TROTT'S CORNER

The Return of the Riemann Surface

Michael Trott

My favorite subject—Riemann surfaces—is revisited through the eyes (functions) of *Mathematica* 6. Old friends will appear in a new light(ing) and new acquaintances with untreated surfaces will be made.

■ Introduction

This is my first column since Version 6 came out a few months ago. Version 6, after being in the works for many years, provides a wealth of new features that are useful in many numeric and symbolic calculations, advanced programs, visualizations, and elsewhere. With so many exciting new possibilities, it was not easy to decide what to discuss in this column. So instead of delving into a new subject, I think the best way to see some of the new features and resulting capabilities of Version 6 is to compare with some corresponding Version 5 results. That is why today I will come back to a subject discussed in earlier columns and demonstrate how to go quite a bit further with Version 6.

In earlier columns, I started discussing the visualization of Riemann surfaces. In this column, I want to come back to this mathematically (and aesthetically) beautiful subject. So far, we discussed the construction of Riemann surfaces of compositions of elementary functions with finitely many branch points (articles IIa to IId [1-4]). With the new 3D graphics system and interactive features of Version 6, new possibilities open up for visualizing Riemann surfaces; the main ones are:

- Specifying vertex normals to obtain smooth-looking surfaces
- Using opacity to get a better look "inside" a Riemann surface (instead of cutting holes in the polygons of the surface)
- Using adaptive refinement of the built-in graphics functions to better resolve branch points
- Using `Manipulate` to easily view a function $w(z) = f(z)$ as a 3D projection of the hypersurface $\{z, w\} \in \mathbb{C}^2 \sim \mathbb{R}^4$ (where showing $\mathrm{Re}(w(z))$ and $\mathrm{Im}(w(z))$ over the complex plane become special cases)
- Using the `Exclusions` option to avoid branch cuts in `Plot3D`, `Para`⋱`metricPlot3D`, and others

These features offer a fresh look at the topic of Riemann surfaces (so, if numbered, this would be article IIdII, not yet IIe, which will appear later and deal with special functions).

Starting with this column, I will also introduce a formatting change: *Mathematica*'s default `StandardForm` is an ideal editing and reading environment for mathematical expressions and very small program snippets (one to two lines). Convenient automatic line breaking, two-dimensional fraction and radical formatting, and other features make writing and reading easy. But for multiline program-like code pieces, the formatted results are frequently less readable than manually formatting the code to avoid unexpected line breaks, alignments, and nonuniform line spacings. A new style in Version 6 (available as ALT-8) is the `"Code"` style. It is the default style for packages (the canonical place to store a program) and can, of course, also be used in notebooks. It does not automatically break lines, but allows for special characters and inherits the new syntax coloring feature to maximize code readability. So, function definitions are given in `"Code"` style (easily identifiable in this notebook by its light blue background) and example uses of the defined functions will be in style `"Input"`. In addition to an easier visual discrimination between actual definitions and example uses, this also allows for a quick selection of all definitions (using RIGHT-ALT) for immediate use without having to run all (sometimes time-consuming) examples to experiment with further Riemann surfaces.

■ The Updated Riemann Surface Package

We start with an updated version of the construction discussed in my earlier columns. Let us quickly recap the construction idea: Starting with a given function $w(z) = f(z)$, a set of coupled nonlinear meromorphic differential equations is derived from $f(z)$ and this set is solved on patches of the z plane. These patches arise from a tensor product decomposition of the complex plane in a cylindrical coordinate system and are separated by branch points. The system of differential equations is solved numerically and the resulting function values are displayed. If at the same time we calculate $f'(z)$, it is possible to construct the normals to obtain a smooth-looking surface. (Calculating the normals of the 3D embedding of $\{z, f(z)\}$ can be done easily using the Cauchy-Riemann conditions.) An updated version of the package, based on the code from *The Mathematica GuideBook for Numerics* [5], is available on the *Mathematica GuideBooks* website (www.mathematicaguidebooks.org/V6/downloads/RiemannSurfacePlot3D.m).

```
In[21]:= Import["http://www.mathematicaguidebooks.org/V6/downloads/
           RiemannSurfacePlot3D.m"]
```

This is the main function defined in the package.

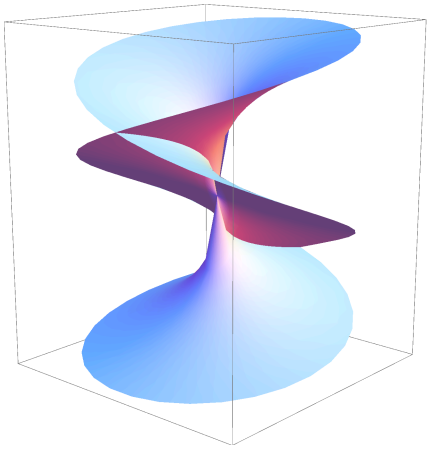
```
In[22]:= ?RiemannSurfacePlot3D
```

RiemannSurfacePlot3D[w == f[z], reim[w[z]], {z, w}] plots a Riemann surface of w as the real or imaginary part reim of w over the complex z–plane.
RiemannSurfacePlot3D[w == f[z], {ζ1, ζ2, ζ3}, {z, w}] plots a Riemann surface of w as {ζ1, ζ2, ζ3} along the Cartesian coordinate axes where ζ1, ζ2, ζ3 can be Re[z], Im[z], Re[w], Im[w] or a linear combination of them.

The first example is $w(z) = z^{1/3}$ displayed as Re(w) over the complex z plane.

```
In[23]:= RiemannSurfacePlot3D[w == z^(1/3), Re[w],
          {z, w}, ViewPoint → {-2.5, -2.2, 0.8}]
```

Out[23]=



The function has a few new options in addition to the built-in 3D graphics options.

```
In[24]:= Complement[Options[RiemannSurfacePlot3D], Options[Graphics3D]]
```

```
Out[24]= {BoxRatios → {1, 1, 1.2}, BranchPointOffset → 1/1 000 000,
  Coloring → Automatic, InterpolationOrder → Automatic,
  InterpolationPoints → Automatic, LogSheets → {-1, 0, 1},
  NDSolveOptions → {}, PlotPoints → {30, 12}, PlotRange → Automatic,
  PlotStyle → Automatic, StitchPatches → False, WorkingPrecision → 25}
```

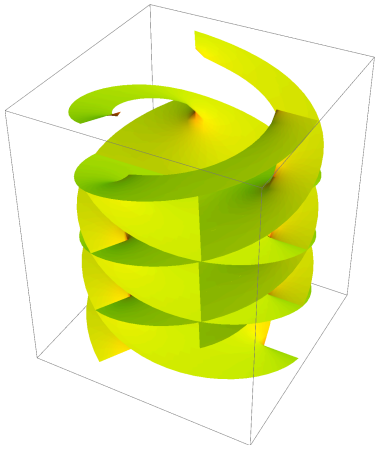
Because 3D graphics in Version 6 are typically no longer made from individual polygons, but rather from `GraphicsComplex` objects that contain polygons and their adjacency information, it is possible to avoid the small gaps that were used in Version 5 between the patches. The function `StitchPatches` "stitches" the `GraphicsComplex` objects together along their boundaries. While in static images, the small (typically 1ppm or smaller) gaps are usually not visible; when interactively rotating graphics, such gaps can potentially become more pronounced.

Note that we do not have to specify a domain of the z plane over which to plot the function. The domain is chosen automatically in such a way to include all finite branch points of the function.

Here is another example of a Riemann surface plotted with `RiemannSurface`[·]. `Plot3D`.

In[25]:=
```
RiemannSurfacePlot3D[w == ArcSin[z^3 - 1],
 Re[w], {z, w}, PlotPoints → {60, 40},
 PlotStyle → Directive[Yellow, Specularity[Red, 20]]]
```

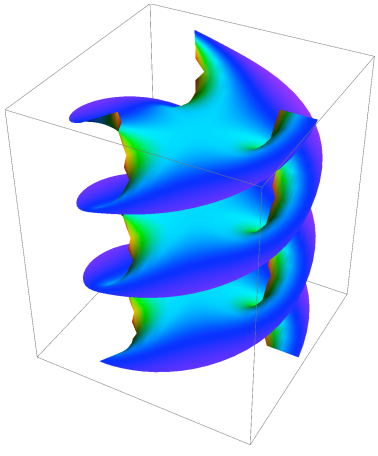
Out[25]=



In the next example, we display the imaginary part of $\log(z^3 - 1)$ over the complex plane and color the surface according to the real part.

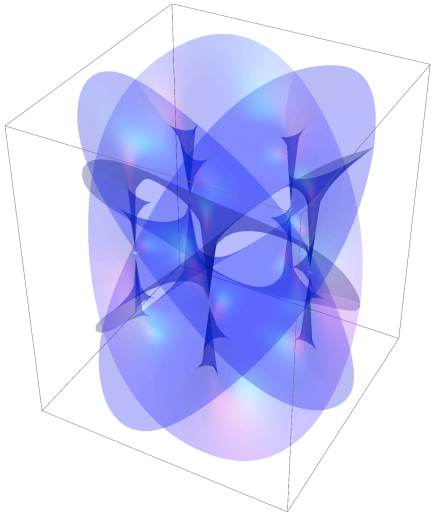
In[26]:=
```
RiemannSurfacePlot3D[w == Log[z^3 - 1],
 Im[w], {z, w}, PlotPoints → {60, 40},
 Coloring → Hue[Rescale[ArcTan[0.8 Re[w]], {-Pi / 2, Pi / 2}]]]
```

Out[26]=



Using a transparent surface makes it easier to see the inner parts of more complicated Riemann surfaces.

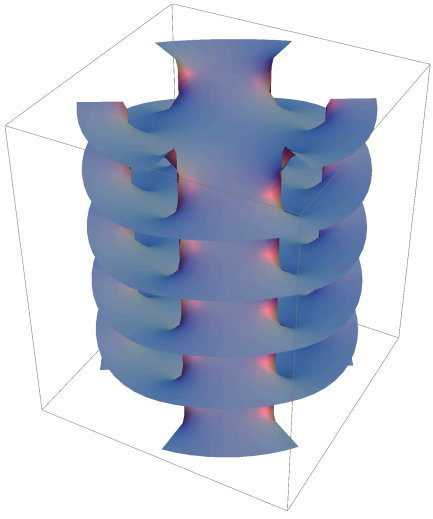
Out[27]=



Here are some more examples.

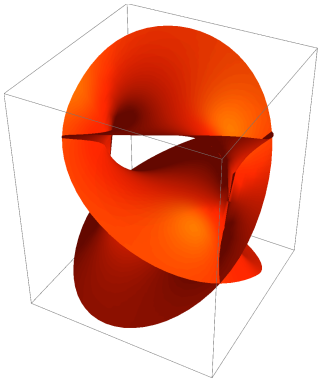
In[28]:=
```
RiemannSurfacePlot3D[w == ArcTan[z^3],
 Re[w], {z, w}, PlotPoints → {60, 40},
 PlotStyle → Directive[Gray, Specularity[Red, 20]],
 StitchPatches → True]
```

Out[28]=




```
In[29]:= RiemannSurfacePlot3D[w == (z^2 - 1)^(1/3),
          Im[w], {z, w}, PlotPoints → {60, 40},
          PlotStyle → Directive[Red, Specularity[Orange, 20]],
          StitchPatches → True]
```

Out[29]=

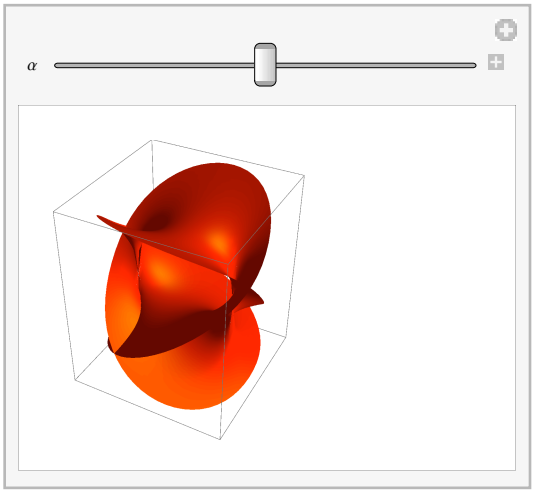


`RiemannSurfacePlot3D` stores some intermediate time-consuming results for the function it has just plotted. As a result, once a function is plotted, calls to `RiemannSurfacePlot3D` with an identical first argument and mostly identical option settings, but a potentially different second argument, are generally quite fast.

Here is a demonstration that allows moving smoothly from the real to the imaginary part.

```
In[30]:= Manipulate[RiemannSurfacePlot3D[w == (z^2 - 1)^(1/3),
           (1 - α) Im[w] + α Re[w], {z, w}, PlotPoints → {60, 40},
           PlotStyle → Directive[Red, Specularity[Orange, 20]]],
         {{α, 0.5}, 0, 1}, SaveDefinitions → True]
```

Out[30]=



By parametrizing the three Cartesian coordinates as a linear form of Re(z), Im(z), Re(w), and Im(w), we can implement an even more general view of the surface.

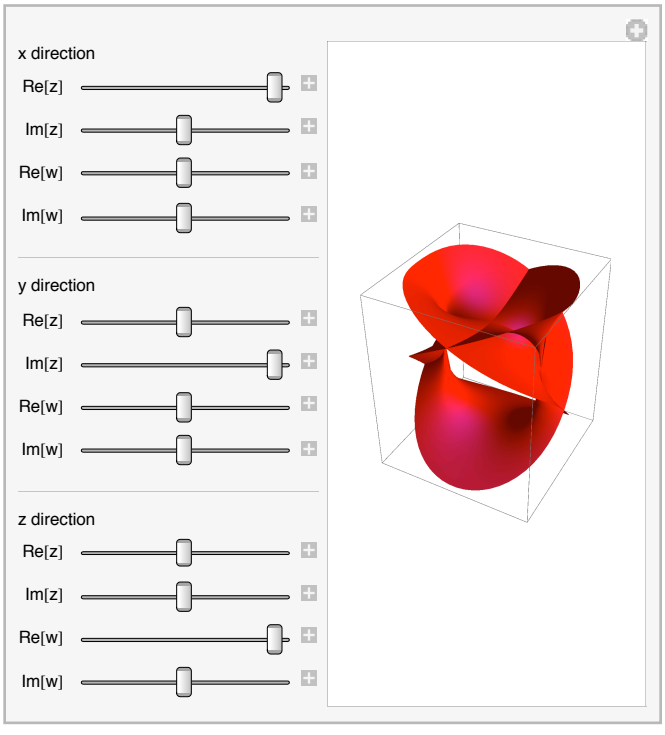
We do this through a `Manipulate` object that makes it easy to change any of the 12 parameters that define the actual function value that is plotted.

In[31]:=
```
ManipulateRSF[w_ == f_, opts___] :=
 Manipulate[RiemannSurfacePlot3D[w == f,
   {a1 Re[z] + a2 Im[z] + a3 Re[w] + a4 Im[w], b1 Re[z] + b2 Im[z] +
     b3 Re[w] + b4 Im[w], c1 Re[z] + c2 Im[z] + c3 Re[w] + c4 Im[w]},
   {z, w}, StitchPatches → True, opts, PlotPoints → {60, 40},
   PlotStyle → Directive[Red, Specularity[Purple, 20]]], Delimiter,
  "x direction", {{a1, 1, Re[z]}, -1, 1, ImageSize → Small},
  {{a2, 0, Im[z]}, -1, 1, ImageSize → Small},
  {{a3, 0, Re[w]}, -1, 1, ImageSize → Small},
  {{a4, 0, Im[w]}, -1, 1, ImageSize → Small}, Delimiter,
  "y direction", {{b1, 0, Re[z]}, -1, 1, ImageSize → Small},
  {{b2, 1, Im[z]}, -1, 1, ImageSize → Small},
  {{b3, 0, Re[w]}, -1, 1, ImageSize → Small},
  {{b4, 0, Im[w]}, -1, 1, ImageSize → Small}, Delimiter,
  "z direction", {{c1, 0, Re[z]}, -1, 1, ImageSize → Small},
  {{c2, 0, Im[z]}, -1, 1, ImageSize → Small},
  {{c3, 1, Re[w]}, -1, 1, ImageSize → Small},
  {{c4, 0, Im[w]}, -1, 1, ImageSize → Small}, SaveDefinitions → True]
```

We give two examples: the first algebraic and the second inverse trigonometric.

In[32]:=
```
ManipulateRSF[w == (z^2 - 1)^(1/3)]
```

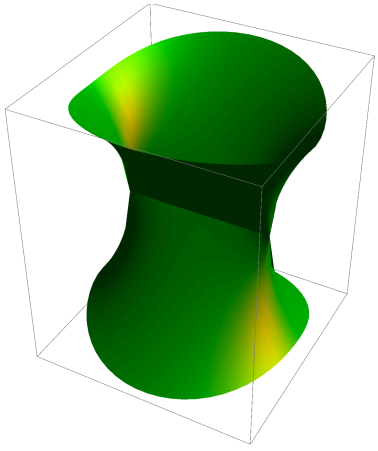
Out[32]=



Here is the inverse trigonometric function.

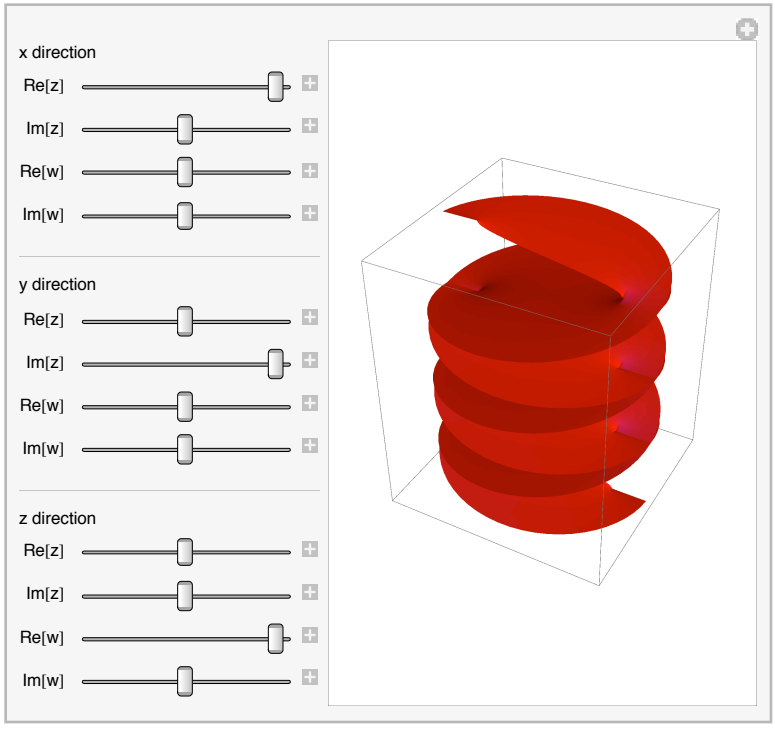
In[33]:=
```
RiemannSurfacePlot3D[w == ArcSin[z],
 Im[w], {z, w}, PlotPoints → {60, 40},
 PlotStyle → Directive[Darker[Green], Specularity[Yellow, 20]]]
```

Out[33]=



In[34]:=
```
ManipulateRSF[w == ArcSin[z]]
```

Out[34]=



For details of the construction, see the package. Within the new package editor of Version 6, it is quite convenient and straightforward to experiment. Input cells are kept in the package and are easily evaluated; comments and examples can be easily embedded, but are invisible when the package is loaded. When developing or investigating package code with the package editor, we do not have to worry about contexts or saving outputs.

`RiemannSurfacePlot3D` fails for functions with infinitely many branch points. Here is an example.

```
In[35]:= RiemannSurfacePlot3D[w == Sqrt[Coth[z]], Re[w], {z, w}]
```

```
RiemannSurfacePlot3D::cantFindAllBranchPoints:
  Unable to calculate all branchpoints for the function Sqrt[Coth[z]].
```

```
Out[35]= RiemannSurfacePlot3D[w == Sqrt[Coth[z]], Re[w], {z, w}]
```

The construction given next deals with functions having infinitely many branch points.

■ The New ContourPlot3D

`ContourPlot3D`, a package function in Version 5, is now a much faster built-in kernel function. This function comes in handy sometimes for polynomials dealing with Riemann surfaces. Here is a simple example that uses `ContourPlot3D` to display the real part of the functions $w = w(z) = \left(1 - z^2\right)^{1/3}$ defined implicitly through a bivariate polynomial.

Eliminating Im(w) yields a (more complicated) trivariate polynomial that can be plotted.

```
In[36]:= {re, im} = ComplexExpand[({Re[#1], Im[#1]} &)[(u + i v)^3 - (x + i y)^2 + 1]]
```

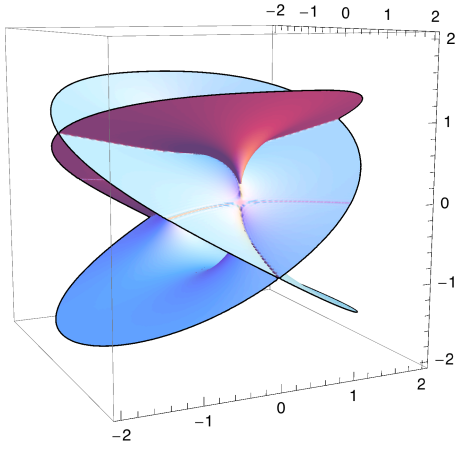
$$\text{Out[36]= } \left\{1 + u^3 - 3\,u\,v^2 - x^2 + y^2,\ 3\,u^2\,v - v^3 - 2\,x\,y\right\}$$

```
In[37]:= res = Factor[Resultant[re, im, v]]
```

$$\begin{aligned}\text{Out[37]= } & 1 - 15\,u^3 + 48\,u^6 + 64\,u^9 - 3\,x^2 + 30\,u^3\,x^2 - 48\,u^6\,x^2 + 3\,x^4 - 15\,u^3\,x^4 - x^6 + 3\,y^2 - \\ & 30\,u^3\,y^2 + 48\,u^6\,y^2 - 6\,x^2\,y^2 - 78\,u^3\,x^2\,y^2 + 3\,x^4\,y^2 + 3\,y^4 - 15\,u^3\,y^4 - 3\,x^2\,y^4 + y^6\end{aligned}$$

In[38]:=
```
ContourPlot3D[Evaluate[res == 0], {x, -2, 2},
 {y, -2, 2}, {u, -2, 2}, PlotPoints → 160, MaxRecursion → 0,
 Mesh → False, RegionFunction → (Norm[{#1, #2}] < 2 &),
 ViewPoint → {3.04, -1.41, 0.58}]
```

Out[38]=

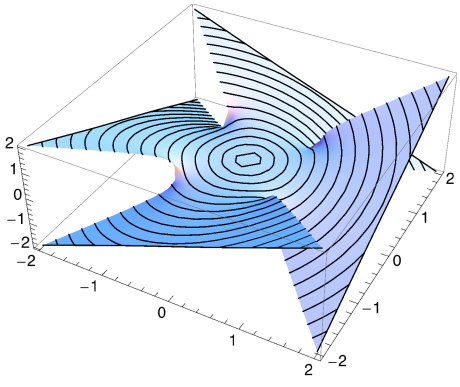


■ Using the Exclusions Options

Now we will use new features of `Plot3D` and `ParametricPlot3D` to construct a Riemann surface. Using these 3D plotting functions on a function that has branch cuts, we now automatically get cuts along the branch cuts.

In[39]:=
```
Plot3D[Im[(1 + (x + i y)^4)^(1/4)], {x, -2, 2},
 {y, -2, 2}, MeshFunctions → {Norm[{#1, #2}] &}]
```

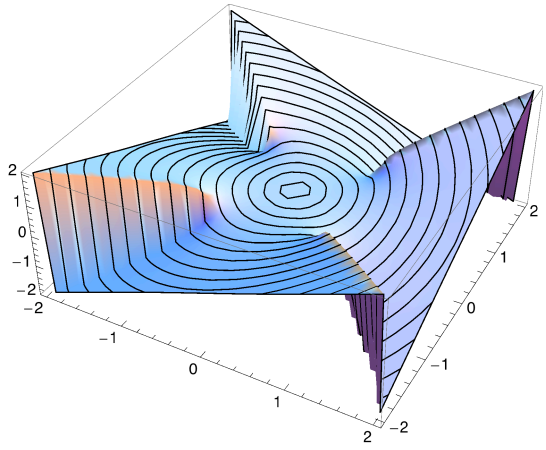
Out[39]=



Modulo the mesh, in Version 5, we would have gotten a graphic similar to the following, much less appealing (but still correct) image.

```
In[40]:= Plot3D[Im[(1 + (x + i y)^4)^(1/4)], {x, -2, 2}, {y, -2, 2},
          MeshFunctions → {Norm[{#1, #2}] &}, Exclusions → None]
```

Out[40]=



We can also obtain descriptions of the excluded cuts using the function `VisualizationDiscontinuities` from the context `Visualization`.

```
In[41]:= Visualization`VisualizationDiscontinuities[Im[Log[x + i y]], {x, y}]

Out[41]= {{-Im[y] + Re[x] ≤ 0, Im[x] + Re[y] == 0}}

In[42]:= Visualization`VisualizationDiscontinuities[Re[(x + i y)^(1/3)], {x, y}]

Out[42]= {{-Im[y] + Re[x] ≤ 0, Im[x] + Re[y] == 0}}
```

This neat feature of plotting functions that recognize branch cuts can be used to construct Riemann surfaces by finding parametrizations of all sheets and then plotting them. Instead of the steep vertical walls in earlier versions of *Mathematica*, we now get small cuts between the sheet patches. The analytic continuation of the logarithm and the power function are straightforward to define through adding multiples of $2\pi i$ and multiplication by $\exp(i\,2\pi/n)$.

```
In[43]:= analyticallyContinue[w_, z_] :=
         Module[{lc = 0, pc = 0,
             (* express inverse trigonometric and hyperbolic function
               through logarithms and square roots *)
             w1 = w //. f:(ArcSin | ArcCos | ArcTan | ArcCot | ArcSec | ArcCsc |
                           ArcSinh | ArcCosh | ArcTanh |
         ArcCoth | ArcSech | ArcCsch) :>
                           Function[ζ,
         Evaluate[TrigToExp[f[ζ]]]]},
             w1 //. {Log[ζ_?(MemberQ[#, z, {0, Infinity}]&)] :>
                   LogContinued[ζ, lc++],

         Power[ζ_?(MemberQ[#, z, {0, Infinity}]&), e:Except[_Integer]] :>
                   PowerContinued[ζ, e, pc++],
                  ProductLog[ζ_?(MemberQ[#, z, {0, Infinity}]&)] :>
                   ProductLogContinued[ζ, lc++]}]
```

For the logarithm, with its infinitely many sheets, we restrict ourselves to (typically) three sheets.

```
In[44]:= logSheetLimits[logSheets_] :=
           Sequence @@
         If[OddQ[logSheets], {-(logSheets - 1)/2, (logSheets - 1)/2},
                             {-logSheets/2, logSheets/2 - 1}]

In[45]:= makeSheetRealizations[LogContinued[z_, j_], logSheets_] :=
           Table[LogContinued[z, j] → Log[z] + 2 k I Pi,
             Evaluate[{k, logSheetLimits[logSheets]}]]

         makeSheetRealizations[PowerContinued[z_, r_Rational, j_], _] :=

         Table[PowerContinued[z, r, j] → Exp[2 Pi I k/Denominator[r]] z^r,
           {k, 0, Denominator[r] - 1}]

         makeSheetRealizations[PowerContinued[z_, r_, j_], logSheets_] :=
           Table[PowerContinued[z, r, j] → Exp[r (Log[z] + 2 k I Pi)],
             Evaluate[{k, logSheetLimits[logSheets]}]]

         makeSheetRealizations[ProductLogContinued[z_, j_], logSheets_] :=
           Table[ProductLogContinued[z, j] → ProductLog[j, z],
             Evaluate[{k, logSheetLimits[logSheets]}]]
```

This list can be easily extended to elliptic integrals, Bessel functions, polylogarithms, and others.

The function `makeSheetList` generates a list of the analytically continued sheets of a multivalued function by forming the outer product of all sheets of all the multivalued functions that occur.

```
In[49]:= makeSheetList[wContinued_, z_, logSheets_:3] :=
         Module[{continuedLogPowers, sheetRealizationRules, theSheets},
           continuedLogPowers = Cases[wContinued,

         _LogContinued | _PowerContinued | _ProductLogContinued,
                           {0, Infinity}];
           sheetRealizationRules =
            If[continuedLogPowers === {}, {},
              Flatten[#, Length[continuedLogPowers] - 1] & @

         Outer[List, Sequence @@ (makeSheetRealizations[#, logSheets] & /@
                                    continuedLogPowers)]];
           theSheets = wContinued //. sheetRealizationRules;
           If[Head[theSheets] === List, theSheets, {theSheets}]]
```

Here is an example. The function $w(z) = \log\left(1 + i + z\sqrt{z+1}\right)$ results in six parametrized sheets: three from the outer logarithm combined with two from the inner square root.

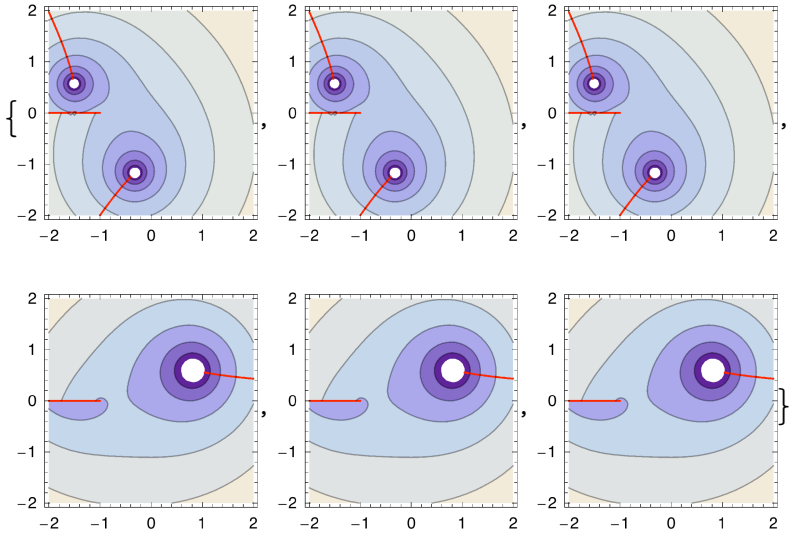
```
In[50]:= sheets1 =
          makeSheetList[analyticallyContinue[Log[1 + i + z Sqrt[z + 1]], z], z]
```

```
Out[50]= {-2 i π + Log[(1 + i) + z Sqrt[1 + z]], Log[(1 + i) + z Sqrt[1 + z]],
          2 i π + Log[(1 + i) + z Sqrt[1 + z]], -2 i π + Log[(1 + i) - z Sqrt[1 + z]],
          Log[(1 + i) - z Sqrt[1 + z]], 2 i π + Log[(1 + i) - z Sqrt[1 + z]]}
```

Here is a quick view on the branch cut structure of these six functions.

```
In[51]:= (ContourPlot[Evaluate[Re[#1 /. z → x + i y]], {x, -2, 2}, {y, -2, 2},
            ExclusionsStyle → {Directive[Thickness[0.006], Red]},
            ImageSize → {144, Automatic}] &) /@ sheets1
```

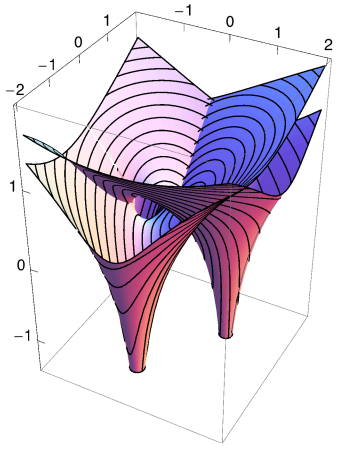
Out[51]=



And here are the real and imaginary parts of the six sheets shown as 3D plots.

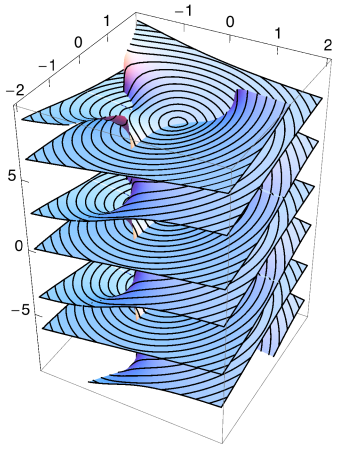
```
In[52]:= Plot3D[Evaluate[Re[sheets1 /. z → x + i y]],
          {x, -2, 2}, {y, -2, 2}, BoxRatios → {1, 1, 1.5},
          PlotPoints → 40, MeshFunctions → {Norm[{#1, #2}] &},
          Exclusions → {Automatic, Offset → 0.02, MaxRecursion → 3}]
```

Out[52]=




```
In[53]:= Plot3D[Evaluate[Im[sheets1 /. z → x + i y]],
          {x, -2, 2}, {y, -2, 2}, BoxRatios → {1, 1, 1.5},
          PlotPoints → 40, MeshFunctions → {Norm[{#1, #2}] &},
          Exclusions → {Automatic, Offset → 0.02, MaxRecursion → 3}]
```

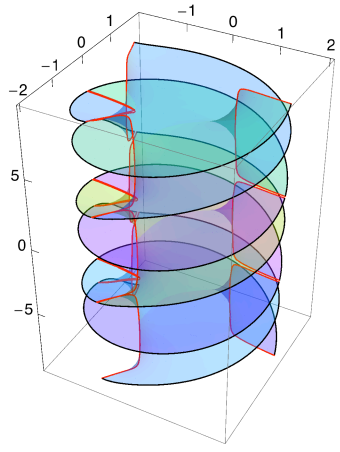
Out[53]=



Emphasizing the cuts and making the surface transparent gives an even better looking graphic.

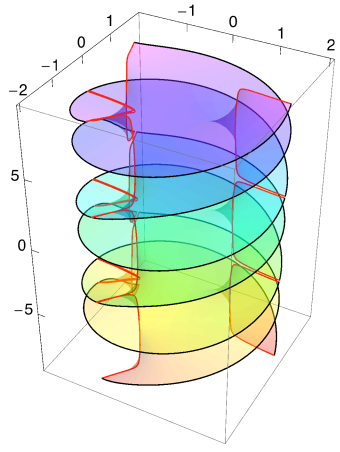
In[54]:=
```
Plot3D[Evaluate[Im[sheets1 /. z → x + ⅈ y]], {x, -2, 2},
 {y, -2, 2}, BoxRatios → {1, 1, 1.5}, Mesh → False,
 RegionFunction → (Norm[{#1, #2}] < 2 &), PlotStyle → Table[
   Directive[Hue[RandomReal[]], Opacity[0.33]], {Length[sheets1]}],
 Exclusions → {Automatic, Offset → 0.02, MaxRecursion → 3},
 ExclusionsStyle → {None, Red}]
```

Out[54]=



In[55]:=
```
Plot3D[Evaluate[Im[sheets1 /. z → x + ⅈ y]],
 {x, -2, 2}, {y, -2, 2}, BoxRatios → {1, 1, 1.5},
 Mesh → False, RegionFunction → (Norm[{#1, #2}] < 2 &),
 ColorFunction → (Directive[Hue[0.8 #3], Opacity[0.33]] &),
 Exclusions → {Automatic, Offset → 0.02, MaxRecursion → 3},
 ExclusionsStyle → {None, Red}]
```

Out[55]=



As a second example, we use $w(z) = \sqrt[5]{-z^4 - z^3 - z + 1}$. This time we have five sheets, all arising from the topmost fifth root.

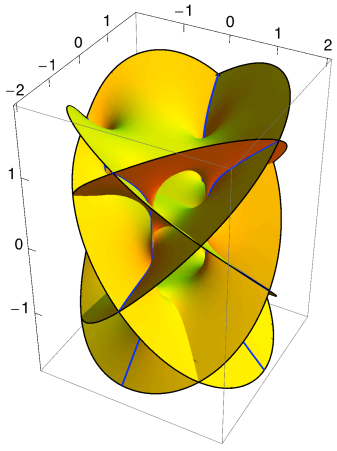
```
In[56]:= sheets2 = makeSheetList[analyticallyContinue[(1 - z - z^3 - z^4)^(1/5), z], z]
```

```
Out[56]= {(1 - z - z^3 - z^4)^(1/5), e^(2 i π/5) (1 - z - z^3 - z^4)^(1/5), e^(4 i π/5) (1 - z - z^3 - z^4)^(1/5),
          e^(-4 i π/5) (1 - z - z^3 - z^4)^(1/5), e^(-2 i π/5) (1 - z - z^3 - z^4)^(1/5)}
```

Here is a plot of the real part over the complex plane.

```
In[57]:= Plot3D[Evaluate[Re[sheets2 /. z → x + i y]], {x, -2, 2},
          {y, -2, 2}, BoxRatios → {1, 1, 1.5}, Mesh → False,
          PlotPoints → 30, RegionFunction → (Norm[{#1, #2}] < 2 &),
          PlotStyle → Directive[Yellow, Specularity[Red, 20]],
          ExclusionsStyle → {None, Blue}]
```

Out[57]=

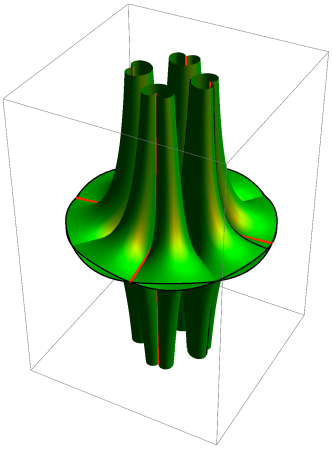


The function `RiemannSheetParametricPlot3D` combines the sheet generation and plotting results. All sheets are included in the first argument of `ParametricPlot3D`, instead of making multiple calls to it. This has the advantage of getting better visual results because the optimal plot range will be calculated by taking all sheets into account.

```
In[58]:= RiemannSheetParametricPlot3D[f_, reim:(Re | Im), {z_, z0_, ρ_},
                    opts:OptionsPattern[]] :=
          Module[{sheets},
              sheets = makeSheetList[analyticallyContinue[f, z], z];
              ParametricPlot3D[Evaluate[{r Cos[φ], r Sin[φ], #}& /@
                              (reim[sheets /. z → z0 + r Exp[I φ]])],
                        {r, 0, ρ}, {φ, 0, 2Pi}, opts,
                        Mesh → False, BoundaryStyle → Black,
                        ExclusionsStyle → {None, Red},

              PlotStyle → Directive[Darker[Green], Specularity[Yellow, 12]],
                        Axes → False, BoxRatios → {1, 1, 1.5}]]
```

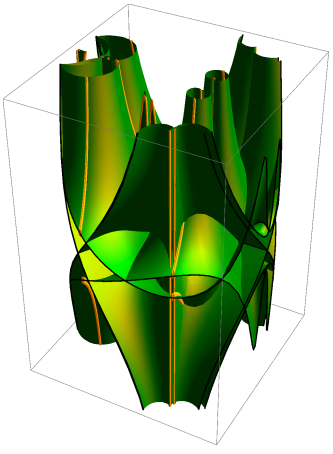
In[59]:=
```
RiemannSheetParametricPlot3D[
 (1 + 4 i) z^2
 ----------------------, Im, {z, 0, Sqrt[2]}, PlotPoints → 80]
 Sqrt[1 - (15 - 8 i) z^4]
```

Out[59]=



In[60]:=
```
RiemannSheetParametricPlot3D[1/Sqrt[z^3 + z^z + 2], Im, {z, 0, Sqrt[2]},
 PlotPoints → 60, ExclusionsStyle → {None, {Yellow, Tube[0.012]}}]
```

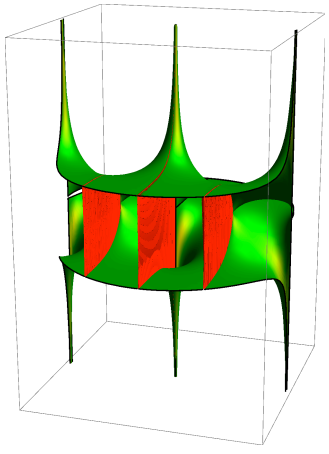
Out[60]=



Using `RiemannSheetParametricPlot3D`, we can now plot the Riemann surface of the function $w = \coth^{1/2}(z)$ that failed earlier. We emphasize the vertical connection along the branch cuts through red polygons.

```
In[61]:= RiemannSheetParametricPlot3D[Sqrt[Coth[z]], Im, {z, 0, Pi},
          PlotPoints → 60, ExclusionsStyle → {Red, None},
          ViewPoint → {-3.1, -0.97, 0.97}]
```

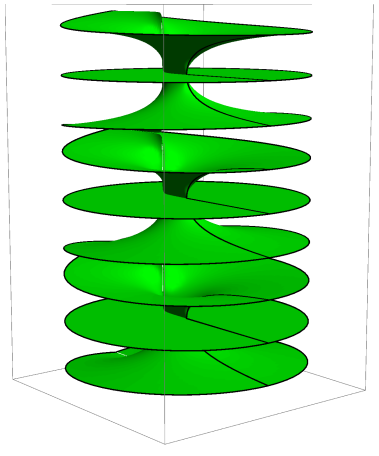
Out[61]=



And here are four considerably more complicated examples. Because of the infinite number of branch points, the function `RiemannSurfacePlot3D` cannot handle them.

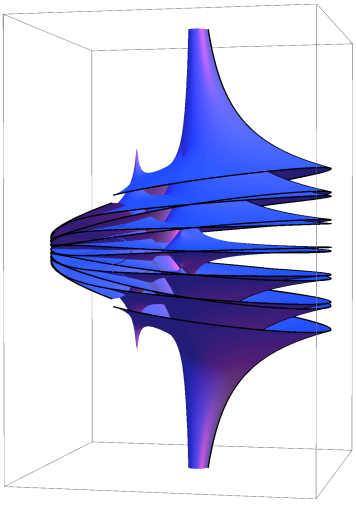
```
In[62]:= RiemannSheetParametricPlot3D[
          Log[(1 - I Sqrt[z])/(1 + I Sqrt[z])], Im, {z, 0, Pi}, PlotPoints → 60,
          PlotStyle → Directive[Green, Specularity[Darker[Green], 20]],
          ExclusionsStyle → {None, None},
          ViewPoint → {2.6, -2.1, 0.51}]
```

Out[62]=



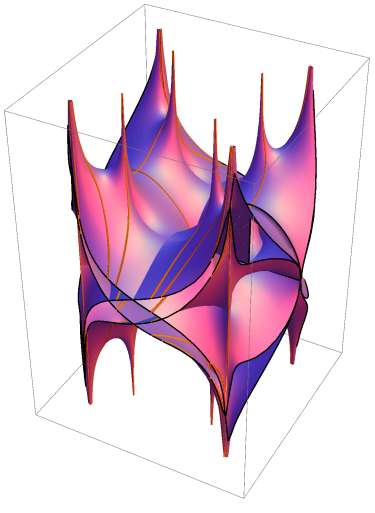
In[63]:=
```
RiemannSheetParametricPlot3D[
 ArcCsc[(1 + I) + z]/Sqrt[z], Re, {z, 0, Pi}, PlotPoints → 60,
 PlotStyle → Directive[Lighter[Blue], Specularity[Red, 24]],
 ExclusionsStyle → {None, None},
 ViewPoint → {0.73, -3.31, 0.052}]
```

Out[63]=



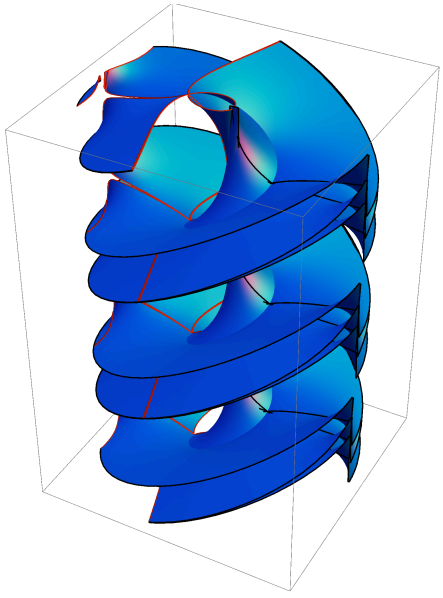
In[64]:=
```
RiemannSheetParametricPlot3D[
 Sqrt[Sin[z]/Sqrt[1 + Exp[-z^2]]], Im, {z, 0, Pi}, PlotPoints → 60,
 PlotStyle → Directive[Lighter[Purple], Specularity[Orange, 16]],
 ExclusionsStyle → {None, {Orange, Tube[0.012]}}]
```

Out[64]=




```
In[65]:= RiemannSheetParametricPlot3D[Log[Sin[z]/Cos[Log[z]]], Im, {z, 0, Pi},
          PlotPoints → 60, PlotStyle → Directive[Blend[{Blue, Green}, 0.3],
            Specularity[Blend[{Yellow, Red}, 0.33], 16]],
          ExclusionsStyle → {None, {Red, Tube[0.016]}}]
```

Out[65]=



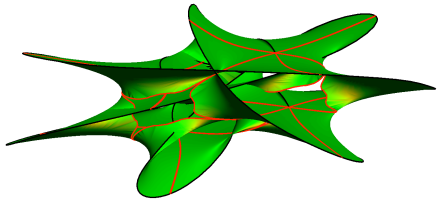
We generalize `RiemannSheetParametricPlot3D` to allow for a matrix as its second argument.

```
In[66]:= RiemannSheetParametricPlot3D[f_, M_?MatrixQ, {z_, z0_, ρ_},
                    opts:OptionsPattern[]] :=
          Module[{sheets},
              sheets = makeSheetList[analyticallyContinue[f, z], z];
              ParametricPlot3D[Evaluate[(M.{r Cos[φ], r Sin[φ],
                              Re[# /. z → z0 + r Exp[I φ]],
                              Im[# /. z → z0 + r Exp[I φ]]})& /@ sheets],
                        {r, 0, ρ}, {φ, 0, 2Pi}, opts,
                        Mesh → False, BoundaryStyle → Black,
                        ExclusionsStyle → {None, Red},

          PlotStyle → Directive[Darker[Green], Specularity[Yellow, 18]],
                        Axes → False,
          BoxRatios → (* all equal *) {1, 1, 1}]]
```

In effect, the next graphic shows $\mathrm{Im}(z) + \mathrm{Im}(w)$ over the $\mathrm{Re}(z) + \mathrm{Im}(z)$, $\mathrm{Re}(w) + \mathrm{Re}(w)$ plane.

```
In[67]:= RiemannSheetParametricPlot3D[(z^3 - 1)^(1/4),
           {{1, 1, 0, 0}, {1, 0, -1, 0}, {0, 2, 0, 2}}, {z, 0, 3/2}],
           PlotPoints → 40, BoxRatios → Automatic,
           Boxed → False, PlotRange → All,
           ViewPoint → {-3.2, -0.9, -0.48}, ViewVertical → {-2.1, 1.2, 0.07}]
```

Out[67]=



To summarize: Using analytically continued sheets and `Plot3D/Parametric`. `Plot3D` with the `Exclusions` option lets you plot Riemann surfaces of compositions of elementary functions. This includes cases where the branch points cannot be calculated in closed form or the case of countably many branch points. Two potential disadvantages of this approach are that calculating the exclusions can be potentially very time consuming and the gaps arising from cutting out thin strips along potential branch cuts cannot be made arbitrarily small with reasonable computational effort.

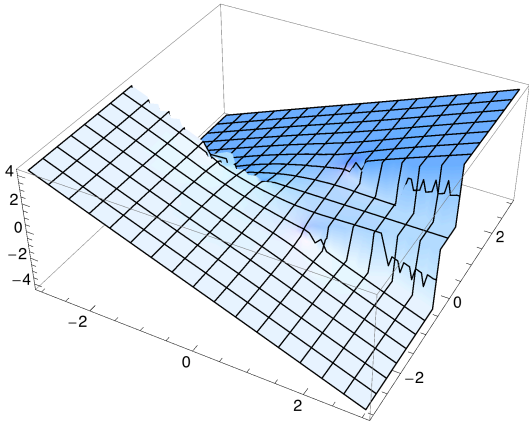
■ Higher-Order Polynomial Roots

Unfortunately, our construction fails for `Root` objects.

```
In[68]:= Options[Solve]

Out[68]= {InverseFunctions → Automatic, MakeRules → False,
          Method → 3, Mode → Generic, Sort → True,
          VerifySolutions → Automatic, WorkingPrecision → ∞}

In[69]:= sol = w /. {ToRules[Roots[w^4 + 2 z^2 w^2 + w - z == 0,
               w, Cubics → False, Quartics → False]]}

Out[69]= {Root[-z + #1 + 2 z^2 #1^2 + #1^4 &, 1], Root[-z + #1 + 2 z^2 #1^2 + #1^4 &, 2],
          Root[-z + #1 + 2 z^2 #1^2 + #1^4 &, 3], Root[-z + #1 + 2 z^2 #1^2 + #1^4 &, 4]}

In[70]:= root = sol[[1]]

Out[70]= Root[-z + #1 + 2 z^2 #1^2 + #1^4 &, 1]
```

```
In[71]:= Plot3D[Evaluate[Im[root] /. z → x + I y], {x, -3, 3}, {y, -3, 3}]
```

Out[71]=



This is not unexpected. The internal function `VisualizationDiscontinuities` yields no exclusions for the `Root` object.

```
In[72]:= Visualization`VisualizationDiscontinuities[
          Re[root] /. z → x + i y, {x, y}]
```

```
Out[72]= {}
```

The reason is the quite large computational effort needed to determine the cuts for parametrized roots. While the branch points are relatively easy to determine as the common root of the function and its derivative through one resultant calculation, calculating the cuts is more expensive.

```
In[73]:= branchPointsF[Root[f_, _], z_] :=
         Module[{w, p}, p = f[w];
         (z /. {ToRules[Reduce[Resultant[p, D[p, w], w] == 0, z]]})]
```

```
In[74]:= bps = branchPointsF[root, z]
```

```
Out[74]= {Root[27 + 544 #1 + 544 #1^2 + 256 #1^3 &, 1]^(1/3),
          -(-1)^(1/3) Root[27 + 544 #1 + 544 #1^2 + 256 #1^3 &, 1]^(1/3),
          (-1)^(2/3) Root[27 + 544 #1 + 544 #1^2 + 256 #1^3 &, 1]^(1/3),
          Root[27 + 544 #1 + 544 #1^2 + 256 #1^3 &, 2]^(1/3),
          -(-1)^(1/3) Root[27 + 544 #1 + 544 #1^2 + 256 #1^3 &, 2]^(1/3),
          (-1)^(2/3) Root[27 + 544 #1 + 544 #1^2 + 256 #1^3 &, 2]^(1/3),
          Root[27 + 544 #1 + 544 #1^2 + 256 #1^3 &, 3]^(1/3),
          -(-1)^(1/3) Root[27 + 544 #1 + 544 #1^2 + 256 #1^3 &, 3]^(1/3),
          (-1)^(2/3) Root[27 + 544 #1 + 544 #1^2 + 256 #1^3 &, 3]^(1/3)}
```

Expanding the first of the roots, we see the existence of three second-order branch points.

```
In[75]:= (Series[root, {z, RootReduce[#], 1}] & /@ bps) // N[#, 2] &
```

> Root::sbr : Because of branch cuts, the series may represent a different root of $-z + \#1 + 2z^2\#1^2 + \#1^4$ & for some values of z. >>

> Root::sbr : Because of branch cuts, the series may represent a different root of $-z + \#1 + 2z^2\#1^2 + \#1^4$ & for some values of z. >>

> Root::sbr : Because of branch cuts, the series may represent a different root of $-z + \#1 + 2z^2\#1^2 + \#1^4$ & for some values of z. >>

> General::stop : Further output of Root::sbr will be suppressed during this calculation. >>

$$\begin{aligned}
\text{Out[75]=}\ &\Big\{-(1.10 + 0.\times 10^{-3}\,i) + (0.03 + 0.39\,i)\,(z - (0.19 + 0.32\,i)) + \\
&\quad O[z - (0.19 + 0.32\,i)]^2,\ -(1.10 - 0.\times 10^{-3}\,i) + \\
&\quad (0.03 - 0.39\,i)\,(z - (0.19 - 0.32\,i)) + O[z - (0.19 - 0.32\,i)]^2, \\
&-0.56 - \sqrt{z + 0.37} - 0.033\,(z + 0.37) + O[z + 0.37]^{3/2}, \\
&-(0.82 + 0.68\,i) + (0.28 + 0.96\,i)\sqrt{z - (0.79 - 0.80\,i)} - \\
&\quad (0.\times 10^{-3} + 0.64\,i)\,(z - (0.79 - 0.80\,i)) + O[z - (0.79 - 0.80\,i)]^{3/2}, \\
&-(0.26 + 0.52\,i) - (0.27 - 0.11\,i)\,(z + (1.09 + 0.28\,i)) + \\
&\quad O[z + (1.09 + 0.28\,i)]^2,\ -(1.7 - 0.2\,i) + \\
&\quad (0.26 + 1.17\,i)\,(z - (0.30 + 1.08\,i)) + O[z - (0.30 + 1.08\,i)]^2, \\
&-(0.82 - 0.68\,i) + (0.28 - 0.96\,i)\sqrt{z - (0.79 + 0.80\,i)} + \\
&\quad (0.\times 10^{-3} + 0.64\,i)\,(z - (0.79 + 0.80\,i)) + O[z - (0.79 + 0.80\,i)]^{3/2}, \\
&-(1.7 + 0.2\,i) + (0.26 - 1.17\,i)\,(z - (0.30 - 1.08\,i)) + \\
&\quad O[z - (0.30 - 1.08\,i)]^2,\ -(0.26 - 0.52\,i) - \\
&\quad (0.27 + 0.11\,i)\,(z + (1.09 - 0.28\,i)) + O[z + (1.09 - 0.28\,i)]^2\Big\}
\end{aligned}$$

A discontinuity of a `Root` object occurs due to a switch in the root numbering. This in turn results from coinciding real parts of different roots. At the point where two roots have a common real part, the real part must have a double root. So carrying out the two resultant calculations gives a sufficient set of algebraic curves along which the roots will exhibit discontinuities.

```
In[76]:= rootExclusions[Root[f_, k_, ___], z_, {x_, y_}] :=
          Module[{pxy, pv, pu, res, u, v},
              (* use explicit real and imaginary
          part for the parameter variable *)
              pxy = f[u + I v] /. z → x + I y;
              (* eliminate imaginary part of the polynomial *)

          pu = Resultant[ComplexExpand[Re[pxy]], ComplexExpand[Im[pxy]], v];
              (* condition for double roots of the real part *)
              res = Resultant[pu, D[pu, u], u, Method → Modular];
              (* return polynomials that change sign *)
              DeleteCases[First/@FactorList[res], _?NumericQ]]
```

Even for just a fourth root, evaluating `rootExclusions` is quite time consuming and the resulting polynomial is large and complex.

```
In[77]:= (rexc = rootExclusions[root, z, {x, y}]); // Timing
Out[77]= {507.655, Null}

In[78]:= Short[rexc, 6]
Out[78]//Short=
{y, 729 + <<49>>, -5 489 031 744 x^3 + 159 190 885 224 x^6 - 687 658 951 372 x^9 -
  6 445 172 140 893 x^12 - 26 594 081 222 808 x^15 + 156 194 821 151 552 x^18 +
  540 341 016 274 944 x^21 + 729 683 032 341 504 x^24 + 558 591 655 043 072 x^27 +
  <<407>> + 789 019 852 013 568 x^12 y^42 - 24 747 670 279 028 736 x^15 y^42 -
  918 302 215 913 865 216 x^18 y^42 + 12 495 536 233 302 196 224 x^21 y^42 -
  901 736 973 729 792 x^13 y^44 - 16 231 265 527 136 256 x^16 y^44 +
  655 662 972 898 639 872 x^19 y^44 + 14 427 791 579 676 672 x^17 y^46}
```

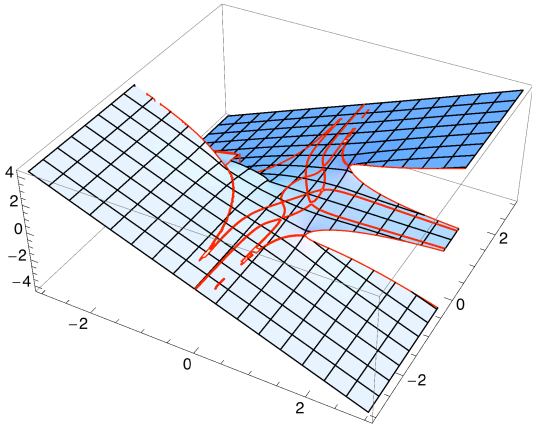
Here is a quick overview of the last polynomial.

```
In[79]:= {ByteCount[rexc], LeafCount[rexc],
          Length[rexc], Exponent[rexc /. {x → τ, y → τ}, τ],
          Floor[Log[10, Max[Abs[Cases[rexc, _Integer, {-1}]]]]]}
Out[79]= {52 984, 2567, 3, {1, 18, 63}, 22}
```

Once the set of exclusions has been calculated, we can use it as the setting for the `Exclusions` option.

```
In[80]:= Plot3D[Evaluate[Im[root] /. z → x + i y], {x, -3, 3},
          {y, -3, 3}, Exclusions → rexc, ExclusionsStyle → {None, Red},
          MaxRecursion → 3, ClippingStyle → None, PlotPoints → 120]
```

Out[80]=



■ Life in Four Dimensions

Up to now, we have seen Re($w(z)$) and Im($w(z)$) over the complex z plane (we plotted a more general graphic earlier). The most complete information of a function $w = w(z)$ is the set of pairs of complex numbers $\{z, w\}$ or the quadruple of real numbers {Re(z), Im(z), Re(w), Im(w)}. Imagining the hypersurface $w = f(z)$ in $\mathbb{C}^2$ (isomorphic to $\mathbb{R}^4$), we are naturally led to Banchoff's tetraview [6] of a complex-valued function. In this section, we implement such a construction. Starting with a plot of the real and imaginary parts of a complex-valued function over the complex plane, we extract the underlying graphic complex, and re-evaluate the function at the x,y values. (Using the already discretized surface in the form of a `GraphicsComplex` gives us the advantage of adaptive subdivision.) Because we will use the values of the derivatives of $w(z)$ with respect to z for calculating the normals, we store the values of the derivatives at the (complex) vertex points in the option setting for `VertexNormals`.

```
In[81]:= to4DGC[pl_, f_] :=
         With[{gc = Cases[pl, _GraphicsComplex, Infinity][[1]], fp = f'},
           GraphicsComplex[Developer`ToPackedArray[N[{#[[1]], #[[2]],
                    Re[f[#.{1., 1. I, 0.}]], Im[f[#.{1., 1. I, 0.}]]}& /@
                                                  gc[[1]]]],
                  gc[[2]], VertexNormals →
                  Developer`ToPackedArray[N[({
                    Re[fp[#.{1., 1. I, 0.}]],
         Im[fp[#.{1., 1. I, 0.}]]}& /@
                                                gc[[1]])]]]]
```

We define a function `RiemannSheetParametricPlots4D` that generates a `GraphicsComplex` with vertex values in $\mathbb{R}^4$. (Here we use the suboption `Time`⁻. `Constraint` in the `Exclusions` option to allow the calculation of more complicated exclusion sets for more complicated functions.) The sheets are generated using the function `analyticallyContinue`.

In[82]:=
```
RiemannSheetParametricPlots4D[
args:PatternSequence[f_, {z_, z0_, ρ_},
                                opts:OptionsPattern[]]] :=
Module[{sheets, pp3d, data},
    sheets = makeSheetList[analyticallyContinue[f, z], z];
    data = Table[
    pp3d = ParametricPlot3D[Evaluate[{r Cos[φ], r Sin[φ],
                        Re[sheets[[k]]] + Im[sheets[[k]]] /.
                        z → z0 + r Exp[I φ]}],
                  {r, 0, ρ}, {φ, 0, 2Pi}, opts,
        PlotPoints → {60, 20}, Mesh → 5, BoundaryStyle → Black,
        PerformanceGoal → "Quality",
        Exclusions →
{Automatic, MaxRecursion → 4, TimeConstraint → 1000},
        ExclusionsStyle → {None, Red}];
     to4DGC[pp3d, Function @@ {z, sheets[[k]]}],
          {k, Length[sheets]}];
    (* for a fast interactive rotation in 4D, cache result *)
    DownValues[RiemannSheetParametricPlots4D] =
      Take[DownValues[RiemannSheetParametricPlots4D], -1];
    RiemannSheetParametricPlots4D[args] = data;
    data]
```

Here is an example. The abbreviated output shows that the actual 4D `Gra`⁻. `phicsComplex` is quite large.

In[83]:=
```
RiemannSheetParametricPlots4D[√z , {z, 0, 2}]
```

Out[83]=

A very large output was generated. Here is a sample of it:

```
{GraphicsComplex[
  {{3.38983×10^-8, 1.121×10^-14, 0.000184115, 3.04428×10^-11},
   {0.0338983, 1.121×10^-8, 0.184115, 3.04429×10^-8},
   <<16 659>>, {-0.166667, -0.288675, 0.288675, -0.5},
   {-0.333333, -0.57735, 0.408248, -0.707107}},
  {<<1>>}, VertexNormals →
   {{2715.7, - <<22>>}, <<16 661>>, {<<19>>, <<19>>}}], <<1>>}
```

Show Less | Show More | Show Full Output | Set Size Limit...

In the absence of a direct 4D viewer in Version 6 (or any support for `Graphics4D`, for that matter—hopefully Version 7 will have at least some), we must project into a 3D subspace to view the surface as a `Graphics3D` object. For a given 4×3 projection matrix, this is straightforward for the vertices and not too difficult for the normals. Fortunately, having the information of the embedded hypersurface in 4D allows a projection with normals in 3D using the Cauchy-Riemann conditions, which will result in a smooth-looking surface. Because we will use the final function interactively, we use `Compile` to carry out the calculation of the 3D vertex coordinates and the 3D normals as efficiently as possible.

```
In[84]:= toPoints3D =
         Compile[{{M, _Real, 2}, {values, _Real, 2}}, M.#& /@ values];

         toNormals3D =
         Compile[{{M, _Real, 2}, {derivatives, _Real, 2}},
             Module[{a = {0., 0., 0.}, b = {0., 0., 0.}},

         Map[(a = M.{1., 0., #[[1]], #[[2]]}; b = M.{0., 1., -#[[2]], #[[1]]};
                #/Sqrt[#.#]& @ {a[[2]] b[[3]] - a[[3]] b[[2]],
                          a[[3]] b[[1]] - a[[1]] b[[3]],
                          a[[1]] b[[2]] - a[[2]] b[[1]]})&, derivatives]]];
```

The function `to3DGraphicsComplex` carries out the projection from 4D to 3D.

```
In[86]:= to3DGraphicsComplex[M_?MatrixQ,
         GraphicsComplex[vertices4D_, body_,

         VertexNormals → vertexNormals4D_, rest___],
                    colorFunction_] :=
         GraphicsComplex[toPoints3D[M, vertices4D], body,
                  VertexNormals → toNormals3D[M, vertexNormals4D],
                  If[colorFunction === Automatic, Sequence @@ {},
                    VertexColors → (colorFunction @@@ vertices4D)]]
```

Putting the last two functions together, we have `TetraviewPlot`. We give it the option `ColorFunction4D`, which operates on the unscaled 4D coordinates to easily color a surface.

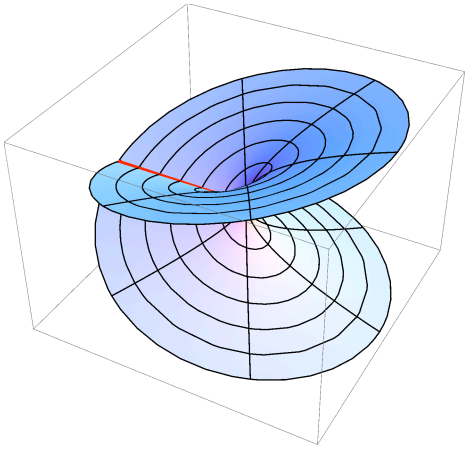
```
In[87]:= Options[TetraviewPlot] =
         Join[Options[Graphics3D], {ColorFunction4D → Automatic}];
```

```
In[88]:= TetraviewPlot[f_, {z_, z0_, ρ_}, M_, opts:OptionsPattern[]] :=
         Module[{rsp4D,cf},
             rsp4D = RiemannSheetParametricPlots4D[f, {z, z0, ρ}]; CC=rsp4D;
             cf = OptionValue[ColorFunction4D];
             Graphics3D[to3DGraphicsComplex[M, #, cf]& /@ rsp4D,
                   Sequence @@ DeleteCases[{opts}, ColorFunction4D → _]]]
```

Here are a few examples of the function `TetraviewPlot`.

```
In[89]:= TetraviewPlot[√z , {z, 0, 2},
          {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}}]
```

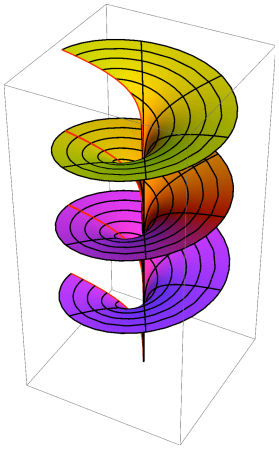
Out[89]=



The next example colors the surface of Im(ln(*z*)) according to values of Re(ln(*z*)).

```
In[90]:= TetraviewPlot[Log[z], {z, 0, 2},
          {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 1}},
                        BoxRatios → {1, 1, 2},
          ColorFunction4D → (Hue[ArcTan[#4] / 10] &),
                        PlotRange → {All, All, {-15, All}}]
```

Out[90]=



A convenient and natural way to parametrize the projection matrix is to use the six rotation angles between the four coordinate planes in $\mathbb{R}^4$.

```
In[91]:= makeRotationMatrix4D[{φxy_, φxu_, φxv_, φyu_, φyv_, φuv_}] :=
          Fold[Dot, IdentityMatrix[4],
           {{{Cos[φxy], Sin[φxy], 0, 0}, {-Sin[φxy], Cos[φxy], 0, 0},
          {0, 0, 1, 0}, {0, 0, 0, 1}},
            {{Cos[φxu], 0, Sin[φxu], 0}, {0, 1, 0, 0},
          {-Sin[φxu], 0, Cos[φxu], 0}, {0, 0, 0, 1}},
            {{Cos[φxv], 0, 0, Sin[φxv]}, {0, 1, 0, 0},
          {0, 0, 1, 0}, {-Sin[φxv], 0, 0, Cos[φxv]}},
            {{1, 0, 0, 0}, {0, Cos[φyu], Sin[φyu], 0},
          {0, -Sin[φyu], Cos[φyu], 0}, {0, 0, 0, 1}},
            {{1, 0, 0, 0}, {0, Cos[φyv], 0, Sin[φyv]},
          {0, 0, 1, 0}, {0, -Sin[φyv], 0, Cos[φyv]}},
            {{1, 0, 0, 0}, {0, 1, 0, 0},
          {0, 0, Cos[φuv], Sin[φuv]}, {0, 0, -Sin[φuv], Cos[φuv]}}}]
```

In the definition for `RiemannSheetParametricPlots4D`, we used the line

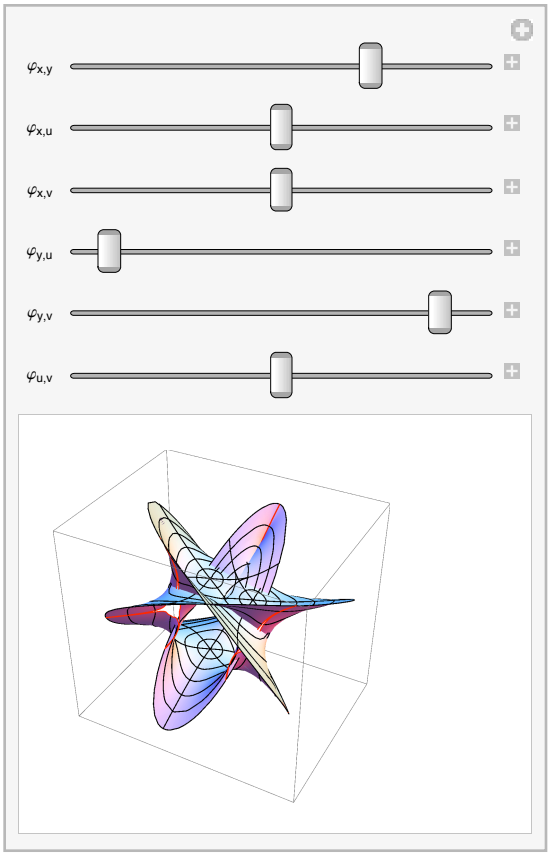
```
DownValues[RiemannSheetParametricPlots4D] =
        Take[DownValues[RiemannSheetParametricPlots4D], -1]
```

to cache the result of the most time-consuming step inside `TetraviewPlot`. This allows us to implement a demonstration that interactively changes the 3D subspace selected.

```
In[92]:= TetraviewPlot[(2 + z^3)^(1/3), {z, 0, 2},
           {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}}]; // Timing
Out[92]= {104.022, Null}
```

```
In[93]:= Manipulate[TetraviewPlot[(2 + z^3)^(1/3), {z, 0, 2},
           Most[makeRotationMatrix4D[{φxy, φxu, φxv, φyu, φyv, φuv}]]],
         {{φxy, 1.48, φx,y}, -π, π}, {{φxu, 0, φx,u}, -π, π},
         {{φxv, 0, φx,v}, -π, π},
         {{φyu, -2.86, φy,u}, -π, π}, {{φyv, 2.64, φy,v}, -π, π},
         {{φuv, 0, φu,v}, -π, π}, SaveDefinitions → True]
```

Out[93]=



This demonstration gives us a feeling for the 2D hypersurface. Branch point neighborhoods in $w = w(z)$ correspond to flat regions in $z = z(w)$, and vice versa. As a result, the initial small gaps along the cuts of $w = w(z)$ widen as we look at the 4D surface from different directions.

This ends our short visit into the world of Riemann surface visualizations with *Mathematica* 6. We will be coming back to this subject in future columns.

References

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

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

[3] M. Trott, "Trott's Corner: Visualization of Riemann Surfaces IIc: Compositions of Elementary Transcendental Functions," *The Mathematica Journal*, **8**(3), 2002 pp. 409-432.

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

[5] M. Trott, *The Mathematica GuideBook for Numerics*, New York: Springer-Verlag, 2006.

[6] T. F. Banchoff, "Computer Graphics in Geometric Research," in *Recent Trends in Mathematics: Conference in Reinhardsbrunn (Reinhardsbrunn 1982)*, (H. Kurke, J. Mecke, H. Triebel, and R. Thiele, eds.), Teubner-Texte Math., **50**, Leipzig: Teubner, 1983 pp. 316-327.

M. Trott, "The Return of the Riemann Surface," *The Mathematica Journal*, 2011.
dx.doi.org/ doi:10.3888/tmj.10.4–1.

Michael Trott
Senior Member, Technical Staff
Wolfram Research, Inc.
mtrott@wolfram.com