

On the Visualization of Riemann Surfaces

Simo Kivelä

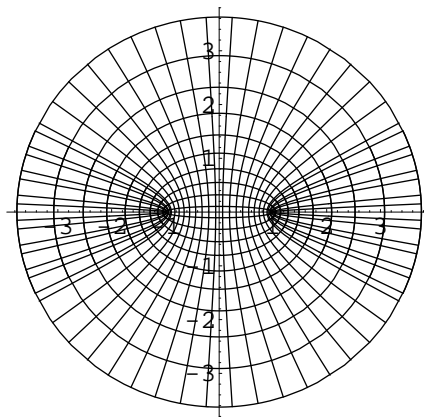
Helsinki University of Technology, Institute of Mathematics, P.O.Box 1100, FI-02015 HUT,
Finland
simo.kivela@tkk.fi

■ Introduction

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

The standard package `Graphics`ComplexMap`` gives tools for visualizing complex-valued function in this way. In the example the function $\sin(z)$ is considered. `CartesianMap` uses a rectangular grid in the domain.

```
<< Graphics`ComplexMap`  
CartesianMap[Sin, {-2, 2}, {-2, 2}, Lines -> {32, 20}]
```



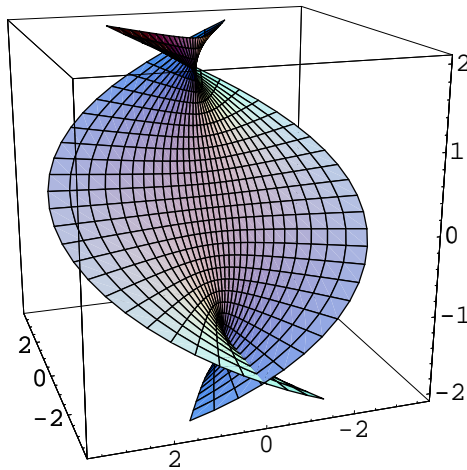
- Graphics -

We can say that the map folds the xy -plane in some way and the result — the image — is flattened out on the uv -plane. As can be guessed in the picture above, the image somehow overlaps itself in the left and in the right part of the picture.

Actually, the graph of a function of this type lives in the four-dimensional space \mathbb{R}^4 with x , y , u and v on the axes. The graph is a two-dimensional manifold, a surface. For getting a better visualization, the surface should not be flattened out. One possibility to do this, is to take a third axis and lift the surface up from the uv -plane.

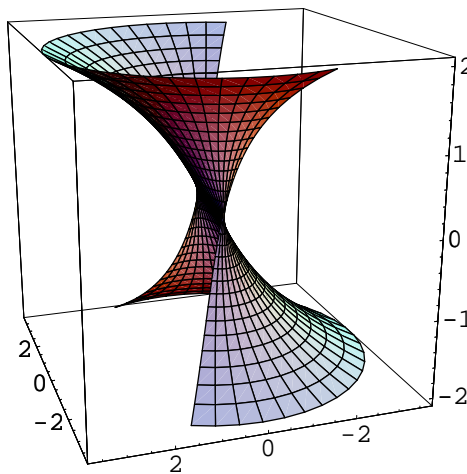
The following examples show the same function as above with x and y as the third (vertical) axis.

```
w = Sin[x + I y];
s1 = ComplexExpand[{Re[w], Im[w], x}]
{Cosh[y] Sin[x], Cos[x] Sinh[y], x}
ParametricPlot3D[s1, {x, -2, 2}, {y, -2, 2},
  BoxRatios -> {1, 1, 1}, ViewPoint -> {-3, 1, 1}]
```



- Graphics3D -

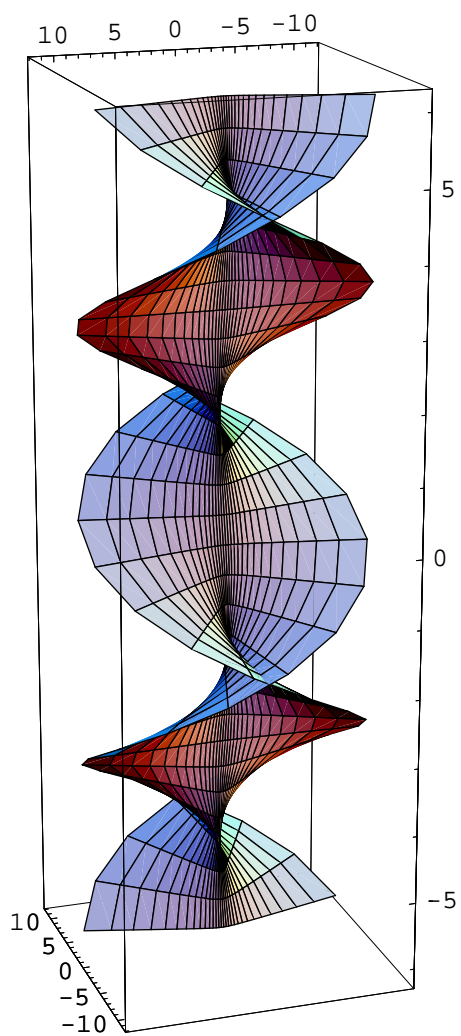
```
s2 = ComplexExpand[{Re[w], Im[w], y}]
{Cosh[y] Sin[x], Cos[x] Sinh[y], y}
ParametricPlot3D[s2, {x, -2, 2}, {y, -2, 2},
  BoxRatios -> {1, 1, 1}, ViewPoint -> {-3, 1, 1}]
```



- Graphics3D -

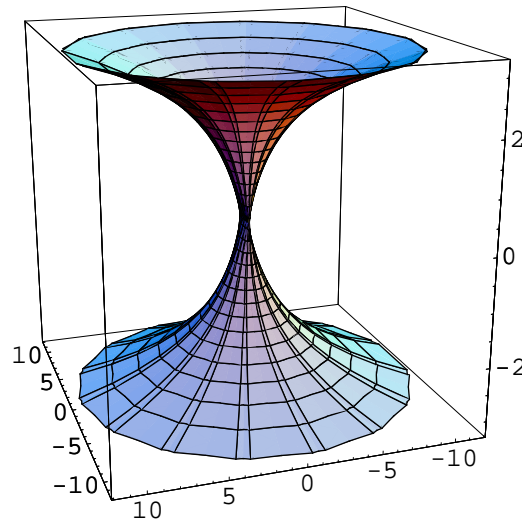
The pictures look very different although the same part of the surface is considered. The difference comes more clear if we take a larger part of the surface:

```
ParametricPlot3D[s1, {x, -6, 6}, {y, -Pi, Pi},  
BoxRatios -> {1, 1, 3}, ViewPoint -> {-3, 1, 1}]
```



- Graphics3D -

```
ParametricPlot3D[s2, {x, -6, 6}, {y, -Pi, Pi},
  BoxRatios -> {1, 1, 1}, ViewPoint -> {-3, 1, 1}]
```



- Graphics3D -

In the latter case, the surface overlaps itself several times and we can't see the characteristic features. The former shows the well-known visualization of $\sin(z)$ called the Riemann surface.

Many good pictures of this type and the analysis of the corresponding surfaces and functions can be found in the articles [5,6,7,8] of Michael Trott and [1] by Corless and Jeffrey.

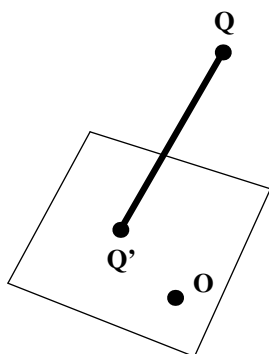
The pictures can be considered from another point of view, too. They are both the images of the graph of the function in a parallel projection $\mathbb{R}^4 \rightarrow \mathbb{R}^3$. In the first case, the projection line is parallel to the y -axis and the image 'plane', i.e. the range of the projection mapping, is the three dimensional subspace spanned by the three other axes. The orthonormal basis used in the subspace consists of the unit vectors of the axes. In the second case, the situation is similar, but the projection line is parallel to the x -axis. Both of the projection mappings are orthogonal, i.e. the image space (or the range of the mapping) is the orthogonal complement of the projection line.

This opens a new way to form pictures of the graph of a function $f: \mathbb{R}^2 \rightarrow \mathbb{R}^2$, $f(x, y) = (u(x, y), v(x, y))$. The graph is a two-dimensional manifold that lives in the space \mathbb{R}^4 and any projection mapping $\mathbb{R}^4 \rightarrow \mathbb{R}^3$ can be used to form its image in the space \mathbb{R}^3 . This is (in general) a usual surface and normal methods can be used to plot a picture of it. The projection mapping can be a parallel projection or a perspective projection. The projection line of the parallel projection may have any direction. The projection space may be the orthogonal complement of the projection line or not, i.e. the parallel projection may be orthogonal or oblique. (Of course, the normal methods like `ParametricPlot3D` also use some projection methods for plotting a two-dimensional picture of a three-dimensional object on the screen. Thus, for seeing on the screen the picture of an object from the four-dimensional space, two consecutive projection mappings have been used.)

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

■ Tools

A parallel projection in the three-dimensional space can be defined by giving the direction of the projection line (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 situation of the plane is unessential 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 $P = I - s n^T / (n^T s)$ 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' . For getting coordinates in the two-dimensional plane, a basis must be chosen and the coordinates transformed appropriately. (See e.g. [2].)

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

```
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];
```

For generating animations we need rotations of a special type in the four-dimensional space. Here, a two-dimensional subspace is invariant and the rotation happens 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:

```
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 given in b .

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

The graph of the function $\mathbb{R}^2 \rightarrow \mathbb{R}^2$ in the four-dimensional $uvxy$ -space is represented by a net of polygons. This is generated by the following function:

```
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 $w=f[r, \text{phi}]$, where

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

or

```
f[r_, phi_] =
  ComplexExpand[{Re[(x + I y)^2], Im[(x + I y)^2], x, y}] /.
  {x -> r Cos[phi], y -> r Sin[phi]};
```

The following test may also be needed:

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

■ Simple example

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

```
f[r_, phi_] =
  ComplexExpand[{Re[(x + I y)^2], Im[(x + I y)^2], x, y}] /.
  {x -> r Cos[phi], y -> r Sin[phi]}
{r^2 Cos[phi]^2 - r^2 Sin[phi]^2,
  2 r^2 Cos[phi] Sin[phi], r Cos[phi], r Sin[phi]}
gr4D =
  genNet4D[f[r, phi], {r, 0, 2, 1/5}, {phi, -Pi, Pi, Pi/36}];
```

We generate a parallel projection for viewing an animation. Projection line 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 line).

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

```
( 1  0  0  0
  0  1  0  0
  0  0  1  0 )
```

In the animation we rotate the graph of the function in the four-dimensional space; the rotation will happen in the xy -plane (i.e. vectors $(1, 0, 0, 0)$ and $(0, 1, 0, 0)$ are invariant):

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

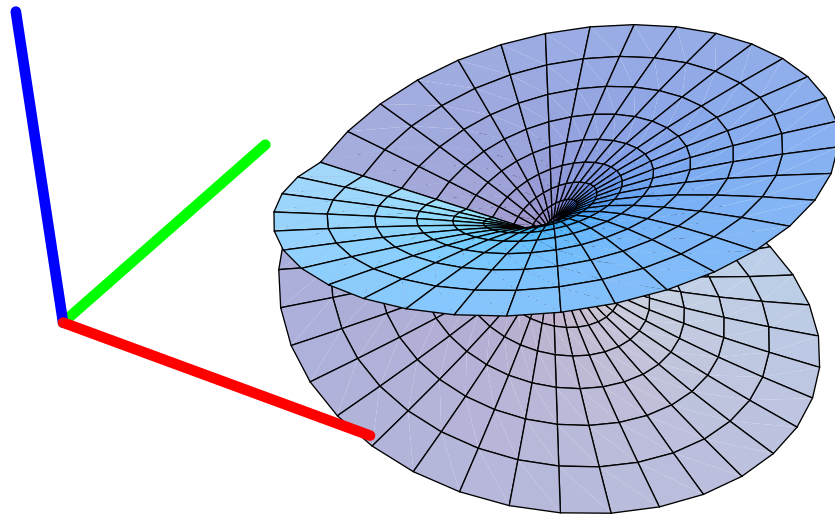
```
( 1.  0.  0.  0.
  0.  1.  0.  0.
  0.  0.  0.984808 -0.173648
  0.  0.  0.173648  0.984808 )
```

Besides the graph of the function we will also plot the axis system of the four-dimensional space. The following gives the necessary definition.

```
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};
```

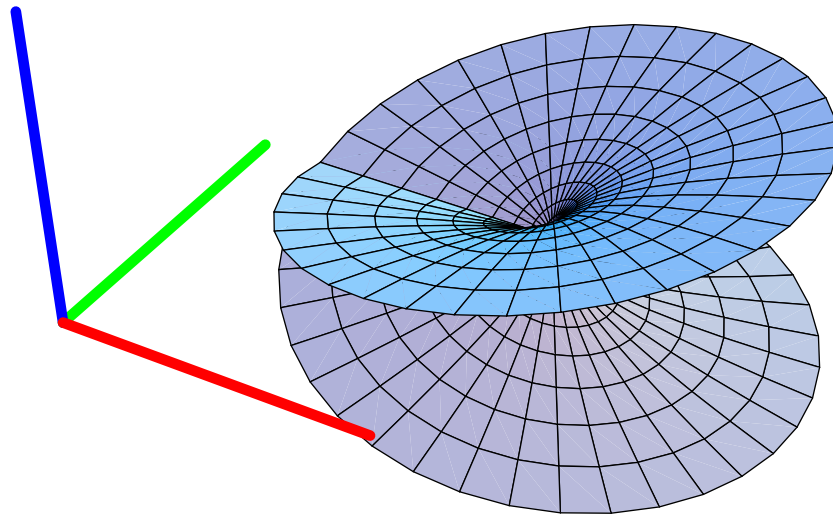
In the animation we may rotate the graph or we may fly around the graph, i.e. rotate the projection mapping. The former gives the following animation:

```
d = 5.2; opts = {Boxed -> False,
  PlotRange -> {{-d, d}, {-d, d}, {-d, d}}, ImageSize -> 400};
gr3D = Table[Graphics3D[{ax4D /. x_?test4D -> p.x,
  gr4D /. x_?test4D -> p.Nest[q.# &, x, k]}, opts], {k, 0, 35}];
Map[
  Show,
  gr3D]
```



And then animating the flight around the graph:

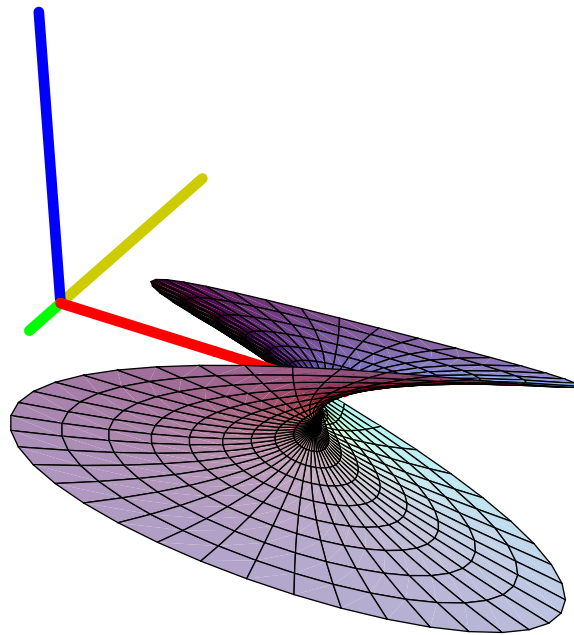
```
gr3D = Table[pk = Apply[genProj, Map[Nest[q.# &, #, k] &, b]];
Graphics3D[{gr4D, ax4D} /. x_?test4D -> pk.x, opts],
{k, 0, 35}];
Map[Show, gr3D]
```



We get a more interesting animation if the rotation happens in the v_y -plane:

```
q = genRot4D[{1, 0, 0, 0}, {0, 0, 1, 0}, Pi / 18.];
gr3D = Table[pk = Apply[genProj, Map[Nest[q.# &, #, k] &, b]];
Graphics3D[{gr4D, ax4D} /. x_?test4D -> pk.x, opts],
{k, 0, 35}];
Map[Show, gr3D]
```

In this case, we have also the following view on the graph. Here, the surface does not intersect itself showing that there is no real intersection in the four-dimensional space.



■ Function $\sin(z)$

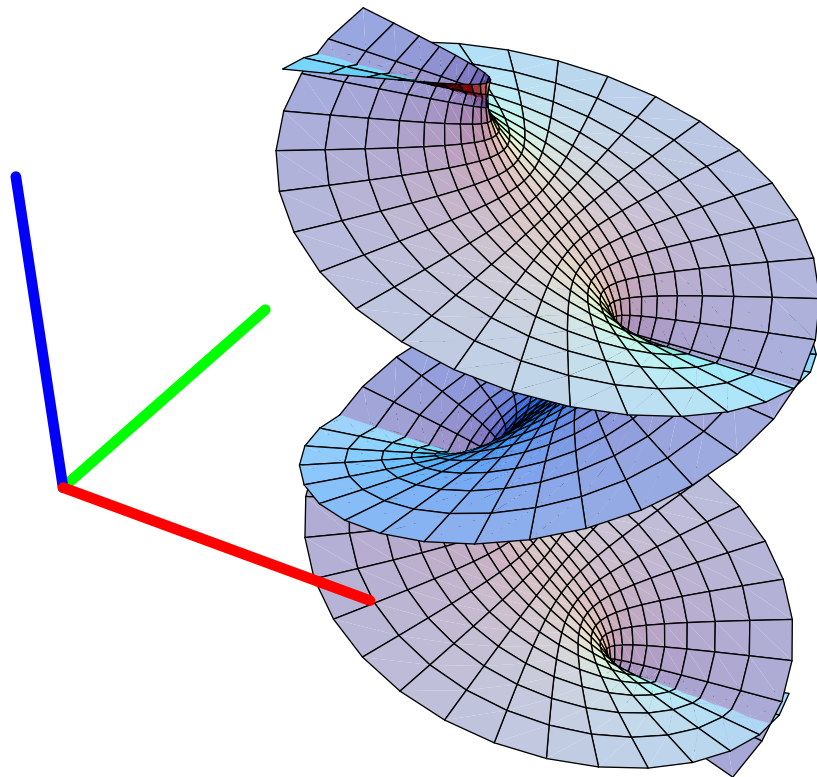
In the similar manner we may analyze the function $\sin(z)$. In this case, it is better to use rectangular coordinates.

```
f[x_, y_] =
  ComplexExpand[{Re[Sin[x + I y]], Im[Sin[x + I y]], x, y}]
{Cosh[y] Sin[x], Cos[x] Sinh[y], x, y}
gr4D = genNet4D[f[x, y], {x, -5, 5, 1/5}, {y, -2, 2, 1/5}];
q = genRot4D[{1, 0, 0, 0}, {0, 1, 0, 0}, Pi/18.];
```

```

gr3D = Table[Graphics3D[{ax4D /. x_?test4D -> p.x,
  gr4D /. x_?test4D -> p.Nest[q.# &, x, k]}, opts], {k, 0, 35}];
Map[
  Show,
  gr3D]

```



■ Toward a better animation

LiveGraphics3D package of Martin Kraus [3] gives the possibility to create animations where 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

$$\mathbf{s} = \{ \sin[t_1] \sin[t_2] \cos[f], \\
 \sin[t_1] \sin[t_2] \sin[f], \sin[t_1] \cos[t_2], \cos[t_1] \};$$

where t_1 , t_2 and f 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:

```
e1 = D[s, t1]; e2 = D[s, t2] / Sin[t1];
e3 = D[s, f] / Sin[t1] / Sin[t2];
b = {s, e1, e2, e3};
b // MatrixForm
```

$$\begin{pmatrix} \cos[f] \sin[t_1] \sin[t_2] & \sin[f] \sin[t_1] \sin[t_2] & \cos[t_2] \sin[t_1] & \cos[t_1] \\ \cos[f] \cos[t_1] \sin[t_2] & \cos[t_1] \sin[f] \sin[t_2] & \cos[t_1] \cos[t_2] & -\sin[t_1] \\ \cos[f] \cos[t_2] & \cos[t_2] \sin[f] & -\sin[t_2] & 0 \\ -\sin[f] & \cos[f] & 0 & 0 \end{pmatrix}$$

The projection mapping is then

```
p = Apply[genProj, b] // Simplify
{{Cos[f] Cos[t1] Sin[t2], Cos[t1] Sin[f] Sin[t2],
 Cos[t1] Cos[t2], -Sin[t1]}, {Cos[f] Cos[t2],
 Cos[t2] Sin[f], -Sin[t2], 0}, {-Sin[f], Cos[f], 0, 0}}
```

We need sliders for controlling the direction of the viewing line, i.e. for controlling the angles $0 \leq f \leq 2\pi$, $0 \leq t_1 \leq \pi$, $0 \leq t_2 \leq \pi$.

```
controls = {Line[{{7, 0, 0}, {7, 2 Pi, 0}}, Point[{{7, 0, 0}},
 Point[{{7, 2 Pi, 0}}, {RGBColor[1, 0, 0], Point[{{7, f, 0}}]},
 Line[{{8, 0, 0}, {8, Pi, 0}}, Point[{{8, 0, 0}},
 Point[{{8, Pi, 0}}, {RGBColor[1, 0, 0], Point[{{8, t1, 0}}]},
 Line[{{9, 0, 0}, {9, Pi, 0}}, Point[{{9, 0, 0}},
 Point[{{9, Pi, 0}}, {RGBColor[1, 0, 0], Point[{{9, t2, 0}}]}];
```

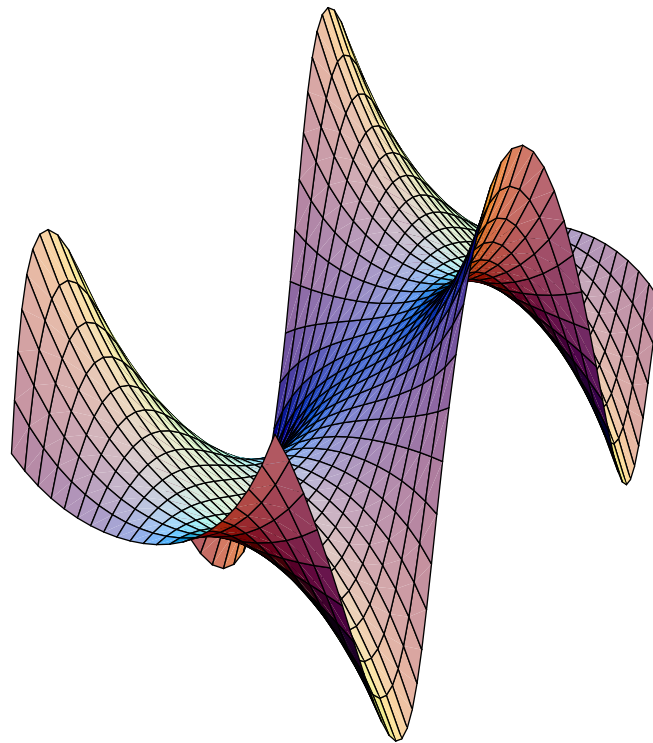
And the graphics element to be exported for **LiveGraphics3D** is as follows:

```
gr3D =
Graphics3D[{gr4D /. x_?test4D => p.x, controls}, Boxed -> False];
<< LiveGraphics3D.m
WriteLiveForm["sin.grf", gr3D]
```

The file `sin.grf` contains the code for the the animation, a `Graphics3D` object.

There are three parameters, the angles, that are controlled by three sliders. The animation is represented as an applet in an html page. The image below gives one possible view in the animation. The red points on the right are the sliders.

```
Show[gr3D /. {f -> 2.5, t1 -> 1.5, t2 -> 1.6}, PlotRange -> All]
```



■ Final remarks

The tools developed here give one possibility to try to understand the nature of complex functions or functions $\mathbb{R}^2 \rightarrow \mathbb{R}^2$. The possibilities are not restricted to the graphs of these functions, but any geometric object of the four-dimensional space can be considered. For example, the properties of the Klein bottle can be studied when a four-dimensional parametric representation is given.

From the technical view point, another approach could also be used: Instead of first projecting the objects to the space \mathbb{R}^3 and then to the two-dimensional screen by means of **LiveGraphics3D**, only one projection mapping $\mathbb{R}^4 \rightarrow \mathbb{R}^2$ could be used. The

functionality needed in the animations could be achieved with **LiveGraphics3D** also in this case. However, there would be problems in the removal of hidden lines (if this is considered necessary).

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

■ References

- [1] Robert M. Corless, David J. Jeffrey, Graphing Elementary Riemann Surfaces, ACM Sigsam Bulletin: Commun. Comput. Algebra 32, 1998.
- [2] I. D. Faux, M. J. Pratt, Computational Geometry for Design and Manufacture, John Wiley & Sons, 1979
- [3] Martin Kraus, LiveGraphics3D Homepage,
<http://www.vis.uni-stuttgart.de/~kraus/LiveGraphics3D/>
- [4] Bernd Thaller, Visualization of Complex Functions, The Mathematica Journal, 7 (2), 1998
- [5] Michael Trott, Visualization of Riemann Surfaces of Algebraic Functions, Mathematica in Education and Research, 6 (4), 1997
- [6] Michael Trott, Visualization of Riemann Surfaces IIa, Compositions of Elementary Transcendental Functions, The Mathematica Journal, 7 (4), 2000
- [7] Michael Trott, Visualization of Riemann Surfaces IIb, Compositions of Elementary Transcendental Functions, The Mathematica Journal, 8 (1), 2001
- [8] Michael Trott, Visualization of Riemann Surfaces IIc, Compositions of Elementary Transcendental Functions, The Mathematica Journal, 8 (4), 2002