An Object-Oriented Framework for Curves and Surfaces with Applications

Philipp Slusallek, Reinhard Klein, Andreas Kolb, and Günther Greiner

Abstract. In computer graphics and geometric modeling one generally faces the problem to integrate a variety of curve and surface types into a single program. Object-oriented design offers the opportunity to use the inherent hierarchical structure of curves and surfaces to solve this problem. This paper presents an object-oriented framework together with its C++ implementation that starts from an abstract class of general differentiable curves and surfaces and in turn refines 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, and blending illustrate the approach.

§1. Introduction

In this paper we present an object-oriented framework together with its C++ implementation that starts from an abstract class of general differentiable curves and surfaces and in turn refines this design to various parametric representations of curves and surfaces [19, 26]. This design includes all of the standard curve and surface types and provides a powerful and uniform interface for applications.

In Section 2 after a short introduction into object-oriented design and classes 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.

Curves and Surfaces II 1
P. J. Laurent, A. Le Méhauté, and L. L. Schumaker (eds.), pp. 1–4.

ISBN 0-12-XXXX.

The main issue will be to extract the operations that identify a certain class of objects and set them apart from objects of other classes. The hierarchical structure will serve as a reference for the derivation of the set of C++ classes, which implement this hierarchy. Here, we assume that the reader is familiar with object-oriented design and C++ in general.

The presented object-oriented framework is supposed to support a wide range of application programs. Some examples are given in Section 3 which illustrates the power of this approach. These examples include visualization of differential geometry properties and the design of blending surfaces.

Section 4 summarizes the achievements of this object-oriented approach and discusses extensions and further research areas.

§2. Design

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

In this paper we restrict ourselves to curves and surfaces in R^3, although most of the presented design also applies to higher dimensions.

Object-Oriented Design and Classes

In object-oriented design the main issue is to identify the operations that can be applied to a certain class of objects. These operations, often called methods, describe objects of a class completely, when seen from its environment – the internal structure is hidden from the user of an object (encapsulation). A class can be derived from another class, by which it inherits all the methods from its superclass (inheritance, code sharing). An object of a derived class can be substituted for one of its super classes and respond to the same set of methods. In this case derived classes can use a different algorithm to implement the same method (polymorphism, virtual methods in C++).

The following section describes our approach to apply these design methods to curves and surfaces. We start with a coarse description of the main curve classes. The description for curves then easily carries over to surfaces.

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 superclass 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).

Figure 1. Schematic view of the class hierarchy.

The whole framework is implemented in C++ [5, 25], which enables an efficient and simple translation of the theoretical results to program code. At this point C++ is used almost exclusively for all projects within our group.

Parameterized Curves

A parameterized curve is a mapping of an interval I to R^3. This type of curves is so common that a class is certainly required, which we 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 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 for 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 better or faster algorithms to obtain these results,

then these methods can always be overwritten.

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

Since we also 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.

Vertex Curves

In CAGD many of the standard curve schemes are based on geometric control vertices. 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, but could still do so by overwriting certain methods, if they have special needs that are not covered by the abstract methods. Only the specific algorithm to calculate a point on the curve using the control points needs to be implemented for these derived classes.

Meta Classes

The application programmer that will use this framework often has many additional methods that he would like to implement for all surfaces. Changing 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 *meta classes*, which operate on objects of other classes.

There is a large set of meta classes, which do not define any actual curves or surfaces by themselves, but reference other curves or surfaces and visualize their properties. For instance the class `CurvaturePlot` is a planar curve that plots the curvature of the associated curve over its arc length. Since `CurvaturePlot` is itself a `ParamCurve` any method of this class also works on it. Thus a `CurvaturePlot` applied to a `CurvaturePlot` is simple.

We found that this concept of meta classes provides so much flexibility and functionality – while being simple and quick to implement – that we have used it throughout the framework.

Parameterized Surfaces

Parameterized surfaces are a bit more difficult and interesting. They are a mapping of a subset $D \subset R^2$ to R^3 and are implemented in a class called `ParamSurface`. All methods that evaluate the surface at a single point, or near a single point for a numerical approximation, 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, 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 given its parameter $u \in D$. 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 overwritten, if better algorithms are available for a specific derived class.

Vertex Surface

Again a large set 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 with the parameter region, this is more difficult for surfaces than for curves, due to the non-linear topology of these control vertices. Common topologies are rectangular, triangular, or regular triangulations.

Our implementation of this abstract class offers only linear access to the set of control vertices. This normally suffices to implement user interaction and other general operations, and 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 other 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 [10].

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 even be non-connected as is often the case for a trimmed surface. To make matters worse the region is often quite complicated and bounded by free-form curves.

In order to solve this problem, we have developed an abstract description of the parameter region of a surface. 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

the region can be tessellated into a set of triangles which offers support for many standard algorithms. Note that this tessellation does not operate on the surface but only on the parameter region. Similarly 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.

§4. Applications

In this section we illustrate the benefits of the above object-oriented approach by applying our framework to the visualization of differential geometry and to the construction of blend-surfaces.

Application 1: Visualizing Differential Geometry

The above framework and the available methods to obtain derivatives can be used to visualize differential geometry properties of curves and surfaces.

The curvature κ of a curve at a given point can be visualized by displaying the 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 analog 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 $\frac{1}{\tau}$. Animating the curvature circle, the torsion cylinder or 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 over the arc length.

For surfaces, sectional curvature is visualized through curvature circles. In order to visualize the variation of the curvature as a function of direction we can also animate these circles.

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

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.

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 [15, 16, 17] 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 [22, 23, 27]. The main idea to construct the blend surface is as follows:

1. It gives a smooth transition to the primary surface at the boundaries,
2. 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, that live on each of the blended surfaces. They describe the geometry of the 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

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

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 [17].

Since the fairing functional is quadratic, the problem reduces to a least square problem for the (inner) control points of the blend surface. Thus the control points of the blend surface are obtained as the solution of a linear system. An example of a blending surface and its curvature distribution is given in Figure 2.

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 that is used for trimming curves of parametric surfaces. This class `SurfaceCurve` of curves on surfaces has 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. But all this is invisible to the blending algorithm using this object.

Figure 2. A blending surface and its curvature plot

§5. Conclusion and Further Work

We have presented an object-oriented framework for applications that work on parametric curves and surfaces. Only the most fundamental method, point evaluation, must be supplied in order to integrate a general new curve or surface into the scheme. All the 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 analysis with support for blending of arbitrary surfaces, differential geometry, scattered data interpolation, tessellation, display, and user interaction is provided.

The support for surface manipulation based on differential geometry or other local operations that do not directly work on control vertices but on the surface itself, have not been studied. 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. Bartels, R. H., J. C. Beatty, and B. A. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modelling*, Morgan Kaufman Publisher, 1987.
2. Beck, J., R. Farouki, and J. Hinds, Surface analysis methods, Computer Graphics & Applications **12** (1986), 18–38.
3. Bloomenthal, J., Polygonization for implicit surfaces, Computer Aided Geometric Design **5** (1988), 341–355.
4. do Carmo, M. P., *Differential Geometry of Curves and Surfaces*, Prentice Hall, Englewood Cliffs, N.J., 1976.
5. Ellis M. A., and B. Stroustrup, *The Annotated C++ Reference Manual*, Addison Wesley, 1990.
6. Farin, G. E., Triangular Bernstein–Bézier patches, Computer Aided Geometric Design **3** (1986), 83–127.
7. Farin, G. E., *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, New York, 2. edition, 1990.
8. Farouki, R. T., Graphical methods for surface differential geometry, in *The Mathematics of Surfaces II*, Martin, R. R. (ed.), Oxford Science Publications, Oxford, 1987, 363–385.
9. A. Forrest. Interactive interpolation and approximation by Bézier polynomials, Computer Journal **15** (1972), 71–79.
10. Fong, Ph., and H.-P. Seidel, An implementation of triangular B-spline surfaces over arbitrary triangulations Computer Aided Geometric Design **10** (1993), 267–275.

11. Fortune, S., Voronoi Diagrams and Delaunay triangulations, in *Computing in Euclidean Geometry*, D. Z. Du, F.Hwang (eds.), World Scientific Publ., 1992, 193–223.
12. Franke, R., and G. Nielson, Scattered data interpolation: A tutorial and survey, in *Geometric Modelling: Methods and Applications*, H. Hagen, D. Roller (eds.), Springer Verlag, New York, 1991, 131–160.
13. Georgiades, P. N., and D. P. Greenberg, Locally manipulating the geometry of curved surfaces. Computer Graphics & Applications **1** (1992), 54–64.
14. Gregory, J., Smooth interpolation without twist constraints, *Computer Aided Geometric Design*, R. Barnhill and W. Riesenfeld (eds.), Academic Press, New York, 1974.
15. Greiner, G., Blending Techniques based on variational principles, in *Curves and Surfaces in Computer Vision and Graphics III*, J. Warren (ed.), Proc. SPIE 1830, 1992, 174–184.
16. Greiner, G, and H.-P. Seidel, Curvature continuous blend surfaces, in *Modeling in Computer Graphics*, B. Falcidieno, T. L. Kunii (eds.), Springer Verlag, 1993, 309–317.
17. Greiner, G., Surface constructions based on variational principles, submitted to *Curves and Surfaces II* P.-J. Laurent, A. Le Méhauté, and L. L. Schumaker (eds.), Chamonix, 1993.
18. Kallay, M., Constrained optimization in surface design, in *Modeling in Computer Graphics*, B. Falcidieno, T. L. Kunii (eds.), Springer Verlag, 1993, 85–94.
19. Klein, R., Ph. Slusallek, An Object-Oriented Framework for Curves and Surfaces, in *Curves and Surfaces in Computer Vision and Graphics III*, J. Warren (ed.), Proc. SPIE 1830, 1992, 284–295.
20. Klass, R., Correction of local surface irregularities using reflection lines. Computer Aided Design **2** (1980), 73–76.
21. Kolb, A., Interpolating scattered data with C^2 surfaces, preprint, Universität Erlangen, 1993.
22. Lounsbery, M., S. Mann, S. and T. deRose, Parametric Surface Interpolation, Computer Graphics & Applications **9**, 1992, 97–115.
23. Moreton, H. P., and C. H. Séquin, Functional optimization for fair surface design, in *Computer Graphics*, E. E. Catmull (ed.), ACM Siggraph, ACM Press, 1992, 167–176.
24. Schumaker, L. L., Triangulation in CAGD, Computer Graphics & Applications **13**, 1993, 47–52.
25. Stroustrup, B., *The C++ Programming Language*, Addison Wesley, 2. edition, 1991.
26. Vermeulen A. H., and R. H. Bartels, C++ spline classes for prototyping, in *Curves and Surfaces in Computer Graphics II*, J. D. Warren (ed.), Proc. SPIE 1830, 1991, 121–131.
27. Welch, W., and A. Witkin, Variational surface modeling, in *Computer Graphics*, E. E Catmull (ed.), ACM Siggraph, ACM Press, 1992, 157–166.

Philipp Slusallek
Andreas Kolb
Günther Greiner
Universiät Erlangen
IMMD IX- Graphische Datenverarbeitung
Am Weichselgarten 9
D-91058 Erlangen, Germany
email: slusallek,kolb,greiner@informatik.uni-erlangen.de

Reinhard Klein
Universität Tübingen, WSI/GRIS
Auf der Morgenstelle 10, C9
D-72076 Tübingen, Germany
email: reinhard@gris.informatik.uni-tuebingen.de