

Shapemonger: A Math Library and Interactive Tool for the Study of 2D Parametric Surfaces in \mathbb{R}^3

Andrew Thall
Comp291
FINAL DRAFT

May 6, 1997

Abstract

Shapemonger is a library of mathematical routines for analyzing parametric surfaces and a user-interface for exploring their properties interactively. It is a stable and extensible collection of analysis and interface tools, useful both for didactic purposes and for research applications. It will be helpful to teachers of 3D shape analysis and differential geometry—both as an educational tool and as a coding platform for course projects—and to others wishing to explore these concepts computationally. This technical report provides user's information on Shapemonger and includes an appendix describing Shapetool, its predecessor.

Contents

1	Overview	4
1.1	Motivation: Differential Geometry for Students of Computer Science	4
1.2	Problems with Shapetool and Goals of Shapemonger	4
2	Shapetool Functionality	5
2.1	Basic Operation	5
2.2	Projects	6
3	Shapemonger Functionality	6
3.1	Code Improvements over Shapetool	6
3.2	Types of Objects Supported	6
3.3	GUI Design	7
3.4	Projects Ported	7
4	Shapemonger Implementation and Organization	8
4.1	SMOOCH: an Object-Oriented Interface to Motif	8
4.2	The Shapemonger Class Library	8
4.2.1	Math Unit—Class <code>Surfmath</code>	9
4.2.2	Math Unit—Class <code>Curvemath</code>	11
4.2.3	Math Unit— <code>Mathtools</code>	11
4.2.4	Class <code>Paramsurface</code>	12
4.2.5	Class <code>Paramcurve</code>	12
4.2.6	Classes <code>Surfrender</code> and <code>Curverender</code>	13
4.2.7	Class <code>Shaperaster</code>	13
4.2.8	<code>Shapeinterface</code>	14
4.2.9	Class <code>EQparser</code>	14
5	User's Information	14

5.1	The Display Tool and Demo Program	15
5.2	Making Extensions, Projects, and Use of the Math Library	15
5.2.1	Working within SMOOCH framework, using Shapemonger library	16
5.2.2	Use of Math Code in Stand-alone Applications	16
5.3	Directions for Further Development	17
5.3.1	Projects to be Ported from Shapetool	17
5.3.2	OpenGL Interface for Rendering	18
5.3.3	Point-and-Click to Select and Back-map Surface Points	18
5.3.4	Other Desirable Interactivity	18
5.3.5	Other Desirable Analysis Tools	18
5.3.6	Other Surface Types—Spline-based Patches	18
5.3.7	Other Surface Types—Implicit Functions and Volumetric Data	19
5.3.8	Other Surface Types—Multiple Patch Surfaces	19
5.3.9	Automatic Help Windows	19
5.4	Known Bugs	19
6	Conclusions	20
A	Appendix: Shapetool Projects	21
B	Appendix: Present Shapetool Implementation	23
B.1	Organization	23
B.1.1	User-Interface Code	23
B.1.2	Project Code	23
B.1.3	HLGL: High-Level Graphics Code	25
B.1.4	PWLL: Low-Level Graphics Code	25
B.2	Known Bugs	26

1 Overview

This paper describes the Shapemonger library and application, a tool designed for teaching differential geometry to students of computer graphics and image processing. Before beginning the discussion of Shapemonger, we will make a brief foray into its reason for being and the work which preceded it.

1.1 Motivation: Differential Geometry for Students of Computer Science

Comp257—Visual Solid Shape—at UNC teaches the concepts of differential geometry by focusing on the visual properties of surfaces in \mathbf{R}^3 . The program *Shapetool* has been developed over the past decade and used extensively in this course, both as a display tool for surfaces and their properties and as a platform for student projects.

Shapetool allows a user to enter an equation in two variables, along with constants and boundary limits, and to see a grid corresponding to the *Monge patch*—the height function—the equation represents. The user selects differential surface-properties (e.g., principal or asymptotic directions) and sees these displayed as vectors at grid vertices. The user rotates and scales the surface using sliders in the menu bar and can select from a pulldown menu student-projects such as those displaying the surface's Gauss-map or asymptotic-spherical maps. Appendices A and B provide details on Shapetool.

1.2 Problems with Shapetool and Goals of Shapemonger

Shapetool has proven its value over the past eight years, but it has become increasingly unwieldy with each new round of project additions and has several other age-related ailments. The user-interface and display code is based on Xview, an older windowing toolkit which is not well-supported on most current platforms. The C code comprising Shapetool is uneven in quality, having been extended and modified by four semesters of student projects. A new version was needed to

- eliminate bugs and superfluous code to produce a more consistent library of math and display tools;
- support the three chief platforms in the department—Sun, HP and SGI;
- encapsulate Shapetool's functionality, using C++ classes to create a cleaner and more easily modified body of code;
- modernize the GUI by replacing the Xview library with an object-oriented interface to Motif, making the application more portable and allowing
 - other graphics toolkits (such as OpenGL),

- advanced rendering features found on high-end machines,
- new user-interface functionality;
- modify the tool to analyze and display surface types in addition to Monge patches.

Shapemonger replaces Shapetool and creates a code base for building other shape-analysis applications. It provides a library of mathematical tools for differential geometric analysis of $2D$ parametric surfaces, and a display application and user-interface tool for the interactive study of such surfaces—not only Monge patches but surfaces-of-revolution and other general parametric surfaces in \mathbf{R}^3 —and parametric space curves as well.

The sections that follow describe the functionality of the library and application and discuss their implementations, explaining briefly the operation of Shapetool and then describing the important differences between Shapetool and Shapemonger.

2 Shapetool Functionality

2.1 Basic Operation

Upon initialization, Shapetool creates two canvases—the control panel and the display window—each of which iconifies separately. The control panel has the following control widgets:

- sliders controlling rotation and scaling;
- data entry fields for the (x,y)-parameterized patch equation; for range values for the x, y, and z axes; for equation parameters; and for S, C, and T values, giving another way to specify a surface using Koenderink’s shape and curvature indices; [Koenderink 90]
- radio buttons to select an axis of rotation, scaling (when non-uniform scaling is selected), or translation;
- toggles to turn on and off asymptotic and principal direction vectors at the grid points;
- reset and quit buttons;
- pull-down menus selecting uniform/non-uniform scaling; choosing between quadrilateral, triangular, and synclastic/anticlastic meshing; displaying normals or Darboux frames at mesh-points; and activating projects.

The display window shows a quadrilateral or triangularized grid of mesh points, 20×20 , and a coordinate axis frame for the x, y, and z directions. The window can be resized using standard X Windows click-and-drag techniques; the image within the window resizes automatically to the new dimensions. The triangularized mesh can be colored to indicate synclastic/anticlastic regions.

2.2 Projects

Shapetool is enhanced by nineteen student-projects¹ which vary in quality and utility. Most students wrote their own math routines to handle the differential geometric analysis—some incorrectly, others correctly but inefficiently. There are numerous other idiosyncrasies (e.g., there is no consistent way to turn off projects), and there are numerous bugs—some malignant, some “mostly-harmless” but unseemly.

Section 5.3.1 lists Shapetool projects that would be worthwhile additions to Shapemonger. Any project included in the new application must use the new math tools and conform to the structure of Shapemonger and the SMOOCH interface library.

3 Shapemonger Functionality

There were substantial changes in the design of the new library and application; these involved improving the structure of the library, creating new classes of analyzable and renderable objects, and improving the interface design.

3.1 Code Improvements over Shapetool

One goal of Shapemonger was to create a class library replacing the chaos of functions and global variables comprising Shapetool. SMOOCH—a high-level, C++ interface to Motif—provided a framework for building this library. SMOOCH was also used to create a better user-interface for the tool, benefiting from the rendering speed of modern workstations while being portable enough to run on on any platform supporting X Windows and Motif.

The other major goal, writing a library of clean math code for Shapemonger applications, was achieved by writing two separate classes for parametric surface and parametric curve analysis. These are designed to work as stand-alone *geometry servers*—making them easily incorporated into other projects and applications—and include robust methods for finding principal and asymptotic directions.

3.2 Types of Objects Supported

Monge patches—surfaces defined as $[x, y, f(x, y)]$ —are the only surfaces handled by Shapetool. Rendering curves and points is done at the application level by each project, using low-level graphics commands drawing directly to the X Windows display widget. The code uses global variables and “optimizations”, making it difficult to understand or modify.

¹Appendix A gives a listing of projects in the most recent version of Shapetool.

In contrast, Shapemonger handles arbitrary 2D-parametric surfaces in \mathbf{R}^3 , as well as points, parametric space-curves, and surface-curves. Shapemonger includes `Surfrender` and `Curverender` interface classes, which create rasterizable primitives for the `Shaperaster` class; `Shaperaster` handles the viewing transforms and maintains the display list, allowing interactive manipulation of displayed objects.

3.3 GUI Design

There were four main changes to the GUI design:

1. SMOOCH allows simple creation of interactive display windows. Rendered objects can be rotated or scaled using click-and-drag mouse manipulations, rather than by awkward sliders in the control panel.
2. SMOOCH has more sophisticated text-entry widgets and cleaner pull-down menus, used for selection of predefined surfaces, projects, and various options. Labelled buttons/toggles and dials (clockface numeric input widgets) are supported primitives.
3. The equation parser has been completely rewritten. The single-pass, bottom-up design was kept but restructured as a C++ class, allowing multiple instances in an application. Problems with incorrect operator precedence were corrected.
4. Originally, SMOOCH did not allow multiple canvases; Shapemonger test applications use a single canvas for the control panel and display window. Recent releases of SMOOCH now allow sub-canvases—necessary for controlling projects and useful for extending to graphical APIs like OpenGL.

3.4 Projects Ported

The new application is limited to displaying the surface mesh, showing *clastic regions*², and displaying surface normals, principal directions, and asymptotic directions in hyperbolic regions. While this is meager compared to the plethora of projects included in Shapetool, higher priority was given to the code library, allowing more projects to be added as time permits.

A separate application tests the display of a space-curves and its properties: Frenet frames, surfaces developed by its tangent vectors, and the loci of points at the centers of curvature of points along it. These capabilities will be included in the final surface analysis tool.

²These contain points with the same local curvature-type—synclastic (elliptic), anticlastic (hyperbolic), or parabolic.

4 Shapemonger Implementation and Organization

Before proceeding with an in-depth discussion of the Shapemonger class library, it will be helpful to begin with a brief discussion of the SMOOCH interface library.

4.1 SMOOCH: an Object-Oriented Interface to Motif

Motif is a powerful toolkit for the X Windows environment, but it is awkward and tricky to program. The *SMOCH* library (**S**imple **M**otif **O**bject-**O**riented **C**lass **H**ierarchy) was created by Alan Liu at UNC-CH as a C++-based library of tools and interface classes above Motif. SMOOCH simplifies the creation of interactive applications, taking over the dirty-work (such as setting and handling user-callbacks) and giving the user a higher-level interface.³

There are both advantages and disadvantages to using a toolkit-on-toolkit approach.

- **Advantages:** SMOOCH makes interface writing easy through the use of the classes `IORaster`, `Button`, `Canvas`, `PullDown`, etc. Each of these has standardized member functions for assigning callbacks, positioning, labelling, and so on; `IORaster`, for instance, has built-in callback pointers allowing user-defined functions to be tied to mouseclicks and other events, and also provides primitives for drawing points, lines, and polylines.
- **Drawbacks:** SMOOCH is not a finished product but is under development at UNC by Graham Gash, Brian Smith, Alan Liu, and Andrew Thall. The current implementation is fairly stable, but many planned features are not yet available. More generally, there are features which will *not* be available in the high-level library—there was no intention of porting more than a small subset of Motif functionality.

4.2 The Shapemonger Class Library

Shapemonger uses class-objects to replace the four functional units of Shapetool: user-interface, project code, and separate high- and low-level graphics. It imposes a logical structure on the GUI, mathematical-analysis tools, projects, and rendering tools—a structure allowing transparent alterations and extensions. There remain four categories of classes, with a total of nine main classes. The class hierarchy was kept simple and without complex inheritance schemes; numerical code must be both correct and fast. The OOP approach was useful mainly for its natural modularity and safe encapsulation of global variables.

The four categories are

³Those interested in working with SMOOCH should contact Graham Gash (gash@cs.unc.edu) at UNC—implementations are currently available for HP-UX, SunOS, IRIX and Linux machines. (Suggestions for a better name for our library are welcomed!)

- the math unit—consisting of **Surfmath**, code for analysis of parametric surfaces; **Curvemath**, code for analysis of parametric curves; and **Mathtools**, code for differentiation and other standard mathematical operations. It also contains optimized 2D and 3D vector and matrix classes since general-purpose, n -dimensional classes are too inefficient.
- **Paramsurface**—maintaining a grid structure, pointers to the parametric functions, bounds of the parameters, etc.
- **Paramcurve**—maintaining a list of points, tangent vectors, bitangents, etc., for the two types of parametric curves implemented.
- GUI and image display classes—**Curverender**, **Surfrender**, **Shaperaster**, **Shapeinterface**, and **EQparser**, supporting user-input and interactive display of surfaces and their properties.

Each of these classes will be discussed briefly, below.

4.2.1 Math Unit—Class Surfmath

Surfmath is the cornerstone of the project, providing a stable computational foundation for constructing shape-analysis tools. Rather than building separate C++ subclasses for each type of surface, **Surfmath** analyzes arbitrary 2D-parametric surfaces, which include Monge patches and surfaces-of-revolution as specific cases. This generality simplified the code, making it easier to debug and to check for correctness of algorithms.

Surfmath provides tools for studying differential properties of points on a 2D parametric patch. There are routines for finding normals and principal directions at a selected point (a *Darboux frame*), for testing *elasticity*—synclastic, anticlastic, or elliptic—and for determining asymptotic directions at an anticlastic point. Whereas Shapetool uses stand-alone C functions, Shapemonger’s class encapsulation allows better organization and optimizations. There are two chief items of interest:

- the caching of intermediate computations for a single point, and
- the patch-based computational methods, especially those computing principal and asymptotic directions.

The caching scheme assumes that multiple queries will often be made for information on a single point—the surface normal, the elasticity, the normal curvature, etc. Since there is a natural ordering of computations needed to find the various properties, preceding computations can be cached and their correctness signaled by a cascade of test flags. For example, to compute the (U, V) parametric coordinates of a tangent vector in \mathbf{R}^3 at a point (u, v) on the surface, one calls the following function:

```
DbVector2 surfmath::tan_3D_to_uv(double u, double v, const DbVector3& vec3D)
```

```

{
    DbVector2 temp2D, result2D;

    if (!(u == current_u) || !(v == current_v) || !(point_initialized))
        initpoint(u, v);
    if (!valid2ndDerivs)
        compute2ndD();
    if (!validForms)
        computeForms();
    if (!validFormIinv)
        computeIinv();

    [Now the pull-back to the parameter space can be done.]
}

```

The valid-flags will be set by previous computations for that same point—if a new point initialized, the flags are zeroed and the functions called as necessary, computing the values (stored as private class members) and setting test-flags accordingly. Subsequent queries on that point will find flags already set, signaling cached values that need not be recomputed.

The other features of interest are the methods for computing principal and asymptotic directions. Both of these properties are derived from the shape operator, and Shapetool performed eigenanalysis, etc., on the 2×2 matrix to find their values. For general parametric surfaces, however, the shape operator in patch-coordinates is not symmetric, which rules out many n -dimensional eigen-solving methods.⁴ Instead, **Surfmath** uses a formalist approach taken from traditional differential geometry. While the ideas of [Koenderink 90] are more intuitive, the mainstream approach presented by [O’Neill 66] is more concise. This uses E, F , and G , l, m , and n to represent dot products of tangent-space basis vectors with each other and with the normal vector; for a patch defined by $X(u, v)$, one computes

$$\begin{aligned}
 E &= X_u \cdot X_u \\
 F &= X_u \cdot X_v \\
 G &= X_v \cdot X_v \\
 l &= \vec{N} \cdot X_{uu} \\
 m &= \vec{N} \cdot X_{uv} \\
 n &= \vec{N} \cdot X_{vv}
 \end{aligned}$$

These are simply the components of the first and second fundamental forms. Given these, one solves for the principal directions analytically, using the fact that an eigenvector $[x, y]^T$ of the

⁴Even for Monge patches, the shape operator will often be asymmetric, a fact which led to bugs in Shapetool routines that assumed that a *symmetric bilinear form* would produce a symmetric matrix when pulled back to parameter space.

shape operator must satisfy

$$Ax^2 + Bxy + Cy^2 = 0,$$

where

$$A = (lF - mE), \quad B = (lG - nE), \quad \text{and} \quad C = (mG - nF).$$

Similarly, a vector $[x, y]^T$ is an asymptotic direction iff it satisfies

$$lx^2 + 2mxy + ny^2 = 0.$$

(See [Oneill 66, p. 230] for derivations.) Using these theorems, the desired shape properties can be found by a quadratic equation solver, without using advanced eigensolving techniques.⁵

4.2.2 Math Unit—Class Curvemath

`Curvemath` is simpler in design than `Surfmath`—though it includes a similar caching scheme for derivatives at curve-points—and is designed for

- parametrically-defined space curves in \mathbf{R}^3 , or
- parametric curves onto the 2D parameter-space for some given patch, and mapping from there by the patch-function into \mathbf{R}^3 .

`Curvemath` has functions for computing the Frenet frame vectors (tangent \vec{T} , normal \vec{S} , and binormal \vec{B}), the curvature κ , and the torsion τ . One issue not addressed is curve parameterization—it would be useful to have a reparameterization function returning an identical curve with arc-length parameterization. This could be implemented in a later version.

4.2.3 Math Unit—Mathtools

`Mathtools` provides math routines that support the `Shapemath` and `Curvemath` classes. These include

- vector and matrix classes—the `SMOCH` library currently does all its arithmetic on `float` rather than `double` values, leading us to implement our own small, optimized class libraries of 2D and 3D vector and matrix routines;
- a derivative server for functions of one or two variables, giving first and second derivatives by central difference approximation, with one-sided difference approximations available for ends of curves and edges of patches;

⁵Cases where the quadratic solver would be numerically unstable indicate special surface properties, which should be checked for in any case.

- a stable quadratic equation solver (no mean feat, actually); and
- a general eigensystem solver, currently broken.

The eigensolver was taken from Shapetool code that assumed the matrices were “nice”, i.e., symmetric, making it unusable for current Shapemath tasks. A stable eigensystem solver will be needed, however, for Eberly-style ridge-traversal methods and other project tasks.

4.2.4 Class Paramsurface

Paramsurface is the base class for all surfaces (2D smooth manifolds in \mathbf{R}^3) currently implemented. A parametric surface is defined by an equation of the form

$$F(u, v) = [X(u, v), Y(u, v), Z(u, v)]$$

for u and v in a specified range. A **Paramsurface** object is essentially an “intelligent” grid and some pointers to functions. The object is initialized with a given equation, either as 3 equations $\mathbf{R}^3 \rightarrow \mathbf{R}$ or as a single equation $\mathbf{R}^3 \rightarrow \mathbf{R}^3$. The object is also initialized with a range for the u and v parameters and with a sampling frequency in u and v . **Paramsurface** then creates a grid (rectangular, but a triangular grid would be a simple modification) containing surface-point positions, normals, principal directions, asymptotic directions, and other curvature information. Each **Paramsurface** has a **Surfmath** object for grid computations and a **Surfrender** object generating rasterizable primitives from the grid information and passing them to the **Shaperaster** for display.

There are two subtypes of parametric surfaces that might benefit from their own child classes:

- **Mongesurface**—defined by a monge patch, as a height function $h(x, y)$ —a **Paramsurface** where $F(u, v) = [u, v, h(u, v)]$.
- **Rotsurface**—defined by a curve $h(x) > 0$ rotated about the x -axis—a **Paramsurface** where $F(u, v) = [u, h(u) \cos v, h(u) \sin v]$.

These are presently implemented using the **Paramsurface** class, but the possibility of surface-specific optimizations might make it worthwhile to implement them as distinct derived-classes.

Other surface types, such as implicit surfaces and multi-patch surfaces, will require more elaborate additions to the class library. Spline-based surfaces fit easily in a **Paramsurface** approach, but might benefit from a more dynamic data-structure; a **Paramsurface** is relatively static—to change sampling density or to alter one of the input equations, one simply creates a new **Paramsurface** with the new characteristics.

4.2.5 Class Paramcurve

As previously stated, there are two classes of parametric curves currently implemented:

- a parameterized space curve $\alpha(t) = [X(t), Y(t), Z(t)]$.
- a parameterized curve into (u,v) parameter space and then into 3D by the patch equation,

$$\alpha(t) = \mathbf{X}(u(t), v(t)) = [X(u(t), v(t)), Y(u(t), v(t)), Z(u(t), v(t))].$$

A **Paramcurve** consists of pointers to functions, bounds and increments for the parameter, and an array containing the points (and their differential properties) generated by traversing the curve in order. Like a **Paramsurface**, a **Paramcurve** has its own **Curvemath** and **Curverender** units. **Paramcurves** can also create two new surfaces, corresponding to the surfaces developed by the tangent vectors along the curve in the plus- and minus-directions.

4.2.6 Classes **Surfrender** and **Curverender**

Surfrender and **Curverender** are *enzymes*—classes which have little or no data of their own, but serve as modules to hold and manage functions used by other classes. **Surfrender** and **Curverender** are interfaces between the **Paramsurfaces** and **Paramcurves**, respectively, and the rasterization unit, taking information stored in object grids or arrays and generating rasterizable primitives for the **Shaperaster**.

This approach encapsulates the task of primitive generation, simplifying modifications needed for other display protocols like OpenGL; it also promotes consistency in techniques and in optimizations for the different curve and surface types.

4.2.7 Class **Shaperaster**

Shaperaster controls the image-display IO and is global to an application. It has four main tasks:

- rasterizing primitives to the display window in the canvas;
- automatically refreshing and resizing this window in accord with X events;
- transformation handling, including mouse-driven rotations and uniform scalings of the objects displayed; and,
- maintaining the display list—labelling objects, making insertions and deletions into the list, declaring primitives visible/invisible, etc.

Shaperaster uses the **SMOCH IORaster** object to create, to draw to, and to receive mouse-events from the display window. **IORasters** but have occasional speed problems, especially with regard to accessing X Windows color tables. This necessitates sorting the primitives by color before entering them into the display list, reducing to a minimum the number of changes of the drawing color. This sorting is done in the **Surfrender** class rather than in **Shaperaster**, since it is **Surfrender** which sets colors for primitives.

4.2.8 Shapeinterface

`Shapeinterface` maintains the canvas for the application program developed from the library. At the time of development, multiple canvases were not supported; consequently, `Shapeinterface` has a lot of balls to juggle. Its main tasks are to maintain

- the `Shaperaster` window,
- text-entry widgets for setting equations, parameter boundaries, etc.,
- pulldown menus for selecting built-in surfaces,
- toggle-buttons for displaying normals, principal and asymptotic directions, clastic and normal grids, etc.,
- lists of object-names passed to the `Shaperaster`, for toggling objects visible or invisible, or for deleting them.

At present, `Shapeinterface` is not a class—the full implementation required tools not implemented in SMOOCH when the code was written. All of the necessary pieces are currently available and tested; implementation of the encapsulating class should be straightforward.

4.2.9 Class EQparser

Shapemonger applications need an equation parser that can take an equation input as an ASCII text string and return a function of several variables. Shapetool has such a parser, taking a text equation in one or two variables and with up to five user-definable parameters, parsing the text string and creating a stack function to evaluate it. In keeping with Shapetool's chaotic design, this parser is a global object, with cryptic magic numbers throughout the code and with bugs in the operator precedence.

Rather than write a new parser from scratch, it was simpler to correct the bugs in the Shapetool code, and use it as a basis to create the class `EQparser`. Each `EQparser` instance creates a function of one or two variables, with up to five user-definable parameters, according to parameter values set and according to the equation text it is given to parse. An `EQparser` evaluation returns a double floating-point result—multiple instances may be used to create functions from $\mathbf{R}^2 \rightarrow \mathbf{R}^3$ for parametric surfaces such as $F(u, v) = [X(u, v), Y(u, v), Z(u, v)]$.

5 User's Information

This section describes the demo program constructed, gives information on programming using Shapemonger and SMOOCH, and points out areas where more work is necessary or desirable—e.g., which Shapetool projects should be ported to Shapemonger.

5.1 The Display Tool and Demo Program

The emphasis so far has been on the library code, especially the math and surface classes. The main purpose of Shapetool, however, was to provide students with an interactive tool for exploring surfaces; consequently, a similar demo program using Shapemonger was created. While not a polished product, it demonstrates the functionality required by a full implementation and shows off improvements of the new library over the old. The main differences are in

- the user-interface,
- built-in surfaces,
- surface properties.

The interface consists of a **Shaperaster** view-window, text-entry windows for equations and parameter values, and toggle buttons for selecting surface properties for display. The student can directly manipulate a surface in the view-window using the mouse as a click-and-drag tool for either rotation or scaling—an improvement over Shapetool, which used an array of sliders for controlling rotations and scalings about specific axes.⁶

Another change from Shapetool is the addition of pulldown menus permitting the selection of predefined surfaces. General parametric equations can create much more complex surfaces than Monge patches; providing a list of predefined surfaces gives the student examples of displayable surfaces along with a list of standard surfaces for assignments.

The implemented surface properties are only a fraction of those available in Shapetool, with its more than twenty specialized tools. The new application has functions displaying principal directions, asymptotic directions, and surface normals, as vectors at grid points. The elasticity of surface regions can also be displayed, by color-coding the (normally green) surface mesh to show regions of positive, negative, and zero curvature. Section 5.3.1 lists the Shapetool projects which should be included in a complete user-interface.

Recently, a second test-bed was created to demonstrate the curve-rendering and analysis tools, but these have not yet been integrated into the primary test platform.

5.2 Making Extensions, Projects, and Use of the Math Library

Shapemonger is straightforward to use for either project extensions or independent applications. The SMOOCH library is designed exactly for this purpose, as are the **Shapemath** and **Curvemath** classes. Below are a few comments on application building in this framework.

⁶Shapemonger now lacks the ability to do non-uniform scaling about specific axes in object-space, but this was a feature seldom used in the old tool, and the gains from the more intuitive interactions outweigh the loss. It would be trivial to add non-uniform-scaling capability to the new tool, if it were desired.

5.2.1 Working within SMOOCH framework, using Shapemonger library

SMOOCH is a remarkably easy-to-use library for application development, providing two class libraries:

- `smooch`—containing the interface-building tools, such as image-widgets, file I/O windows, pull-down and pop-up menus, buttons and sliders, and text-entry widgets; and
- `tools`—providing mathematical tools such as vectors, matrices, and B-splines, as well as display and graphics-oriented tools such as transformation matrices, homogeneous-coordinate routines, and camera/viewpoint models.

A potential problem with SMOOCH is that it currently lacks classes for double-precision vector and matrix operations and that its single-precision code was designed for generality rather than efficiency. A numerical application requiring extended precision might benefit from a smaller, independently created double-precision class tuned to the particular task. This is the tact taken with Shapemonger.

Application building in SMOOCH is simple: one first creates a `Canvas` object, then creates buttons, sliders, text-entry widgets, and `IORasters` with desired attributes and locations, and finally enters the event loop to drive the program. All the details of Motif window management are encapsulated in the member functions and are hidden from view—the user simply indicates which callback functions to attach to which events.

The Shapemonger code library is built on top of this. The class `Shaperaster` is the only derived class, a `public IORaster`; the rest simply use classes from SMOOCH as member elements. The Makefile has been kept as platform independent as possible, with platform-dependent code isolated in separate include files. Class dependencies are handled using `#define D_<classname>` in the `.C` files for each class needed, followed by a single call to `#include "Shapemonger.H"`, which checks dependencies and includes all necessary header files. (SMOOCH itself uses a similar scheme.)

5.2.2 Use of Math Code in Stand-alone Applications

`Surfmath` and `Curvemath` were designed to be as independent as possible from SMOOCH and Shapemonger code. To use them as stand-alone modules, one needs only the classes themselves, the derivative server and linear algebra routines in the files `Deriv.{C,H}` and `LinAlg.H`, and a few of the definitions from the class header file `Shapeheader.H`.

Note that many optimizations (e.g., in regard to caching) were application specific and can be eliminated if unnecessary. Note, too, that optimizations for special surface-types were not made. For example, if a user wants principal directions on a surface-of-revolution, these require no computation at all, but are simply the parametric coordinate axes X_u and X_v . A user building an efficient application should examine the routines carefully for such optimizations.

5.3 Directions for Further Development

Shapemonger still requires a lot of work before it will match and surpass the utility of Shapetool. Necessary tasks are

- porting projects from the Shapetool,
- enhancing the interface and creating an OpenGL window class,
- creating other surface types than parametric, and
- creating other analysis tools.

Each of these will be discussed briefly.

5.3.1 Projects to be Ported from Shapetool

Of the existing Shapetool projects, the following are ones that should be included in the finished Shapemonger application. While it was not possible to implement these in the time available, Shapemonger has the support apparatus allowing their inclusion. See Appendix A for a full description of these and other Shapetool projects.

1. Level curves and range-map.
2. Animations of Principal/Asymptotic spherical maps and Gauss-map.
3. Parabolic curves on surface—current implementation non-functional.
4. Specular reflections—not an easy port, but a nice tool that should be included.
5. Point explorer—too slick not to include in new tool.
6. Principal mesh—current implementation works badly.
7. Geodesic strips.
8. Maximum-height, Koenderink, and vertex-curve ridges.
9. Morphogenesis. (The Shapetool code was not functional, but a working version would be a useful addition.)

5.3.2 OpenGL Interface for Rendering

Shapemonger's `Shaperaster` makes X Windows calls for all drawing to the `IORaster`. While `Shaperaster` has been optimized to achieve interactive speeds, there is no double-buffering nor support for other rendering modes, i.e., filled, shaded polygons, fog for depth-reference, etc. K.C. Low has incorporated OpenGL windows into SMOOCH interfaces, but SMOOCH needs a `GLRaster` object with functionality similar to `IORaster`. OpenGL is available in this department on the SGI workstations, and the MESA library provides the same interface and functionality (though not speed) on all other platforms. A `GLRaster` would give Shapemonger much more sophisticated rendering capability.

5.3.3 Point-and-Click to Select and Back-map Surface Points

A user often needs to select a single point on a surface—a location for surface analysis, a starting point for a surface curve, etc. One method allows a mouse-click to select the pixel where the desired point is projected. The pixel must be back-mapped onto the 3D surface in space and then onto the parametric plane to find its (u, v) coordinates. A number of different schemes are feasible—some limited to Monge-patches, but others effective for general parametric surfaces.

5.3.4 Other Desirable Interactivity

In an independent project done in 1996 by Hansong Zhang, a user could move the mouse-cursor over a screen region representing the parameter space of the surface; a point on the surface in the main display window would then track this position. Similar functionality was included in Brian Morse's Point-Explorer project several years ago. Such an interface would work well for selecting points, in lieu of (or superior to) a system such as the point-and-click scheme above.

5.3.5 Other Desirable Analysis Tools

The analytic tools in `Surfmath` principally use the shape operator and its eigenvalues and eigenvectors. We need additional tools for finding geodesics, etc, and for searching regions for ruffles, flecnodal curves, and other tasks which require ridge-following or minimization approaches.

5.3.6 Other Surface Types—Spline-based Patches

A spline-based patch is similar in properties to other parametric patches. John Keyser, in 1996, created a spline-based surface class to be included in the Shapemonger library. This will be installed in the next version of the code.

5.3.7 Other Surface Types—Implicit Functions and Volumetric Data

Given a function $F : \mathbf{R}^3 \rightarrow \mathbf{R}$, defining a scalar function $F(x, y, z)$ over a volume of space, one defines an *isosurface* of the function over that volume as all points (x, y, z) where $F(x, y, z) = c$, for some constant c . Differential geometry can be done on an isosurface as on a parametric surface—finding principal directions, hyperbolic regions, etc. Such techniques have applications in a number of different areas (e.g., [Interrante 96]).

A new math class would be needed for computation on non-parametrically defined surfaces. New surface classes would be necessary, with `Surfrender` extensions to create rasterizable primitives for volumetrically-derived data. This is a major task, but perhaps easier given a `GLRaster` display to help with the rendering. Decisions on volume-rendering techniques would also be necessary.

5.3.8 Other Surface Types—Multiple Patch Surfaces

Most interesting shapes are not easily modeled by a single patch but can be assembled using multiple patches to cover the surface. Although more sophisticated display tools and interfaces are necessary before such work can be done elegantly, most of the math code is already in place. Analyses of patch-based objects are identical to those for other parametric surfaces, but care is needed in regions of overlap between patches.

5.3.9 Automatic Help Windows

A major problem with Shapetool was the lack of online help for projects, many of which had confusing, non-intuitive interfaces. SMOOCH can now create and destroy sub-canvases from within the main event-loop; it would be useful to have a dialog-box subclass whose sole function is the display of error and help messages. This is standard in on many PCs and would be a useful addition to Shapemonger.

5.4 Known Bugs

Shapemonger currently has several known bugs, which will be eradicated as time permits. These include

- mysterious domain errors from a `sqrt()` function when creating develop-able surfaces from a space curve, which neither crash the program nor produce visible inaccuracies;
- noticeable inaccuracies in computation of the curvature κ for space-curves: when the loci at the radius of curvature for a right circular helix is computed, its position overshoots the true axis by a constant amount;

- lack of error-checking for operations performed at coordinate singularities—e.g., at the poles of a sphere in spherical coordinates. These can cause zero-divides and other anomalies in arithmetic routines, and a robust application will need to check for them.

6 Conclusions

The Shapemonger library and application code makes marked improvements over its predecessor and shows much promise for future development. Shapemonger should be of interest to anyone doing work in computational differential geometry, but especially to teachers and students seeking a platform for developing their insights and their own applications. Parties interested in using the tool (or the math libraries as stand-alone code) should contact thall@cs.unc.edu. Shapemonger is still very much a work-in-progress, and input from potential and actual users is welcome.

A Appendix: Shapetool Projects

Shapetool projects vary in both quality and usefulness. There is no consistent way to turn off many of them: some must be selected a second time from the menu and rerun with null values; others have quit buttons in their dialog-boxes; others remain until the surface formula is changed. There are also numerous bugs, both large and small.

The following projects are available via the project menu:

1. Level curves: plots level curves on the surface for 10 user-input z values. BUG: draws level curves 90 degrees out of alignment with the surface. Reselect and recompute will toggle off the computed curves.
2. Principal directions map animator: turns off by quitting. BUG: problems with points at which principal directions seem to change discontinuously, distorting map produced. Quitting the dialog sub-window turns off the project.
3. Asymptotic directions map animator: similar to last, but no apparent bugs.
4. Hyperbolics A and B: A seems to give just hyperbolic directions, B adds some confusing yellow curves in space off of surface. Window update events (i.e., any use of control panel) turn off the project. BUG: No apparent purpose.
5. Gauss-map: maps the mesh onto an invisible Gaussian sphere. BUG: any update event cancels selection, so can't rotate resulting map to get a better view of it.
6. Range-map: selecting a position from the dialog sub-window generates a set of level curves for fixed distances from it. BUG: color of level curves hard to see (faint blue on the green surface mesh). This is the case for item 1 as well. It is turned off by changing the surface's formula.
7. Parabolic: should display parabolic curves on surface, or curvature feature? BUG: doesn't seem to work...control panel ambiguous.
8. Curvature: unclear what it does. Application dies when error in input function (attempted compute on null entries).
9. Gauss-map animation: similar to other animations—can toggle on and off a unit sphere about the origin. Works well.
10. Specular reflections: a fancy project, positioning two locators to give light position, and positioning the eye-point, upon computation, the tool draws a small cluster of vectors on the surface at the point of the specular highlight(s). The animate slider then moves the light source between the the two locators, showing the change in specular highlight position at each interpolated location. BUG: operation is *not* self-evident; as with many of the projects, a help panel feature would be good.

11. Near & Far points: BUG: doesn't work at all. Bus error kills program.
12. Point explorer: A slick one...creates a canvas with a large (x,y) grid, and small areas showing
 - principal curvatures K_1 and K_2 , graphically, as circles (or arcs of circles) of radius proportional to the curvature,
 - K and H , and indicated by a "thermometer"-style widget,
 - the normal, as a small dot on a grid, showing the (x,y) projection of the vector,
 - the F_1 and F_2 principal frame vectors, as projected onto the (x,y) plane,
 - the principal curvatures K_1 and K_2 plotted on a grid, with their ratio K_1/K_2 given by the slope of the vector from the origin to the point,
 - Koenderink's shape and curvedness indices S and C , plotted on a grid

When the user positions the mouse on the (x,y) grid, the other fields display their respective values for that indicated point. A toggle places a normal vector on the surface in the display window at the point being analyzed. BUG: with normal vector displayed, runs SLOW...Xevents pile up and the lag becomes pronounced. (Simply porting to a faster architecture may solve this.) It is turned off by a button in the window.

13. Principal mesh: gives integral curves of principal direction. BUGS: behaves strangely, seems limited to the curves emanating from the origin on some surfaces; trouble at umbilics, and at boundary of region. Toggled off by second calling of project from menu.
14. Asymptotic spherical map: works occasionally, drawing projections of asymptotic directions in red and blue onto (invisible) sphere. Dialog window disappears when map is generated and there's no safe way to turn off the map. BUG: calling the project a second time and setting all *To Show* fields off will clear the display window, but upon trying to call the project again, program dies with obscure error message.⁷
15. Geodesic strip: nice...given either a starting and ending point, or a point and a vector in the (x,y) plane, this computes and displays a geodesic between the points or in the tangent direction given by the vector. Turned off by a button in window. BUG: red line on green grid wreaks havoc with the human visual system...doesn't seem that line is lying on grid.
16. Maximum height ridges: finds a maximal height ridge on the surface, coloring its portions according to whether they're in elliptic or hyperbolic regions. There is strange behavior on spherically symmetric surfaces like $-(x^2 + y^2)$, but I am not sure if that should be considered a BUG. Project is turned off by toggling selection. Ridges thus computed are maxima of height along lines of principal curvature.
17. Koenderink ridges: similar behavior to item 16, but no explanation of exactly what's going on. Likewise get strange results on $-(x^2 + y^2)$, more clearly numerical error in this case.

⁷"In generategrid(). I never expected cl_dealwithzerocase() to be called!"

(MH ridges on this were especially odd—one side of hemisphere was completely crosshatched, other side clean.) Project is turned off by toggling selection. Ridges computed are maxima of curvature along lines of principal curvature.

18. Vertex curve ridges: similar to above, same functionality. Similar type of error to Koenderink ridges on spherically symmetric surface, though different pattern of ridges. Ridges thus computed are maxima of curvature along level curves on surface.
19. Morphogenesis: unsure of purpose—dialog window has wrong title (still says “Gauss map animation”), and no hints as to the correct use of it. BUG: changing parameters can also result in frozen X-servers. Not usually desirable.

B Appendix: Present Shapetool Implementation

B.1 Organization

Code in the Shapetool library can be classified into four groups: user-interface code, project code, high-level graphics code, and low-level graphics code. A brief overview will be presented, so that users needing code or algorithms from Shapetool will know where to find them. A preliminary listing was done by Ron Azuma or Graham Gash (AGG).

B.1.1 User-Interface Code

The user-interface code consists of the following files:

- `main.c` — creates initial windows and widgets
- `callbacks.c` — callbacks to support the widgets, including project calls
- `mesh.c` — creates quad-meshes, triangles, normals, etc.
- `render.c` — draws meshes into display window
- `eval.c` — part of Tim Cullip’s parser routines
- `parser.c` — reads in a text description of an equation.

B.1.2 Project Code

Support code for the projects is provided by the following files; note that much of this code is undocumented.

- `AM_anim.c` — based on Azuma's `gm_anim.c`, maps asymptotic directions onto unit sphere. By AGG.
- `H_ridge.c`, `K_ridge.c`, `V_ridge.c` — Charlie Kurak's ridge-finding code.
- `PM_anim.c` — as above, but maps principal directions.
- `asyp.c` — code by AGG to calculate asymptotic directions, and draw them to the mesh. Called by `render()`.
- `clastic.c` — code to calculate clasticity of a point.
- `crit.c` — code by Barbara Winkler, calculates critical points of the range map—near, far, and saddle-points. Includes some math code.
- `eigen.c` — eigenanalysis code from Numerical Recipes.
- `eigenvalues.c` — computes eigenstuff for 2x2 matrices. Brian Morse.
- `explore.c` — Brian Morse's point-explorer code.
- `frame.c` — code to classify points synclastic/anticlastic, and to calculate a Darboux frame-field over an object.
- `gaussm.c` — computer the gauss-map of the surface.
- `geodesic.c` — Peter Brown's code to draw geodesic strip on the surface.
- `gm_anim.c` — Azuma's Gaussmap animation code.
- `hyp.c` — finds and displays hyperbolic regions of the surface, with code for the asymptotic spherical map as well.
- `ii.c` — by AGG, calculates principal directions and curvature at a point on the surface, nose-dive and twist, the 2nd fundamental form, etc.
- `level.c` — draw level curves on the surface.
- `math.c` — essential math functions—mostly vector math.
- `morph.c` — animate the morphing of the surface...bad documentation, and code perhaps doesn't work at all.
- `oliverlevel.c` — same functionality as `level.c`.
- `parabolic.c` — more entirely undocumented code. Duplicates some of `level.c`.
- `point_shape.c` — Morse's point-shape calculation code for explorer. Duplicates much of the functionality of `ii.c`.

- `princip.c` — AGG code. Calculate and draw principal direction crosses at mesh points.
- `prmesh.c` — AGG. Calculate a mesh by following principal directions.
- `quadratic.c` — Morse. Finds real roots for quadratic eq. Numerically unstable method (doesn't eliminate catastrophic cancellation.)
- `range.c` — compute and display range level-curves from a specified eye position.
- `shape_index.c` — Rik Faith code. Interfaces with Shapetool parser to allow specification of surface by `shape_index` values.
- `specular.c` — Fredericksen code. Creates interface panel and computes location/orientation animation of specular highlights.
- `sphere_map.c` — code to compute asymptotic spherical map of function. Supposed to show map for parabolic curve, flecnodal, and tangent plane to each ruffle. Not fully implemented.
- `sub1.c` — subroutine for `range.c`. Code to plot a graphical object, given global Xform matrix and environment, and to calculate z values for Monge grid, distance maps to eyepoint, etc.
- `vector.c` — Morse's vector creation and math code.

There are also associated header files to go along with these. The ones most important to the design of the library are:

- `shape.h` — definition of globals for grid points' z-values, curvatures, as well as UI-globals, definitions of "objects" for drawing to window, and global flags.
- `Math.h` and `vector.h` — basic definitions and declarations of vector functions found in `math.c`.
- `devgl.h` and `dgl.h` — the interface functions for the graphics routines.

B.1.3 HLGL: High-Level Graphics Code

This is the high-level graphics library, designed to be device independent. It is a 3D library, which allows the maintenance of a stack of transformation matrices and handles colors, clipping, and the basic drawing and windowing commands. By Judd Reid in 1988.

B.1.4 PWLL: Low-Level Graphics Code

These are the device dependent drawing and windowing operations, called by the `hgl` routines. Converted to X routines in 1990.

B.2 Known Bugs

Aside from the ones in the functionality section and below, there is an error in the rasterization of normal vectors—it is done with the same transformation as that used for points and tangent vectors, rather than using the inverse transpose of that matrix. This gives incorrect normals for cases of non-uniform scaling.

The following is a list compiled by AGG and given here verbatim; some of these were listed in the functionality section.

1. The Level Curves option of the Projects menu, which used to be invoked by a button on the main panel, does not work properly. Wei-Jyh Lin said it should be replaced with a call to the range map code from a viewpoint on the z axis.
2. The Hyperbolics A option of the Projects menu has been disabled. Apparently, Hyperbolics A is a subset of Hyperbolics B, however.
3. Quadratic.c and eigenvectors.c should be excised. However, note that on the surface $x^2 - y^2$, the Point Explorer code gives domain errors, probably in solving the quadratic equation. This means that code needs an additional check for clasticity.
4. Some projects, such as Hyperbolics B, are not using XView correctly, and their results are erased by a refresh of the surface window. This means the Projects menu may not overlap the surface window.
5. Several routines have the same sphere-drawing function. At least PM_anim.c, AM_anim.c and gm_anim.c do.
6. Originally, evaluate() was a float function with float arguments. It is now a double function with double arguments. Some functions, e.g. hyp.c, use unnecessary casts with evaluate. Specular.c uses a defined type of FPrecision to select between float and double. This method should be used throughout and the typedef should be put in shape.h.
7. Some functions (B. Morse's) compile using gcc. The others use cc.
8. It is difficult to sort out the asymptotic spherical map (AM_anim.c), and asymp.c is modified to assist in this. The commented out part (lines 155 - 188) contains code that should instead be used. The head option of the place_asymp_cross function should perhaps be user-configurable.
9. Prmesh.c is a prototype. While the Principal mesh option seems to work, it is not certain that it cannot hang, and it is not drawing a complete mesh. A. G. Gash has done additional work on this problem.
10. Shape declares fixed-length arrays for the objects. These should be malloced.
11. There is no consistent and globally used toolkit for geometrical calculations. The files math.c and ii.c come closest to forming a nucleus for this.

12. The oliver*.c files should be excised.
(The above notes were all recorded on or before Jan 2, 1991.)
13. Say an equation is displayed with principal directions turned on, and part of the mesh is clipped. Then, if the Minimum or Maximum values are changed so none of the mesh is clipped, the principal directions are not recomputed for the new points. Possibly, the entire grid should be computed, ignoring clipping, and then only the unclipped portion be displayed. AGG, 30 Sep 1994.
14. If an equation is entered correctly and then changed to be incorrect, say by adding a "+" at the end, then nothing else can be done to recompute points, but some operations, such as rotation, still work. Probably the program should not allow any action except correction of the equation or quitting. AGG, 30 Sep 1994.
15. When the show button of the Point Explorer is used, a small yellow line remains on the screen after that project is exited. AGG, 30 Sep 1994.
16. The "curvature" project, file oliverlevel.c, seems to give no output. However, output can be produced by closing and reopening the graph window after pressing compute. Note that no function in this file is being called at the end of render.c. AGG, 5 Oct 1994.
17. The "Level Curves" project does not draw. However, after pressing compute, if the graph window is closed and reopened, the output will be seen. AGG, 5 Oct 1994.
18. Project "Asymp sphere map" does not work for equation $x * y$.

References

- [Gray 93] Gray, A., *Modern Differential Geometry of Curves and Surfaces*, CRC Press, 1993.
- [Interrante 96] Interrante, Victoria L., “Illustrating Transparency: Communicating the 3D Shape of Layered Transparent Surfaces via Texture,” Doctoral Dissertation, UNC-CH, May 1996.
- [Koenderink 90] Koenderink, J.J., *Solid Shape*, MIT Press, 1990.
- [Lipschutz 69] Lipschutz, M., *Theory and Problems of Differential Geometry*, Shaum’s Outline Series, McGraw-Hill, 1969.
- [O’Neill 66] O’Neill, B., *Elementary Differential Geometry*, Academic Press, 1966.
- [Press 92] Press, W., Teukolsky, S., Vetterling, W., Flannery, B., *Numerical Recipes in C, 2nd Ed.*, Cambridge University Press, 1992.