SISL

The SINTEF Spline Library

(version 4.4)

User's Guide

SINTEF ICT, Applied Mathematics

September 21, 2010

Contents

Chapter 1

Preface

1.1 A word of welcome

Welcome to the SISL 4.4 user's guide! This document is written to make you able to use the powerful routines of SISL in as short time as possible. SISL stands for ***S**intef **S**pline **L**ibrary*, and has been gradually developed and enhanced for more than two decades by the geometry group at SINTEF in Oslo. Although it is very comprehensive, its organisation is simple. There are but a few structures, and its nearly four hundred main functions can usually be employed directly and individually. The SISL 4.4 distribution comes with a comprehensive reference manual, which organises and explains the main routines. However, much of this information can also be found directly in the code in the form of commentaries.

The complete software package you have in your hands should contain the following:

- The SISL 4.4 distribution and reference guide
- The User's Guide (the document you are reading now)
- Supplementary routines for writing SISL objects to streams (including file streams) in a simple ASCII format called `Go` (**G**eometric **O**bject)
- A selection of *sample programs*, designed to demonstrate functionalities and use of SISL
- Source code for a *viewer* that can be used to view geometric objects stored in the `Go`-format. This allows visual inspection of SISL-generated curves and surfaces, as well as points

1.2 The structure of this document

Chapter 2 is a general introduction to SISL and its programming style. Since it is strongly recommended that the user has some general knowledge of splines, this chapter also contains a couple of sections introducing the subject of spline curves and surfaces. The text in chapter 2 can also be found in the SISL 4.4 reference guide.

Chapter 3 goes through the provided sample programs and explain what these do, and what the user can expect to learn from them. There are a total of 15 sample programs, ranging from very basic to intermediate complexity.

The goal of **Chapter 4** is to explain the use of the *viewer program*, which is a small but handy tool for visually inspecting results from SISL routines.

Finally there is an **annex**, citing the text of the General Public License.

1.3 The structure of the software package

There are five directories:

- `sisl/` - the source code of the 4.4 release of SISL
- `doc/` - documentation (reference manual and user's guide)
- `streaming/` - source code for the routines that can read and write SISL objects to a stream
- `examples/` - sample programs making use of the SISL 4.4 source code
- `viewer/` - source code for a viewer that can be used to view SISL objects saved in the `Go`-format

Compiling libraries and example programs

Each of these directories, with the exception of `doc/`, contains a Makefile, which can be used to compile the code in that directory. The programs in the `examples/` directory links with libraries generated from `sisl\_4.4/` and `streaming/`, so these should be compiled first. In order to compile the source code, this is what you should do:

- To compile *SISL 4.4*, enter the `sisl_4.4/` directory and type '`make lib`' .

- To compile *the streaming routines*, enter the `streaming` directory and type '`make lib`' .
- To compile an *example program*, enter the `examples/` directory and type '`make exampleXX`', where `XX` is a number from 01 to 15. In order to link successfully, make sure you have already compiled the streaming routines as well as SISL 4.4.
- To compile the *viewer*, enter the `viewer/` directory and type '`make viewer`'.

By default, the files will be compiled in optimized mode. If you want to compile in non-optimized mode, you should add the argument `OPT_TAG=nopt` to your command line.

1.4 Licensing information

SISL is distibuted under the *General Public License* (GPL). The license text is given in its entirety as an annex to this document. Commercial licenses are also available from SINTEF. You can contact Tor Dokken (tor.dokken@sintef.no) for more information.

Chapter 2

General Introduction

SISL is a geometric toolkit to model with curves and surfaces. It is a library of C functions to perform operations such as the definition, intersection and evaluation of NURBS (Non-Uniform Rational B-spline) geometries. Since many applications use implicit geometric representation such as planes, cylinders, tori etc., SISL can also handle the interaction between such geometries and NURBS.

Throughout this manual, a distinction is made between NURBS (the default) and B-splines. The term B-splines is used for non-uniform non-rational (or polynomial) B-splines. B-splines are used only where it does not make sense to employ NURBS (such as the approximation of a circle by a B-spline) or in cases where the research community has yet to develop stable technology for treating NURBS. A NURBS require more memory space than a B-spline, even when the extra degrees of freedom in a NURBS are not used. Therefore the routines are specified to give B-spline output whenever the extra degrees of freedom are not required.

Transferring a B-spline into NURBS format is done by constructing a new coefficient vector using the original B-spline coefficients and setting all the rational weights equal to one (1). This new coefficient vector is then given as input to the routine for creating a new curve/surface object while specifying that the object to be created should be of the NURBS (rational B-spline) type.

To approximate a NURBS by a B-spline, use the offset calculation routines with an offset of zero.

The routines in SISL are designed to function on curves and surfaces which are at least continuously differentiable. However many routines will also handle continuous curves and surfaces, including piecewise linear ones.

SISL is divided into seven modules, partly in order to provide a logical structure, but also to enable users with a specific application to use subsets of SISL. There are three modules dealing with curves, three with surfaces, and one module to perform data reduction on curves and surfaces (this last module is largely in

Fortran). The modules for curves and surfaces focus on functions for creation and definition, intersection and interrogation, and general utilities.

The three important data structures used by SISL are SISLCurve, SISLSurf, and SISLIntcurve. These are defined in the Curve Utilities, Surface Utilities, and Surface Interrogation modules respectively. It is important to remember to always free these structures and also to free internally allocated structures used to pass results to the application, otherwise strange errors might result.

Each chapter in this manual contains information concerning the top level functions of each module. Lower level functions not usually required by an application are not included. Each top level function is documented by describing the purpose, the input and output arguments and an example of use. To get you started, this chapter contains an Example Program.

2.1 C Syntax Used in Manual

This manual uses the K&R style C syntax for historic reasons, but both the ISO/ANSI and the K&R C standards are supported by the library and the include files.

2.2 Dynamic Allocation in SISL

In the description of all the functions in this manual, a convention exists on when to declare or allocate arrays/objects outside a function and when an array is allocated internally. *NB! When memory for output arrays/objects are allocated inside a function you must remember to free the allocated memory when it is not in use any more.*

The convention is the following:

- If [] is used in the synopsis and in the example it means that the array has to be declared or allocated outside the function.
- If $*$ is used it means that the function requires a pointer and that the allocation will be done outside the function if necessary.
- When either an array or an array of pointers or an object is to be allocated in a function, two or three stars are used in the synopsis. To use the function you declare the parameter with one star less and use & in the argument list.
- For all output variables except arrays or objects that are declared or allocated outside the function you have to use & in the argument list.

2.3 Creating a Program

In order to access SISL from your program you need only one inclusion, namely the header file sisl.h. The statement

```
#include "sisl.h"
```

must be written at the top of your main program. In this header file all types are defined. It also contains all the SISL top level function declarations.

To compile the calling program you merely need to remember to include the name of the directory where sisl.h resides. For example, if the directory is called sisldir then,

```
$ cc -c -Isisldir prog1.c
```

will compile the source code prog1.c to produce prog1.o.

In order to build the executable, the *c* parts of the SISL library libsislc.a must be included. Thus

```
$ cc prog1.o -Lsisldir -lsisl -o prog1
```

will build the test program prog1.

2.4 B-spline Curves

This section is optional reading for those who want to become acquainted with some of the mathematics of B-splines curves. For a description of the data structure for B-spline curves in SISL, see the SISL 4.4 manual.

A B-spline curve is defined by the formula

$$\mathbf{c}(t) = \sum_{i=1}^{n} \mathbf{p}_i B_{i,k,\mathbf{t}}(t).$$

The dimension of the curve $\mathbf{c}$ is equal to that of its *control points* $\mathbf{p}_i$. For example, if the dimension of the control points is one, the curve is a function, if the dimension is two, the curve is planar, and if the dimension is three, the curve is spatial. Usually the dimension of the curve will be at most three, but SISL also allows higher dimensions.

Thus, a B-spline curve is a linear combination of a sequence of B-splines $B_{i,k,\mathbf{t}}$ (called a B-basis) uniquely determined by a knot vector $\mathbf{t}$ and the order k. Order is equivalent to polynomial degree plus one. For example, if the order is two, the degree is one and the B-splines and the curve c they generate are (piecewise) linear. If the order is three, the degree is two and the B-splines and the curve are quadratic. Cubic B-splines and curves have order 4 and degree 3, etc.

The parameter range of a B-spline curve $\mathbf{c}$ is the interval

$$[t_k, t_{n+1}],$$

and so mathematically, the curve is a mapping $\mathbf{c} : [t_k, t_{n+1}] \to \mathbb{R}^d$, where d is the Euclidean space dimension of its control points.

The complete representation of a B-spline curve consists of

dim : The dimension of the underlying Euclidean space, $1, 2, 3, \ldots$.

n : The number of vertices (also the number of B-splines)

k : The order of the B-splines.

$\mathbf{t}$: The knot vector of the B-splines. $\mathbf{t} = (t_1, t_2, \ldots, t_{n+k})$.

$\mathbf{p}$: The control points of the B-spline curve. $p_{d,i}$, $d = 1, \ldots, dim$, $i = 1, \ldots, n$.
e.g. when $dim = 3$, we have $\mathbf{p} = (x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_n, y_n, z_n)$.

We note that arrays in c start at index 0 which means, for example, that if the array t holds the knot vector, then $t[0] = t_1, \ldots, t[n + k - 1] = t_{n+k}$ and the parameter interval goes from $t[k - 1]$ to $t[n]$. Similar considerations apply to the other arrays.

The data in the representation must satisfy certain conditions:

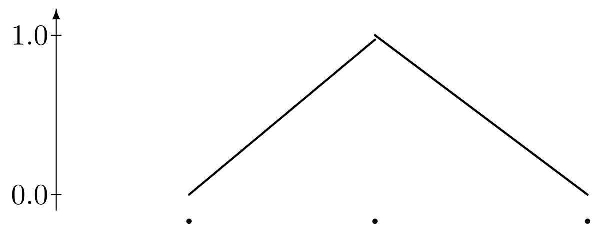

Figure 2.1: A linear B-spline (order 2) defined by three knots.

- The knot vector must be non-decreasing: $t_i \leq t_{i+1}$. Moreover, two knots t_i and t_{i+k} must be distinct: $t_i < t_{i+k}$.
- The number of vertices should be greater than or equal to the order of the curve: $n \geq k$.
- There should be k equal knots at the beginning and at the end of the knot vector; that is the knot vector $\mathbf{t}$ must satisfy the conditions $t_1 = t_2 = \ldots = t_k$ and $t_{n+1} = t_{n+2} = \ldots = t_{n+k}$.

To understand the representation better, we will look at three parts of the representation: the B-splines (the basis functions), the knot vector and the control polygon.

2.4.1 B-splines

A set of B-splines is determined by the order k and the knots. For example, to define a single B-spline of degree one, we need three knots. In figure 2.1 the three knots are marked as dots. Knots can also be equal as shown in figure 2.2. By taking a linear combination of the three types of B-splines shown in figures 2.1 and 2.2 we can generate a linear spline function as shown in figure 2.3.

A quadratic B-spline is a linear combination of two linear B-splines. Shown in figure 2.4 is a quadratic B-spline defined by four knots. A quadratic B-spline is the sum of two products, the first product between the linear B-spline on the left and a corresponding line from 0 to 1, the second product between the linear B-spline on the right and a corresponding line from 1 to 0; see figure 2.4. For higher degree B-splines there is a similar definition. A B-spline of order k is the sum of two B-splines of order $k-1$, each weighted with weights in the interval [0,1]. In fact we define B-splines of order 1 explicitly as box functions,

$$B_{i,1}(t) = \begin{cases} 1 & \text{if } t_i \leq t < t_{i+1}; \\ 0 & \text{otherwise,} \end{cases}$$

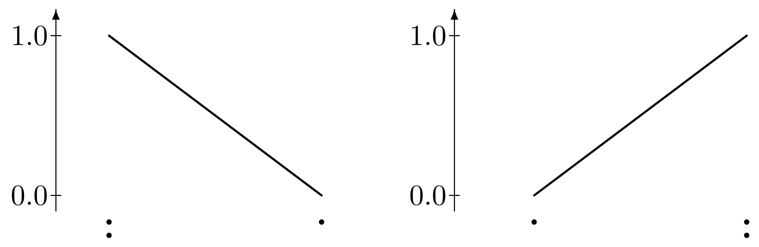

Figure 2.2: Linear B-splines of with multiple knots at one end.

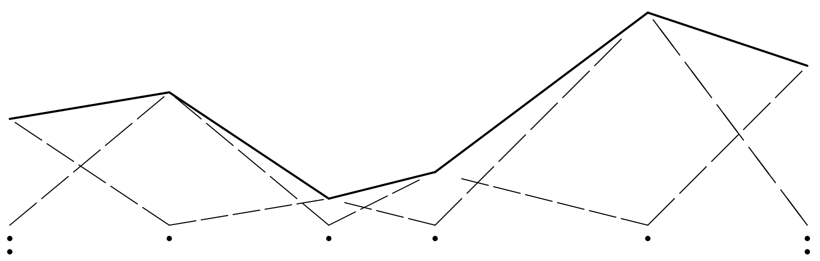

Figure 2.3: A B-spline curve of dimension 1 as a linear combination of a sequence of B-splines. Each B-spline (dashed) is scaled by a coefficient.

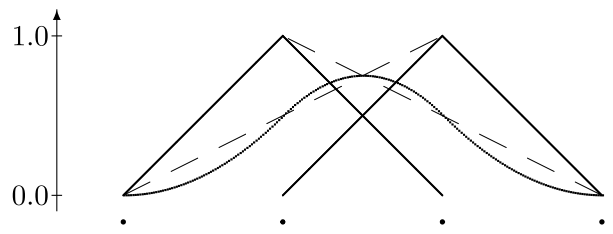

Figure 2.4: A quadratic B-spline, the two linear B-splines and the corresponding lines (dashed) in the quadratic B-spline definition.

and then the complete definition of a k-th order B-spline is

$$B_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} B_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} B_{i-1,k-1}(t).$$

B-splines satisfy some important properties for curve and surface design. Each B-spline is non-negative and it can be shown that they sum to one,

$$\sum_{i=1}^{n} B_{i,k,\mathbf{t}}(t) = 1.$$

These properties combined mean that B-spline curves satisfy the *convex hull property*: the curve lies in the convex hull of its control points. Furthermore, the support of the B-spline $B_{i,k,\mathbf{t}}$ is the interval $[t_i, t_{i+k}]$ which means that B-spline curves has *local control*: moving one control point only alters the curve locally.

Due to the demand of k multiple knots at the ends of the knot vector, B-spline curves in SISL also have the *endpoint property*: the start point of the B-spline curve equals the first control point and the end point equals the last control point, in other words

$$\mathbf{c}(t_k) = \mathbf{p}_1 \qquad \text{and} \qquad \mathbf{c}(t_{n+1}) = \mathbf{p}_n.$$

2.4.2 The Control Polygon

The control points $\mathbf{p}_i$ define the vertices The *control polygon* of a B-spline curve is the polygonal arc formed by its control points, $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n$. This means that the control polygon, regarded as a parametric curve, is itself piecewise linear B-spline curve (order two). If we increase the order, the distance between the control polygon and the curve increases (see figure 2.5). A higher order B-spline curve tends to smooth the control polygon and at the same time mimic its shape. For example, if the control polygon is convex, so is the B-spline curve.

Another property of the control polygon is that it will get closer to the curve if it is redefined by inserting knots into the curve and thereby increasing the number

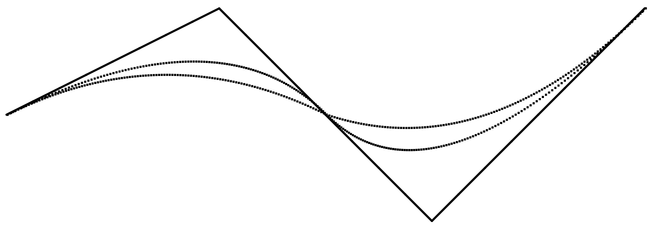

Figure 2.5: Linear, quadratic, and cubic B-spline curves sharing the same control polygon. The control polygon is equal to the linear B-spline curve. The curves are planar, i.e. the space dimension is two.

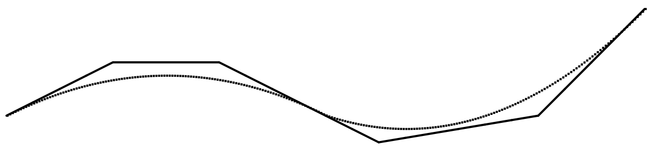

Figure 2.6: The cubic B-spline curve with a redefined knot vector.

of vertices; see figure 2.6. If the refinement is infinite then the control polygon converges to the curve.

2.4.3 The Knot Vector

The knots of a B-spline curve describe the following properties of the curve:

- The parameterization of the B-spline curve
- The continuity at the joins between the adjacent polynomial segments of the B-spline curve.

In figure 2.7 we have two curves with the same control polygon and order but with different parameterization.

This example is not meant as an encouragement to use parameterization for modelling, rather to make users aware of the effect of parameterization. Something close to curve length parameterization is in most cases preferable. For interpolation, chord-length parameterization is used in most cases.

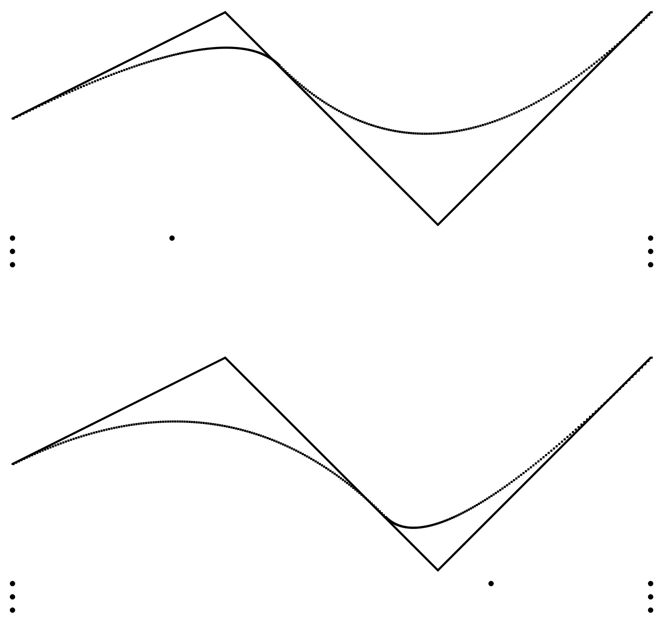

Figure 2.7: Two quadratic B-spline curves with the same control polygon but different knot vectors. The curves and the control polygons are two-dimensional.

The number of equal knots determines the degree of continuity. If k consecutive internal knots are equal, the curve is discontinuous. Similarly if $k-1$ consecutive internal knots are equal, the curve is continuous but not in general differentiable. A continuously differentiable curve with a discontinuity in the second derivative can be modelled using $k-2$ equal knots etc. (see figure 2.8). Normally, B-spline curves in SISL are expected to be continuous. For intersection algorithms, curves are usually expected to be continuously differentiable (C^1).

2.4.4 NURBS Curves

A NURBS (Non-Uniform Rational B-Spline) curve is a generalization of a B-spline curve,

$$\mathbf{c}(t) = \frac{\sum_{i=1}^{n} w_i \mathbf{p}_i B_{i,k,\mathbf{t}}(t)}{\sum_{i=1}^{n} w_i B_{i,k,\mathbf{t}}(t)}.$$

In addition to the data of a B-spline curve, the NURBS curve $\mathbf{c}$ has a sequence of weights $w_1, \ldots, w_n$. One of the advantages of NURBS curves over B-spline curves is that they can be used to represent conic sections exactly (taking the order k to be three). A disadvantage is that NURBS curves depend nonlinearly on

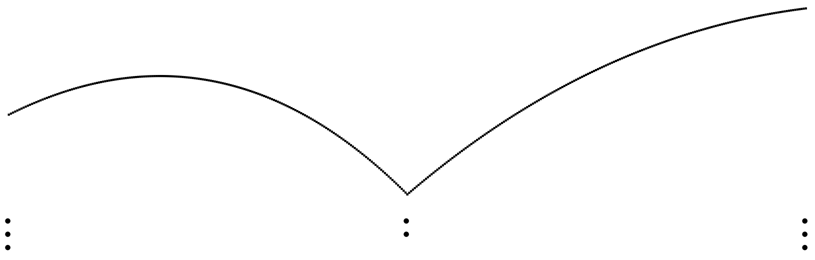

Figure 2.8: A quadratic B-spline curve with two equal internal knots.

their weights, making some calculations, like the evaluation of derivatives, more complicated and less efficient than with B-spline curves.

The representation of a NURBS curve is the same as for a B-spline except that it also includes

$\mathbf{w}$: A sequence of weights $\mathbf{w} = (w_1, w_2, \ldots, w_n)$.

In SISL we make the assumption that

- The weights are (strictly) positive: $w_i > 0$.

Under this condition, a NURBS curve, like its B-spline cousin, enjoys the convex hull property. Due to k-fold knots at the ends of the knot vector, NURBS curves in SISL also have the endpoint property.

2.5 B-spline Surfaces

This section is optional reading for those who want to become acquainted with some of the mathematics of tensor-product B-splines surfaces. For a description of the data structure for B-spline surfaces in SISL, see the reference manual.

A tensor product B-spline surface is defined as

$$\mathbf{s}(u,v) = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \mathbf{p}_{i,j} B_{i,k_1,\mathbf{u}}(u) B_{j,k_2,\mathbf{v}}(v)$$

with control points $\mathbf{p}_{i,j}$ and two variables (or parameters) u and v. The formula shows that a basis function of a B-spline surface is a product of two basis functions of B-spline curves (B-splines). This is why a B-spline surface is called a tensor-product surface. The following is a list of the components of the representation:

dim : The dimension of the underlying Euclidean space.

n_1 : The number of vertices with respect to the first parameter.

n_1 : The number of vertices with respect to the second parameter.

k_1 : The order of the B-splines in the first parameter.

k_2 : The order of the B-splines in the second parameter.

u : The knot vector of the B-splines with respect to the first parameter, $\mathbf{u} = (u_1, u_2, \ldots, u_{n_1+k_1})$.

v : The knot vector of the B-splines with respect to the second parameter, $\mathbf{v} = (v_1, v_2, \ldots, v_{n_2+k_2})$.

p : The control points of the B-spline surface, $c_{d,i,j}$, $d = 1, \ldots, dim$, $i = 1, \ldots, n_1$, $j = 1, \ldots, n_2$. When $dim = 3$, we have $\mathbf{p} = (x_{1,1}, y_{1,1}, z_{1,1}, x_{2,1}, y_{2,1}, z_{2,1}, \ldots,$ $x_{n_1,1}, y_{n_1,1}, z_{n_1,1}, \ldots,$ $x_{n_1,n_2}, y_{n_1,n_2}, z_{n_1,n_2})$.

The data of the B-spline surface must fulfill the following requirements:

- Both knot vectors must be non-decreasing.
- The number of vertices must be greater than or equal to the order with respect to both parameters: $n_1 \geq k_1$ and $n_2 \geq k_2$.
- There should be k_1 equal knots at the beginning and end of knot vector **u** and k_2 equal knots at the beginning and end of knot vector **v**.

The properties of the representation of a B-spline surface are similar to the properties of the representation of a B-spline curve. The control points $\mathbf{p}_{i,j}$ form a *control net* as shown in figure 2.9. The control net has similar properties to the control polygon of a B-spline curve, described in section 2.4.2. A B-spline surface has two knot vectors, one for each parameter. In figure 2.9 we can see *isocurves*, surface curves defined by fixing the value of one of the parameters.

2.5.1 The Basis Functions

A basis function of a B-spline surface is the product of two basis functions of two B-spline curves,

$$B_{i,k_1,\mathbf{u}}(u)B_{j,k_2,\mathbf{v}}(v).$$

Its support is the rectangle $[u_i, u_{i+k_1}] \times [v_j, v_{j+k_2}]$. If the basis functions in both directions are of degree one and all knots have multiplicity one, then the surface basis functions are pyramid-shaped (see figure 2.10). For higher degrees, the surface basis functions are bell shaped.

Figure 2.9: A B-spline surface and its control net. The surface is drawn using isocurves. The dimension is 3.

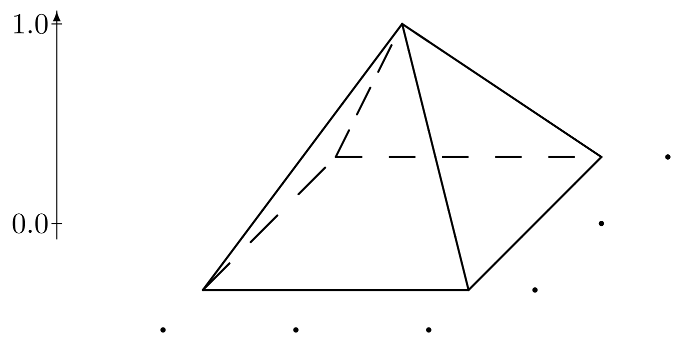

Figure 2.10: A basis function of degree one in both variables.

2.5.2 NURBS Surfaces

A NURBS (Non-Uniform Rational B-Spline) surface is a generalization of a B-spline surface,

$$\mathbf{s}(u,v) = \frac{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} w_{i,j}\mathbf{p}_{i,j}B_{i,k_1,\mathbf{u}}(u)B_{j,k_2,\mathbf{v}}(v)}{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} w_{i,j}B_{i,k_1,\mathbf{u}}(u)B_{j,k_2,\mathbf{v}}(v)}.$$

In addition to the data of a B-spline surface, the NURBS surface has a weights $w_{i,j}$. NURBS surfaces can be used to exactly represent several common 'analytic' surfaces such as spheres, cylinders, tori, and cones. A disadvantage is that NURBS surfaces depend nonlinearly on their weights, making some calculations, like with NURBS curves, less efficient.

The representation of a NURBS surface is the same as for a B-spline except that it also includes

w : The weights of the NURBS surface, $w_{i,j}$, $i = 1, \ldots, n_1$, $j = 1, \ldots, n_2$, so $\mathbf{w} = (w_{1,1}, w_{2,1}, \ldots, w_{n_1,1}, \ldots, w_{1,2}, \ldots, w_{n_1,n_2})$.

In SISL we make the assumption that

- The weights are (strictly) positive: $w_{i,j} > 0$.

Chapter 3

Tutorial programs

This release of SISL is bundled with a number of sample programs which are intended to make the user more familiar with the use of the API, as well as demonstrating some of its capabilities.

3.1 Compiling the programs

Makefiles are provided to compile all provided source code, included the sample programs. Refer to (1.3) for details on using the provided makefiles. Please note that since these executables link both with SISL 4.4 and the 'streaming' library, these must be compiled first.

The compilation of the example code has been verified for the GCC 3.3.3 compiler and the Microsoft Visual C++ .NET 2002 compiler, but it should be general enough to compile relatively troublefree on most platforms.

3.2 Description and commentaries on the sample programs

The example programs are named `example01` through `example15`. Each of the program demonstrates the use of a single or a couple of SISL functions. The programs produces output files that contain geometric objects in the `Go`-format, which can then be visualised by the provided KRULL-viewer.

To keep things as simple as possible, the example programs (with the exception of `example15`) take no command line arguments. Instead, upon execution they inform the user about what they are about to do, and which files will be read from and written to. The names of the input and output files are hard-coded in each example, but the user can experiment by changing the name of these files if she wants to. Several of the sample programs rely upon files generated by earlier examples, so the user should make sure she runs through them in chronological order.

3.2.1 example01.C

What it does

This program demonstrates how to directly specify a spline curve by providing the position of control points and a knotvector (parametrization). It generates such a curve by using hard-coded values as input to the SISL `newCurve` routine.

What it demonstrates

1. How control points and knotvectors are specified in memory
2. How to use the `newCurve` routine
3. How to clean up memory using `freeCurve`

Input/output

The program takes no input files.
The program generates the files `example1_curve.g2` and `example1_points.g2`. The former contains the curve object and the latter contains the control points, expressed in the `Go`-format.

3.2.2 example02.C

What it does

This program demonstrates one of the simplest *interpolation* cases for spline curves in SISL. A sequence of 6 3D-points are provided (hardcoded), and the routine generates a spline curve that fits exactly through these points. Note that this is a simple example of a more general routine, which can also take into consideration tangents, end point conditions, etc.

What it demonstrates

1. The use of the SISL routine `s1356` for interpolating points with a curve

Input/output

The program takes no input files.
The program generates the file `example2_points.g2` and `example2_curve.g2`. The first file contains the points to be interpolated, and the second file contains the generated curve.

3.2.3 example03.C

What it does

This program creates a so-called *blend-curve* between two other curves, creating a smooth connection between these. In this program, the blend curve connects the *end points* of the two other curves, but in its generality, the routine can be used to create blend curves connecting to any point on the other curves.

What it demonstrates

1. What a blend curve is and how it can be specified

2. The use of the SISL routine `s1606` which computes the blend curve

3. The use of the SISL routine `s1227` which evaluates points (and derivatives) on a spline curve

4. How to directly access data members of the `SISLCurve` struct.

Input/output

The program takes as input the files `example1_curve.g2` and `example2_points.g2`, which are respectively generated by the programs `example01` and `example02`. The generated blend curve will be saved to the file `example3_curve.g2`.

3.2.4 example04.C

What it does

This program generates an *offset curve* from another curve. An offset curve is specified as having a fixed distance in a specified direction from the original curve. The generated offset curve will not be exact, as this would in general be impossible using spline-function. We can however obtain an approximation within a user-specified tolerance.

What it demonstrates

1. What an offset curve is and how it can be specified
2. The way in which many SISL routines deal with geometric tolerances
3. The use of the SISL routine `s1360` which computes the offset curve within a specified, geometric tolerance

Input/output

The original curve is read from the file `example1_curve.g2`, which is generated by the program `example01`. The resulting approximation of the offset curve will be written to the file `example4_curve.g2`.

3.2.5 example05.C

What it does

This program generates a family of conic section curves, which are represented as rational splines. Conic sections can be *exactly* represented with such splines, so no geometric tolerance specification is needed. The program will generate three ellipse segments, one parabola segment and three hyperbola segments, based on internal, hard-coded data.

What it demonstrates

1. The use of the SISL routine `s1011` to generate all kinds of conic sections
2. The important fact that conic sections can be exactly represented by rational splines
3. How a single *shape* parameter can specify whether the generated curve will be an ellipse, a parabola or a hyperbola

Input/output

The program takes no input files.
The program generates the file `example5_curve.g2` which contains all the generated curves.

3.2.6 example06.C

What it does

This program generates two curves (from internal, hardcoded data), and computes their intersections. Computation of intersections is an extremely important part of SISL, although the intersection of two curves is a minor problem in this respect.

What it demonstrates

1. The use of the SISL routine `s1857` for computing the intersection points between two given spline curves

2. Underlines the fact that the detected intersection points are returned as parameter values, and have to be evaluated in order to find their 3D positions

3. How to clean up an array of intersection curves (SISLIntcurve), although, in this example, this array will already be empty

Input/output

The program takes no input files (the data for the curves is hard-coded).
The generated curves will be written to the files `example6_curve_1.g2` and `example6_curve_2.g2`. The intersection point positions will be written to the file `example6_isectpoints.g2`.

3.2.7 example07.C

What it does

This is a very short and simple program that calculates the arc length of a curve.

What it demonstrates

1. The use of the SISL routine `s1240` for computing the length of a spline curve

Input/output

The curve whose length is calculated is read from the file `example6_curve_1`, which has been generated by the sample program `example06`. The calculated length will be written to standard output.

3.2.8 example08.C

What it does

This program generates two non-intersecting spline curves (from internal, hard-coded cata), and computes their mutual closest point. The call is very similar to the one in `example06`, where we wanted to compute curve intersections.

What it demonstrates

1. The use of the SISL routine `s1955` for locating the closest points of two curves.

Input/output

As the curves are specified directly by internal data, no input files are needed. The two generated curves will be saved to the two files `example8_curve_1.g2` and `example8_curve_2.g2`. The closest points will be written to the file `example8_closestpoints.g2`.

3.2.9 example09.C

What it does

This program generates four different surfaces interpolating an array of spatial points. The surfaces have different spline order, so that even though they interpolate the same points, they have different shapes.

What it demonstrates

1. The use of the SISL routine `s1537` for generating an interpolating surface to a grid of points

2. The effect of the spline order on the interpolating surface

Input/output

The program takes no input files (the points to be interpolated are hard-coded). The program creates two data files: `example9_points.g2`, which contains all the interpolated points, and `example9_surf.g2`, which contains the four generated surfaces.

3.2.10 example10.C

What it does

This program generates a sequence of spline curves. Moreover, it generates a *lofted surface* interpolating these curves. The lofted surface has the original sequence of curves as isoparametric curves in one of its parameters.

What it demonstrates

1. The use of the SISL routine `s1538` for generating lofted spline surfaces
2. Gives a good example of what a lofted surface looks like

Input/output

The program takes no input files (the curves to be interpolated are hard-coded).
The program creates two data files: `example10_curves.g2`, containing the generated sequence of curves, and `example10_surf.g2`, containing the lofted surface.

3.2.11 example11.C

What it does

This program generates a cylindrical surface with an oval base.

What it demonstrates

1. The use of the SISL routine `s1021` for generating cylindrical surfaces
2. The fact that cylindrical surfaces are exactly representable as rational spline surfaces

Input/output

The program takes no input files.
The program creates one data file: `example11_surf.g2`, containing the generated surface.

3.2.12 example12.C

What it does

This program finds the intersection points between a curve and a surface. The curve and the surface in question have been defined by previous example programs.

What it demonstrates

1. The use of the SISL routine `s1858` for computing intersection points between a curve and a surface

Input/output

The curve and the surface in question are read from the files `example4_curve.g2` and `example10_surf.g2`, respectively generated by the sample programs `example04` and `example10`. The found intersections are written to the file `example12_isectpoints.g2`.

3.2.13 example13.C

What it does

This program computes all intersection curves between two surfaces. This is a nontrivial task in geometrical modeling. The problem is twofold. The first problem is to determine the number of intersections, and their topology. The region of an intersection can be either a point, a curve and a surface. In the two latter cases, the shape of the region can usually only be approximated. We do not know a priori how many separate intersections there exists between two surfaces, so we have to look systematically for them. Intersection curves can take the form either as closed loops on the interior of the surfaces, or as curves running from the surface edges. When we have successfully determined the topology of the intersections, the second problem is to determine their acutal shape. This is usually done by *marching techniques*. However, we may run into problems with 'degenerated' surfaces, or surfaces being close to coplanar in the intersection.

What it demonstrates

1. The use of the SISL routine `s1859` for determining the topology of the intersections between two spline surfaces

2. The use of the SISL routine `s1310` for marching out the detected curves after their topologies have been determined

Input/output

The two surfaces have been generated by the previous sample programs `example10` and `example11`, and can be found in the files `example10_surf.g2` and `example11_surf.g2`. The resulting intersection curves will be written to the file `example13_isectcurves.g2`.

3.2.14 example14.C

What it does

This program demonstrates one of the data reduction techniques of SISL. As input data, it first generates a dense point set by sampling from a (predefined) spline curve. Then, using this data, it attempts to generate a new spline curve that fits closely to these samples, while using as few control points as possible. Since we know that in this case the data points come from a simple spline curve, it should be no surprise that the generated curve will have approximately the same expression as the sampled curve (and thus reduce the quantity of data substantially compared to what is needed to store the points). However, data reduction can be obtained on any sufficiently smooth point set, even if it originates from other processes.

What it demonstrates

1. The use of the SISL routine `s1961` for generating approximating spline curves through a set of data, using as few control points as possible.

2. The power of this data reduction technique on smooth point data.

Input/output

The program takes no input files, as the curve to be sampled from is hard-coded. The sampled points will be written to the file `example14_points.g2`, and the obtained curve will be stored in `example14_curve.g2`.

3.2.15 example15.C

What it does

This is the last of the sample programs, and by far the most complicated. It aims not only to demonstrate a certain feature of SISL, but to show how this feature can be used for a purpose (raytracing). Moreover, it demonstrates two ways of achieveing this, one slow and robust method and one rapid but fragile method.
Raytracing can be seen as the process of determining what an object 'looks like' from a certain viewpoint, through a certain 'window', as illustrated below. Lines

('rays') are extended from the viewpoint through a dense grid of points on the window, and checked for intersection with the object. If such an intersection exists, it should be registered as a point on the object 'visible' from the viewpoint. In

computer graphics, these points are projected back on the window, which becomes a 2D image that can be displayed on the computer screen. For our purposes, we refrain from doing this projection, and store the full 3D coordinates of the detected point.
Note that a ray may intersect the object more than once. In these cases, the intersection point closest to the viewpoint is chosen, as the other points are 'hidden' by it. As mentioned above, there are two raytracing routines in this example program. The robust routine calculates all possible intersection points for each ray, and then choses the nearest one. This should always work, but can be slow since no information is re-used. When we have found an intersection point for a given ray, we can usually expect that the next, neighouring ray will intersect in a point close to the one already found. If this is the case, it would be speedier to use a local algorithm that converges on the intersection point quickly given a good initial guess. This is the basis for our 'quick' routine. This routine uses the robust raytracing algorithm to find the first point on a surface, and then it switches over to the fast method as long as it is possible to do so. However, since the quick method never finds more than one intersection point, and since a ray may generally intersect an object more than once, we have no guarantee that the point found is the one truly visible from the viewpoint. There are some checking procedures that make things better, but we still have no guarantee. If the user inspects the results obtained, he will notice this problem even on the simple example given here. In general, it can be said that the rapid algorithm should only be used in some special cases, where we know for a fact that any ray from the viewpoint will not intersect the surface more than once.

This is the only of the example programs that can be run with a command line argument. If the first argument is `q`, then the quick raytracing routine will be invoked. Else, the robust and slow routine is used.

What it demonstrates

1. The basic setting and principe of a raytracer, with a defined viewpoint, window and intersection with rays.
2. The use of the SISL routine `s1856`, which calculates all intersections between a spline surface and a line.
3. The use of the SISL routine `s1518`, which converges to an intersection between a spline surface and a line, given a good initial guess.

Input/output

The surface to be raytraced is read from the file `example10_surf.g2`, generated by the `example10` program. The other parameters necessary for the raytracing are hard- coded (viewpoint, view window, resolution, etc.). The resulting points are written to the file `example15_points.g2`.

Chapter 4

The object viewer program

4.1 General

The object viewer program bundled with this distrubtion of SISL is intended to be a simple but handy tool for visualising curves and surfaces generated by SISL. The supported file format is the `Go` (**G**eometric **O**bject) format, which is a simple, ASCII-based format defined by SINTEF. The viewer is based on OpenGL. The object(s) to be viewed are specified on the command line when starting the program. Once the program is started, the user cannot open other files containing SISL objects. The viewer allows the user to zoom, pan and rotate the objects with the mouse, and some other useful commands can be accessed through the keyboard.

In the viewer window, several curves and surfaces can be displayed simultaneously. At all times, exactly *one* surface and *one* curve are defined as being *active* (the other ones being *passive*). With keyboard commands, the user can change the currently active surface/curve. An object just becoming active will flash for a few seconds. With other keyboard commands, the user can *enable/disable* surfaces and curves. This refers to turning the display of these objects on or off. For details, refer to the section on keyboard commands.

4.2 Command line arguments

When starting up the viewer, the options listed below can be used. If no option is specified, a short text listing the available options is printed on screen.

- `s` *filename* - view the surface(s) contained in the file *filename*. Note: this command line option can be used repetitively if the user wants to inspect several surfaces at once.

- `c` *filename* - view the curve(s) contained in the file *filename*. Note: this command can be used repetitively if the user wants to inspect several curves at once.

- `p` *filename* - view the point(s) contained in the file *filename*. Note: this command line option can be used repetitively if the user wants to inspect several surfaces at once.

- `r` *integer* set surface refinement factor (number of facets in each direction on the surface). Default value is 100. Higher values gives smoother drawing of the surface. NB: this option has to *precede* the 's' option!

- `e` *string* the string contains keypresses to execute directly upon start (see the section on keyboard control keys for details).

- `hotkeys` does not start the viewer, but displays a list of keyboard commands that can be used when viewing.

A file can contain one or several curves, or one or several surfaces. Files containing both curves and surfaces are not supported. The viewer can read several files to be viewed at once. On the command line, each "curve" file should be preceded with the letter 'c', and each "surface" file should be preceded with the letter 's'. After launch, all the objects contained in the given files are shown simultaneously. The user can disable the view of certain curves and surfaces if he or she wants to.

4.3 User controls

After program launch, the viewing of curves and surfaces can be controlled with the mouse and keyboard. The mouse is used to define viewing angle, direction and zoom factor, while keyboard keys are used to turn on/off objects and to change certain view parameters.

4.3.1 Mouse commands

It is assumed that a 3-button mouse is used. By dragging the mouse while holding down the *left button*, the user can rotate the current view in an intuitive way. By dragging with a certain speed, the view will continue to rotate even after the left button is released. The *middle button* is used for zooming. Hold down this button and move the mouse forwards and backwards in order to zoom in and out. Holding down the *right button* while dragging the mouse moves the view up and down.

4.3.2 Keyboard commands

The available keyboard commands are:

- `q` - quit the viewer program
- `<space>` - change the currently active curve (cycles through each of them)
- `<tab>` - change the currently active surface (cycles through each of them)
- `w` - turn on/off the wireframe display for surfaces
- `B` - toggle between black and white color for backgrounds
- `A` - toggle drawing of coordinate axes on/off
- `S` - toggle drawing of surfaces
- `e` - toggle visibility of currently active surface
- `a` - make all loaded surfaces visible
- `d` - hide all surfaces except the currently active one
- `<ctrl>-e` - toggle visibility of currently active curve
- `<ctrl>-a` - make all loaded curves visible
- `<ctrl>-d` - hide all curves except the currently active one
- `O` - center all objects around origo, and rescale objects so that they fit inside the unit volume (does not preserve aspect ratio)
- `o` - center all objects around origo, no rescaling
- `+` - increase thickness of axes
- `-` - decrease thickness of axes
- `>` - increase size of points
- `<` - decrease size of points
- `/` - decrease length of axes
- `<esc>-w-[n]` - store viewpoint in slot [n], where [n] is a number from 0 to 9. The viewpoint will be saved to file, and can such be preserved from one session to another.
- `<esc>-r-[n]` - load a previously saved viewpoint from slot [n], where [n] is a number from 0 to 9.

Appendix A

The GNU General Public License

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer

of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy,

modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to dis-

No Warranty

SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> one line to give the program's name and a brief idea of what it does.
> Copyright (C) yyyy name of author
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) yyyy name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
> This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> signature of Ty Coon, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.