An Algorithmic Approach to Manifolds

An analytical approach to form modelling as an introduction to computational morphology

Rémi Barrère

An algorithmic approach to manifolds is presented. The initial purpose was to blend geometric and symbolic aspects, so as to equip computer assisted design with symbolic capabilities. Nevertheless, this investigation aims more generally at providing a uniform treatment of analytical geometry and field analysis, in view of applications to physics, system modelling and morphology. This computational approach is based on a reification of parametric plotting commands.

After a presentation of the data structure, the core of the paper describes a range of operators for the manipulation of manifolds. It stresses their potential use in shape design and scene description, in particular their ability to supersede several graphic packages. As such, the data type constitutes the foundations of a computational morphology. Then, various extensions are discussed: distributions and fields, mesh generation for finite element software and the prospect of an extension of the vector analysis package, with emphasis on tensors and differential forms.

■ Introduction

Computer algebra and symbolic programming have introduced analytical capabilities into many areas of scientific computing: discrete systems, algebra and summation, calculus and differential equations… Nevertheless, little benefit has been gained in shape design; researches in that domain have stimulated the evolution of computer assisted design but so far, these tools have included no or little symbolic capabilities and most CAD software is still developed with procedural languages and numerical methods. Besides, geometric problems have been tackled so far mainly by means of algebraic or theorem proving methods [1], thus leading to an underdevelopment of analytical methods.

Yet the analytical approach to geometry constitutes the foundations of many physical questions, especially those linked to space or space–time analysis by means of field theory. Hence the attempt to introduce analytical capabilities into geometry, initially in view of applications to shape description, then with the purpose to lay the foundations of further uses in differential geometry and field analysis. Although geometry is commonly thought of as the federating mathematical abstraction underlying those questions, morphology appears as an encompassing common denominator able to take into account questions outside the field of geometry, such as scene description, linkage design or finite element analysis.

◻ Morphology as a Transverse Concept

Although shape and form are more or less considered equivalent in common language, shape is rather devoted to geometric external aspects whereas form is rather devoted to more general internal structuring aspects (geometric or not). For instance, curves, surfaces and volumes as such are shapes, while considered together with some other characteristics, such as a field, they tend to be thought of as forms. So a form description is a piece of information about the way in which an object occupies and structures space. In practice, it is indissoluble from the way the object (or its form) is generated and can be transformed or combined with other ones.

Moreover, as a result of our cognitive capabilities, we tend to understand objects in contrast with a background, and also to perceive them as structured in container (wrapping structure) and content (internal structure) [2]. On the contrary, modern ideas about space or spacetime tend to identify space with its structuring content, space being constituted by the relationships between objects. To some extent, these opposite ideas can be made compatible by blending continuous aspects (figures as expansions) and discrete ones (figures as objects),

◻ Mathematical and Algorithmic Requirements

Many form descriptions have been developed so far, in view of more or less specific applications: drawing software, geometric reasoning, computer assisted design, fractal image generation, data visualization…There is probably no universal form description, for those methods need always, to some extent, be optimized to best serve some specific problems. However, the trend to develop ad hoc optimized solutions for practical needs yields "a widely scattered conglomeration of disparate and, at first sight, unrelated methods" [3]. This subsequently tends to severely decrease the so called orthogonality, i.e., the capability to cross fertilize disciplines by information exchange and object combinations thanks to generic types.

By quoting Lord and Wilson [3], one may even stress the need for a mathematical foundation for a science of morphology unifying various approaches, a need that has not been fully recognized yet. Today, the requirement that the proposed method should lend itself well to an algorithmic treatment must be taken into account. It should also have a broader scope than the so called mathematical morphology, which is more or less restricted to the "pixel level" methods used for image processing. In the following, we actually focus on form synthesis rather than form analysis. The solution put forward derives from a common mathematical tool, slightly adapted to an algorithmic purpose, i.e., a new glance at a classical theory [4].

◻ Manifolds as a Unifying Approach to Morphology

We bet on manifolds, because of their sound analytical foundation, their strong geometric flavor and their adaptation to the algorithmic treatment, especially computer algebra and symbolic programming. In particular, a manifold determines a codomain as a subset of some (often euclidean) space. By regarding the codomain as the object and the surrounding space as the background, it actually determines a shape. By considering various structuring elements such as a coordinate system (domain) or a field over the manifold, it determines as many forms. Moreover, the boundary of a manifold constitutes an element of a container–content approach. More generally, other structuring tools will be described below, such as atlases that enable scene descriptions.

Because of its visual aspects, shape design is in natural relationship with the underlying graphic capabilities [5, 6]. In Mathematica, these rely upon a few graphic primitives (`Line`, `Polygon`…) and a range of standard plotting commands (`Plot`, `ParametricPlot`…), plus a variety of complementary commands defined in various packages (`Shapes‘`, `SolidOfRevolution‘`…). This leads to a functional approach where shapes are produced by plotting commands that build them by assembling low level graphic primitives, thus excluding higher level objects.

On the contrary, resorting to manifolds leads to a reification of shape: these and other geometric entities like fields, can be defined as quite compact symbolic objects, rather than huge assemblies of raw graphic primitives resulting from plotting commands. This facilitates the introduction of both higher level entities, able to describe objects as wholes, and higher level symbolic functions able to manipulate and combine these entities. This also enables the consistent gathering of graphic tools scattered around various graphic packages and their extension not only to shape design but also to field analysis.

```
Needs["Manifolds‘Morphology‘"]
```

An experimental set of packages (available material) is developed to investigate the ideas presented in the paper. `Manifolds‘Morphology‘` is simply intended to load the whole directory. Extracts or simplified versions, which should not be evaluated, are occasionally shown in the paper.

■ An Algorithmic Approach to Manifolds

Although a manifold is usually defined as a collection (called an atlas) of patches, i.e., of homeomorphisms from an open subset of $\mathbb{R}^n$ onto an open subset of a topological space $\mathcal{M}$ [7], the algorithmic approach will rather focus on $\mathbb{R}^n$ as a curvilinear coordinate systems on $\mathcal{M}$, which in most cases will be some subset of $\mathbb{R}^p$. Indeed, the notion of a manifold is deeply rooted in analytical geometry, i.e., in the practical need to describe curves, surfaces and volumes, or their n–dimensional generalizations, both from the analytical and the geometrical points of view.

□ The Data Structure

As a reflect of this origin, the algorithmic approach aims at blending the analytical and geometric aspects, especially their graphic counterpart. Basically, the type `Manifold` introduced hereafter denotes a patch; then an atlas is simply a collection of manifolds, without any continuity or differentiability requirement. The type `Manifold` consists of two entities: a set of expressions that should be thought of as a list of parametric equations, together with a domain specification for the variables, which are the local coordinates on the manifold. The type being the head of the expression, this leads to the informal expression template : `Manifold[list of expressions, domain specification]`. The domain specification follows the syntax of continuous domains in plotting commands: a triple or a sequence of triples `{c, cmin, cmax}` denoting a coordinate symbol with a minimum value and a maximum value.

This design is adapted from a first attempt by Gray [8] who actually adopted a more general viewpoint by considering mappings. Here is an example, which may be thought of as a parametrized parametric representation of a segment of a ring with the symbolic parameter r.

```
Manifold[{ρ Cos[θ], ρ Sin[θ]}, {ρ, 1, r}, {θ, 0, 3 π / 2}]
```

In that case, the codomain is a subset of $\mathbb{R}^2$ and the expressions are usually interpreted as the parametric representation (equations) of a geometric domain. Nevertheless, manifolds describe in principle abstract entities that entail no assumption about their nature, except they are member of a topological space. In particular, the parametrized objects need not be points nor vectors. Any parametrized family of entities is susceptible of a description as a manifold, provided it has some differentiability or at least continuity properties. For instance, a parametrized family of matrices or a parametrized family of functions can be investigated as manifolds. However, there is no standard visualization procedure for such manifolds and their graphic representation may require assumptions or tricks.

When the codomain is a subset of $\mathbb{R}^p$, it needs not be euclidean; manifold theory is basically a relativistic theory of abstract spaces. Nevertheless, in many applications, manifolds are interpreted from an absolute point of view: the codomain is supposed to be some subset of an euclidean space with an orthogonal cartesian coordinate system and the domain is viewed as a curvilinear coordinate system for the codomain. Such an absolute interpretation is almost unavoidable when graphic representations are considered.

However this assumption is not mandatory and manifolds may have other interpretations: for instance, in another context, the codomain may be the subset of an euclidean space with a curvilinear coordinate system or even the subset of a space with no metric property. So we try to maintain flexibility by avoiding useless strong assumptions regarding the absolute or relative interpretations of manifolds, thus leaving the user choose the interpretation only when needed. In particular, commands like `Connect` are introduced, that facilitate the switching between the absolute and relative interpretations.

Although it may be required in some reasonings, the distinction between open or closed intervals is useless in this algorithmic context ; hence the use of lists for domains. As a remark, these algorithmic versions of manifolds need not be differentiable, i.e., they describe varieties as well (for instance, fractal varieties). Finally, a manifold with no domain is a point. Its syntax is `Manifold[list of expressions]`. Taking into account this limit case is useful in some generic applications. Actually, when working with manifolds, there is no longer any notion of point, curve, surface or volume as types since all are manifolds. The number of coordinates gives the nature of the figure, the dimension of the codomain specifying the immersion space.

□ The Selectors

The selectors extract the various arguments of a typed expression [9]. They were initially introduced in computer science to isolate the interface specification from the internal representation. Actually, when the representation is stable, argument extraction can be done directly with patterns in the left hand sides of transformation rules. Nevertheless, some generic programs are more easily designed by resorting to selectors.

By thinking of applications to geometry or field analysis, we use `Coordinates` for the variables, `Domain` for the coordinate ranges and `Codomain` for the parametric expressions (thus identifying the functions and their values).

```
Coordinates[Manifold[_, d__List]] ^:= Map[First, {d}]
Domain[Manifold[_, d__List]] ^:= {d}
Codomain[Manifold[e_, __]] ^:= e
```

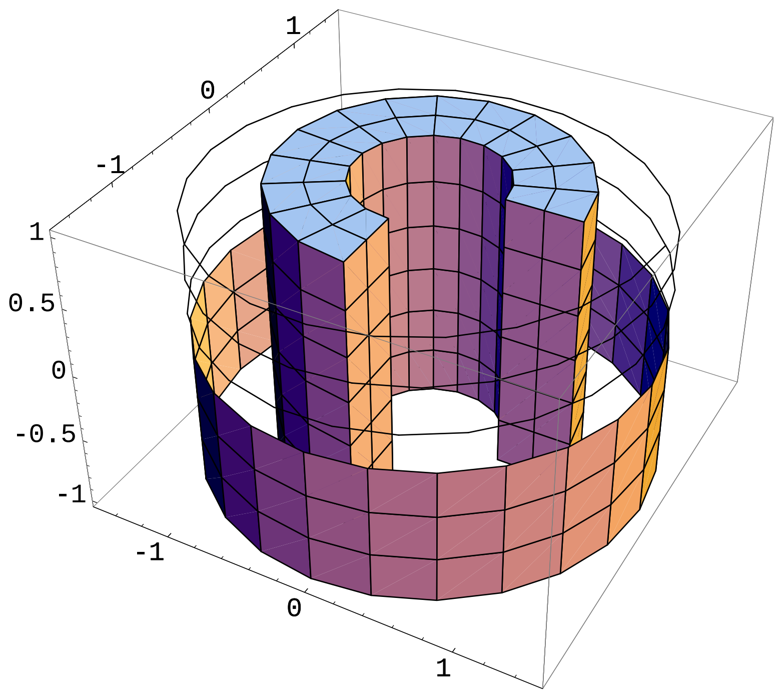
There might be an ambiguity between domain and codomain, which in this context are the domain and codomain of the associated mapping, while the codomain can also be thought of as a domain in a geometric sense. We nevertheless maintain domain and codomain, because of their common use in mathematics as well as computer science and we will resort to patch or region to denote a geometric domain.

□ Variants

Atlases and geometric scenes

In the frame of this algorithmic approach, an atlas is simply a list of manifolds; so one departs from the mathematical approach where a differentiable (at least continuous) overlapping is required. An atlas is basically a piecewise manifold; it is actually slightly more general for it enables the accumulation of manifolds with different dimensions. The manifolds of an atlas must nonetheless have codomains with the same dimensions.

```
Draw[Atlas[{
   DrawingStyle[Manifold[Cylindrical[{r, θ, h}], {r, 1 / 2, 1},
     {θ, 0, 3 π / 2}, {h, -1, 1}], PlotPoints -> {3, 15, 7}],
   DrawingStyle[Manifold[Cylindrical[{3 / 2, θ, h}],
     {θ, 0, 2 π}, {h, -1, 0}], PlotPoints -> {25, 4}],
   Manifold[Cylindrical[{3 / 2, t, t / 6 / π}], {t, 0, 6 π}]
  }], BoxStyle → GrayLevel[0.5]]
```



```
- Graphics3D -
```

Combined with `Draw`, an `Atlas` can be viewed as an adaptation to manifolds of the command `StackGraphics` from the package `Graphics‘Graphics3D‘`, which directly applies to graphic objects.

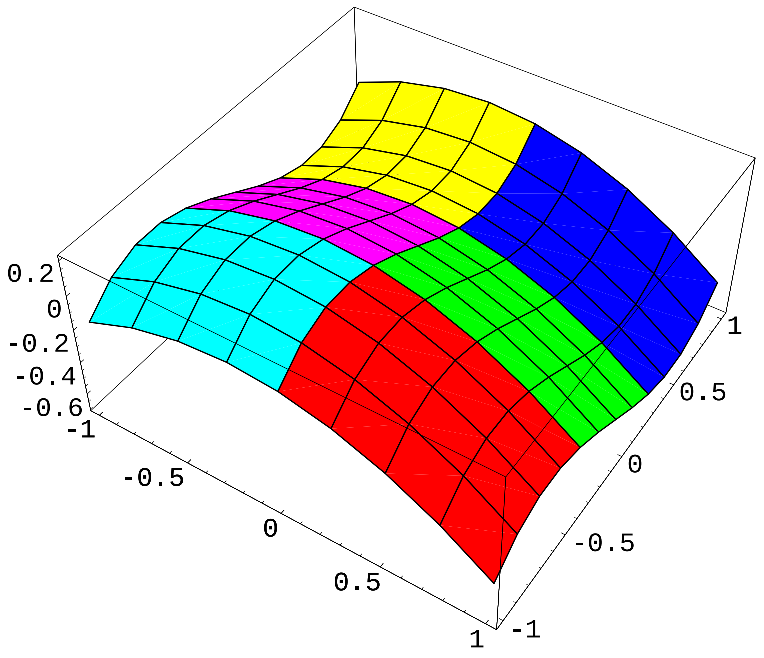
Atlases are especially useful for the manipulation of sets of manifolds as wholes. As such, they constitute a natural data structure for compound shapes or geometric scenes, the atlas then being thought of as a set of figures. Manifolds then appear as the natural primitives for scene description, in association with the type `Atlas` as a composition tool. As such they enable the combination of continuous and discrete aspects.

Many−valued manifolds

A many−valued manifold differs from an atlas in the sense that there is a common domain for a list of parametrizations; nevertheless, it can possibly be rewritten as an atlas of manifolds. The multiple parametrizations can not only be arranged in lists but also matrices or higher dimensional tables. Roughly, many−valued manifolds implement coverings.

Many−valued manifolds are mainly used to parametrize figures with several branches (e.g. hyperbole), in particular in the case of manifolds computed as inverse mappings. Applications of many−valued manifolds to fractals are given below. These can also be used to build curved rasters.

```
R2D2 = Raster2D[
   {{RGBColor[0, 1, 1], RGBColor[1, 0, 1], RGBColor[1, 1, 0]},
    {RGBColor[1, 0, 0], RGBColor[0, 1, 0], RGBColor[0, 0, 1]}},
   Manifold[{u, v, (v^3 - v) / 2 - (u^2 + u / 2) / 3},
    {u, -1, 1}, {v, -1, 1}]];

Draw[R2D2, PlotPoints → 5,
 Lighting → False, ViewPoint -> {1.3, -2., 2.3}]
```



```
- Graphics3D -
```

Coordinate systems and manifolds

In view of a generic treatment of fields, coordinate systems should be regarded as particular cases of coordinate systems on manifolds. Nevertheless, for convenience and consistency with the vector analysis package, a specific data structure for usual coordinate systems is maintained, in the form of a type wrapping a list of coordinate symbols, e.g., `Polar[{r, θ}]`. There is nonetheless no default symbol for coordinates: freely chosen by the user, these are specified where needed.

```
SetAttributes[CoordinateRanges, HoldFirst]
SetAttributes[CoordinateSystemQ, HoldFirst]
CoordinateSystemQ[_Polar] := True
Polar[{r_, θ_}] := {r Cos[θ], r Sin[θ]}
CoordinateRanges[Polar[{r_, θ_}]] ^:= {{r, 0, ∞}, {θ, 0, 2 π}}
```

The associated transformation rules must be given for every common coordinate system. A symbol like `Polar` may express either a coordinate system (absolute viewpoint) when processed as a type or a change of coordinates (relative viewpoint) when processed as a function. For that purpose, the corresponding argument of some operators is maintained held with an appropriate attribute. If need be, the auxiliary command `ToManifold` generates the associated manifold with the standard coordinate ranges.

In particular, `Manifold[Polar[{r, θ}]]` yields `Manifold[{r Cos[θ], r Sin[θ]}]`, i.e., the cartesian coordinates of the point; so it is equivalent to the command `CoordinatesToCartesian` of the vector analysis package, the analog of `CoordinatesFromCartesian` being obtained by introducing reciprocal coordinate systems (respectively coordinate changes), e.g., `InversePolar[{u, v}]`.

```
Manifold[InversePolar[{u, v}]]
```

```
Manifold[{√(u² + v²), ArcTan[u, v]}]
```

With these conventions, the expression `Manifold[Polar[{f[t], g[t]}], {t,tmin,tmax}]` describes the curve with parametric expressions `f[t]` and `g[t]` in polar coordinates. Surfaces in three dimensional coordinate systems can be described in a similar way. Although it is not common, this way of expressing manifolds by their parametric representations in some coordinate system is concise.

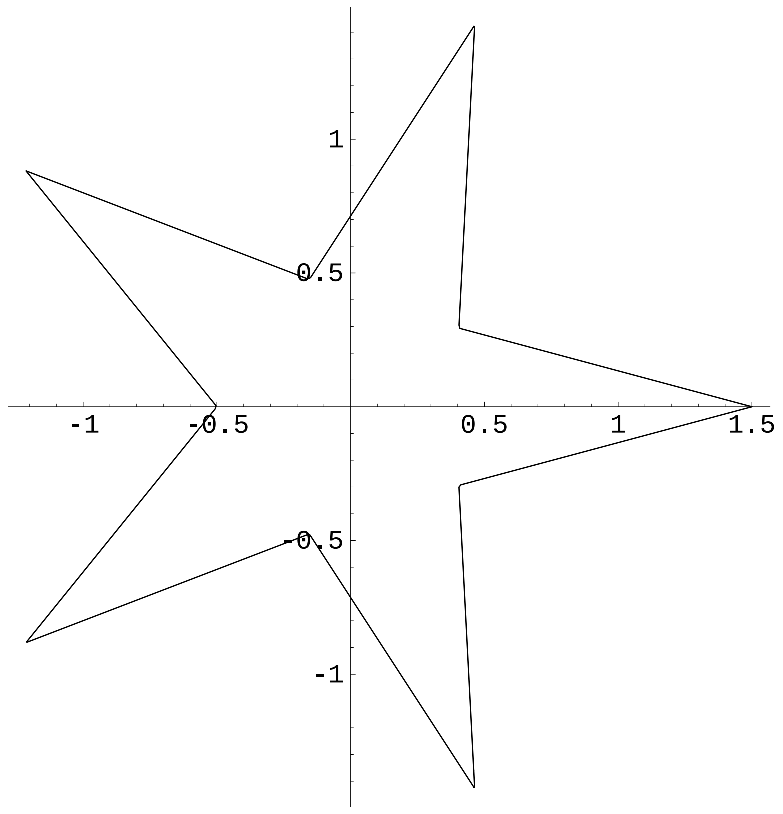
Manifolds with interpolating functions

If need be, a manifold can be defined in terms of interpolating functions. Convenient in the case of differential equation solving (`NDSolve`), such manifolds are occasionally useful to built more or less intricate shapes.

```
ip = {Interpolation[Table[{t, (1 + Cos[5 t] / 2) Cos[t]},
    {t, 0., 2 π, π / 5}], InterpolationOrder → 1],
  Interpolation[Table[{t, (1 + Cos[5 t] / 2) Sin[t]},
    {t, 0., 2 π, π / 5}], InterpolationOrder → 1]}
```

```
{InterpolatingFunction[{{0., 6.28319}}, <>],
 InterpolatingFunction[{{0., 6.28319}}, <>]}
```

```
Draw[Manifold[Through[ip[t]], {t, 0, 2 π}],
  AspectRatio → Automatic];
```



■ Visualization of Manifolds

The visualization of manifolds mainly relies upon the `ParametricPlot` and `ParametricPlot3D` commands, plus two novel functions (`SolidParametricPlot` and `SolidParametricPlot3D`) developed for 2D and 3D shapes. A generic command `Draw` is introduced, in the form of polymorphic transformation rules.

□ The Command Draw

Both the functions to be plotted and the domains being part of the data structure, manifolds require a single graphic command with a single argument plus possible options; let us call it `Draw`. Only manifolds the codomain of which have a geometric interpretation in a two or three dimensional space are drawn, the length of the first argument determining the dimension of the representation space. Then the number of variables determines the nature of the represented object: point, curve, surface, volume or higher dimensional object. Moreover, the codomain is supposed to be a subset of an euclidean affine space equipped with a canonical orthogonal cartesian coordinate system, which has consequently the statute of an “absolute” reference space. Then the domain is that of the curvilinear coordinate system of the manifold.

Most drawings are “subcontracted” to standard parametric plotting commands. The various combinations are summarized in the following table.

n (domain)	p (codomain)	plotting function
0	2	Show (Graphics)
0	3	Show (Graphics3D)
1	2	ParametricPlot
1	3	ParametricPlot3D
2	2	SolidParametricPlot
2	3	ParametricPlot3D
3	3	SolidParametricPlot3D

Extensions of the family `ParametricPlot` are required for the 2–2 and 3–3 cases. They have been temporarily called `SolidParametricPlot` and `SolidParametricPlot3D`, although polymorphism should allow to process them as specific cases of `ParametricPlot` and `ParametricPlot3D`. `SolidParametricPlot3D` draws the boundary, which is valid only provided the jacobian of the parametric equations does not vanish. The case p=1 is not processed by `Draw` because it is of weak visual interest. Nevertheless, thanks to a lifting, it can be rearranged into a 2D or a 3D manifold so as to be visualized the same way functions are visualized with `Plot` or `Plot3D`.

```
Lift[Manifold[{h[t]}, {t, -1, 1}]]
```

```
Manifold[{t, h[t]}, {t, -1, 1}]
```

◻ Higher Dimensional Manifolds

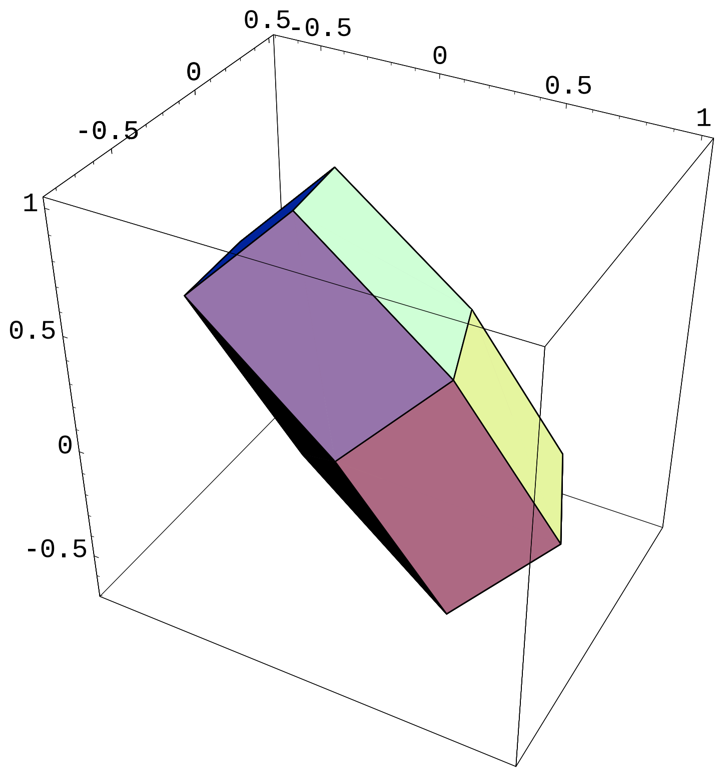
Higher dimensional manifolds are processed by means of the 2D or 3D projection of their boundary. Here is the example of the 4–dimensional unit hypercube.

```
UnitHyperCube[s_List] :=
 Manifold[s, Apply[Sequence, Thread[List[s, 0, 1]]]]
uhc4 = UnitHyperCube[{a, b, c, d}]
```

```
Manifold[{a, b, c, d}, {a, 0, 1}, {b, 0, 1}, {c, 0, 1}, {d, 0, 1}]
```

Below, a 3D projection of the 4D unit hypercube is drawn. Transparency effects would be welcome. Analog applications to complex–valued functions are also possible.

```
Draw[Boundary[Transform[uhc4,
   Shadowing[{{1, -1, 0, 0}, {1, 0, -1, 0}, {1, 0, 0, -1}},
    {1, 2, 1, 2}]]], PlotPoints → 2]
```



```
- Graphics3D -
```

□ Manifolds and Fractals

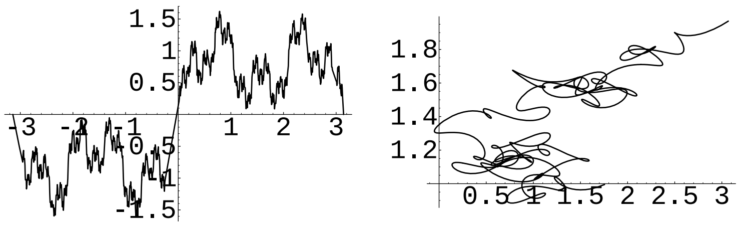
Manifolds and fractal functions

Fractal functions generalize the process by which the Weierstrass function is built. Fractal curves (or more generally fractal varieties) can be built by means of fractal functions, obtained by summing scale transformed versions of an initial function, typically a trigonometric function [10]. The command `FractalFunction` below computes the nth approximation of the theoretical version.

```
FractalFunction[ω_, δ_, n_][e_, t_] :=
 Sum[ScaleTransform[ω, δ, k][e, t], {k, 0, n}]
```

Fractal functions can be used to generate parametric approximations to fractal curves. Even when not useful for scientific purpose, they may have aesthetically pleasing properties.

```
Draw[{
  Manifold[
   {t, FractalFunction[3, 0.5, 7][Sin[t], t]}, {t, -π, π}],
  Manifold[FractalFunction[{3, 4}, {0.3, 0.5}, 5][
    {Cos[t / 9], Cos[t / 25]}, t], {t, 0, 2 π}]}]
```



```
- GraphicsArray -
```

Manifolds and iterated function systems

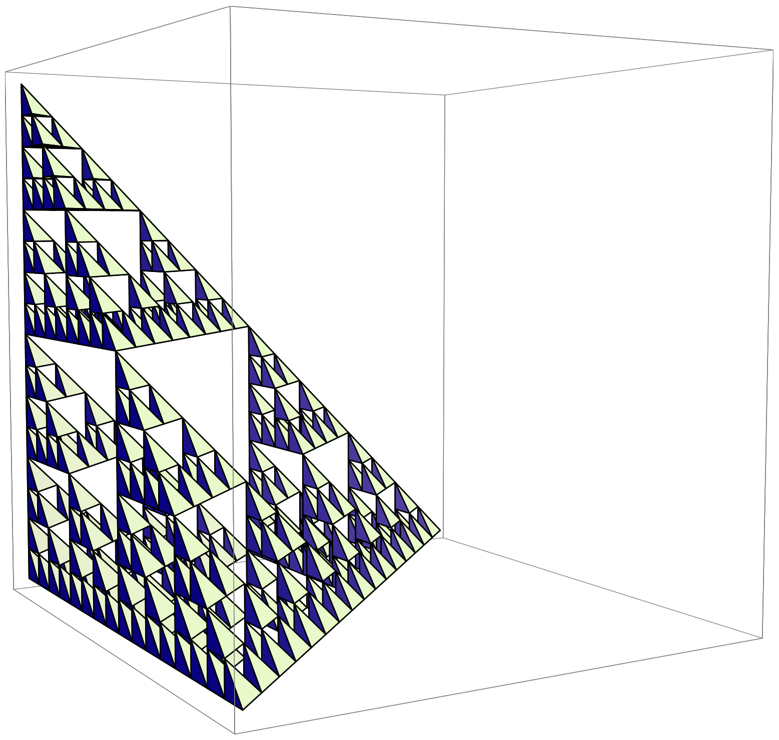
Iterated function systems (IFS) constitute another well known means to generate fractals [11]. Briefly, an iterated function system is a set of contracting affine transformations that are iteratively applied to any initial figure. The command `ManifoldIFS` uses the direct iteration [12] rather than its variant called "chaos game" [12, 13]: it computes the nth approximation of an IFS, from any initial manifold, in the form of a many–valued manifold.

```
OneStepIFS[ifs_][s_] := Flatten[Map[Through[ifs[#]] &, s], 1]
IterateIFS[ifs_, init_, n_] := Nest[OneStepIFS[ifs], {init}, n]
ManifoldIFS[ifs_, Manifold[p_, c___], n_] :=
 Manifold[IterateIFS[ifs, p, n], c]
```

Here is the example of a Sierpinsky sponge. Giving a low value to `PlotPoints` is necessary to avoid a lengthy computation. The process appears to be a very general one that actually builds a variety where even continuity properties are abandoned.

```
Sierpinsky3D[1][v_] := 0.5 v
Sierpinsky3D[2][v_] := 0.5 v + {0.5, 0, 0}
Sierpinsky3D[3][v_] := 0.5 v + {0, 0.5, 0}
Sierpinsky3D[4][v_] := 0.5 v + {0, 0, 0.5}
Sierpinsky3DIFS := Array[Sierpinsky3D, 4]
```

```
Draw[ManifoldIFS[Sierpinsky3DIFS,
  Manifold[{u, (1 - u) v, (1 - u) (1 - v) w}, {u, 0, 1},
   {v, 0, 1}, {w, 0, 1}], 4], PlotPoints → 2, Axes → None,
 BoxStyle → GrayLevel[0.5], ViewPoint -> {2.4, -1.4, 0.2}]
```



```
- Graphics3D -
```

The following of the paper describes a variety of tools that extend the game by enabling numerous manifold manipulations.

■ Shapes as Manifolds

Thanks to their graphic representations, manifolds can be used for shape description and visualization. As such, they constitute a means to introduce symbolic aspects in shape design. This relies upon a number of primitive manifolds (respectively shapes) such as cylinder, sphere or torus plus manifold (respectively shape) generators that compute boundaries, embeddings, connections, extrusions… or that operate by combining or transforming them.

□ Primitive Manifolds (respectively Shapes)

Primitive manifolds are defined in a way similar to graphic primitives, except that domains with coordinate symbols and bounds must be specified in the form of triples: symbol followed by two values that denote a coordinate ranging from a minimum value to a maximum value. When they are parametrized, subvalues are used for the definitions. For instance, in the definition of a parallelogram below, u and v denote two vectors while λ and μ denote the coordinates.

```
Parallelogram[{u_, v_}][dλ : {λ_, __}, dμ : {μ_, __}] :=
 Manifold[λ u + μ v, dλ, dμ]
```

As far as possible, the types remind of the corresponding geometric figures. Nevertheless, we are faced with two problems. Firstly, conflicts with the standard graphic primitives must be avoided, so `Line`, `Circle` and `Disc` were abandoned to the benefit of `Segment`, `Ring` or `Ellipse`. Then, the same words (ellipse, cylinder, cone…) denote both lines and surfaces (respectively surfaces and volumes). This polysemy of the natural language has polymorphic transformation rules as computational counterpart: according to the argument configuration, a single value denotes a fixed parameter while a triple with one symbol and two values denotes a parameter range.

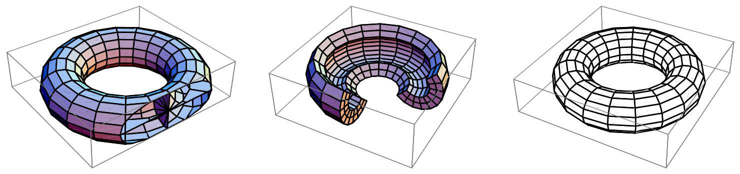
```
Cylinder[dr : {r_, __}, dθ : {θ_, __}, dh : {h_, __}] :=
 Manifold[{r Cos[θ], r Sin[θ], h}, dr, dθ, dh]
Cylinder[r_, dθ : {θ_, __}, dh : {h_, __}] :=
 Manifold[{r Cos[θ], r Sin[θ], h}, dθ, dh]

Cylinder[r0, {θ, 0, π}, {h, 0, 1}]

Manifold[{r0 Cos[θ], r0 Sin[θ], h}, {θ, 0, π}, {h, 0, 1}]
```

In some cases, n–dimensional generalizations are added to the set: they are preferable for programs that must be generic with respect to dimension. Objects are centered at the origin and have canonical (standard) orientations; if need be, they can be moved or deformed with affine or more general transformations that are introduced below.

☐ A Small Gallery of Manifolds (respectively Shapes)

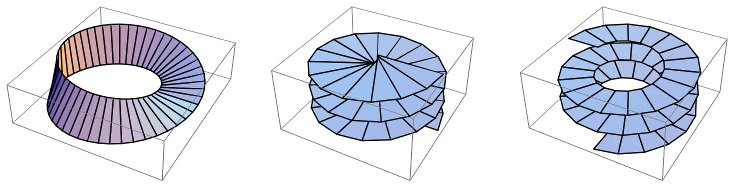
Together with the drawing command, this set of primitive manifolds can supersede the `Shapes`` package: here are a few examples drawn from the package. Other examples can be found in the literature, e.g., [7] or [14].

```
Draw[{
  DrawingStyle[Torus[3][{ρ, 0, 1}, {θ, 0, 2 π}, {φ, 0, 2 π}],
   PlotPoints → {2, 20, 15}, PlotRange →
    {{-4, 3}, {-4, 4}, {-1, 1}}, ViewPoint -> {2., -1.5, 1.7}],
  DrawingStyle[Torus[3][{ρ, 1, 2}, {θ, 0, 3 π / 2},
    {φ, -3 π / 4, π / 4}], PlotPoints → {3, 15, 10}],
  DrawingStyle[Torus[3][1, {θ, 0, 2 π}, {φ, 0, 2 π}],
   PlotPoints → {20, 15}, Shading → False]
 }, Axes → None, BoxStyle → GrayLevel[0.5]]
```



```
- GraphicsArray -
```

```
Draw[{
  MoebiusStrip[3][{ρ, -1, 1}, {θ, 0, 2π}],
  Helicoid[1/3][{r, 0, 1}, {θ, 0, 6π}],
  Helicoid[1/3][{r, 1/2, 1}, {θ, -π/2, 5π}]
 }, PlotPoints → {2, 50}, Axes → None, BoxStyle → GrayLevel[0.5]]
```



```
- GraphicsArray -
```

Operations on Manifolds

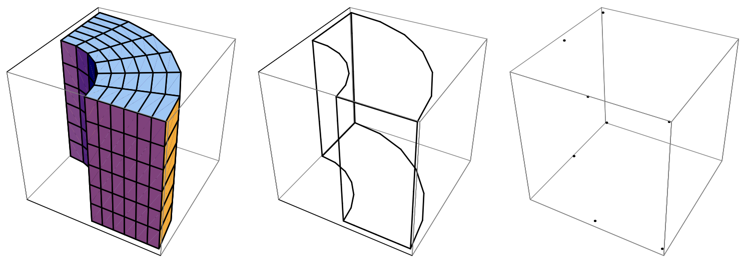
Boundary extraction

We actually define an iterated version of the boundary operator. In the 2–dimensional case, the first order boundary computes edges while the second order one computes vertices. In the 3–dimensional case, the first order boundary computes faces, the second order one edges and the third order one vertices; and so forth. This is a boundary from the domain viewpoint. It does not necessarily identify with the geometrical boundary when the codomain self overlap.

```
b0 = Manifold[{r Cos[t], r Sin[t], h},
   {r, 1, 2}, {t, 0, Pi/2}, {h, -1, 1}];
b1 = Boundary[b0]; b2 = Boundary[b0, 2];
b3 = Boundary[b0, 3]; Short[b2, 2]
```

```
Atlas[{Manifold[{1, 0, h}, {h, -1, 1}],
   <<10>>, Manifold[{0, r, 1}, {r, 1, 2}]}]
```

```
Draw[{b1, b2, b3}, PlotPoints → 7,
 Ticks -> None, BoxStyle → GrayLevel[0.5]]
```



```
- GraphicsArray -
```

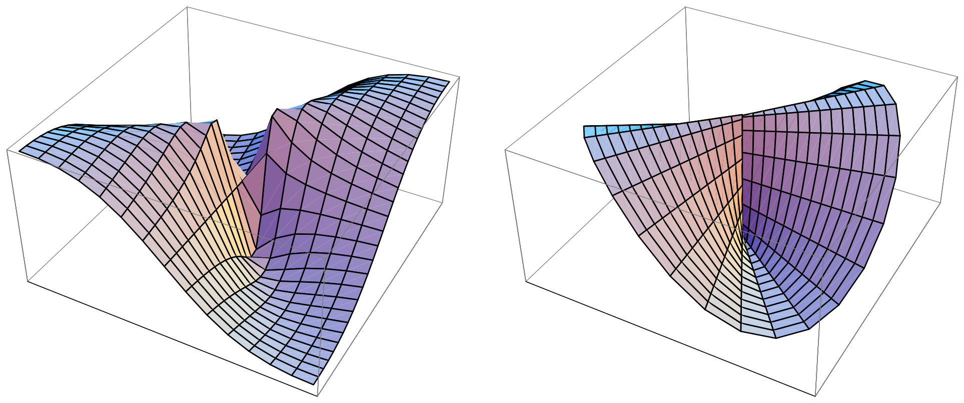
Manifold reshaping as right composition

A manifold reshaping consists of both a change of coordinates and a domain modification. It is expressed by means of a manifold passed as a second argument, assumed to be outlined on the first manifold. The operation amounts to testing the domain of the former and the codomain of the latter and then make a substitution. There are two more transformation rules for many–valued manifolds and atlases.

```
Reshape[m : Manifold[p_, oldc__], Manifold[f_, newc__]] /;
  Length[{oldc}] == Length[f] ^:=
 Manifold[p /. Thread[Rule[Coordinates[m], f]], newc]
```

Goetz and Wagon [15] used such changes of coordinates as a means to carry out adaptive surface plotting.

```
m1 = Manifold[{x, y, x y / (x^2 + y^2)}, {x, -1, 1}, {y, -1, 1}];
m2 = MapAt[Simplify, Reshape[
   Manifold[{x, y, x y / (x^2 + y^2)}, {x, -2, 2}, {y, -2, 2}],
   Manifold[{r Cos[θ], r Sin[θ]}, {r, 0, 1}, {θ, 0, 2 π}]], 1]

Manifold[{r Cos[θ], r Sin[θ], Cos[θ] Sin[θ]},
 {r, 0, 1}, {θ, 0, 2 π}]

Draw[{m1, m2}, Ticks -> None,
 BoxStyle → GrayLevel[0.5], PlotPoints -> {15, 30}]
```

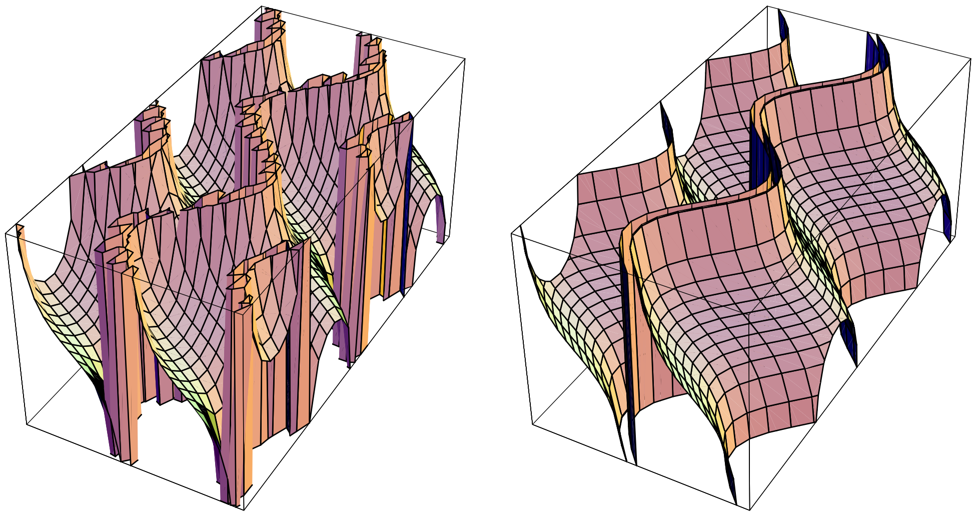


```
- GraphicsArray -
```

The following example is adapted from Kuzniarek [16].

```
m1 = Manifold[{x, y, Cot[x + Sin[y]]}, {x, -π, π}, {y, -2 π, 2 π}];
m2 = Reshape[
  Manifold[{x, y, Cot[x + Sin[y]]}, {x, -2 π, 2 π}, {y, -2 π, 2 π}],
  Manifold[{u - Sin[v], v}, {u, -π, π}, {v, -2 π, 2 π}]]

Manifold[{u - Sin[v], v, Cot[u]}, {u, -π, π}, {v, -2 π, 2 π}]
```

```
Draw[{m1, m2}, Ticks -> None,
 PlotRange -> {{-π, π}, {-2 π, 2 π}, {-3, 3}}]
```



```
- GraphicsArray -
```

Manifold connection as right composition

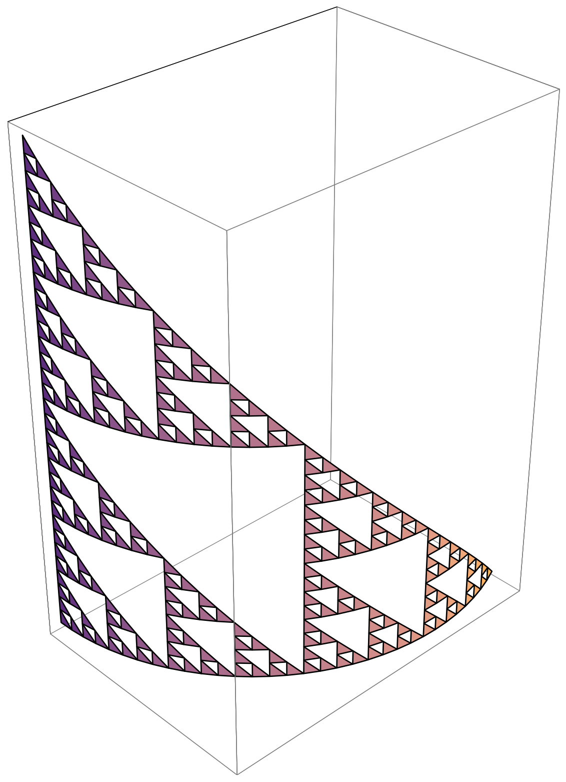
A manifold connection consists in the identification of the codomain of the former with the coordinate system of the latter, which is actually a right composition. It can be used to define a manifold (as a geometric domain or a figure) on some other carrier manifold (as a coordinate system). Manifolds can be connected provided the codomain of the first one and the domain of the second one have the same dimension, which may require a preliminary embedding.

```
Connect[Manifold[f_?VectorQ, d___], m : Manifold[p_, c___]] /;
  Length[f] == Length[Coordinates[m]] ^:=
 Manifold[p /. Thread[Rule[Coordinates[m], f]], d]
```

There are two more transformation rules for many−valued manifolds and atlases. Thanks to the connection of a many−valued manifold, a Sierpinsky triangle can be pasted onto a surface.

```
Sierpinsky2D[1][v_] := 0.5 v
Sierpinsky2D[2][v_] := 0.5 v + {0.5, 0}
Sierpinsky2D[3][v_] := 0.5 v + {0, 0.5}
Sierpinsky2DIFS := Array[Sierpinsky2D, 3]

cylindricalSierpinsky = Connect[ManifoldIFS[Sierpinsky2DIFS,
    Manifold[{u, (1 - u) v}, {u, 0, 1}, {v, 0, 1}], 5],
   Manifold[Cylindrical[{0.5, 2 θ - π / 2, h}], θ, h]];
```

```
Draw[cylindricalSierpinsky, Ticks → None, PlotPoints → 2,
 BoxStyle → GrayLevel[0.5], ViewPoint -> {2., -1.9, 1.7}]
```



```
- Graphics3D -
```

Transformations as left compositions

In a geometrical context where manifolds are intended to describe figures or shapes, transformations constitute a useful manifold generator. General (linear or non−linear) transformations are left compositions ; so in the context of Mathematica, they are basically defined as vector functions that apply to n−uples (lists) of coordinates, according to the informal scheme:

someTransform[{s_1 _, s_2 _...}] := {**expressions of the** s_i }

As a consequence, they do not directly apply to manifolds but require a construct like the following one, where `tr` denotes such a vector function.

```
Transform[Manifold[parametric_?VectorQ, domain___], tr_] :=
 Manifold[tr[parametric], domain]
```

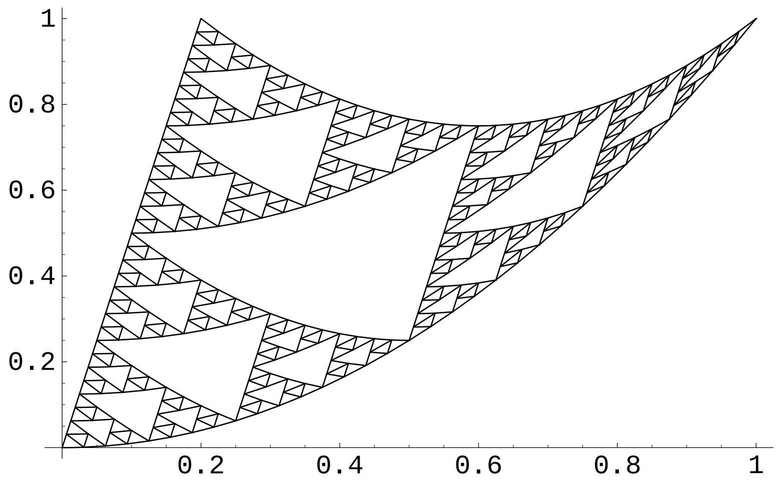
In the example, the pure function `{#[[1]]+#[[2]]/5, (#[[2]]+#[[1]]^2)}&` could be used in place of deform.

```
deform[{x_, y_}] := {x + y / 5, y + x^2}
Transform[Manifold[{x, y}, {x, 0, 1}, {y, 0, 1}], deform]
```

Manifold$\left[\left\{x+\frac{y}{5},\ x^2+y\right\},\ \{x,0,1\},\ \{y,0,1\}\right]$

The mechanism works with atlases and many−valued manifolds (the one in the example implementing a fractal variety) as well.

```
Draw[
 Transform[ManifoldIFS[Sierpinsky2DIFS, Manifold[{u, (1 - u) v},
    {u, 0, 1}, {v, 0, 1}], 5], deform], PlotPoints → 2]
```



```
- Graphics -
```

This general scheme is supplemented by a set of common transformations for use in geometry, shape design, CAD or engineering.

A collection of transformations

In order to facilitate a later symbolic treatment of transformations (composition and other operations), we focus on transformations as typed objects that can be combined according to algebraic or heuristic rules, i.e., a reified approach to transformations. So there are at least three levels: typed symbolic objects implementing transformations, their definitions as vector functions, plus their associated matrices for convenience. Once they are defined as vector functions, the transformations also apply to manifolds thanks to the scheme `Transform[manifold, transformation]`.

Transformations are firstly defined as typed objects, e.g., `Translation[v]` with v a list, so transformation rules can possibly be given for various transformation combinations. In view of their use as vector functions, transformation have an associated subvalue, according to the informal scheme `SomeTransform[params][v_]:=expression`. In the case of affine transformations, a complementary command defines the associated matrix. Between brackets, affine transformations also apply to graphic primitives (except occasionally circles and disks). Actually, for each affine transformation, there is a linear case and a strict affine case which is thought of as the linear version applied "about" some point. For instance here is the case of a scaling transformation, where k is either a scalar or a list.

```
Stretching[k_, pt_: 0][v_] := k * (v - pt) + pt

Transform[Manifold[{x, y}, {x, 0, 1}, {y, 0, 1}],
 Stretching[{a, b}, {-1, 1}]]

Manifold[{-1 + a (1 + x), 1 + b (-1 + y)}, {x, 0, 1}, {y, 0, 1}]
```

In the case of rotations, there are different means to specify a 3–dimensional rotation: Euler angles, nautical angles, axis and angle. These rotations can themselves be decomposed into precession, pitching, spin… That is why we introduce typed entities, so `Precession[`φ`]` denotes a precession of angle φ or `Euler[`φ, θ, ψ`]` denotes a rotation with the specified Euler angles. The type `Rotation` is restricted to the 2–dimensional case.

The spinning direction can be given either by a vector or by two spherical angles. Precession and winding are actually one and the same thing; so are nutation and rolling. Here is the rotation matrix associated with the given spinning parameters.

```
Matrix[Spinning[{0, 1, 0}, α]] // MatrixForm
```

$$\begin{pmatrix} \text{Cos}[\alpha] & 0 & -\text{Sin}[\alpha] \\ 0 & 1 & 0 \\ \text{Sin}[\alpha] & 0 & \text{Cos}[\alpha] \end{pmatrix}$$

Projections, shadowings and reflections are defined in a similar way. These annex definitions basically supersede and extend those found in the standard package `Geometry‘Rotations‘`.

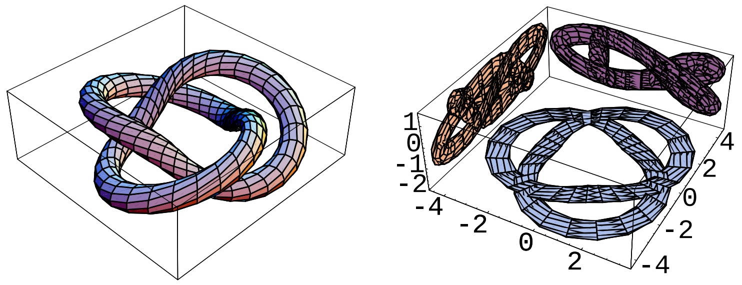
Applications to manifolds and shapes

Applying various transformations to standard manifolds constitutes a means to generate all kinds of shapes by deformation. In particular, `Projection` is more or less equivalent to `Project` from the package `Graphics‘Graphics3D‘` defined in the case of 3D graphics and projections onto coordinate planes. As an example, let us rewrite in terms of manifolds the function `TwistedTube` introduced by Edwards [17].

```
TwistedTube[Manifold[{x_, y_}, dt_],
  dθ : {θ_, θmin_: 0, θmax_: (2 π)}, r_, twist_, n_: 1] :=
 Manifold[{0, 0, y Cos[twist θ] + x Sin[twist θ]} +
   (r + x Cos[twist θ] - y Sin[twist θ]) {Cos[θ], Sin[θ], 0}, dt, dθ]
```

```
tube =
  TwistedTube[Manifold[{Cos[u] / 2, 1 + Sin[u] / 2}, {u, 0, 2 Pi}],
   {θ, 0, 4 π}, 3, 3 / 2];
```

```
Draw[{
  DrawingStyle[tube, Axes -> None,
   PlotPoints -> {12, 70}, ViewPoint -> {1.9, -1.9, 2.0}],
  DrawingStyle[Atlas[{
     Transform[tube,
      Projection[{{1, 0, 0}, {0, 1, 0}}, {0, 0, 1}, {0, 0, -2}]],
     Transform[tube, Projection[{{0, 1, 0}, {0, 0, 1}},
       {1, 0, 0}, {-4.5, 0, 0}]],
     Transform[tube, Projection[{{0, 0, 1}, {1, 0, 0}},
       {0, 1, 0}, {0, 4.5, 0}]]}], PlotPoints -> {12, 50}]}]
```



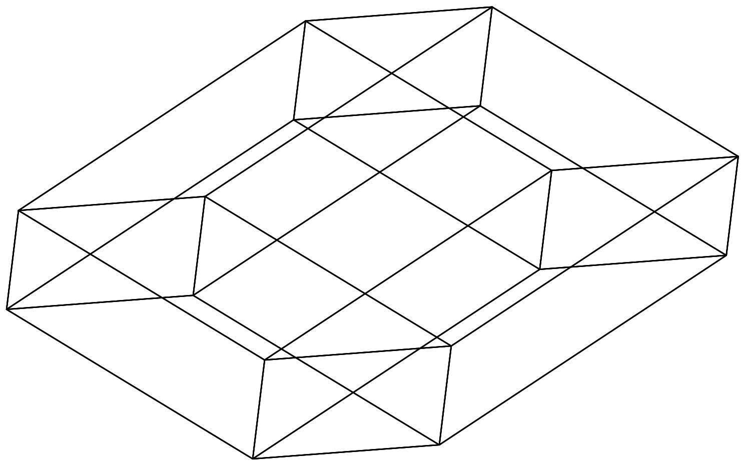
```
- GraphicsArray -
```

Applying a non–linear transformation to an affine object generally yields a curved object. This would not be possible with graphic primitives. With their analytical representations, manifolds constitute the natural primitives for non–linear operations, while built–in graphic primitives are more or less restricted to affine operations.

Applications to higher dimensional manifolds

There is an interesting use of transformations for the 2D or perspective 3D visualization of higher dimensional manifolds. As an example, here is a 2D projection of the edges of a 4–dimensional hypercube.

```
Draw[Boundary[Transform[UnitHyperCube[{a, b, c, d}],
   Shadowing[{{1.2, 1, 0.7, 0}, {0, -1.2, 1, 0.7}},
    {{1, 0, -0.7, 1.2}, {-1, 0.7, 0, 1.2}}]], 2],
 PlotPoints → 2, Axes → None]
```



```
- Graphics -
```

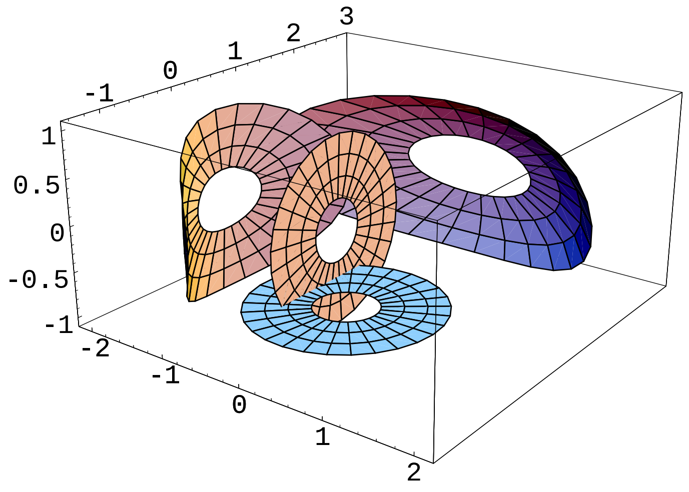
One can also compute a 3D projection which is then displayed with the Mathematica internal projection engine (section Higher dimensional manifolds).

□ Other Shape Generators

Embeddings

One sometimes needs to embed some manifold (respectively many–valued manifold or atlas) into a higher dimensional space. This is the role of `Embed`. A manifold can be embedded by connecting it to any coordinate subspace of a coordinate system with dimension greater than that of the embedded manifold. An embedding is determined by a list of coordinates that specify the orientation of the embedded manifold and a point where its origin is posted. Actually, a manifold can also be embedded by connecting it to any coordinate subspace of another manifold having a greater dimension.

```
e0 = Ring[{r, 1 / 3, 1}, {θ, 0, 2 π}];
e1 = Embed[e0, Cartesian[{x, y, z}], {x, y}, 1 / 2 {1, -1, -1}];
e2 = Embed[e0, Cartesian[{x, y, z}], {y, z}];
e3 = Embed[e0, Cylindrical[{r, θ, h}], {θ, h}, {3 / 2, π, 0}];
e4 = Embed[e0, Torus[1.5][{ρ, 0, 1}, {θ, 0, 2 π}, {φ, 0, 2 π}],
    {θ, φ}, {1, π / 2, 0}];
Draw[Atlas[{e1, e2, e3, e4}], PlotPoints → {4, 30},
 PlotRange → {{-2.2, 2.2}, {-1.5, 3}, {-1.1, 1.1}},
 ViewPoint -> {1.7, -2.3, 1.1}]
```

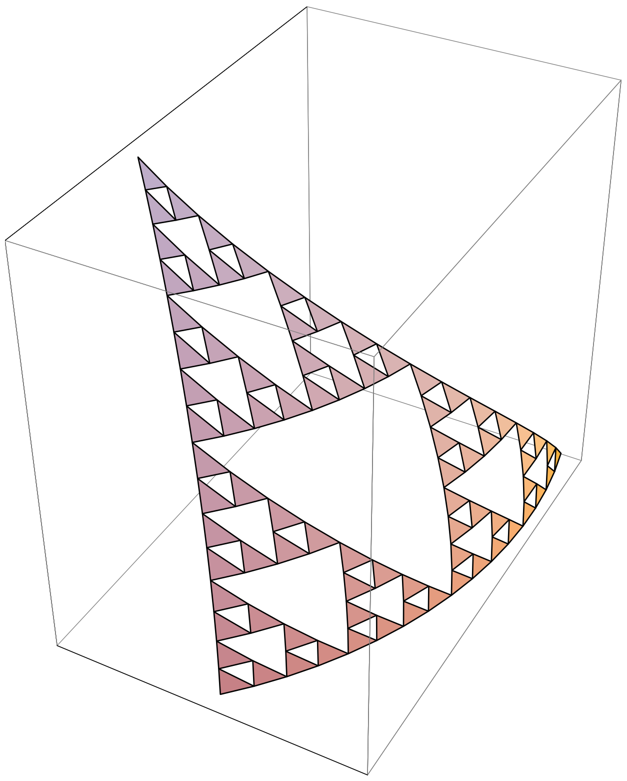


```
- Graphics3D -
```

To permute coordinates, one may play with the order of symbols in the third argument of `Embed`. Here is an embedding of a many−valued manifold implementing a fractal.

```
Embed[ManifoldIFS[Sierpinsky2DIFS,
   Manifold[{u, (1 - u) v}, {u, 0, 1}, {v, 0, 1}], 4],
  Spherical[{r, θ, φ}], {θ, φ}, {1, -π / 4, 0}];
```

```
Draw[%, PlotPoints → 2, Ticks → None, BoxStyle → GrayLevel[0.5]]
```



```
- Graphics3D -
```

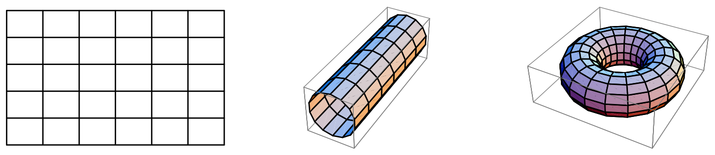
As a remark, a connection is a particular embedding where the dimensions of the embedded and embedding spaces are identical and the point is the origin.

```
carrier = Manifold[{u + v, u - v, u v}, {u, -1, 1}, {v, -1, 1}];
Connect[Manifold[{r Cos[t], r Sin[t]},
   {r, 1 / 3, 1}, {t, -3 Pi / 4, 3 Pi / 4}], carrier] ==
 Embed[Manifold[{r Cos[t], r Sin[t]}, {r, 1 / 3, 1},
   {t, -3 Pi / 4, 3 Pi / 4}], carrier, {u, v}, {0, 0}]

True
```

The embedding constitutes a powerful shape generators. In the following example, a torus is built by two successive embeddings: a rectangle is first embedded into a cylindrical coordinate system with a convenient axis, which yields a cylinder that in turn, is embedded into another cylindrical coordinate system with an orthogonal axis, which yields the torus.

```
c0 = Parallelogram[{{1, 0}, {0, 1}}][{u, 0, 2 π}, {v, 0, 2 π}];
c1 = MapAt[RotateLeft,
   Embed[c0, Cylindrical[{r, θ, z}], {θ, z}, {1, 0, 0}], 1];
c2 = Embed[c1, Cylindrical[{r, θ, z}], {r, θ, z}, {2, 0, 0}];
```

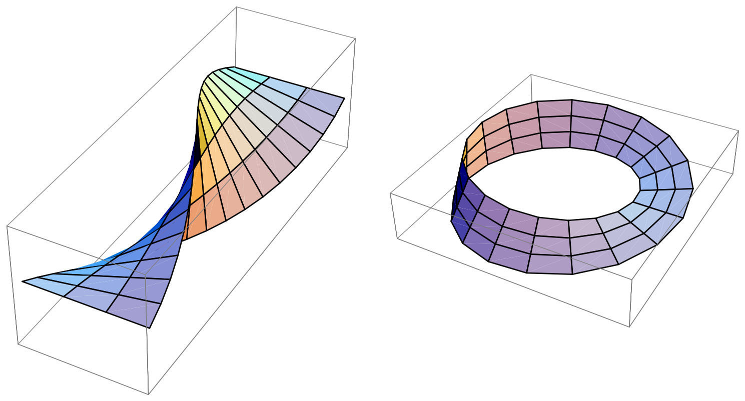
```
Draw[{
  DrawingStyle[c0, PlotPoints → {7, 6}],
  DrawingStyle[c1, PlotPoints → {15, 7}],
  DrawingStyle[c2, PlotPoints → {15, 20}]
 }, Axes → None, BoxStyle → GrayLevel[0.5]]
```



```
- GraphicsArray -
```

Similarly, a Moebius ribbon is built by embedding an helicoid into a cylindrical coordinate system.

```
h1 = Manifold[
    {2 + t Cos[ λ / 2], λ, t Sin[λ / 2]}, {t, -1, 1}, {λ, 0, 2 π}];
h2 = Embed[h1, Cylindrical[{r, θ, z}], {r, θ, z}, {2, 0, 0}];
Draw[{h1, h2}, PlotPoints → {4, 20},
 Axes -> None, BoxStyle → GrayLevel[0.5]]
```



```
- GraphicsArray -
```

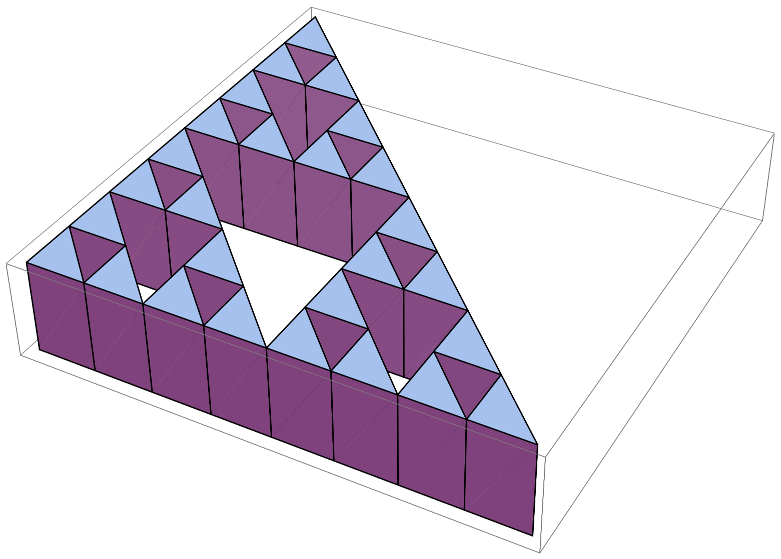
Extrusions

Extrusion is a feature commonly encountered in drawing or CAD applications. The analytical approach to manifolds lends itself well to an extension of the common extrusion process by means of parametrized transformations.

```
Extrude[Manifold[e_, d___], tr_, p__List] /;
  MemberQ[tr, Apply[Alternatives, Map[First, {p}]], {-1}] :=
 Manifold[tr[e], d, p]
```

The ordinary extrusion is no more no less than a single parameter transformation extrusion in the case of a translation. An embedding may be necessary to initiate the process. Many–valued manifolds as well as atlases can also be extruded.

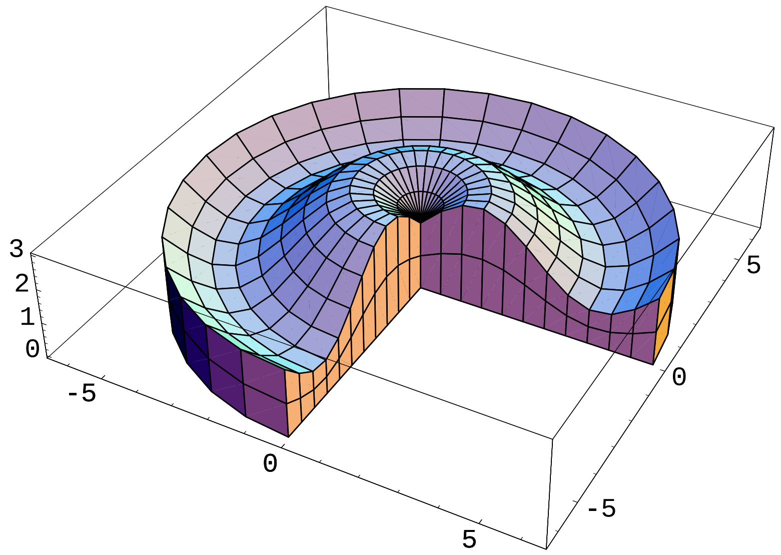
```
Draw[Extrude[
  Embed[ManifoldIFS[Sierpinsky2DIFS, Manifold[{u, (1 - u) v},
     {u, 0, 1}, {v, 0, 1}], 3], Cartesian[{x, y, z}], {x, y}],
  Translation[{0, 0, h}], {h, 0, 1 / 5}], PlotPoints -> 2,
 Axes -> None, BoxStyle → GrayLevel[0.5]]
```



```
- Graphics3D -
```

Ruled surfaces, cylindrical shapes, objects of revolution, conic objects… are obtained by specific extrusions that deserve specific operators named `Slide`, `Revolve`… `Revolve` more or less supersedes `SurfaceOfRevolution`, especially when used with the spinning transformation, whose first parameter corresponds to the revolution axis, and it naturally extends to solids of revolution.

```
Draw[
 Revolve[Manifold[{x, 0, z (2 + Sin[x])}, {x, 0, 2 π}, {z, 0, 1}],
  Spinning[{0, 0, 1}, θ], {θ, 0, 3 π / 2}],
 PlotPoints → {12, 3, 25}]
```

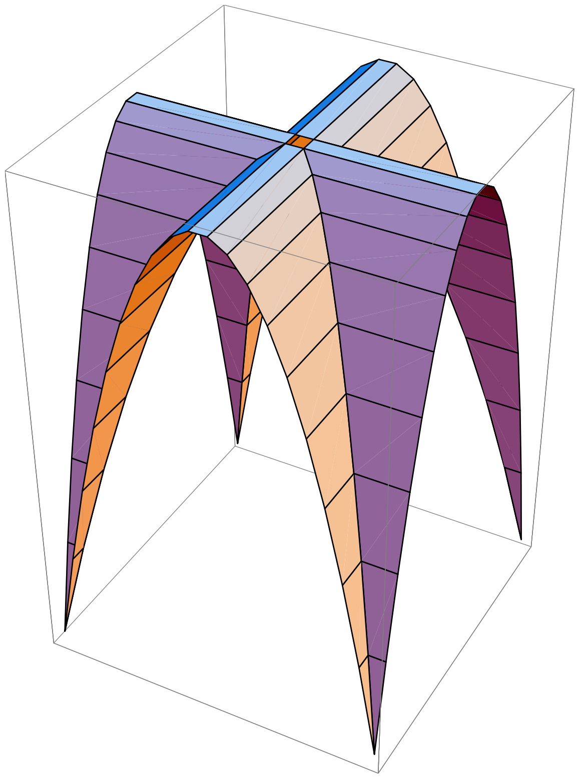


```
- Graphics3D -
```

Duplications

A duplication is an accumulation of transformed manifolds but it can also be viewed as a stepwise extrusion; so `Duplicate` has basically the same structure as `Extrude`. By default, it generates a many−valued manifold.

```
Draw[Duplicate[Manifold[{(1 - u) Abs[v] + u, v, 3 (1 - v^2)},
   {u, 0, 1}, {v, -1, 1}], Precession[θ], {θ, 0, 3 π / 2, π / 2}],
 PlotPoints -> {2, 20}, Axes → None, BoxStyle → GrayLevel[0.5]]
```



```
- Graphics3D -
```

Possible applications to the geometry of vaults or more general problems in architectural design are mentioned by Cerny [18].

Animations and ray tracing

`Movie` has basically the same structure as `Duplicate`, except it generates an animation by successively displaying the generated pictures. It is intended to supersede the various movie–plotting commands of the package `Graphics‘Animation‘`. It nevertheless reuses the basic command `Animate`.

```
Movie[Manifold[{u, Cos[v] Sin[u], Sin[v] Sin[u]},
   {u, -Pi, Pi}, {v, 0, 2 Pi}], Precession[θ],
  {θ, 0, π, π / 2}, Axes → None, Boxed → False];
```

Finally, a tool developed by Maeder to convert and export surface graphics data to ray tracing programs deserves to be mentioned [19].

Homotopy (morphing) and interpolation

Provided linear operations can be defined with respect to some type of objects, the homotopic transformation (1–k) O_1 + k O_2, k∈[0,1], defines all intermediaries between the objects O_1 and O_2, i.e., a linear interpolation between these two objects. In the case of manifolds, the idea can be generalized to non–linear interpolation, provided weighting functions are specified. It can also be generalized to polynomial interpolation.

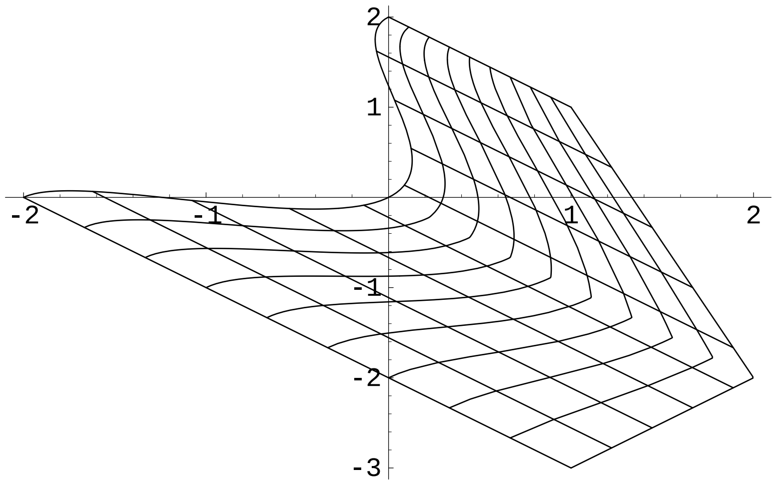
Homotopy is a powerful shape generator that produces n–dimensional manifolds by generating all the intermediaries between two n–1 dimensional manifolds (valid only for single−valued manifolds). If the domains are not identical, the manifolds must firstly be normalized.

In some cases, geometric domains are presented in the form of the so called cylindrical decompositions: v1min≤v1≤v1max, v2min[v1]≤v2≤v2max[v1], v3min[v1,v2]≤v3≤ v3max[v1,v2]… That is typically the way integration domains are specified. Homotopy then transforms those patches into manifolds. This idea was suggested by Tavouksoglou and Freed in [20, 21].

```
Manifold[
 Patch[{x, y, z}, {x, 0, 1}, {y, 0, 1 - x}, {z, 0, 1 - x - y}]]
```

```
Manifold[{x, (1 - x) y, (1 - x - (1 - x) y) z},
 {x, 0, 1}, {y, 0, 1}, {z, 0, 1}]
```

```
Draw[Manifold[Patch[{u + v, u - v},
   {u, -1, 1}, {v, u^4 - 2 u^2, 2 - (u + Abs[u])}]]]
```



```
- Graphics -
```

■ Future Directions

Thanks to their analytical potential, manifolds constitute a natural foundation for differential geometry, field theory and also some modelling applications. A data structure is introduced to describe scalar, vector or tensor fields (or any other sort of field) over manifolds. The principle of an extension of vector analysis to tensor analysis is described, that apply to manifolds as well as to coordinate systems. Finally, a link with a finite element package is presented.

□ Fields and Distribution Theory

Let us process fields as distributions over manifolds. For that purpose, we introduce the Dirac distribution over a manifold, which is defined by the rule [22, 23]:

$$\forall\varphi, < \delta_m , \varphi > = \int_m \varphi \, d m$$

with m some manifold, φ belonging to some functional space. When the domain and the codomain have the same dimension, the Dirac distribution is no more no less than the characteristic distribution of the manifold. The properties of the Dirac distribution over a manifold yield the following transformation rule (for experimental purpose, we use `Delta` rather than `DiracDelta`).

```
SimplifyDirac[alpha_ * Dirac[m : Manifold[p_, c___]][var_]] :=
 (alpha /. Thread[var → p]) * Dirac[m][var]
```

Then, one recovers the classical field computations in the form of manipulations with the Dirac distribution. Indeed, the distribution $\alpha\, \delta_m$ describes a field (or the restriction of a field) α over m, so the construct can be used as a data structure for fields. Then, many physical fields are expressed as the convolution product of some Green kernel by a Dirac delta distribution. This is the case of the electric potential resulting from a uniform density of charge over a circular domain.

```
GreenKernel[v_, ϵ_: ϵ] := 1 / Simplify[4 π ϵ Sqrt[v.v]]
Convolution[GreenKernel,
 Dirac[Manifold[R {Cos[θ], Sin[θ], 0}, {θ, 0, 2 π}]],
 {r, 0, h}, GenerateConditions -> False]
```

$$\frac{\text{EllipticK}\left[-\frac{4 r R}{h^2+(r-R)^2}\right]}{\pi \sqrt{h^2+(r-R)^2}\, \epsilon}$$

□ Differential Geometry

The basic ingredients of differential geometry are the tangent manifold (jacobian computation) and the metric. The Christoffel coefficients and other differential characteristics (curvature, torsion…) can be treated in a similar way.

Tangent manifold

One must distinguish the tangent manifold computed at some point, which is an (hyper)–plane, from the field of tangent manifolds, which can be viewed as a manifold over an abstract space with twice the initial dimension (the tangent bundle).

```
TangentManifold[m : Manifold[e_, __], pt_List] :=
 Manifold[(JacobianMatrix[e, Coordinates[m]] /.
     Thread[Rule[Coordinates[m], pt]]).Map[Dt, Coordinates[m]],
  Apply[Sequence, Map[Dt, Coordinates[m]]]]

TangentBundle[m : Manifold[e_, __]] :=
 Manifold[Dt[e], Join[Coordinates[m], Map[Dt, Coordinates[m]]]]

m = Manifold[{x^2 + y^2, -2 x y}, x, y];

TangentManifold[m, {-1, 1}]

Manifold[{-2 Dt[x] + 2 Dt[y], -2 Dt[x] + 2 Dt[y]}, Dt[x], Dt[y]]

TangentBundle[m]

Manifold[{2 x Dt[x] + 2 y Dt[y], -2 y Dt[x] - 2 x Dt[y]},
 {x, y, Dt[x], Dt[y]}]
```

Metric

When the codomain is an euclidean n–dimensional space with its canonical metric, this determines an induced metric on the manifold.

```
EuclideanMetric[m : Manifold[p_, c__]] :=
 With[{jac = Transpose[JacobianMatrix[p, Coordinates[m]]]},
  Simplify[Outer[Dot, jac, jac, 1]]]
```

```
Simplify[EuclideanMetric[Torus[R][r, {θ, 0, π}, {φ, 0, 2π}]]] //
 MatrixForm
```

$$\begin{pmatrix} (R + r\,\mathrm{Cos}[\varphi])^2 & 0 \\ 0 & r^2 \end{pmatrix}$$

When the codomain has no prior metric structure, an explicit metric can be given.

```
HalfPlanePoincareMetric[
  Manifold[{_, _}, {u_, -∞, +∞}, {v_, 0, ∞}]] :=
 λ^2 v^2 IdentityMatrix[2]
```

Vector and tensor analysis

In order to extend to manifolds the classical formulation of vector analysis in terms of coordinate systems, let us introduce the following polymorphic scheme, given in the case of `CovariantD` but valid for such operators as `Grad`, `Div`, `Curl` or `Laplacian`. Fields over manifolds are processed by means of the `Dirac` distribution introduced above. Actually, the extension to tensor analysis must take into account non orthogonal coordinate systems, so it requires the introduction of an operator computing the covariant derivative (`CovariantD`); for antisymmetric tensors, the exterior derivative (`ExteriorD`) should be introduced too.

```
SetAttributes[CovariantD, HoldRest]
CovariantD[scalar_, cs_[coord_]] := Map[D[scalar, #] &, coord]
CovariantD[scalar_ * DiracDelta[m_][v_]] /;
  Apply[And, Map[FreeQ[field, #] & , v]] :=
 Map[D[scalar, #] &, Coordinates[m]] * DiracDelta[m][v]
```

```
CovariantD[r Cos[θ], Polar[{r, θ}]]
```

```
{Cos[θ], -r Sin[θ]}
```

```
CovariantD[s[ρ, θ] * Dirac[
    Manifold[Polar[ρ, θ], {ρ, 1, r}, {θ, 0, 3π / 2}]][{x, y}]]
```

$$\left\{\mathrm{DiracDelta}\left[\mathrm{Manifold}\left[\mathrm{Polar}[\rho,\theta],\{\rho,1,r\},\left\{\theta,0,\frac{3\pi}{2}\right\}\right]\right][\{x,y\}]\;s^{(1,0)}[\rho,\theta],\; \mathrm{DiracDelta}\left[\mathrm{Manifold}\left[\mathrm{Polar}[\rho,\theta],\{\rho,1,r\},\left\{\theta,0,\frac{3\pi}{2}\right\}\right]\right][\{x,y\}]\;s^{(0,1)}[\rho,\theta]\right\}$$

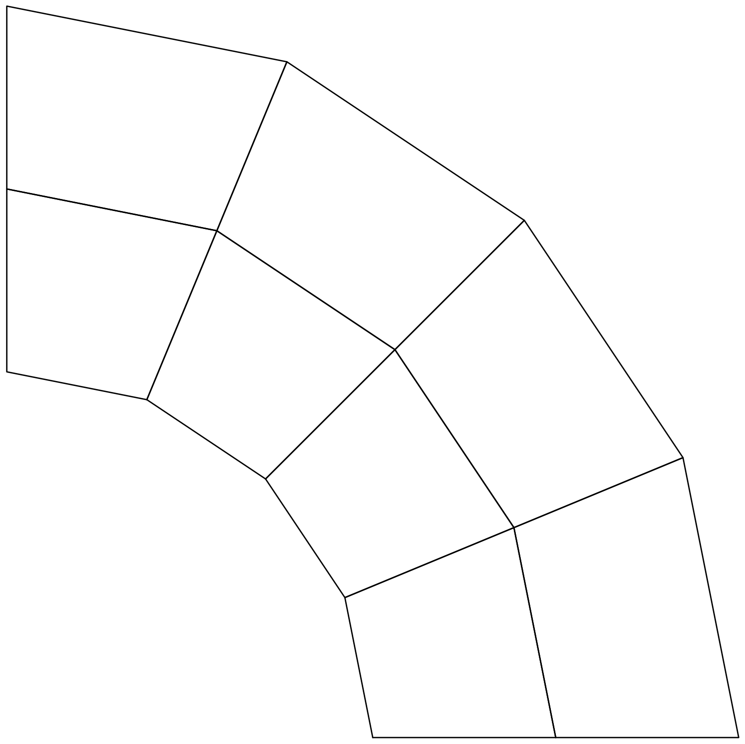
□ Applications to Mechanical Engineering

Beyond obvious applications to space or space–time modelling, manifolds have uses in mechanical engineering, especially by means of interactions with finite element packages. The details depend on the data structure used for finite elements. Here is an example with the “IMTEK Mathematica Supplement” package [24, 25].

```
Needs["Manifolds`Mesher`"]
m2D = Manifold[{r Cos[t], r Sin[t]}, {r, 1, 2}, {t, 0, π / 2}];
nexus2D = ToImsNexus[m2D, {r, 3}, {t, 5}]
```

```
- imsNexus -
```

```
Show[Graphics[imsDrawElements[nexus2D]],
 AspectRatio → Automatic]
```



```
- Graphics -
```

nexus2D is not only a mesh but a structure differentiating boundary nodes from interior nodes, able to take into account boundary values in view of further computations.

■ Discussion and Prospects

Founded on a reification of parametric representations, this computational approach to manifolds leads to a uniform treatment of questions that arise in differential geometry and field theory plus other domains such as shape design, fractal generation, scene description or mesh generation. In particular, it has the potential to supersede various graphic packages. Numerous operators generate more or less intricate or distorted manifolds by twisting and combining simpler ones.

As opposed to the functional approach of plotting commands that builds shapes by assembling low level graphic primitives, manifolds lead to a higher level description of shapes that can be defined by compact symbolic expressions, rather than huge assemblies of raw graphic primitives. This symbolic layer paves the way for a concise representation of many–level systems, then processed as wholes, such as spatial scenes, linkages or other assembled systems.

Nevertheless, the set of packages developed for that purpose is primarily intended for investigation about the feasibility of this project: testing remained minimal, there are no messages, exceptions have not been investigated thoroughly and the packages might have to be reorganized. The system of options associated with manifolds, atlases and the Draw command should be improved with a better filtering mechanism. The option specifications `DrawingStyle` and `DrawingOptions` are experimental features that should be tested thoroughly.

In the case of intricate systems, a mechanism for naming and retrieving the subexpressions describing the corresponding subsystems would be welcome. For instance, naming individual figures would occasionally be useful in the case of geometric scenes. Also, it would be interesting to enable links with geometry packages like Geometrica [26]

Parametric representations do not lend themselves well to the algebraic approach, from which a substantial part of the power of computer algebra derives, which weakens the idea of a full symbolic treatment of form. Because the parametrization of implicitly defined manifolds remains a basically unsolved problem (except in particular cases) [27], the relationship between parametric representations and implicit definitions (cartesian descriptions or inequalities) remains loose. Consequently, the analytical treatment of manifolds does not lend itself well to boolean operations, which require algebraic computations.

The `Dirac` distribution associated with manifolds has not been investigated thoroughly. In particular, its compatibility with the `DiracDelta` defined in the kernel should be analyzed. More generally, the recourse to the Dirac distribution to represent fields over manifolds is an experimental feature that should undergo further analysis.

Nevertheless, the major role of the analytical approach in physics gives importance to this way of representing and manipulating manifolds; so does its capability to blend and supplement a variety of tools, scattered about various packages.

■ Conclusion

Founded on reified parametrizations, this algorithmic approach to manifolds leads to a generic treatment of form modelling, encompassing shape design, differential geometry and field analysis, with direct applications to mechanical engineering. It has also the potential to supersede several packages. It introduces a unified viewpoint that not only gathers and supplements a variety of graphic tools, scattered about several packages, but also enables a symbolic approach to form, that encodes in a very concise way the various entities encountered in form modelling. As such, it constitutes a possible foundation for a computational morphology.

■ References

[1] X−S. Gao, D. Wang: Mathematics Mechanization and Applications, Academic Press, 2000.

[2] G. Lakoff, R. Nunez: Where Mathematics come from, Basic Books, 2000.

[3] E. A. Lord, C. B. Wilson: The mathematical description of shape and form, Ellis Horwood, 1986.

[4] R. Barrère, The Structuring Power of Mathematica in Mathematics and Mathematical Education, International Mathematica Symposium IMS’99, 1999 ; http://south.rotol.ramk.fi/keranen/IMS99/ims99papers/ims99papers.html.

[5] T. Wickham−Jones: Mathematica Graphics, Springer−Verlag, 1994.

[6] M. Trott: The Mathematica Guidebook for Graphics, Springer−Verlag, 2004.

[7] A. Gray: Modern Differential Geometry of Curves and Surfaces with Mathematica, CRC Press, 1998.

[8] J. W. Gray: Mastering Mathematica, Academic Press, 1994.

[9] R. Maeder: Computer Science with Mathematica, Cambridge, 2000.

[10] M. Attéia, J. Gaches: Approximation Hilbertienne, Editions de Physique et Presses Universitaires de Grenoble, 1999

[11] M. Barnsley: Fractals Everywhere, Academic Press, 1988.

[12] S. Wagon: *Mathematica* in Action, Freeman, 1991, 102−108.

[13] J. M. Gutiérrez, A. Iglesias, M. A. Rodriguez, V. J. Rodriguez: Fractal image generation using iterated function systems, International Mathematica Symposium IMS'95; in V. Keränen, P. Mitic: Mathematics with Vision, Computational Mechanics Publications, 1995, 175−182.

[14] D. von Seggern: CRC Standard Curves and Surfaces, CRC Press, 1993.

[15] R. Goetz, S. Wagon: Adaptive Surface Plotting: A Beginning, Mathematica in Education and Research, 5(3), 1996, 74−83.

[16] A. Kuzniarek: Making the Perfect Picture, The Mathematica Journal, 4(4), 54−60.

[17] C. H. Edwards: Twisted Tubes, The Mathematica Journal, 3(1), 10−13.

[18] J. Cerny: Geometry and Architecture, in Proceedings of the 9th European Seminar on Mathematics in Engineering Education, Arcada Polytechnic, 1998, 27−30.

[19] R. Maeder: Ray Tracing and Graphics Extensions, The Mathematica Journal, 4(3), 48−55.

[20] A. N. (Tom) Tavouktsoglou, B. Freed: Parametric Representations of Surfaces over Arbitrary Domains, Mathematica in Education and Research, 3(1), 1994, 20−23.

[21] A. N. (Tom) Tavouktsoglou, B. Freed: Drawing Mathematical Solids, Mathematica in Education and Research, 3(4), 1994, 22−24.

[22] L. Schwartz: Théorie des distributions, Hermann, 1966.

[23] L. Pinchard, Electromagnétisme classique et théorie des distributions, Ellipses, 1990.

[24] O. Rübenkönig, Z. Liu, J. Korvink: Integrated Engineering Development Environment, International Mathematica Symposium IMS'05, 2005, https://internationalmathematicasymposium.org/IMS2005/

[25] O.Rübenkönig, J. Korvink, "IMTEK *Mathematica* Supplement (IMS)", 2002−2005, IMTEK Mathematica Supplement (IMS).

[26] B. Autin: Geometry and reality in Geometrica05, Wolfram Technology Conference, 2005, http://library.wolfram.com/infocenter/Conferences/5846.

[27] D. Cox, J. Little, D. O'Shea: Ideals, Varieties and Algorithms, Springer, 1997.

■ Available Material

An experimental set of packages is being developed to investigate the ideas presented in the paper. It is available at http://macmaths.ens2m.fr/Mathematica/packages/Manifolds.zip (see also http://macmaths.ens2m.fr/Mathematica, menu "Packages").

■ Acknowledgments

On the occasion of academic projects, students have contributed to experiment with drafts of the material presented here. The author would like to thank them. A few reports of student projects are available at http://macmaths.ens2m.fr/students, menu “Published Projects”.

About the Author

Rémi Barrère received a French engineering degree in 1980 and a doctorate in mathematics in 1985 at the University of Franche–Comté (Besançon, France). He obtained a position at the University of Paris XIII and then at the University of Franche–Comté where he developed the use of computer algebra and symbolic programming techniques for mathematical modelling. He teaches applied mathematics and mathematical modelling, and experiments an integrated teaching of scientific computing in the form of student projects with *Mathematica*. He has been using this program since the early nineties in teaching as well as research, especially in the domains of symbolic approximations and foundational questions. He is author of a book about *Mathematica*.

Rémi Barrère
University of Franche−Comté, ENSMM
26 chemin de l’Epitaphe
F−25000 Besançon, France
rbarrere@ens2m.fr