On the Visualization of Riemann Surfaces

Simo Kivelä

The graphs of complex-valued functions $f:\mathbb{C}\to\mathbb{C}$ or functions of the type $f:\mathbb{R}^2\to\mathbb{R}^2$ are in general two-dimensional manifolds in the space $\mathbb{R}^4$. The article presents a method for the visualization of such a graph. The graph is first projected to three-dimensional space with parallel projection and the image—the surface in three-dimensional space—is rendered on the screen in the usual way. The visualization can be improved in two ways: the graph can be rotated in four-dimensional space or the direction line of the projection can be changed, which means that the observer flies around the graph in four dimensions. The animation and manipulation capabilities of *Mathematica* are appropriate tools for the purpose.

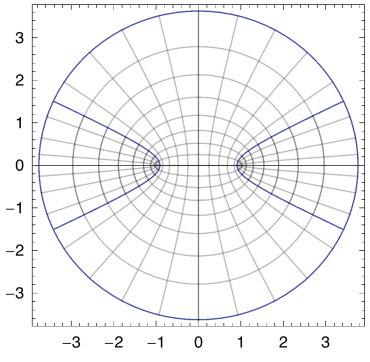
■ Introduction

Complex-valued functions $f:\mathbb{C}\to\mathbb{C}$, that is, functions of the form $f(z)=f(x+i\,y)=u(x,\,y)+i\,v(x,\,y)$, can be visualized by putting a rectangular or a polar grid in the x-y plane and plotting the image of the grid in the u-v plane. The type of grid used depends on the function. More generally, the method can be used for any function $f:\mathbb{R}^2\to\mathbb{R}^2$, $f(x,\,y)=(u(x,\,y),\,v(x,\,y))$.

The standard package `Graphics`ComplexMap`` had tools for visualizing complex-valued functions in this way. The new two-parameter form of `ParametricPlot` now provides the functionality of `Graphics`ComplexMap``. As an example we consider the function $\sin(z)$.

```
In[1]:= ParametricPlot[Through[{Re, Im}[Sin[x + I y]]],
          {x, -2, 2}, {y, -2, 2}, PlotStyle → None]
```

Out[1]=



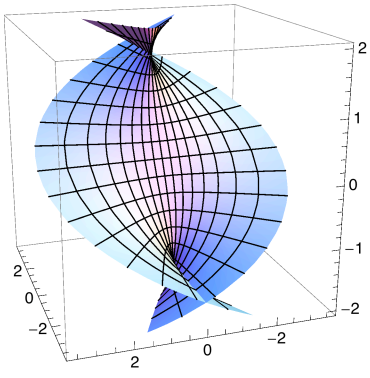
We can say that the map "folds" the x-y plane in some way and the result—the image—is flattened out on the u-v plane. As you can guess from the plot, the image somehow overlaps itself on the left and the right.

Actually, the graph of this type of function is a two-dimensional manifold (a surface) in four-dimensional space $\mathbb{R}^4$ with x, y, u, and v as the axes. To get a better visualization, the surface should not be flattened out in two dimensions. One way to do this is to take a third axis and lift the surface up from the uv plane.

The following two examples show the same $\sin(z)$ function with x and y as the third (vertical) axis.

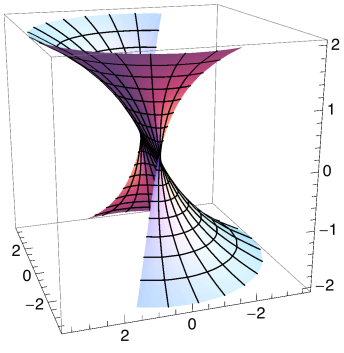
```
In[2]:= w = Sin[x + I y];

In[3]:= s1 = ComplexExpand[{Re[w], Im[w], x}]

Out[3]= {Cosh[y] Sin[x], Cos[x] Sinh[y], x}

In[4]:= ParametricPlot3D[s1, {x, -2, 2}, {y, -2, 2},
         BoxRatios → {1, 1, 1}, ViewPoint → {-3, 1, 1}]
```

Out[4]=




```
In[5]:= s2 = ComplexExpand[{Re[w], Im[w], y}]

Out[5]= {Cosh[y] Sin[x], Cos[x] Sinh[y], y}

In[6]:= ParametricPlot3D[s2, {x, -2, 2}, {y, -2, 2},
         BoxRatios → {1, 1, 1}, ViewPoint → {-3, 1, 1}]
```

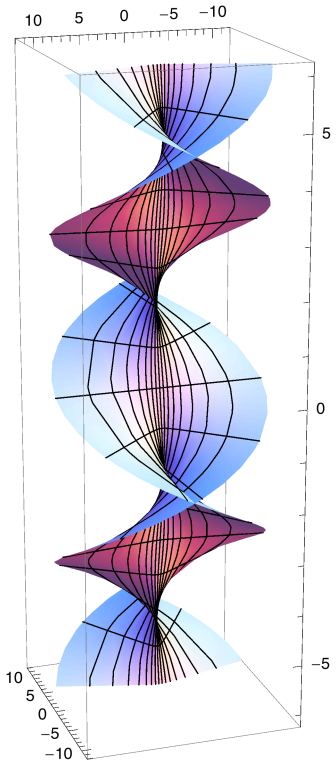
Out[6]=



The plots look very different although the same part of the surface is considered. The difference becomes clear if we take a larger part of the surface.

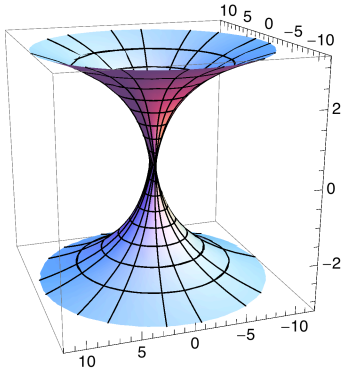
In[7]:=
```
ParametricPlot3D[s1, {x, -6, 6}, {y, -Pi, Pi},
 BoxRatios → {1, 1, 3}, ViewPoint → {-3, 1, 1}]
```

Out[7]=



In[8]:=
```
ParametricPlot3D[s2, {x, -6, 6}, {y, -Pi, Pi},
 BoxRatios → {1, 1, 1}, ViewPoint → {-3, 1, 1}]
```

Out[8]=



In the second plot with y as the vertical axis, the surface overlaps itself so many times that we cannot see its characteristic features. The first plot with x as the vertical axis shows the well-known visualization of $\sin(z)$ as a Riemann surface.

Many good plots of this type and analysis of the corresponding surfaces and functions can be found in [1-4] by Michael Trott and [5] by Corless and Jeffrey.

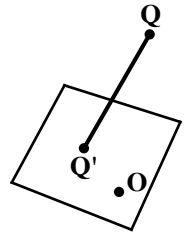
The two plots can be considered from another point of view. They are both images of the graph of the function in a parallel projection $\mathbb{R}^4 \to \mathbb{R}^3$. In the first case, the projection direction is parallel to the y axis and the image (the range of the projection mapping) is the three-dimensional subspace spanned by the three other axes. In the second case, the projection direction is parallel to the x axis. Both projection mappings are orthogonal, that is, the image space (or the range of the mapping) is the orthogonal complement of the projection direction.

This opens a new way to show the graph of a function $f : \mathbb{R}^2 \to \mathbb{R}^2$, $f(x, y) = (u(x, y), v(x, y))$. The graph is a two-dimensional manifold in the space $\mathbb{R}^4$ and any projection mapping $\mathbb{R}^4 \to \mathbb{R}^3$ can be used to form its image in $\mathbb{R}^3$. This is (in general) an ordinary surface and usual methods can be used to plot it. The projection mapping can be a parallel projection or a perspective projection. The parallel projection may have any direction. The parallel projection may be orthogonal or oblique. (Of course, `ParametricPlot3D` also uses some projection methods to plot a two-dimensional graphic of a three-dimensional object on a computer screen. Thus, to display an object from four-dimensional space, two consecutive projection mappings are used.)

In the following, we first develop some tools and then use them to create images and animations of some geometric objects from four-dimensional space.

■ Tools

A parallel projection in three-dimensional space can be defined by giving the direction of the projection (say, a column vector s with three components) and the normal vector of the two-dimensional image plane (a column vector n with three components). The placement of the plane is not important because the image projected on the plane does not change when the plane is moved without changing its direction. Therefore, we may assume that the origin is in the plane. Then, the matrix of the projection mapping is given by $P = I - \frac{s\,n^T}{n^T\,s}$, where I is the identity matrix.



If x is the coordinate vector of the point Q, then $P\,x$ gives the (three-dimensional) coordinate vector for the image point Q′. To get coordinates in the two-dimensional plane, a basis must be chosen and the coordinates transformed appropriately. (See [6] for examples.)

The approach can be generalized to the four-dimensional case and we give the following code to compute the matrix of a parallel projection. Here `s` is the direction of the projection line and `b` stands for the basis vectors of the three-dimensional image space.

```
In[9]:= genProj[s_, b__] := Module[{n, p}, n = Cross[b];
          p =
           IdentityMatrix[Length[s]] - (1 / n.s) Outer[Times, s, n];
          Drop[Inverse[Transpose[{b, s}]].p, -1]]
```

We need a special type of rotation to generate animations in four-dimensional space. Here, a two-dimensional subspace is invariant and the rotation acts in the orthogonal complement of this subspace. If the vectors `a` and `b` span the invariant subspace and `w` is the angle of rotation, the matrix of the rotation mapping can be generated as follows.

```
In[10]:= genRot4D[a_, b_, w_] :=
          Module[{ker, c, d, r, b0, b1}, ker = NullSpace[{a, b}];
           {{c, d}, r} = QRDecomposition[Transpose[ker]];
           b0 = {a, b, c, d};
           b1 = {a, b, Cos[w] c - Sin[w] d, Sin[w] c + Cos[w] d};
           Transpose[Inverse[b0].b1]]
```

We may also need orthonormal bases for orthogonal complements of given vectors. The following function gives an orthonormal basis where the first vectors span the same subspace as the vectors in `b`.

```
In[11]:= ortBasis[b_List] := QRDecomposition[
           Transpose[Flatten[{b, NullSpace[b]}, 1]]][[1]]
```

The graph of the function $\mathbb{R}^2 \to \mathbb{R}^2$ in four-dimensional u-v-x-y space is represented by a net of polygons that is generated by the following function.

```
In[12]:= genNet4D[w_, {x_, x1_, x2_, dx_},
           {y_, y1_, y2_, dy_}] := Module[{t},
           t = N[Table[w, {x, x1, x2, dx}, {y, y1, y2, dy}]];
           Table[Polygon[{t[[i, j]], t[[i + 1, j]],
              t[[i + 1, j + 1]], t[[i, j + 1]], t[[i, j]]}], {i, 1,
            Dimensions[t][[1]] - 1}, {j, 1, Dimensions[t][[2]] - 1}]]
```

Here `w` is the four-dimensional expression of the function with `x` and `y` as variables. For example, `w=f[x,y]` or `w=f[r,phi]`, where

```
In[13]:= f[x_, y_] =
          ComplexExpand[{Re[Sin[x + I y]], Im[Sin[x + I y]], x, y}]

Out[13]= {Cosh[y] Sin[x], Cos[x] Sinh[y], x, y}
```

or

```
In[14]:= f[r_, phi_] =
          ComplexExpand[{Re[(x + I y)^2], Im[(x + I y)^2], x, y}] /.
           {x → r Cos[phi], y → r Sin[phi]}

Out[14]= {r^2 Cos[phi]^2 - r^2 Sin[phi]^2,
          2 r^2 Cos[phi] Sin[phi], r Cos[phi], r Sin[phi]}
```

The following test may also be needed.

```
In[15]:= test4D[x_] := And[VectorQ[x, NumericQ], Length[x] == 4]
```

■ Simple Example

First, we consider the function $f(z) = z^2$. We use polar coordinates to get beautiful pictures.

```
In[16]:= f[r_, phi_] =
          ComplexExpand[{Re[(x + I y)^2], Im[(x + I y)^2], x, y}] /.
           {x → r Cos[phi], y → r Sin[phi]}

Out[16]= {r^2 Cos[phi]^2 - r^2 Sin[phi]^2,
          2 r^2 Cos[phi] Sin[phi], r Cos[phi], r Sin[phi]}

In[17]:= gr4D = genNet4D[f[r, phi],
            {r, 0, 2, 1 / 5}, {phi, -Pi, Pi, Pi / 36}];
```

We generate a parallel projection to view an animation. The projection direction is parallel to the vector (0, 0, 0, 1). The three other vectors form the basis of the image space (which is the orthogonal complement of the projection direction).

```
In[18]:= b = {{0, 0, 0, 1}, {1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}};
         p = Apply[genProj, b];
         p // MatrixForm

Out[19]//MatrixForm=
          ⎛1  0  0  0⎞
          ⎜0  1  0  0⎟
          ⎝0  0  1  0⎠
```

In the animation we rotate the graph of the function in four-dimensional space. The rotation fixes the u-v plane (i.e., the vectors (1, 0, 0, 0) and (0, 1, 0, 0) are invariant).

```
In[20]:= q = genRot4D[{1, 0, 0, 0}, {0, 1, 0, 0}, Pi / 18.];
         q // MatrixForm

Out[21]//MatrixForm=
          ⎛1.  0.     0.         0.       ⎞
          ⎜0.  1.     0.         0.       ⎟
          ⎜0.  0.  0.984808  -0.173648     ⎟
          ⎝0.  0.  0.173648   0.984808     ⎠
```

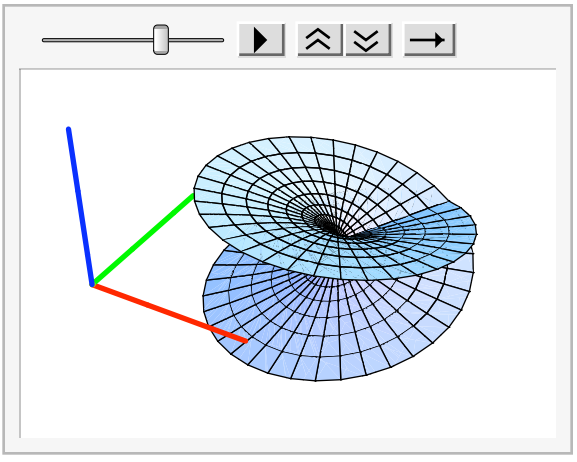
Next to the graph of the function we also plot the four axes of the four-dimensional space. The following gives the necessary definition.

```
In[22]:= d = 5; dst = 5; ax4D = {Thickness[0.01],
           {RGBColor[1, 0, 0], Line[{{0, 0, 0, 0}, {d, 0, 0, 0}}]},
           {RGBColor[0, 1, 0], Line[{{0, 0, 0, 0}, {0, d, 0, 0}}]},
           {RGBColor[0, 0, 1], Line[{{0, 0, 0, 0}, {0, 0, d, 0}}]},
           {RGBColor[0.8, 0.8, 0],
            Line[{{0, 0, 0, 0}, {0, 0, 0, d}}]}} /.
         x_?test4D :> x + {-dst, -dst, 0, 0};
```

In the animation we may rotate the graph or we fly around it, that is, rotate the projection mapping. Rotating the graph gives the following animation.

```
In[23]:= d = 5.2; opts =
          {Boxed → False, PlotRange → {{-d, d}, {-d, d}, {-d, d}}};

In[24]:= gr3D = Table[Graphics3D[{ax4D /. x_?test4D :> p.x, gr4D /.
                x_?test4D :> p.Nest[q.# &, x, k]}, opts], {k, 0, 35}];
         ListAnimate[
          gr3D]
```

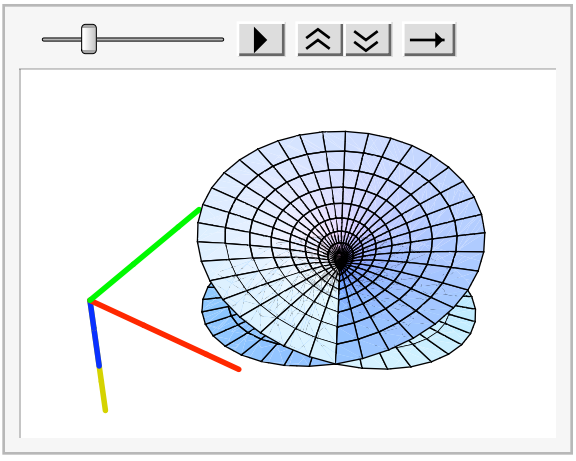
Out[25]=



Here is an animation of the flight around the graph.

```
In[26]:= gr3D = Table[pk = Apply[genProj, Map[Nest[q.# &, #, k] &, b]];
           Graphics3D[{gr4D, ax4D} /. x_?test4D :> pk.x, opts],
           {k, 0, 35}];
         ListAnimate[gr3D]
```

Out[27]=

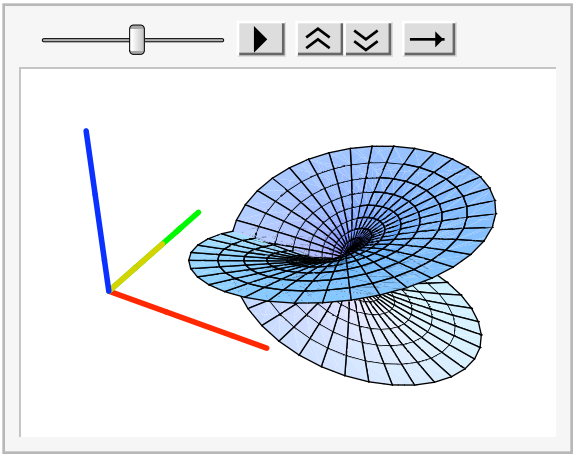


We get a more interesting animation if the rotation happens in the v-y plane.

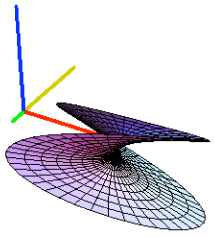
```
In[28]:= q = genRot4D[{1, 0, 0, 0}, {0, 0, 1, 0}, Pi / 18.];
```

```
In[29]:= gr3D = Table[pk = Apply[genProj, Map[Nest[q.# &, #, k] &, b]];
           Graphics3D[{gr4D, ax4D} /. x_?test4D :> pk.x, opts],
           {k, 0, 35}];
         ListAnimate[gr3D]
```

Out[30]=



This frame shows that the surface does not intersect itself, so there is no real intersection in four dimensions, either.



■ Function sin (*z*)

We may analyze the function sin(*z*) in a similar manner. In this case, it is better to use rectangular coordinates.

In[31]:=
```
f[x_, y_] =
 ComplexExpand[{Re[Sin[x + I y]], Im[Sin[x + I y]], x, y}]
```

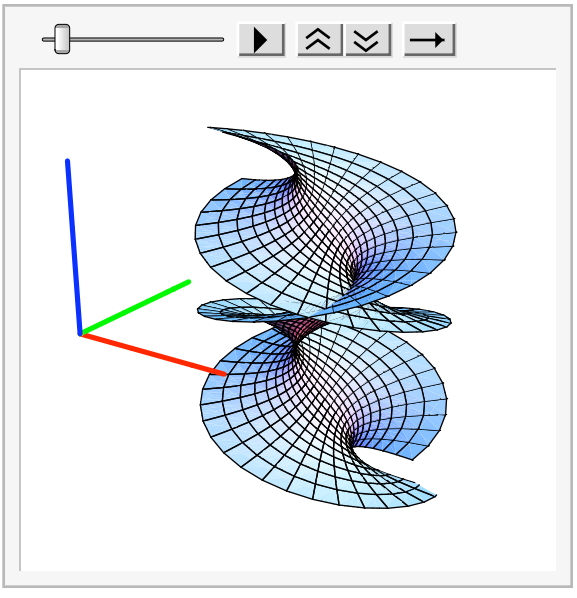
Out[31]=
```
{Cosh[y] Sin[x], Cos[x] Sinh[y], x, y}
```

In[32]:=
```
gr4D = genNet4D[f[x, y], {x, -5, 5, 1 / 5}, {y, -2, 2, 1 / 5}];
```

In[33]:=
```
q = genRot4D[{1, 0, 0, 0}, {0, 1, 0, 0}, Pi / 18.];
```

In[34]:=
```
gr3D = Table[Graphics3D[{ax4D /. x_?test4D :> p.x, gr4D /.
       x_?test4D :> p.Nest[q.# &, x, k]}, opts], {k, 0, 35}];
ListAnimate[
 gr3D]
```

Out[35]=



■ Toward a Better Animation

With the function `Manipulate` we can create animations in which the user may easily control the view.

We will use four-dimensional spherical coordinates for defining the necessary parallel projection mappings. The viewing direction—or the direction of the projecting line—is then

```
In[36]:= s = {Sin[th1] Sin[th2] Cos[ph],
              Sin[th1] Sin[th2] Sin[ph], Sin[th1] Cos[th2], Cos[th1]};
```

where `th1`, `th2`, and `ph` are the three angles of spherical coordinates. The three-dimensional image space is the orthogonal complement of this direction. It is spanned by the following orthonormal vectors:

```
In[37]:= e1 = D[s, th1]; e2 = D[s, th2] / Sin[th1];
         e3 = D[s, ph] / Sin[th1] / Sin[th2];

In[38]:= b = {s, e1, e2, e3};
         b // MatrixForm
```

```
Out[39]//MatrixForm=
⎛ Cos[ph] Sin[th1] Sin[th2]   Sin[ph] Sin[th1] Sin[th2]   Cos[th2] Sin[th1]   Cos[th1] ⎞
⎜ Cos[ph] Cos[th1] Sin[th2]   Cos[th1] Sin[ph] Sin[th2]   Cos[th1] Cos[th2]   -Sin[th1] ⎟
⎜     Cos[ph] Cos[th2]             Cos[th2] Sin[ph]           -Sin[th2]           0    ⎟
⎝        -Sin[ph]                     Cos[ph]                    0                0    ⎠
```

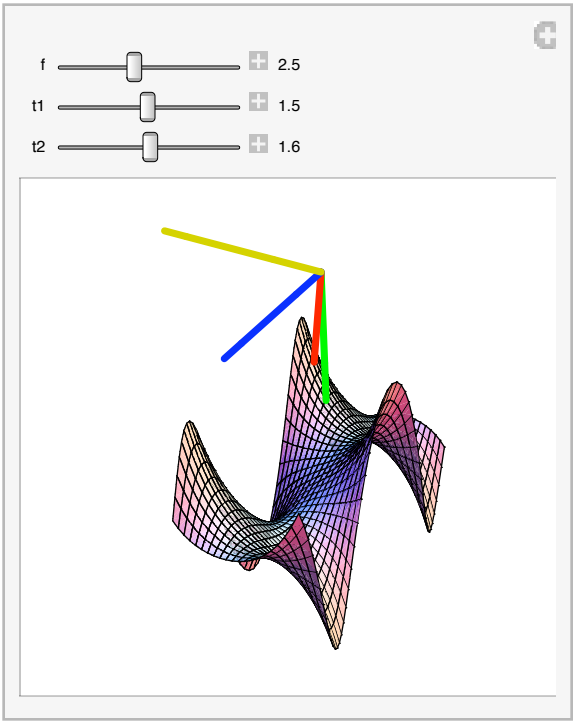
The projection mapping is

```
In[40]:= p = Simplify[Apply[genProj, b]]

Out[40]= {{Cos[ph] Cos[th1] Sin[th2],
           Cos[th1] Sin[ph] Sin[th2], Cos[th1] Cos[th2], -Sin[th1]},
          {Cos[ph] Cos[th2], Cos[th2] Sin[ph], -Sin[th2], 0},
          {-Sin[ph], Cos[ph], 0, 0}}
```

and with `Manipulate` we get an animation in which the user may change the viewing direction in four dimensions—that is, fly freely around the graph.

```
In[41]:= d = 7.0; opts = {Boxed → False,
           PlotRange → {{-d, d}, {-d, d}, {-d, d}}, ImageSize → 400};

In[42]:= MManipulate[q = p /. {ph → f, th1 → t1, th2 → t2}; Graphics3D[
           {ax4D /. x_?test4D :→ q.x, gr4D /. x_?test4D :→ q.x}, opts],
         {{f, 2.5}, 0, 2 Pi, 0.1, Appearance → "Labeled"},
         {{t1, 1.5}, 0, Pi, 0.1, Appearance → "Labeled"},
         {{t2, 1.6}, 0, Pi, 0.1, Appearance → "Labeled"},
         SaveDefinitions → True,
         AutorunSequencing → {{1, 60}, {2, 30}, {3, 30}},
         ContinuousAction → False]
```

Out[42]=



■ Final Remarks

The tools developed in this article present one way to understand the nature of complex functions or functions $\mathbb{R}^2 \to \mathbb{R}^2$. These methods apply not only to the graphs of these functions, but to any geometric object in four-dimensional space. For example, the properties of the Klein bottle can be studied given a four-dimensional parametric representation.

From the technical viewpoint, another approach could also be used: instead of first projecting the objects to the space $\mathbb{R}^3$ and then to the two-dimensional computer screen, only one projection mapping $\mathbb{R}^4 \to \mathbb{R}^2$ could be used.

The methods and material presented here were developed in a Finnish project MatTaFi, matta.hut.fi/mattafi. The aim of the project is to study and produce computer-based materials for basic courses of mathematics at the university level.

■ References

[1] M. Trott, "Visualization of Riemann Surfaces of Algebraic Functions," *Mathematica in Education and Research*, **6**(4), 1997 pp. 15-36.

[2] M. Trott, "Visualization of Riemann Surfaces IIa, Compositions of Elementary Transcendental Functions," *The Mathematica Journal*, **7**(4), 2000 pp. 465-496.

[3] M. Trott, "Visualization of Riemann Surfaces IIb, Compositions of Elementary Transcendental Functions," *The Mathematica Journal*, **8**(1), 2001 pp. 50-62.

[4] M. Trott, "Visualization of Riemann Surfaces IId, Compositions of Elementary Transcendental Functions," *The Mathematica Journal*, **8**(4), 2002 pp. 532-562.

[5] R. M. Corless and D. J. Jeffrey, "Graphing Elementary Riemann Surfaces," *ACM SIGSAM Bulletin: Communications in Computer Algebra*, **32**(1), 1998 pp. 11-17.

[6] I. D. Faux and M. J. Pratt, *Computational Geometry for Design and Manufacture (Mathematics and Its Applications)*, New York: John Wiley & Sons, 1979.

[7] B. Thaller, "Visualization of Complex Functions," *The Mathematica Journal*, **7**(2), 1998 pp. 163-181.

Simo Kivelä, "On the Visualization of Riemann Surfaces," *The Mathematica Journal*, 2010. dx.doi.org/doi:10.3888/tmj.11.3–6.

About the Author

Simo K. Kivelä is an emeritus lecturer of mathematics. He has used mathematical software like muMath, Matlab, and *Mathematica* in several courses, beginning in the 1970s. He has also been the supervisor of projects where information and communication technologies are used in teaching and studying mathematics. He has continued this work as a part-time researcher after retirement.

Simo Kivelä
Helsinki University of Technology,
Institute of Mathematics,
P.O. Box 1100,
FI-02015 HUT, Finland
simo.kivela@tkk.fi