1

An Object-Oriented Approach to Curves and Surfaces

Philipp Slusallek
Reinhard Klein
Andreas Kolb
Günther Greiner

Applications in computer graphics and geometric modeling generally require the integration of a variety of curve and surface types into a single system. Object-oriented design offers the opportunity to use the inherent hierarchical structure of curves and surfaces to solve this problem. This paper presents a top down approach to the design of an object-oriented framework for curves and surfaces together with its C++ implementation. We start from an abstract class of general differentiable curves and surfaces and in turn refine this design to various parametric representations of curves and surfaces. This design includes all of the standard curve and surface types and provides a powerful and uniform interface for applications. Examples from differential geometry, blending, and scattered data interpolation illustrate the approach.

1 Introduction

In this paper we present a top down approach to the design of an object-oriented framework for curves and surfaces together with its C++ implementation. We start from an abstract class of general differentiable curves and surfaces and in turn refine this design to various parametric representations of curves and surfaces [19, 25]. This design includes all of the standard curve and surface types, and provides a powerful and uniform interface for applications.

In Section 2 we present our approach to order the types of curves and surfaces into a hierarchical structure and review implementation features and selected curve and surface classes.

The main issue is to extract the operations that identify a certain class of curve and surface representations and set them apart from objects of other classes. This hierarchical structure serves as a reference for the derivation of a set of C++ classes implementing this hierarchy.

To an application programmer, this derived class hierarchy offers a unified view onto the various types of curves and surfaces representations. One only needs to know about the methods offered by the abstract classes and not about their internal implementation in derived classes.

Even more important in a research environment is our design decision to already supply most functionality at the abstract level by resorting to numerical techniques. As a result, a new curve or surface representation only requires to implement a method for point evaluation, and all other functionality is provided through numerical approximations in the abstract classes. Another benefit is the reuse of code in the class hierarchy, since similar functionality of certain curve and surface representations is already provided through common base classes (e.g. management of control points, operations on the parameter region, etc.).

The presented object-oriented framework is supposed to support a wide range of applications. Some examples are given in Section 3 which illustrate the power of this approach. These examples include visualization of differential geometry properties, the design of blending surfaces, and scattered data interpolation. In Section 4 the experience with this object-oriented approach is summarized and extensions and further research areas are discussed.

2 Design

We start with a general overview of curves and surfaces and explain how they can be grouped into a hierarchical scheme. We use this scheme as a guideline for the implementation of a set of abstract C++ classes. These abstract classes are used to derive the classes for concrete curve and surface types like a B-spline curve or a specific tensor-product patch.

2.1 Overview

An overview of our class hierarchy is shown in Figure 1. The important abstract classes together with some classes of special curve or surface types are given. Some less important classes have been removed from this figure to clarify the approach. Solid arrows mark derivations from super class to subclass. Dotted arrows mark classes that take a references to other classes, which either implement certain parts of this object (e.g. `ParameterRegion` for a surface) or which specify the class of objects that this class can operate on (e.g. `CompositeCurve` has `ParamCurves` as sub-curves).

The whole framework is implemented in C++ [5, 24], which enables an efficient and easy translation of our theoretical results to program code. At this point C++ is used almost exclusively for all projects within our group.

In the following subsections we present the more important classes of the hierarchical structure for curves and surfaces. For each class we describe the concepts implemented by this class and the set of methods providing this functionality.

Additionally, we identify those basic methods which must be implemented by all derived classes and those which may be implemented in the base class using the provided numerical methods.

2.2 Parameterized Curves

A parameterized curve C is a mapping of an interval I to R^3. This type of curve is so common in computer graphics that a class is certainly required. In our framework this class is called `ParamCurve`. The most fundamental methods for this type of curves are to obtain the parameter range I and to evaluate the curve at a given parameter $t \in I$ to derive a point $C(t)$ on the curve.

Applications in Computer Aided Geometric Design (CAGD) and other areas often require methods to obtain the derivatives $C^r(t)$, curvature, torsion, arc length or the Frenét Frame at a given parameter value t. These methods can be implemented numerically, using

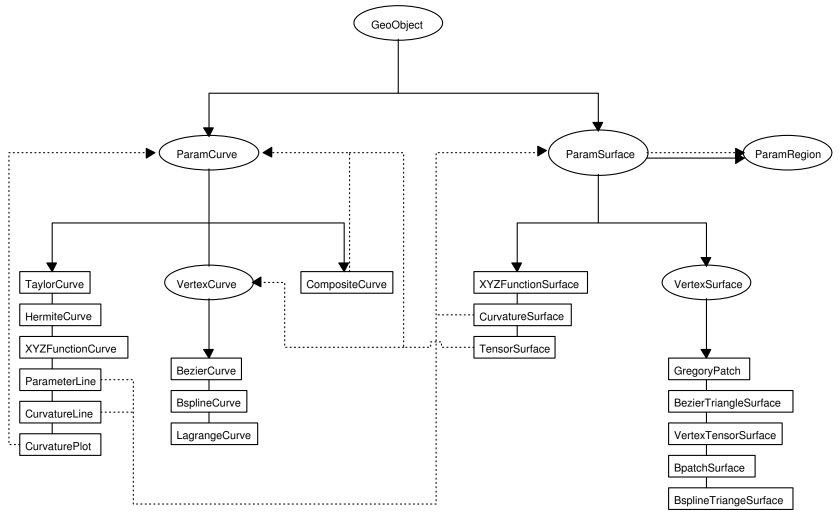

Figure 1: Schematic view of the class hierarchy. Ovals indicate abstract classes, rectangles actual implementations, solid arrows derivation in the class hierarchy, and dotted arrows references between related classes.

only the point evaluation method of the curve.

To offer all this functionality for any curve that at least knows how to evaluate a point $C(t)$, we have chosen to implement this functionality in the abstract class `ParamCurve`. At this level the methods use numerical approximation techniques, to obtain results for a general differentiable curve. If a subclass of a curve provides more accurate or faster algorithms to obtain these results, then the methods can always be overridden.

This implementation on the abstract level frees the programmer of a new curve class from the burden to implement all these probably difficult algorithms and instead rely on numerical approximation. Other algorithms can then be substituted at a later stage of the design process.

Since we want to visualize the curve, we need a way to output the curve to a graphics display. We have therefore implemented a method to generate a piecewise linear approximation. The accuracy, that should be met by the approximation, is specified by the user or the application program. This accuracy description is a separate class, with methods to query for criteria like 'flatness', number of segments, etc., whichever is more appropriate for the given curve or surface type. Again a default implementation is provided by the `ParamCurve` class.

2.3 Vertex Curves

In CAGD many of the standard curve schemes are based on geometric control points. The shape of the curve is then derived from these control points by approximation or interpolation techniques. This common property of many curve types motivates another abstract class derived from `ParamCurve`, called `VertexCurve`.

This class handles methods like management and user interaction of the control points already on the abstract level. Thus instantiations like Bézier- or Lagrange curves do not need to handle those operations explicitly. Of course, if there are special needs which are not covered by the abstract methods, they can be met by overriding the appropriate methods. Only the specific algorithm to calculate a point on the curve using the control points needs to be implemented for these derived classes.

The set of control points is implemented as an object of a separate class, which is then referenced in the curve classes. Many curves also need a set of knot values (e.g. B-spline-, Lagrange curves), which is implemented in the same way. Because the class of control vertices or knot values is itself derived from `ParamCurve`, it shares all its functionality. This includes output, user interaction, and subdivision.

The method to obtain a linear approximation of a curve is overridden for many vertex curves. Instead of using the point evaluation method to obtain enough points on the curve for an approximation, we use the more efficient (recursive) subdivision schemes, which are available for many types of vertex curves.

2.4 Meta Objects

The application programmer who wants to use this framework often has many additional methods which he would like to implement for all surfaces. Changing the abstract classes in the framework might not be the best way to do this, due to a probable inflation of methods. Instead we have chosen to implement this functionality using classes of *meta object* (not to be confused with other definitions associated with the term "meta" in object-oriented languages). Instead of having their own representation of geometry, they derive their geometry from objects of other classes.

There is a large set of classes for meta objects, which reference other curves or surfaces and visualize their properties. For instance the class `CurvaturePlot` is a planar curve that plots the curvature of another curve over its arc length. Since `CurvaturePlot` is itself derived from `ParamCurve` any method of this class also works on it. Thus a `CurvaturePlot` applied to a `CurvaturePlot` is easy.

Another example for meta objects is the following: We could have implemented a method that returns the derivative of the curve A over the whole parameter range as another curve object. Instead, we have used a meta object: A new meta object B (of class `Hodograph` in this case) is instantiated with a reference to the curve A. Whenever a point of this curve B is queried, it queries curve A for the derivative and returns it. Thus hodographs [7, 10] are available for any curve type. Other functionality like offset curves are implemented using the same technique.

An alternative design to meta objects would have been the use of action tables [23]. This design allows a more general extension of arbitrary classes. Since this is not required in our case, we found that our concept of meta objects provides so much flexibility and functionality – while being simple and quick to implement – that we have used it throughout the framework.

2.5 Parameterized Surfaces

Parameterized surfaces are a bit more difficult and interesting. They are mappings from a subset $D \subset R^2$ to R^3. They are implemented in the class `ParamSurface`. All methods relying on evaluating the surface at a single point are nearly identical to the curve methods, except that we now have to calculate partial derivatives, etc.. Problems arise when a non-local method needs to be applied to a surface (e.g. triangulation), since the method needs knowledge about the whole parameter region over which the surface is defined. This is more difficult than in the one dimensional case.

Again the most fundamental method is to obtain a point on the surface corresponding to a parameter value $u \in D$. As in the curve class all the other local methods are implemented at the abstract level in the class hierarchy as numerical approximations using point evaluation. These methods include calculation of partial derivatives and cross derivatives, normal vectors, Gaussian-, minimal and maximal curvatures, and the Fundamental Forms [4, 8] of the surface. This functionality is offered for any derived class, but can again be overridden, if better algorithms are available for a specific type of surface.

2.6 Vertex Surface

Similar to curves, a large group of surface types use geometric control points to specify the geometry of a surface, which is adequately reflected as a separate abstract class called `VertexSurface`. But as for the parameter region, management of control points is more difficult for surfaces than for curves, due to the non-linear arrangement of the control vertices. Common arrangements form regular rectangular or triangular meshes, but arbitrary triangulations are also available.

Our implementation of this concept in an abstract class offers only linear access to the set of control vertices. This normally suffices to implement user interaction and other general operations, as we can access any set of control points regardless of their topology. The classes for the other topologies are derived from this class and offer different access methods and storage implementations to efficiently implement special arrangements.

Special instantiations of these surface types are tensor-product surfaces, triangular Bézier patches [6], and multivariate B-splines [9]. Again, we benefit from our hierarchical class structure, since for a new derived class only the point evaluation method must be implemented. All other functionality is already provided by the base classes.

2.7 Parameter Region

Non-local operations require knowledge about the domain of the surface. The problem is that the domain region is not a one-dimensional interval, but a region in two dimensions that could be non-connected and bounded by free-form curves. This is often the case for trimmed free-form surface.

In order to solve this problem, the concept of a parameter region is implemented in an abstract class `ParamRegion`. This class can be queried for such information as point in region, the border as a set of bounding curves, a 2D-bounding box, etc..

The region can also return a tessellation of itself. The tessellation can either be a simpler representation (bounded by piecewise linear curves) or a set of triangles. This offers support for many standard algorithms. Note, that this tessellation method does not know about the surface but only operates on the parameter region. Similar meshing methods are implemented for surfaces, which then have access to the surface properties to obtain an adaptive tessellation, e.g. based on the surface curvature.

In this section we have outlined our framework for curves and surfaces, its most important concepts, and their implementation as C++ classes. From these many other classes have been derived which either implement special curve and surface representations or which - as meta objects - operate on other geometric objects. Several advanced algorithms for tesselating parameter regions and surfaces have also been implemented.

3 Applications

In this section we illustrate the benefits of this object-oriented framework for three different applications:

For the visualization of concepts from differential geometry we have implemented several classes of meta objects. They allow the calculation of curvature and torsion for a curve as well as mean and Gaussian curvature for a surface. This enables the user to create curvature plots, lines of curvature, etc., thus supplying tools for quality control of curves and surfaces.

The second application is the construction of blend-surfaces through variational design. In this case we use the framework to obtain informations about the boundary conditions for a new blending surface. Given the position and cross derivatives at the boundary, we generate a smooth blending surface. The tools developed for the first application can be used for visualizing the smoothness of the resulting surface and thus determine its quality.

The last application illustrates how the framework can help designing new surface schemes for scattered data interpolation. The problem is to find a smooth surface interpolating a given set of unorganized points. Again, the visualization of the smoothness of the interpolating surface is vital for the development of good algorithms.

The hierarchical structure of curve and surface representations in our framework allows us to apply nearly all operations of these applications to any curve or surface. As a result implementing the applications was greatly simplified, because the algorithms need not deal with many special cases, as they were already handled in the specific methods of the framework (e.g. derivatives near the boundary of a surface).

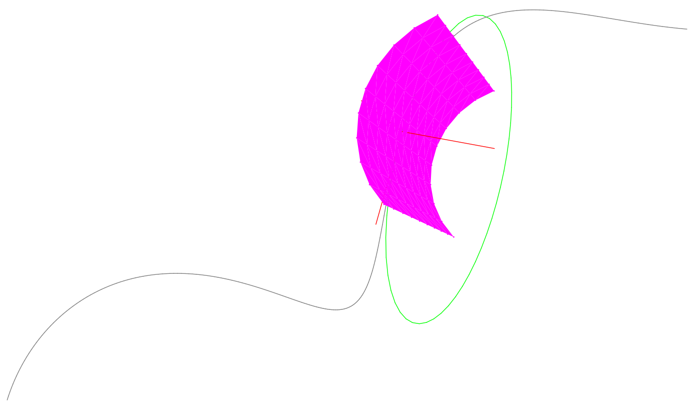

Figure 2: Visualizing differential geometry: A helix with a curvature circle, a Frenét frame, and a section of the torsion cylinder

3.1 Application 1: Visualizing Differential Geometry

Some properties of curves and surfaces can be visualized through appropriate geometric primitives. For instance, the curvature κ of a curve at a given point can be visualized by displaying a curvature circle, also called the osculating circle. This circle lies in the osculating plane spanned by the tangent t and the main normal n and has the radius $\frac{1}{\kappa}$.

In an similar way we visualize the torsion τ of a curve by a cylinder through the point on the surface and with axis parallel to the binormal and having radius $\frac{1}{\tau}$ (Figure 2). Animating the curvature circle, the torsion cylinder, and the Frenét frame along the curve results in a method for displaying their variation.

A simple and convenient method for displaying the variation of the scalar curvature and torsion values is by means of a color-coded map. Another method for displaying the curvature is through curvature plots. The curvature plot is a two dimensional graph which plots the curvature as a function of the arc length of the curve. For surfaces, sectional curvature may again be visualized through curvature circles.

The variation of the scalar-valued Gaussian- and mean curvature, as well as minimal and maximal curvature over the surface can be visualized by means of a color-coded map (Figure 3). There are several other elaborated techniques which produce good results [2, 8].

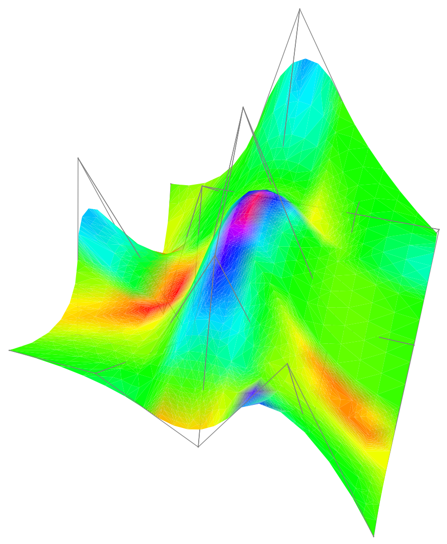

Figure 3: A B-spline surface with imposed color-coded Gauss-curvature

An informative method for analyzing the variation of the principal directions across the surface is to incorporate a family of lines of curvature [2] into the display. A line of curvature is a curve on the surface whose tangent direction at each point coincides with one of the principal directions (Figure 4).

The framework has proven to support this kind of application very well, because it is very simple to create various kinds of meta objects for visualizing different aspects of any of the supported surface representations.

3.2 Application 2: Constructing Blending Surfaces

Another application that uses this framework is the construction and visualization of blending surfaces. Given two *primary surfaces* the problem is to construct a smooth transitional surface. Such a surface is called a *blend surface*. Our method [14, 15, 16] is based on a variational principle or an optimization problem. These methods have become quite popular in recent years in different areas in computer graphics [21, 22, 26]. The main idea to construct the blend surface is as follows:

- It gives a smooth transition to the primary surface at the boundaries, and
- a *fairing functional*, which somehow measures total mean curvature, is minimized.

The boundary curves and the derivatives along those curves are given by special curve objects, which live on each of the primary surfaces. They describe the geometry of the

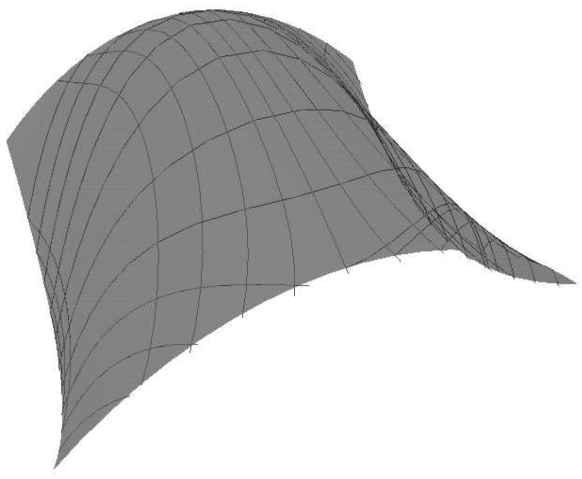

Figure 4: A Bézier surface with curvature lines visualizing the direction of minimum and maximum curvature at each point along the curve

problem completely and together with the functional ensure a unique solution to the blending problem.

So far this method has been implemented for tensor product B–spline surfaces (TPS). For two primary TPS and specified boundaries (which are B-spline curves), a TPS is constructed such that it meets the primary surfaces at the boundaries. In addition, the cross-boundary derivatives of blend surface and a primary surface coincide at the boundary where they meet. The fairing functional J which will be minimized is of the form [17]

$$J(F) = \int_\Omega \sum_{i\,\alpha\,\beta} w_{i\alpha\beta} \frac{\partial^\alpha S_i}{(\partial u)^\alpha} \frac{\partial^\beta S_i}{(\partial v)^\beta}$$

where α and β are multi-indices of order ≤ 2 , and S_i $(i = 1, 2, 3)$, denote x- y- and z component of the surface S. The weight functions $w_{i\alpha\beta}$ depend on the geometry of the region to be blended and are chosen via a parameter transformation between the parameter space of the TPS (rectangle) and a more natural parameter space. This has the effect, that the fairing functional J is a good approximation for the total mean curvature (in mean square sense). The details are given in [15].

The implementation of the blending operation relies on a set of classes that describe the boundary curves and the derivatives along those curves.

The boundary curves all lie on the blended surfaces. So they are implemented using curves that map a parameter interval to the two dimensional parameter region of the surface. This is the same technique which is used for trimming curves of parametric surfaces. This class `SurfaceCurve` for curves on surfaces offers additional methods to calculate derivatives of the surface along the curve. A `SurfaceCurve` object can be queried for a derivative and will return a new object of a class derived from `SurfaceCurve` called `SurfaceDeriv`. Evaluating a point on this curve results is the requested derivative.

Encapsulating the derivatives in another object allows us to trade accuracy for speed without changing any other algorithm in the framework: The class `SurfaceDeriv` can query the surface for derivatives at a few points and can then use interpolation to obtain intermediate results, which can result in large speedups. All this is invisible to the blending algorithm using this object.

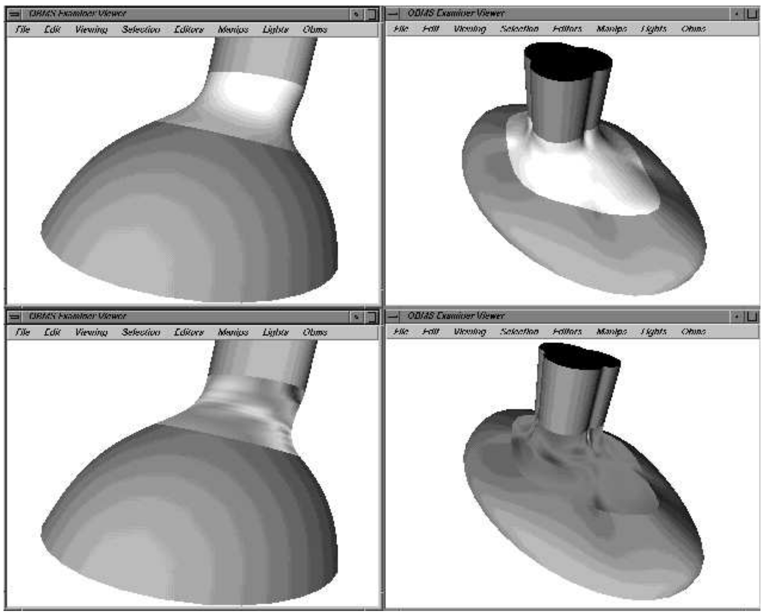

Figure 5: Two blend surfaces and their curvature plots

An example of a blending surface and its curvature distribution is given in (Figure 5). On the upper left side two primary surfaces (green) and the blend surface(red). On the upper right side the same setting with a curvature plot for Gaussian curvature for the blend surface. Below another setting of two primary and one blend surface. In this case the upper primary surface is an elliptical cylinder.

3.3 Application 3: Minimal Norm Network Interpolants

In various areas of research one is confronted with the problem of scattered data interpolation (to reconstruct a scalar valued function $F : \Omega \mapsto \mathrm{R}$, $\Omega \subseteq \mathrm{R}^2$, in two variables knowing its values only at a finite number of points $\mathbf{u}_i \in \Omega$, where the points $\mathbf{u}_i$ do not have any regular structure). It is generally difficult to ensure a priori that an interpolation scheme produces a surfaces with good overall shape. Therefore curvature plots are an indispensable tool for judging the overall quality of the resulting shape [18].

One approach to solve this problem is the so-called *Minimal Norm Network method* (MNN). This method has several advantages: it can be used on non-convex domains Ω and the shape of the interpolant can be controlled in a predictable manner.

The MNN method for constructing an interpolating function consists of three steps (for more details see [12, 20]):

1. The data points are used to construct a *triangulation* of the domain Ω. Since thin triangles are not desired for numerical reasons, we usually take the well known *Delaunay* triangulation [11].

2. A *curve network* whose domain is the union of all edges of the triangulation is uniquely defined as follows: The single curves must meet with a certain degree of continuity. Furthermore, we choose among all these network functions those whose norm with respect to a given functional is minimal. The functionals are usually of

the type:

$$\sigma(S) = \sum_{e \in T} \int_e \left\{ \left(\frac{\partial^2 F}{\partial e^2} \right)^2 + \alpha^2 \left(\frac{\partial F}{\partial e} \right)^2 \right\} de$$

or

$$\sigma(S) = \sum_{e \in T} \int_e \left\{ \left(\frac{\partial^3 F}{\partial e^3} \right)^2 + \gamma^2 \left(\frac{\partial^2 F}{\partial e^2} \right)^2 \right\} de,$$

where the triangulation T is defined as a set of edges $\{e\}$.

3. The network is extended to the interior of each triangle by means of a *triangular interpolant.* For each vertex in the considered triangle, one patch is defined by interpolating the function values and derivatives in this vertex and along its opposite edge. The final triangular interpolant is obtained by a convex combination of these three patches.

A first preliminary implementation of this scheme showed quite good results as far as mean and maximum errors were concerned. However, after the scheme was introduced into this framework and analyzed using the tools for visualizing the curvature of the interpolating surfaces, the quality of curvature plots turned out to be quite poor. Thus, the visualization of curvature inspired us to do further investigation. This finally led to an improved MNN-interpolation scheme [20] with significantly better curvature distribution (Figure 6).

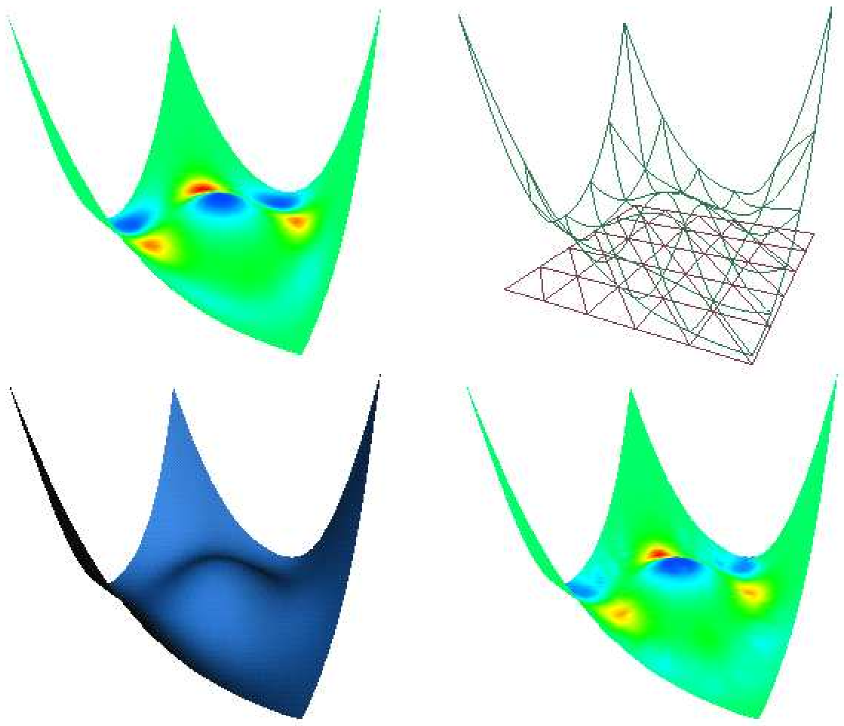

Figure 6: A curvature plot the original surface, the curve network, the MNN interpolant, and its curvature plot

4 Conclusion and Further Work

We have presented a top-down approach to the design of an object-oriented framework for parametric curves and surfaces. Only the most fundamental method, point evaluation, must be supplied in order to integrate a new type of curve or surface into the scheme. All other methods are already implemented in abstract base classes. Thus the programmer is free to experiment without worrying about details such as implementing derivatives or similar operations, but still has the ability to use better methods as they become available.

A complete set of methods for curve and surface design and analysis with support for blending, scattered data interpolation, differential geometry, tessellation, and display is provided.

The support for surface manipulation based on differential geometry or other local operations which do not directly work on control vertices but on the surface itself, have not been studied [13]. This is certainly a very interesting research area, but it is yet unclear if and how these operators can be applied to arbitrary abstract surface classes.

References

[1] R.H. Bartels, J.C. Beatty, and B.A. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modelling.* Morgan Kaufman Publisher, 1987.

[2] J. Beck, R. Farouki, and J. Hinds. Surface Analysis Methods. *Computer Graphics & Applications*, 6(12):18–38, December 1986.

[3] J. Bloomenthal. Polygonization fo implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[4] M. P. do Carmo. *Differential Geometry of Curves and Surfaces.* Prentice Hall, Englewood Cliffs, N.J., 1976.

[5] Margaret A. Ellis and Bjarne Stroustrup. *The Annotated C++ Reference Manual.* Addison Wesley, 1991.

[6] Gerald E. Farin. Triangular Bernstein–Bézier patches. *Computer Aided Geometric Design*, 3:pp. 83–127, 1986.

[7] Gerald E. Farin. *Curves and Surfaces in Computer Aided Geometric Design.* Academic Press, 1988.

[8] R. T. Farouki. Graphical Methods for Surface Differential Geometry. In R. R. Martin, editor, *The Mathematics of Surfaces II*, pages 363–385. Oxford Science Publications, Oxford, 1987.

[9] Philip Fong and Hans-Peter Seidel. Control Points for Multivariate B-spline Surfaces over Arbitrary Triangulations. *Computer Graphics Forum*, 10:309–317, 1991.

[10] A. R. Forrest. Interactive Interpolation and Approximation by Bézier Polynomials. *The Computer Journal*, 15:pp. 71–79, 1972.

[11] S. Fortune. Voronoi Diagrams and Delaunay Triangulations. In D. Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*, pages 193–223. World Scientific Publ., 1992.

[12] R. Franke and G. Nielson. Scattered Data Interpolation: A Tutorial and Survey. In H. Hagen and D. Roller, editors, *Geometric Modelling: Methods and Applications*, pages 131–160, New York, 1991. Springer Verlag.

[13] Priamos N. Georgiades and Donald P. Greenberg. Locally Manipulating the Geometry of Curved Surfaces. *IEEE Computer Graphics & Applications*, 12(1):54–64, January 1992.

[14] G. Greiner. Blending Techniques Based on Variational Principles. In J. Warren, editor, *Curves and Surfaces in Computer Vision and Graphics III*, Proc. SPIE 1830, pages 174–184. SPIE, 1992.

[15] G. Greiner. Surface Contructions Based on Variational Principles. In P.-J. Laurent, A. Le Méhauté, and L. L. Schumaker, editors, *Curves and Surfaces II*, Chamonix, 1993.

[16] G. Greiner and H.-P. Seidel. Curvature Continuous Blend Surfaces. In B. Falcidieno and T. L. Kunii, editors, *Modelling in Computer Graphics*, pages 309–317. Springer Verlag, 1993.

[17] M. Kallay. Constrained optimization in surface design. In B. Falcidieno and T. L. Kunii, editors, *Modeling in Computer Graphics*, pages 85–94. Springer Verlag, 1993.

[18] R. Klass. Correction of Local Surface Irregularities Using Reflection Lines. *Computer Aided Design*, 12(2):73–76, February 1980.

[19] R. Klein and Ph. Slusallek. An Object-Oriented Framework for Curves and Surfaces. In J. Warren, editor, *Curves and Surfaces in Computer Vision and Graphics III*, Proc. SPIE 1830, pages 284–295. SPIE, 1992.

[20] Andreas Kolb. Interpolationg Scattered Data with C^2 surfaces. Technical report, Universität Erlangen, 1993.

[21] M. Lounsbery, S. Mann, and S. and T. deRose. Parametric Surface Interpolation. *Computer Graphics & Applications*, 9:97–115, 1992.

[22] H. P. Moreton and C. H. Séquin. Functional Optimization for Fair Surface Design. In E. E. Catmull, editor, *Computer Graphics*, pages 167–176. ACM Siggraph, ACM Press, 1992.

[23] Paul S. Strauss and Rikk Carey. An Object-Oriented 3D Graphics Toolkit. *Computer Graphics*, 26(1):341–352, July 1992.

[24] Bjarne Stroustrup. *The C++ Programming Language.* Addison Wesley, 2. edition, 1991.

[25] Allan H. Vermeulen and Richard H. Bartels. C++ Spline Classes for Prototyping. In *Curves and Surfaces in Computer Graphics II*, pages 121–131. SPIE, 1991.

[26] William Welch and Andrew Witkin. Variational Surface Modeling. In Edwin E Catmull, editor, *Computer Graphics*, pages 157–166. ACM Siggraph, ACM Press, 1992.