Realistic Visualization of 3D and 4D Mathematical Objects

by

Dr. Mirek Majewski
PNG University of Technology
Department of Mathematics & Computer Science
PMB Lae,
Papua New Guinea
E-mail majewski@maths.unitech.ac.pg

Abstract

Most of the popular mathematical packages like Derive, Mathematica or Maple display 3D graphs as an opaque wire frame object. These graphs are usually a finite-point approximation of the real 3D objects. This means that in many cases such graphs are inaccurate and incomplete. In some of these programs, we have an opportunity to change the display and add some shades, change colors or enlightening, however, it does not improve most of the graphs noticeably.

In my paper, I show how to obtain realistic images of many types of 3D mathematical objects: functions of two variables, implicit functions of three variables, parametric functions, some 3D fractals, etc. For this purpose, I use famous mathematical program Maple V and ray-tracing language POV-Ray. In many cases, I use some basic information taken from Maple: equation of a surface, range of variables, camera position, etc. I transform this information into a POV-Ray program and render it on a PC. I can apply also any kind of texture, light and animation, to show correctly all details of my object.

Introduction

I have been teaching advanced mathematics for a long time. Very often, I had to show my students a graph of a function of two variables or a graph of the surface represented by an implicit or parametric equation. Many of these graphs, I could draw by hand. To obtain some others I had to use computer software. Soon, I started using Derive and later Maple to display graphs. In most of the cases graphs obtained this way were good enough to show an approximate shape of the surface. However three problems occurred very soon.

1. All mathematical programs draw 3D graphs as an opaque wire-frame. I observed that some of my students also started to draw graphs using this method. In their mind, a smooth surface has been replaced by an irregular mesh of points and lines.
2. Another problem occurred while modeling some quartics. In many cases the graph of a quartic had unexpected holes or gaps (see fig. 1). This error can be easily explained. Maple, to obtain a graph of a surface, uses a finite 3D array of points, that are calculated from the equation of a surface. These points are obtained as numerical solutions of some equations. Solving of equations numerically can be a

very complicated problem. In many cases the iterating sequence is divergent even in the closest neighborhood of a root.

3. Finally, there are surfaces with small bumps, needles or corns that are thinner than the distance between points of the array representing the surface. In such a case, these tiny elements disappear from the Maple plot.

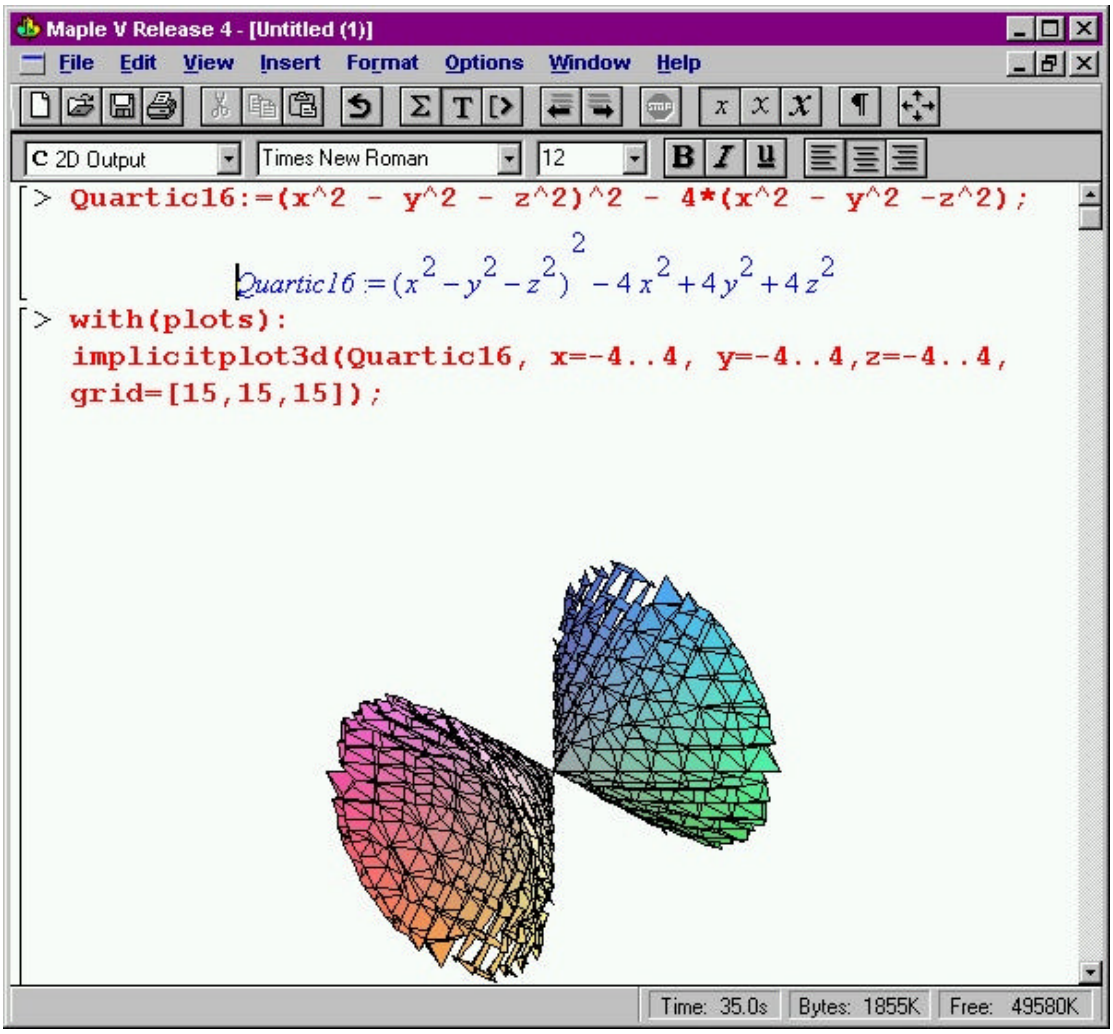

Fig. 1. Maple plot with missing parts of the surface

Due to the problems mentioned, I started looking for tools that would give me an opportunity of realistic and accurate modeling 3D graphs. Realistic – it means with smooth surface, pleasant colors and reflections. Accurate – means without major errors. I decided to use POV-Ray, which is a very powerful language for realistic ray-tracing 3D objects. POV-Ray has a very large library of primitives including polynomial expressions up to order 7 (see [7]). This ray-tracing package is mostly used to obtain pictures of artistic nature and doesn't have a build-in parser of mathematical formulae. I started discussing my problem trough Internet and very soon three people helped me. Daniel Skarda and Tomas Bily from Charles University (Prague, the Czech Republic) made a patch for POV-Ray adding to it a very good parser of mathematical expressions and commands for rendering graphs of implicit and parametric equations. Another person, Ryoichi Suzuki from Electrotechnical Laboratory in Tsukuba, Japan, made another patch for POV-Ray where he included his unique, but very accurate algorithm for rendering isosurfaces. Finally both patches were compiled together and

we got POV-Iso, which is an incredibly strong tool for realistic visualization of 3D graphs of various mathematical equations.

Method

As a first-choice tool, I use Maple V R4, where I obtain a rough shape of the surface, range of variables used in the equation, and the coordinates of that point in a view from which the graph looks the best (see [1]). Then, I print the Maple worksheet with the graph. Maple worksheet can also be saved as an ASCII file and used later to write a POV-Ray description of a scene with my object. In a POV-Ray file I have to declare:

1. appropriate texture for the surface,
2. light sources that enlighten the surface properly,
3. axes of the coordinate system, if necessary,
4. appropriate background,
5. camera (our point of view).

Such a scene, I render using very low resolution and quality of the output. This is only to make sure that all elements of the scene are properly chosen and located. Finally, I render the scene into a very high resolution and good quality picture. The final rendering can be very slow, and the best way to do it is just leave it for overnight.

Some algorithmic objects, I obtain often in a quite different way. First, I write a Pascal program that implements the algorithm and calculates positions of primitives used to design the object. The output from such a Pascal program is written to a POV-Ray file that I later render with appropriate lights, texture, etc. I use this method to obtain graphs of Sierpinski and Ford fractals (see [2], [3], [6]).

Selected Examples

Example 1

Produce the graph of a surface represented by the following equation:

x^2 - y^2 + z^2 - 1=0.

The above equation is an implicit equation of a quadric and represents a hyperboloid of one sheet. Here are the Maple statements necessary to plot the graph of that surface.

```
> MyPoly:=  x^2 - y^2 + z^2 -1:
> with(plots):
> setoptions3d(
>     grid=[25,25,25],
>     labels=[X, Y, Z],
>     axes=BOXED,
>     scaling=CONSTRAINED,
>     shading=XYZ,
>     titlefont=[HELVETICA, BOLD,16],
>     axesfont=[TIMES,ROMAN,12],
>     orientation=[30,70],
>     projection=0.8,
>     labelfont=[HELVETICA,BOLD,14]
> );
> implicitplot3d(MyPoly, x=-1.5..1.5, y=-1.5..1.5, z=-1.5..1.5);
```

Figure 2 depicts the result of the Maple command **`implicitplot3D`**

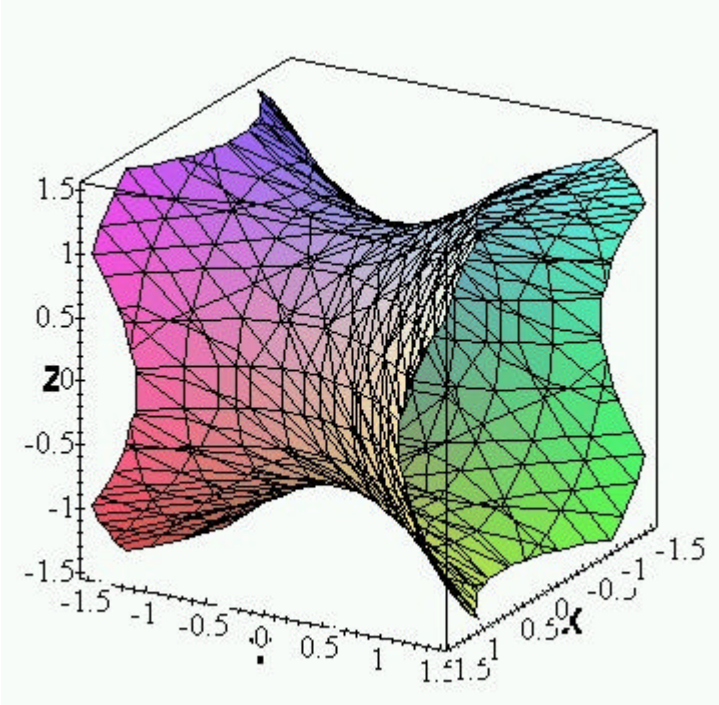

Fig. 2. Maple plot of the equation x^2 - y^2 + z^2 - 1=0

Here is a POV-Iso scene with the same hyperboloid.

```
// Persistence Of Vision raytracer version 3.0 parametric
global_settings {
  ambient_light rgb <1,1,1>*0.8
}

#include "colors.inc"
background {color rgb <1,1,0.8>}

//-- declaration of a Cartesian coordinate system
#declare LGreen = color rgb <0.184314,0.509804,0.184314>
#declare LWhite  = color red 1 green 1 blue 1
#declare L=2
#declare R=0.03
#declare Cart = union {
  cylinder { x*L,-x*L, R
    pigment {
      gradient <1 00>
      color_map {
        [0.0 0.5 colorLWhite color LWhite]
        [0.5 1.0 color red 1  color  red 1]
      }
    }
    no_shadow
    }
  cylinder { y*L,-y*L, R
    pigment {
      gradient <0 1 0>
      color_map {
        [0.0 0.5 colorLWhite  color LWhite ]
        [0.5 1.0 colorLGreen color  LGreen]}
      }
    no_shadow
  }
  cylinder { z*L,-z*L, R
```

```
    pigment {
      gradient <0 0 1>
      color_map {
        [0.0 0.5 color LWhite color LWhite]
        [0.5 1.0 color blue 1 color blue 1]}
      }
    no_shadow
  }
  cone{x*5*R,0,0,2*R pigment{color Red} translate x*L
      no_shadow}
  cone{y*5*R,0,0,2*R pigment{color LGreen}translate y*L
      no_shadow}
  cone {z*5*R,0,0,2*R pigment{color Blue}translate z*L
      no_shadow}
}
object { Cart}

// declarations of lights
light_source {<20, 0,  5> color White}
light_source {< 0, 1.2,  0.8> color White}
light_source { <5,  0, 20> color White}

//-------------------------------------------------
// Camera declarations
#declare theta=radians(30)
#declare psi=radians(70)
#declare d=25
#declare x_pos=d*sin(psi)*cos(theta)
#declare y_pos=d*sin(psi)*sin(theta)
#declare z_pos=d*cos(psi)
camera {
  sky       z
  up        z
  right     -y*1.33
  angle 15
  location  <x_pos, y_pos , z_pos>
  look_at   <0.0, 0.0, 0>
}
isosurface {
   function (x^2) - (y^2) + (z^2) - 1
   clipped_by { box{ <-1.5,-1.5,-1.5>, <1.5,1.5,1.5>}}
   texture {
     pigment {color Gold}
     finish {phong 0.8 }
   }
}
```

Figure 3 shows the output obtained from the above POV-Iso file.

Large part of the above file contains declarations of lights and coordinate system. It is necessary to mention about differences in coordinate systems used by both programs. Maple uses the most classical right-handed Cartesian coordinate system, shown in many geometry textbooks (see [1], [5]). In this system OZ is a vertical axis, OX points to the left-down of the observer and the OY axis points to the right of the observer. Position of the observer is given by two angles, theta (here 30^0) and psi (here 70^0). The triplet (r, theta, psi) are spherical coordinates of an observer position, r is calculated according to the size of the box bounding the graph and required perspective.

POV-Ray uses as a default the left-handed coordinate system with OY as a vertical axis, OX the horizontal axis pointing to the right, and OZ another horizontal axis

pointing into the computer screen. The three POV-Ray statements in camera description:

```
sky       z
up        z
right     -y*1.33
```

I use to redefine the default POV-Ray coordinate system into the right-handed coordinate system used in Maple. The POV-Iso declaration **`isosurface`** is a straightforward translation of the Maple command **`implicitplot3D`**

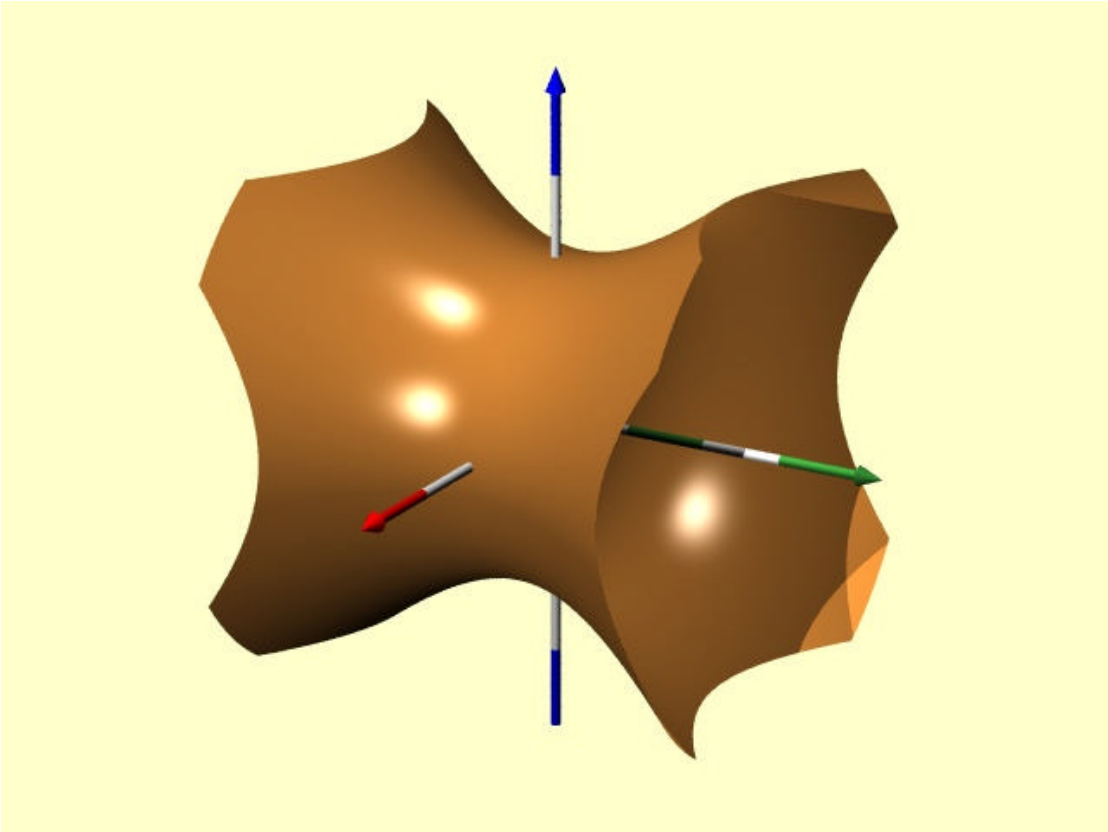

Fig. 3. POV-Iso *graph of the equation x^2 - y^2 + z^2 - 1=0*

Example 2

One of the most remarkable quartic surfaces is the Steiner Surface, which contains ∞^2 conics (see [8]). The surface is named after Steiner who studied it during a visit to Rome, and it is sometimes called the Roman surface. The general equation of the surface is of the form:

$$ay^2z^2 + bz^2x^2 + cx^2y^2 + xyz(fx + gy + hz + k) = 0$$

Let us obtain the picture of the Steiner Surface for a=b=c=1, k=2 and f=g=h=0.

POV-Iso declaration of the above surface may be as follow:

```
//Steiner Surface
isosurface {
  function ((y^2)*(z^2)+(z^2)*(x^2)+(x^2)*(y^2)+ 2*x*y*z)
  clipped_by { box{ <-1.5,-1.5,-1.5>, <1.5,1.5,1.5>}}
  texture {
    Gold_Metal
```

```
  }
}
```

The remaining declarations of the above scene are the same as in Example 1. The surface is located inside the cube 2x2x2 with <0,0,0> as a center. Figure 4 shows the Steiner surface obtained by rendering the above isosurface declaration.

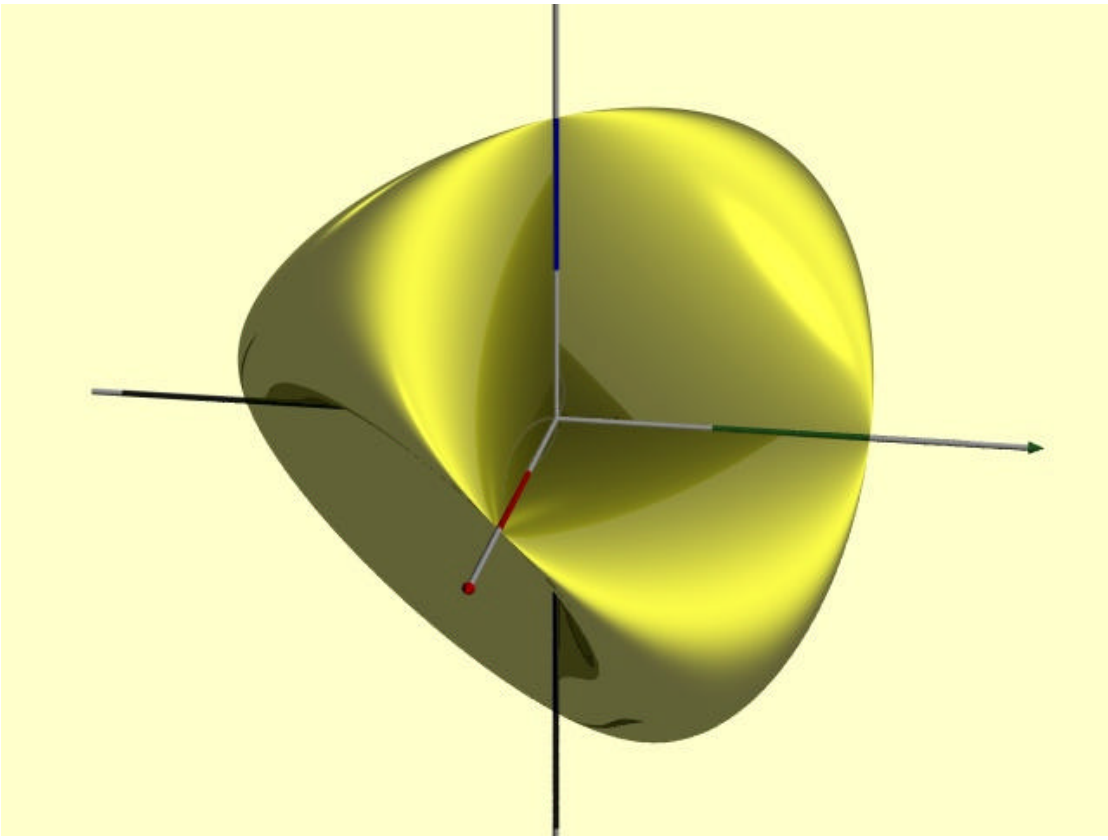

Fig. 4. The POV-Iso graph of the Steiner Surface.

Example 3

Most of the parametric equations represent very interesting shapes. Those in 3D are especially worth of exploring. Here I present only one such equation and the graph of this equation obtained by using POV-Iso. Let us take the parametric equation:

x=sin(u),
y=sin(v),
z=sin(u^2 +2*v^2),

where -3<u<3 and -3<v<3 are real variables. Maple pictures demonstrate that the shape of the surface is full of small elements that must be obtained with a very high accuracy. Maple plot is not good enough for this job. Even using very large grid values, like grid=[100, 100, 100] we still loose some tiny details. POV-Iso may give us a much more correct output. Here is the appropriate POV-Iso declaration.

```
#declare A=3
//Parametric function
   parametric {
     function
     sin(u), sin(v), sin((u^2)+2*(v^2))
     <-A,-A>, <A,A>
```

```
    clipped_by { box{ <-4,-4,-4>, <4,4,4>} }
    texture {
      pigment {color Gold}
      finish {phong 0.8 }
    }
  }
```

The constant A used in the above declaration plays a very important role. I use it to define the range of both variables. Changing this constant, I cut or extend the domain of the parametric function. In such a case, I can loose or add some peripheral parts of the graph. Figure 5 depicts the output from the above POV-Iso declaration.

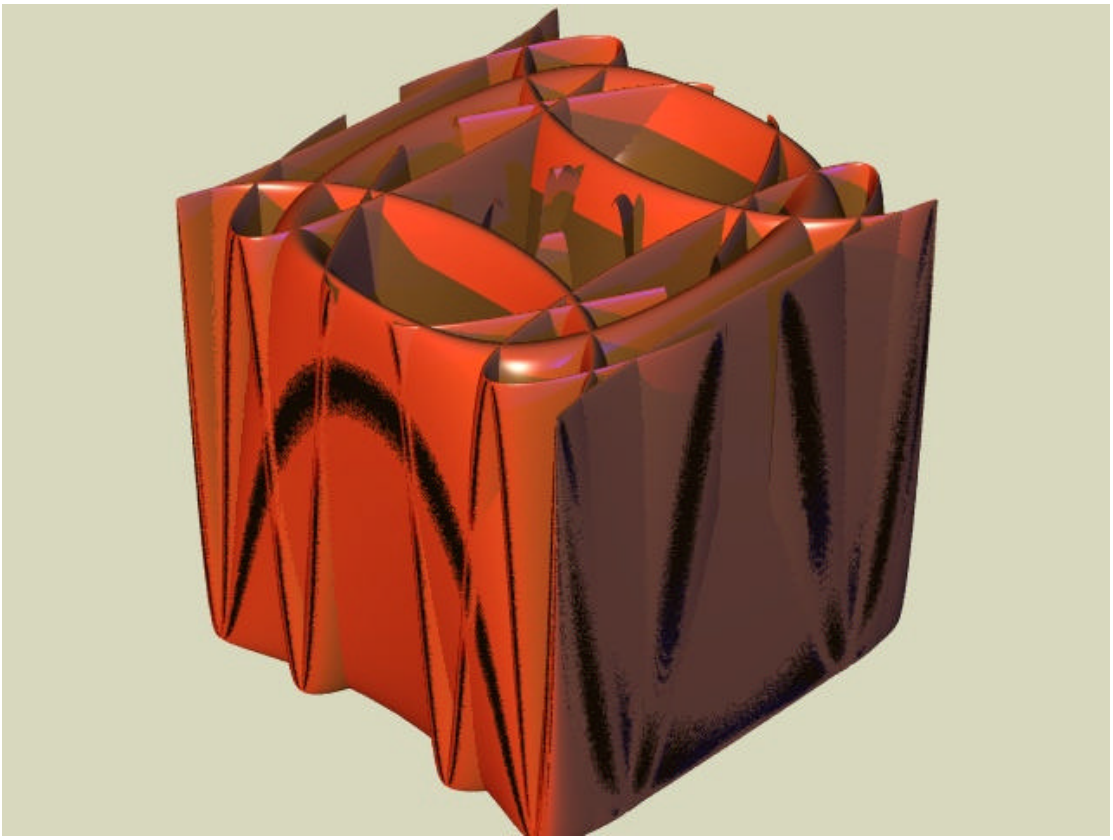

Fig. 5. Surfaces obtained from parametric equations are perhaps the most exciting mathematical objects.

Example 4

POV-Iso language may not be sufficient to define some more complex mathematical equations. Some of them may need an iteration process or very sophisticated external functions. R. Suzuki introduced into POV-Iso the possibility of using external functions that can be defined in any high-level programming language and compiled into a DLL file. Below, enclosed is a graph of one of the 3D density functions. Definition of that function is much more complicated that the functions in all the above examples and can be found in one of the POV-Iso files (for more information, see the last chapter of this paper). Picture 6 illustrates that function obtained by rendering the following POV-Iso declaration:

```
isosurface {
  library   "i_dat3d","test1.dat", <7,7,7,0>
  function  'bdd_sphere", <1.2, 5., 0.879>
  bounded_by {sphere {<0, 0, 0>, 14}}
```

```
  eval
  max_gradient 2.5
  accuracy 0.02
  pigment {color rgb <0.8934, 0.3489,0.654> }
  finish {phong 0.6}
}
```

File “i_dat3d” is a DLL file with definition of the function “odd_sphere”. Test1.dat is an additional file of numerical data required for “i_dat3d.dll”. Finally, the vector <1.2,5.,.879> is a vector of parameters for the function “odd_sphere”.

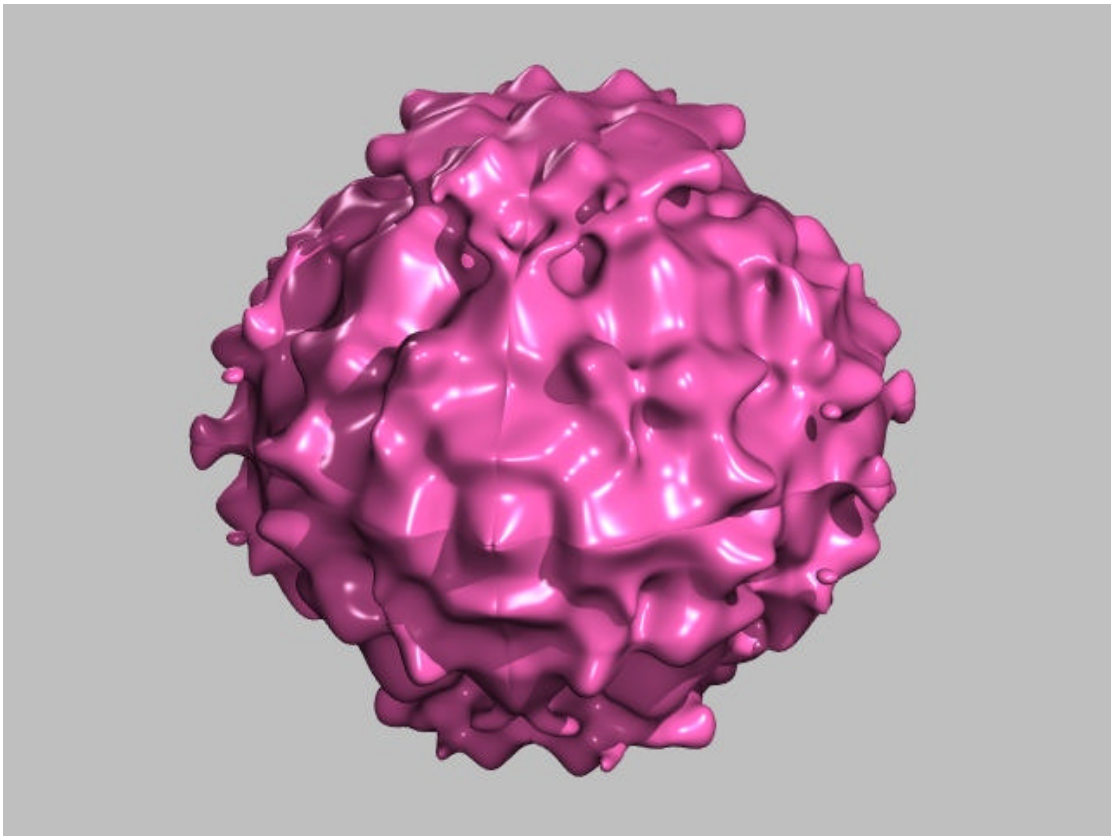

Fig. 6. The graph of the 3D density function.

Going into Higher Dimensions

It is very difficult for us to imagine an object in 4D or more dimensional space. One of the possibilities is to obtain 3D slices of such object. This process is very easy to describe. Suppose, we have an equation F(x,y,z,t)=0 of a 4D surface. Substituting one of the variables with a constant, say t=2, we obtain a projection of the surface into a 3D space. Using different values for variable t we may develop an idea about different stages of the surface. There is a very natural interpretation of that process. The variable t can be considered as a time and for various values of t we obtain a graph of the surface in the various moments of the time. From this stage, there is only one step to computer animation.

Suppose $t_0<t<T$ is an interval of a time, and $t_0<t_1<t_2<\ldots<t_n=T$ is a sequence of real values. For any value t_k we can render the graph of the function $F(x,y,z,t_k)=0$ and further, obtain an animation using these pictures as consecutive frames of that animation. Such process can be extremely time-consuming, however obtained results

may give us very interesting information about the surface over some period of time. In mathematical parlance: we observe how the graph changes along an interval of the OT axis. It is very natural to interpret t as a time, however any other variable can be used instead. Such animation will show the changes of the graph along the other axis. Below, I give only a single and perhaps one of the most interesting examples of a 4D object.

Julia Fractals in 4D Space

The classic Julia fractal is an object generated by using some iteration process in the plane of complex numbers. The Julia fractal object is calculated using an algorithm that determines whether an arbitrary complex number z is inside or outside the object. The algorithm requires generating the sequence of complex numbers:

H(0)=z, and H(n+1)=H(n)*H(n) + p,

where n=0, 1, 2, …, max_iteration, and p is a fixed complex number. I call the number p a seed of the Julia fractal. For detailed discussion of the classic Julia fractal, see [4] or almost any other book about fractals.

The complex number z that begins the sequence is considered inside the Julia fractal if none of the numbers of its sequence H(n) escapes a circle of some fixed radius and with its center at the origin before the iteration reaches the max_iteration constant. The points of the Julia fractal can be colored using different colors for points that escape at the specified level of iteration. Very colorful pictures of 2D Julia fractals are displayed in many books and scientific magazines. These images can be obtained using the well-known program Fractint.

Now let us think how we can generalize the idea of the Julia fractals into higher dimensions. For the beginning we need a bit of history. A great mathematician, Sir William Hamilton was interested in complex numbers. For about 10 years, he tried to extend this concept in order to define a complex volume by searching a second imaginary axis. Such a number would have three components: one real and two imaginary. This, however, for some mathematical reasons, could not be done. Finally, Hamilton discovered that three rather than two imaginary units were needed, with the following properties: i^2=j^2=k^2= -1, ij=k and jk= -k. Hamilton called the number a quaternion (see [9]).

The classic idea of the Julia fractal can be easily applied to quaternions. Instead of complex numbers, we use quaternions and instead of a circle, we use a 4D sphere defined by the equation |q|<R, where R>0 is the radius of the sphere. The resultant object is a 4D Julia fractal. Now we can obtain 3D slices of the 4D Julia fractal or we can design an animation showing how the 4D Julia fractal changes along one of the axes. Using POV-Ray I obtain 3D slices of the Julia fractals. I use here a plural because for different seeds and different values of max_iteration we obtain quite distinct Julia fractals. Instead of the function H(n)*H(n) any other function may be used, such as: H(n)^3, exp, sin, cos, etc. Here is an example of the POV-Ray declaration of a Julia fractal:

```
julia_fractal {
  <.49,.5,-.34,0.1>
  quaternion
  max_iteration 7
  precision 400
```

```
  slice <0,0,0,1>, 0
  texture {
    pigment { colorrgb <.7,.9,.8> }
      finish {phong .3 phong_size 200 }
  }
}
```

Other declarations of the Julia fractal scene, i.e. lights, camera and background, are the same as in Example 1. The above declaration uses a seed <.49,.5,-.34,0.1> which is a quaternion q=0.49 + 0.5*i - 0.34*j + 0.1*k. Using a very low level of iteration, here max_iteration=7, we obtain an interesting mixed-pastry-like shape. For higher values, instead of a continuous object we obtain a fractal dust. I use here a default projection into a 3D space normal to the vector <0,0,0,1> and passing through origin. It is also worth mentioning that the above declaration uses quaternions, however, hypercomplex numbers, that are another generalization of complex numbers, can also be used. Figure 7 shows a 3D Julia fractal slice obtained by rendering the above POV-Ray declaration.

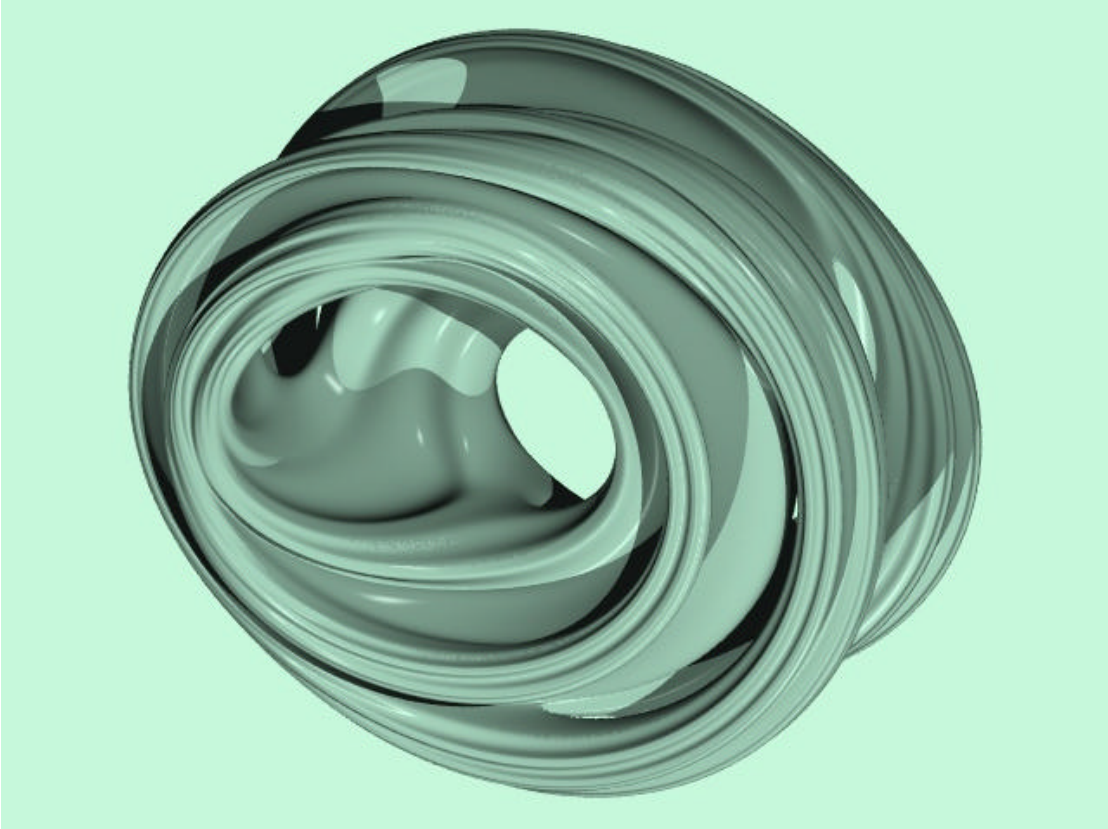

Fig. 7. A 3D slice of the 4D Julia fractal.

Final Information

To describe methods used in my paper, I need much more space and time than I am allowed to use for the conference article, however, more information can be found in the cited literature.

The current version of POV-Ray can be found on the Internet:

http://www.povray.org/ftp/pub/povray/

This site contains links to other POV-Ray pages, groups, utilities, etc.

Information about all POV-Ray features, commands, etc. may be found in POV-Ray documentation. Using POV-Ray for Windows, choose Help/POV-Ray Documentation/Introduction and then print all. Full documentation for POV-Ray is a large document and be prepared to print about 700 pages.

Information about isosurfaces can be obtained from Ryoichi Suzuki:

http://www.etl.go.jp/etl/linac/public/rsuzuki/e/povray/

http://www.public.usit.net/rsuzuki/e/povray/

These are also the Internet sites where you can obtain POV-Iso executable file replacing main POV-Ray file.

Information about other POV-Iso elements: parametric plot, etc. can be obtained from Daniel Skarda:

http://atrey.karlin.mff.cuni.cz/~0rfelyus

Here you can also find information about macro-language that was implemented into POV-Ray by Daniel. This language seems to be very useful in rendering algorithmic 3D structures.

I thank the POV-Ray team for their marvelous work. Without it, my research would not be possible. For me POV-Ray is not only a tool for modeling various mathematical ideas, but also a wonderful adventure that sometimes occupies 24 hours of my day and seven days of the week. Thank you guys for all.

I thank Daniel Skarda, Thomas Bily and Ryoichi Suzuki for their help, continuos improvement of POV-Ray, implementing some of my ideas and sending me new versions of POV-Iso by post (unfortunately Unitech is one of a few universities that do NOT have interactive access to the Internet). I am sure that sometimes I bothered them to much and I wish to thank them for their patience. I believe that our cooperation has been useful for all of us.

Bibliography

1. Hall K.M., Hansen M.L., Rickard K.M., Maple V - Learning Guide, Springer-Verlag, New York, 1996.
2. Majewski Mirek, A Tutorial on the Realistic Visualization of 3D Sierpinski Fractals, (in print) Computers & Graphics, Elsevier Science Ltd.
3. Majewski Mirek, Realistic Visualization of 3D Sierpinski Fractals, PNGJMCE vol.2 (1996), pp. 31-44.
4. McGuire Michael, An Eye for Fractals, Addison Wesley, 1991.
5. Mortenson M.E., Computer Graphics, an introduction to the mathematics and geometry, Heineman Newnes, Oxford, 1989.
6. Pickover Clifford A, Keys to Infinity, John Wiley & Sons., New York, 1995.
7. POV-Ray team, Welcome to POV-Ray (documentation enclosed with POV-Ray version 3 for Windows), 1996.
8. Sommerville D.M., Analytical Geometry of Three Dimensions, Cambridge the University Press, 1947.
9. Watt Alan & Watt Mark, Advanced Animation and Rendering Techniques, theory and practice, Addison-Wesley, 1994.