (type: off|fastest|nicest|dont_care, default: fastest).

point_smooth (type: off|fastest|nicest|dont_care, default: fastest).

polygon_smooth (type: off|fastest|nicest|dont_care, default: fastest).

8 Light parameters.

The lights created using the **light** command will use the current value of these parameters:

light_ambient ambient color for the lights.

light_diffuse diffuse color for the lights.

light_specular specular color for the lights.

9 Key bindings.

A special point named “the target” is placed in front of the observer. Initially the target is the origin, that is the center of the initial cube.

→ rotate the surfaces to the right around the target.

← rotate the surfaces to the left around the target.

↑ rotate the surfaces upward around the target.

↓ rotate the surfaces downward around the target.

Esc|x|X quit the program immediately.

Home|h|H reset the observer position to its initial value.

PageUp|> multiply the distance to the target by $\sqrt{2}$.

PageDown|< divide the distance to the target by $\sqrt{2}$.

4 rotate the observer counter clockwise.

6 rotate the observer clockwise.

2 rotate the observer upward.

8 rotate the observer downward.

1 rotate the observer to the left.

3 rotate the observer to the right.

Space move forward toward the target (in curve mode, from the initial position, you move toward the plane $z = 0$ and never traverses it).

b|B move backward from the target.

c|C toggles the curve mode (see the **Slide** mode of the mouse bindings).

e|E displays the edges of all triangles.

f|F switch between full-screen and normal mode.

m|M see the mouse control section.

p|P prints the position of the observer and the target (the orientation of the observer is also printed as the right, up and front vector).

q|Q quit the current drawing (specified by a `draw` command and processes the next one (or wait until it is ready).

r|R produces a POVray file describing the current scene named “`xxx.d.pov`” where “`xxx`” is set by the **file_prefix** global parameter (default “`image`”), “`d`” is an integer. If the `povray` command is in the PATH, GlSurf calls POVray on this file to produce an image file named “`xxx.d.png`”.

s|S save the current image in a file. The filename is “`xxx.d.yyy`” where “`xxx`” is set by the **file_prefix** global parameter (default “`image`”), “`d`” is an integer and “`yyy`” is set by the **file_format** global parameter (default “`png`”). The supported format are those supported by `camlimages` (some format require that specific library be installed prior to the compilation of `camlimages`).

Note: preexisting files are not replaced nor removed.

10 Mouse control

There are four ways to control the display using the mouse. In all the cases, we use the same principle, if the mouse cursor is where you pressed the button nothing moves. Movement speed is proportional to the distance from this position. This may be confusing at the beginning, but it is nice when you learn how to use it.

Here are the four control modes

Slide mode (activated using the left mouse button) slides the observer right/left/up/down according to the mouse cursor position. In curve mode (see the key binding for “`c`”), from the initial position (use the Home key), you can really drag the curve.

Rotate mode (activated using the right mouse button) rotate the object around the “target point” according to the mouse cursor position (see the PageUp and PageDown key). Initially the target is at position (0,0,0)

Direction mode (activated using the middle mouse button) rotates the observer on its vertical axes when moving the mouse cursor right/left; moves the observer forward when moving the mouse cursor up/down.

Fly mode (activated by pressing the “m—M” key and deactivated by pressing the same key again) rotate the observer up, down, right or left (around horizontal axes) according to the mouse cursor position. You can use this mode together with the “space” key to fly like in a flight simulator (needs some training). Remark: the mouse cursor disappear in this mode.

11 Compilation

The glsurf program can call a C compiler to compile function. The variable “compile” should be true for the program to compile function.

The access to the C compiler is controlled by the following environment variables:

CC (default “gcc”) the compiler name. If the value of CC is NONE, then GlSurf will not try to use a C compiler. This is useful on windows 98 where it seems that GlSurf can not detect the absence of the C compiler automatically.

CCTEST (default “-v”) arguments send to the compiler to test if it works.

CCSHOPTS (defaults “-shared -lm”) the compilation option to compile a shared library from a C file containing the definition of a few functions using floats.

12 Interface with POVRay

GlSurf can produce file for POVRay a free ray tracer.

13 An example.

```
(* construction of a quartic with 10 components *)
(* some initial parameters *)
let compile = true;
let size = 4.2; let min_size =1e-5;
let max_angle = pi/6;

(* four planes on the face of a regular tetrahedron *)
let tetra_angle = pi - (109.47122063449069174*pi/180);
let cz = cos(tetra_angle);
let cy = sin(tetra_angle);

let pt1(x,y,z) = z - 1;
let pt2(x,y,z) = -1*cy*y + cz*z + 1;
let pt3(x,y,z) = sin(2*pi/3)*cy*x -cos(2*pi/3)*cy*y + cz*z + 1;
let pt4(x,y,z) = sin(4*pi/3)*cy*x -cos(4*pi/3)*cy*y + cz*z + 1;

begin (*****)
  vector front_diffuse = (0.2, 0.2, 0.8, 0.5);
  vector back_diffuse = (0.7, 0.7, 0.2, 0.5);
  vector background = (0.2, 0.2, 0.2);

  surface p1 = pt1(x,y,z);
```

```

    surface p2 = pt2(x,y,z);
    surface p3 = pt3(x,y,z);
    surface p4 = pt4(x,y,z);
end (*****)

draw p1,p2,p3,p4;

(* a well chosen sphere *)
let q0(x,y,z) = pt1(x,y,z)*pt2(x,y,z)*pt3(x,y,z)*pt4(x,y,z);

let r1 = 2;
let s1(x,y,z) = x^2 + y^2 + z^2 - r1^2;

begin (*****)
    vector front_diffuse = (0.7, 0.4, 0.4, 0.5);
    vector back_diffuse = (0.7, 0.4, 0.4, 0.5);
    let transparent = true;

    let max_angle = pi/24;
    surface s1 = s1(x,y,z);
end (*****)

draw p1,p2,p3,p4,s1;

(* the intersection of the planes and the sphere *)
curve [on s1] i1 = pt1(x,y,z);
curve [on s1] i2 = pt2(x,y,z);
curve [on s1] i3 = pt3(x,y,z);
curve [on s1] i4 = pt4(x,y,z);

draw i1,i2,i3,i4,s1;

(* the final quartic: the square of the sphere pertubated by the four planes *)
let ra = 0.05;
let a = 0.5;

begin (*****)
vector front_diffuse = (0.2, 0.2, 0.8, 0.5);
vector back_diffuse = (0.7, 0.7, 0.2, 0.5);
let transparent = false; let max_angle = pi/6;

surface q0 = q0(x,y,z) + a*s1(x,y,z)^2 + ra;
end (*****)

print q0(x,y,z);
draw q0,s1,i1,i2,i3,i4;
draw q0,i1,i2,i3,i4;
delete i1,i2,i3,i4;
draw q0;
topology q0;
quality q0;

(* what if we change the sign of the pertubation ? *)
let size = 6.5;

```

```
let ra = 0.01;

begin (*****)
vector front_diffuse = (0.2, 0.2, 0.8, 0.5);
vector back_diffuse = (0.7, 0.7, 0.2, 0.5);
let transparent = false; let max_angle = pi/6;

surface q0 = q0(x,y,z) - a*s1(x,y,z)^2 + ra;
end (*****)

draw q0,s1;
draw q0;
topology q0;
quality q0;
```