

XISL—A Development Tool for Construction of Implicit Surfaces

Július Parulek* and Pavol Novotný†
Faculty of Mathematics, Physics and Informatics
Comenius University
Mlynska Dolina, Bratislava, Slovakia

Miloš Šrámek‡
Austrian Academy of Sciences
Donau-City-Strasse 1
A-1220 Vienna, Austria

Abstract

This paper presents the XISL (XML based scripting of Implicit Surfaces) package aimed to develop applications based on implicit surfaces. First, we demonstrate our implicit surface library written in C++. The system of implicit primitives includes analytic, geometric, convolution and other special implicit objects. The operations comprise unary, binary and their generalization— n -ary operations. Second, we propose an XML based language to script arbitrary implicit surfaces. The simple API syntax allows developers easily include XISL to their applications. Moreover, the XISL package contains tools for processing and converting of our XML implicit surfaces to other formats.

CR Categories: I.3.4 [COMPUTER GRAPHICS]: Graphics Utilities—Graphics packages; I.3.6 [COMPUTER GRAPHICS]: Methodology and Techniques—Languages;

Keywords: implicit surfaces, geometrical modeling, C++ API library, script language

1 Introduction

Implicit surfaces (*implicit*s) are useful geometric modeling tools for image synthesis and computer-aided geometric design. The set of techniques, known today as implicit modeling, was used for the first time by Blinn [Blinn 1982]. Currently, there are several types of implicit modeling systems that are oriented towards specific classes of objects.

The first class stands for skeletal models. A simplest skeleton is formed from a set of isolated points. Implicit surfaces built up from skeletal points are also known as *blobs* [Blinn 1982], *soft objects* [Wyvill et al. 1986] and *metaballs* [Nishimura et al. 1985]. Techniques presented by point skeletons were later extended by lines, curves and polygons and generalized to *convolution surfaces* [Bloomenthal and Shoemake 1991; Bloomenthal and Wyvill 1990].

The second class is represented by implicit surfaces defined by analytic functions. These include algebraic (e.g. plane, quadrics) and non-algebraic functions (e.g. superquadrics) [Barr 1981].

Pasko *et al.* generalized the representation of implicit, by combination of the aforementioned models [Pasko et al. 1995]. The inequality

$$f(x_1, \dots, x_n) \geq 0 \quad (1)$$

is called a *functional representation* or *F-rep* of the geometric object.

To modify a given implicit surface function (*implicit function*), one can use basic unary modifiers that include density change, complement and offsetting [Velho et al. 1998; Pasko et al. 1995]. Other forms of unary operations are affine transformations and deformations (*warps*, [Wyvill and van Overveld 1997; Bloomenthal et al. 1997]).

Complex objects can be created via Constructive Solid Geometry (CSG) operations by Boolean set-theoretic operations. The basic set-theoretic operations can be defined using the *min* and *max* operators. Analytical expressions that approximate these operators were proposed by Ricci [Ricci 1972]. The other analytical definitions of the set-theoretic operations are known as *R-functions* [Shapiro 1991; Pasko et al. 1995].

Implicit surfaces are particularly well suited for construction of blends. A blend is a surface that forms a smooth transition between intersecting surfaces. The blends can be divided into global and local. In general, global blends include linear, hyperbolic and super-elliptic ones [Velho et al. 1998]. Local blends limit the domain in which the blending operation is performed by definition of an extra displacement function which is added to the given set-theoretic operation [Pasko and Savchenko 1994; Dekkers et al. 2004].

Implicit surfaces are suitable for approximation of real world data. Muraki used the blobby model to fit volumetric data [Muraki 1991]. The reconstruction of surface models using the method based on thin plate splines [Duchon 1977] and radial basis interpolants [Floater and Iske 1996] from unparallel slices was explored in several works [Savchenko et al. 1995; Turk and O'Brien 1999; Carr et al. 2001].

Implicit surfaces offer a free-form modeling methodology, and thus several modeling environments for implicit surfaces have been developed [Witkin and Heckbert 1994]. The general progress of computing power and speed has accelerated existing polygonization algorithms [Lorensen and Cline 1987; Bloomenthal 1988], which now allow users to model implicit surfaces interactively and see their results in nearly real-time.

Nevertheless, with all this advancement and progress, implicit surfaces still have not found widespread use. It is caused by several reasons. One reason is that it is difficult to create a desired implicit surface model in an intuitive way. Another reason is lack of efficient methods for representation of complex implicit objects. The third reason is an insufficient practical integration of the existing implicit surface tools to custom applications.

1.1 Problem Statement

The topic of this paper is to specify a way for storing, processing and design of different kinds of implicit models. The problem can be stated not only for implicit surfaces but can be generalized to any modeling environment. We can say that there are three basic techniques to create a scene composed of various objects.

First, a GUI modeler can be written. Such a technique allows users to create models interactively. Nevertheless, in the case of general implicit surfaces, design of efficient GUI modelers is currently still a challenge.

Second, a full programming language can be used. This provides considerable possibilities especially in the case when the API library is well written. However, the build cycle (compile, link and run) using a programming language is not a suitable solution for efficient development of geometric models. The alternative way is to use an interpreted programming language. The drawback is that interpreted programming languages suffer from slow computation.

*e-mail: julius.parulek@savba.sk

†e-mail: novotny@sccg.sk

‡e-mail: milos.sramek@oeaw.ac.at

Third, a declarative text file format can be used to define the complex models. It is the most common approach. The problem with this technique is that it is difficult for beginners to learn the syntax. To make it easier, it is useful to adopt a well-accepted standard when developing the file format and eventually to use a GUI frontend for the file specification.

Our contribution is based on the third approach. We propose a new, simple and extensible language, based on the XML technology aimed at definition of complex implicit models.

The remainder of this paper is organized in the following manner. Related work is described in Section 1.2. Section 2 gives an overview of our library of implicit functions and introduces XISL language. The basic XISL functionality is discussed in Section 3. A brief description of XISL application to current research is discussed in Section 4. The availability and basic requirements are described in Section 5. Finally, conclusion and future work are discussed in Section 6.

1.2 Related work

A powerful development system *BlobTree* aimed at implicit surfaces was proposed by Wyvill *et al.* [Wyvill *et al.* 1999]. They organized blending, warping and Boolean operations on implicit surfaces into a CSG system. They structured implicit surface models as a hierarchical combination of primitives organized in a tree. Its leaves are characterized as skeletal primitives (generated by skeletal elements) and the nodes stand for Boolean, blending and warping operators. The evaluation of such a function is achieved by recursive traversal of the tree and evaluation of the field functions at each node. Specification of complex models is achieved using a scripting language. They use Python for the description and rendering of models and scenes. Due to the fact that Python is an interpreted programming language, rendering and processing of such models is rather slow. Orientation of the *BlobTree* system only to skeletal based surfaces can be thought of as a drawback, however working only with skeletons has enabled to build an interactive modeler.

Based on the theory of functional representation [Pasko *et al.* 1995], Adzhiev *et al.* built a modeling system architecture—Hyperfun [Adzhiev *et al.* 1999]. Hyperfun is a modeling language designed as a high level programming tool that provides for construction of *F-rep* objects by means of a library, which contains definitions of primitives, operations and relations. The HyperFun interpreter parses an input text file, which contains a symbolic function evaluation process, and gives a single function as an output.

Some support of implicits is also available in the *vtk* toolkit [Schroeder *et al.* 1996] including objects of the quadric subclass and several other primitive classes. Another tool that supports implicit surfaces is The Persistence of Vision Ray-Tracer (POV-Ray) [Povray 1996]. It reads in a text file containing information describing the scene (objects, lighting and a camera) and generates an image by ray-tracing. However, it is only suitable for rendering purposes.

2 XISL overview

The main contribution of this paper is introduction of the XISL (XML based scripting of Implicit Surfaces) package assisting developers in construction of arbitrary implicit models using a declarative textual language. The package supports properties, which we find to be important from the point of view of our experience.

First, the user should clearly and easily understand the given complex implicit function from its definition (declarative text file). The format specification thus must be very intuitive and robust. A possibility should exist to build complex implicit objects using previously defined ones (eventually in distinct files).

Second, the API of the supporting library should also be intuitive and simple. Its functionality must encapsulate both input file parsing and processing of arbitrary input functions (objects). The user should be able to easily extend the package by custom objects.

The XISL package provides tools to construct various classes of implicit surfaces and fulfills the aforementioned requirements (Fig. 1). In XISL, the implicit models are described in declarative

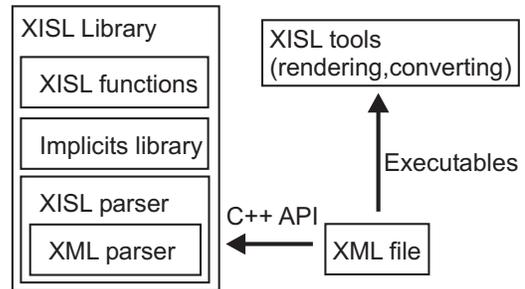


Figure 1: The basic concept of the XISL library.

text files by means of the extensible markup language (XML). The library is implemented in C++, which is well accepted among computer graphics developers and, moreover, integrates with the XML technology very well. The basic package includes methods for parsing the XML files into a library of implicit functions that represent the largest part of the XISL package. The tools stand for voxelization and polygonization of implicit objects, which can be used to convert the implicit representation to volumetric or polygonal one for rendering and other purposes.

2.1 Library of implicit functions

Each implicit function is represented via an n -ary hierarchical tree, leaves of which stand for arbitrary implicit primitives and inner nodes stand for unary, set-theoretic, blending and interpolation operations (Fig. 2). Each tree node corresponds to a C++ implicit function

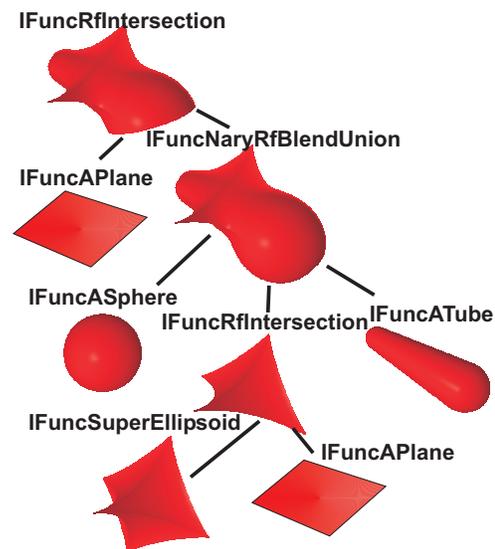


Figure 2: The n -ary tree represents the construction of a complex object by means of the XISL library of implicit functions.

class, which is derived from the base class *IFunction*:

```

class IFunction {
public:
    virtual float    eval(const Vec3f& x) = 0;
    virtual Vec3f    grad(const Vec3f& x);
    ...
}

```

The member `eval()` is a pure virtual method of the `IFunction` class and must be implemented by all subclasses. The `eval()` method defines implicit function by the following equation

$$f(\mathbf{x}) \geq 0, \quad (2)$$

where $\mathbf{x} = (x_1, x_2, x_3) \in E^3$. The `grad()` member computes gradient of the implicit function by means of central differences.

We organize all implicit functions in a C++ class hierarchy. The pure abstract classes representing various categories of implicits (derived directly from `IFunction`) are demonstrated in Fig. 3.

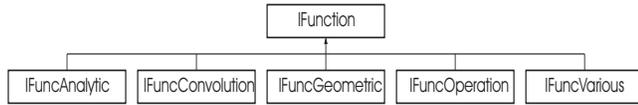


Figure 3: Abstract classes, derived directly from the class `IFunction`, represent possible XISL categories of implicits.

2.1.1 Analytic implicits

The derivatives of the *Analytic class* include algebraic and non-algebraic implicit functions. The algebraic objects are represented by a list of coefficients required polynomial. The polynomial of the first order stands for a plane (`IFuncAPlane`) and the polynomials of the second order represent quadrics (e.g. `IFuncACone`, `IFuncAEllipsoid`). The non-algebraic classes stand for super-shapes [Gielis et al. 2003] that are obtained by exponentiation of quadrics (e.g. `IFuncSuperEllipsoid`, `IFuncSuperToroid`).

2.1.2 Geometric implicits

Implicits of the *Geometric class* are defined by functions that return a signed distance to the surface. It provides computation of the exact Euclidean measure to a surface, which is a useful tool in modeling real world structures. The basic geometric objects comprise offset surfaces and geometric quadrics.

Given a skeletal primitive P , an offset surface (e.g. `IFuncGSphere`, `IFuncGTube`) is defined as

$$f(\mathbf{x}) = r(\mathbf{x}') - d(\mathbf{x}, \mathbf{x}'), \quad (3)$$

where \mathbf{x}' is a nearest point to \mathbf{x} on P , r defines the width of P in \mathbf{x}' , and d returns the Euclidean distance between the point \mathbf{x} and the point \mathbf{x}' .

Representation of quadrics as distance surfaces was proposed by Hart [Hart 1996] (e.g. `IFuncGCone`, `IFuncGTorus`).

2.1.3 Convolution implicits

The *Convolution class* includes implicit functions based on convolution of a kernel function and skeleton primitives:

$$f(\mathbf{p}) = \int_{R^3} S(\mathbf{p})h(\|\mathbf{s} - \mathbf{p}\|)ds, \quad (4)$$

where h is a kernel function and $S: \mathbf{R}^3 \rightarrow \mathbf{R}$ represents the geometry of a modeling primitive P , $S(\mathbf{p}) = 1$ if $\mathbf{p} \in P$ and $S(\mathbf{p}) = 0$ otherwise.

We provide two types of kernel functions. The first one, Cauchy kernel has an infinite domain of the influence, and is defined as

$$h(r) = \frac{1}{(1 + s^2 r^2)^2}, \quad (5)$$

where r is the distance from a given point and s controls the width of the kernel. McCormack and Sherstyuk enumerated convolution (4) of this kernel (5) with basic skeletal primitives [McCormack and Sherstyuk 1998]. We adopt two sets of skeletal primitives—points and lines (`IFuncConvPointCK`, `IFuncConvLineCK`). Jin *et al.* [Jin et al. 2001] presented an analytical solution for convolving line-segment skeletons with a variable Cauchy kernel modulated by a polynomial function (`IFuncConvLineWeightCK`).

The second one, quartic kernel, with a finite domain of the influence, is defined as

$$h(r) = \begin{cases} \left(1 - \frac{r^2}{R^2}\right)^2 & r \leq R \\ 0 & r > R \end{cases}, \quad (6)$$

where R is the effective radius of the kernel. Jin and Tai proposed an analytical solution of convolution (4) for varying quartic kernel and line-segments in [Jin and Tai 2002] (`IFuncConvPointQK`, `IFuncConvLineQK`, `IFuncConvLineWeightedQK`).

An important property that makes convolution surfaces suitable for modeling is superposition. Superposition implies that summing (blend union) of the convolution surfaces generated by two separate skeletons yields the same surface as that generated by convolution from the combined skeleton.

2.1.4 Various implicits

The derivatives of *Various class* stand for non-categorized implicit primitives that can be used specifically.

Surface approximation from the real world data is one of the practical employment of implicit surfaces. Savchenko *et al.* [Savchenko et al. 1995] proposed a method based on a so-called *carrier solid*, which interpolates scattered points in $2D$ or $3D$. The general idea of their approach is to introduce the carrier solid with a defining function f_c and to construct variational implicit function f_v that interpolates values of the function f_c in the given points. Sequentially, the algebraic difference between f_v and f_c describes a reconstructed solid.

To avoid the precondition of the carrier solid, we have also implemented the method proposed by Turk and O'Brien [Turk and O'Brien 1999]. This method uses so-called offset constraints that represents non-zero weight points.

Another subclass that belongs to the *various class* consists of planar implicit shapes. The $2D$ implicit shapes can be transformed to $3D$ implicits by operations of interpolation or extruding. The useful technique of creating planar implicit shapes is the method of implicit polygons [Pasko et al. 1996]. This method represents a non-self intersecting polygon by the implicit function that takes zero values at polygon edges. Other possibility is to use a method of scattered points interpolation in $2D$.

2.1.5 Operation classes

The class of operations includes basic CSG operations (union, intersection and subtraction), blending operations, blending as extension to CSG operations, unary affine transformations, deformations and interpolations. There are three basic container classes that stand for unary, binary and n -ary abstract classes.

```

class IFuncUnaryOp : public IFuncOperation
{
protected:
    IFunction* child;
    ...
};
class IFuncBinOp : public IFuncOperation
{
protected:
    IFunction* child[2];
    ...
};
class IFuncNaryOp : public IFuncOperation
{
protected:
    IFuncVector child;
    ...
};

```

Each operation has to be derived from one of these classes. We divide the CSG operations into two categories.

The first category is defined by means of the *min* and *max* operators

$$\begin{aligned} \text{union}(f_1, f_2) &= \max(f_1, f_2) \\ \text{intersect}(f_1, f_2) &= \min(f_1, f_2). \end{aligned} \quad (7)$$

This class is useful to define sharp connections of intersecting geometric objects. The drawback is that the functions (7) have a C^1 discontinuity where $f_1 = f_2$. The blending form of equations (7) is obtained by the addition of the displacement term:

$$D(x = |f_1 - f_2|, n) = \begin{cases} n(\frac{x}{n} - \frac{1}{4})^2 & \text{for } x < \frac{n}{4} \\ 0 & \text{for } x \geq \frac{n}{4} \end{cases} \quad (8)$$

that is based on [Dekkers et al. 2004]. The positive feature is that one can determine the size of the blending domain.

The second category of CSG operations is defined as an analytical approximation by means of *R-functions*. We have implemented the in practice most useful form of *R-functions*:

$$\begin{aligned} \text{union}(f_1, f_2) &= f_1 + f_2 + \sqrt{f_1^2 + f_2^2} \\ \text{intersect}(f_1, f_2) &= f_1 + f_2 - \sqrt{f_1^2 + f_2^2}. \end{aligned} \quad (9)$$

The functions (9) have C^1 discontinuity only in points where $f_1 = f_2 = 0$. The blending form is defined by the displacement term [Pasko et al. 1995]:

$$D(f_1, f_2) = \frac{a_0}{1 + \left(\frac{f_1}{a_1}\right)^2 + \left(\frac{f_2}{a_2}\right)^2}, \quad (10)$$

where a_0 defines the displacement of the blending surface with respect to both input objects ($a_0 < 0$ for subtraction and $a_0 > 0$ for addition), and values $a_1 > 0$ and $a_2 > 0$ influence the distance between the blended surface and the original surfaces of objects f_1 and f_2 respectively.

Further, *n*-ary operations comprise summation, interpolation and generalization of the aforementioned CSG operations.

Unary operations include affine transformations (translation, rotation, scaling), deformations (twist, bend, etc.) and operations that modifies value of the implicit function:

$$\begin{aligned} \text{density}(f) &= d * f(\mathbf{p}) \\ \text{complement}(f) &= -f(\mathbf{p}) \\ \text{offset}(f) &= f(\mathbf{p}) + c, \end{aligned} \quad (11)$$

where $d, c \in \mathbf{R}$ ($d > 1$ stands for a density expansion, $d < 1$ for a compression, $c > 0$ represents a dilatation and $c < 0$ represents an erosion).

All the operations derived from the *IFunction* class are also subclasses of *IFunction* class and can be used as an input for other operations.

2.2 The XISL language

XISL language is based on XML, which is a well defined and widely used industrial standard. Moreover, XML is a simple way to delimit text data.

Each implicit function class (a primitive, an operation, etc.) is defined by its appropriate XISL tag(s). This ensures the requirement of clear notation of complex implicits. With respect to the fact that XML has its own strict tag rules, the script code is primarily validated by the XML parser itself. The script in Figure 4 demonstrates the XML notation of a compound implicit object (Fig. 2, Fig. 5).

```

<intersectionRf>
  <blendedUnionRf a0="0.8" a1="0.3" a2="0.3">
    <aSphere>
      <vec4 x1="0" x2="0" x3="0" x4="1.3"/>
    </aSphere>
    <aTube>
      <vec4 x1="0" x2="0" x3="0" x4="0.8"/>
      <vec4 x1="0" x2="4" x3="0" x4="0.5"/>
    </aTube>
    <intersectionRf>
      <superEllipsoid>
        <vec3 x1="0" x2="2" x3="0"/>
        <vec3 x1="2.5" x2="2.5" x3="2.5"/>
        <vec2 x1="3.3" x2="2.3"/>
      </superEllipsoid>
      <aPlane>
        <vec3 x1="0" x2="2" x3="0"/>
        <vec3 x1="1" x2="0" x3="0"/>
      </aPlane>
    </intersectionRf>
  </blendedUnionRf>
  <aPlane>
    <vec3 x1="0" x2="0" x3="0"/>
    <vec3 x1="0.5" x2="0.5" x3="0"/>
  </aPlane>
</intersectionRf>

```

Figure 4: The XML script defines the compound implicit object.

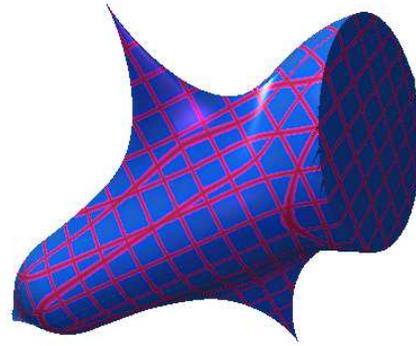


Figure 5: Image of the implicit object that is defined in the figure 4.

The tag language is kept very simple. It supports common implicit primitives, operations, interpolations and transformations, which gives rise to a high number of possibilities in modeling. Besides, XISL tag language supports all implicit functions described in Section 2.1, e.g. Analytic implicits — `<aPlane>`, `<superEllipsoid>`,..., Geometric implicits — `<gTube>`, `<gTorus>`,... Convolution implicit `<convLineCK>`, `<convLineWeightedQK>`,... Various implicits — `<visCS3D>`, `<implicitPg>`,... Operations — `<translate>`, `<sum>`, `<blendedUnion>`,...

2.2.1 Object reference

Each object in XISL has to get assigned its reference name in a XML file using the tag `<defObject name="some_name"/>`. The system also enables to use previously defined objects by calling its reference name `<getObject name="some_name"/>` (Figure 6). Furthermore, it is possible to include various files into a single file project using the tag `<includeXMLFile fileName="test.xml"/>`.

```
<defObject name="A">
  <sum T="0.5">
    <convLineCK>
      <vec4 x1="0" x2="0" x3="0" x4="0.3"/>
      <vec4 x1="0" x2="5" x3="0" x4="0.2"/>
    </convLineCK>
  </sum>
</defObject>
<defObject name="B">
  <blendedUnionRf a0="0.45" a1="0.3" a2="0.3">
    <getObject name="A"/>
    <rotation x="0" y="0" z="40">
      <getObject name="A"/>
    </rotation>
    <rotation x="0" y="0" z="-40">
      <getObject name="A"/>
    </rotation>
  </blendedUnionRf>
</defObject>
```

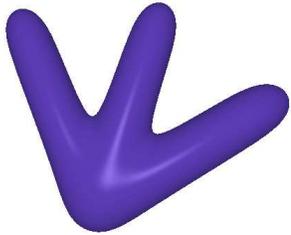


Figure 6: Top: the XML Script defines two implicit objects A and B, where B is defined by means of the object A. Bottom: depicted implicit object B.

3 XISL functionality

The whole XISL package is written in C++. Library itself includes the classes of implicit surfaces, provides high-level C++ API for parsing of XISL tags, voxelization and polygonization of `IFunction` class and its derivatives.

3.1 XISL device

The basic C++ code for obtaining implicits from an XML file is very simple. There is a single pure abstract class called

`xislDevice` that provides the whole XISL functionality. The following code demonstrates the basic routines that have to be called for loading and processing (polygonization) a given `test.xml` file.

```
#include "xisl.h"
#include <vector>
using namespace xisl;
int main()
{
  std::vector<TRIANGLE_VEC> triangles, normals;
  xislDevice* xislDev = new createXISLDevice();
  if (xislDev->loadXMLFile("test.xml"))
  {
    xislDev->parseFile();

    IFunction* func1 = xislDev->getIFunctionByName("A");

    xislDev->polygonize(func1, Vec3i(40, 40, 40),
                      triangles, normals);
  }
  delete xislDev;
}
```

Each parsed implicit function can be called by its reference name (`xislDevice::getIFunctionByName(.)`). The other methods provide voxelization and polygonization of implicit surfaces. Polygonization currently includes two methods.

The first is the Bloomenthal's polygonization [Bloomenthal 1988] method that adaptively samples an implicit function by a spatial partitioning. The partitioning is represented by an octree, which may either converge to the surface or track it. The octree converges to the surface by subdivision of those cubes that intersect the surface. The surface tracking is established from the *seed cell* that intersects the surface. New cells propagate along established cell edges that intersect the surface. This method may fail in the case when the implicit surface is divided into several isolated surfaces.

The second is based on the marching cube algorithm [Lorenson and Cline 1987] that uniformly processes the whole implicit function domain.

3.2 Voxelization

Voxelization is conversion from a geometric representation of a synthetic model into a grid of voxels. There are several ways how to perform voxelization of an implicit function f . The simplest method is to write directly the value of f into each voxel. The drawback of this approach is that the whole volume must be filled by the function values, and so both the processing and memory requirements are very high.

As we are typically interested just in the surface of the implicit solid, it is reasonable to compute and store values of the implicit function only in voxels situated in the surface vicinity—those which are necessary for the surface reconstruction. This idea was presented in [Sramek and Kaufman 1999; Sramek and Kaufman 2000] where voxelization of implicit solids is based on the *truncated distance fields (TDFs)* representation for volume storage. The distance $d(\mathbf{x})$ of a point $\mathbf{x} = (x_0, x_1, x_2)$ to the surface can be approximated here using the formula

$$d(\mathbf{x}) = \frac{f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}, \quad (12)$$

where $\|\nabla f(\mathbf{x})\|$ is the gradient magnitude. In the TDFs representation, density $D(\mathbf{x})$ stored in the volume depends on the distance $d(\mathbf{x})$ as follows:

$$D(\mathbf{x}) = \begin{cases} 1 & \text{for } d(\mathbf{x}) < -r \\ 0 & \text{for } d(\mathbf{x}) > r \\ \frac{1}{2} \left(1 - \frac{d(\mathbf{x})}{r}\right) & \text{elsewhere} \end{cases}, \quad (13)$$

where r is radius of the transitional area—a thin layer of voxels located near the object surface. The linear approximation (12) is acceptable provided that f is C^1 continuous, which is not the case in edge areas of many objects. So, this method is suitable for objects with smooth surfaces, which do not contain any sharp features. If we try to voxelize solids with sharp details, such as edges and vertices, they result in disturbing artifacts (jaggy edges) clearly visible, e.g., in rendered images (Figure 7a).

A technique called *Sharp Details Correction (SDC)* was presented in [Novotny and Sramek 2005] to address the problem of objects with sharp features. The goal of this method is to voxelize these solids using TDFs in a way to get correct data, i.e., the reconstruction error should be suitably bounded. It is known from the sampling theory that in the discrete space it is possible to capture data details just up to a certain level. In other words, the highest feasible frequency is given by the sampling density. As edges are details with unbounded frequency, it is not possible to represent them in discrete data. As we have mentioned, if we try to voxelize the objects with edges anyway, we encounter the problem of jaggy edges. A more correct approach is to modify data during the voxelization process to make edges suitably rounded—their curvature should correspond to the maximal permissible frequency, which is given by a representability criterion stated in [Bærentzen et al. 2000].

In [Novotny and Sramek 2005] the problem is solved by dividing the voxelization into two stages. In the first one values of voxels are computed in the whole volume in a standard way [Sramek and Kaufman 1999] and *critical voxels* (those located in the vicinity of sharp details) are marked. In the second stage the critical voxels acquire new values according to the representability criterion by extrapolating voxel values from their non-critical neighbors using a modified version of the Fast Marching Method (FMM) [Bærentzen 2001] and by applying ideas of the paper [Novotny et al. 2004], where CSG operations with voxelized solids were discussed. Solids voxelized using this method do not produce disturbing artifacts in rendered images, presented in Figure 7b.

3.3 Tools

The XISL package includes tools for polygonized preview (`xislViewer`) of implicits defined in XML files (Figure 8 left). Along the XISL viewer the package contains the conversion tools to the POV-Ray format (`xisl2pov` — Figure 8 middle) and volumetric representation in `f3d` format (`xisl2f3d`, `xisl2f3dMulti` — Figure 8 right)[Sramek and Dimitrov 2002]. The POV-Ray support is based on the version 3.0 with the Suzuki’s POV-Ray isosurface patch [Suzuki 1999]. The volumetric model, depicted in Figure 8, is rendered using `f3dvr` [Červeňanský 2004].

4 Application to current research

Nowadays, methods based on implicit surfaces provide a useful platform for modeling of various biological structures. Up to present time, we have used the XISL library in two projects from the area of biological modeling.

First, we have used the XISL library to build a heart surface model from unparallel sets of slices [Parulek and Zimanyi 2005]. First, the *carrier solid* (implicit ellipsoid) f_c is constructed that approximates the given set of slices. Second, variational implicit function f_v is created that interpolates values of f_c at slices contours. The final heart implicit function (Figure 9) is as follows:

$$f(\mathbf{x}) = f_v(\mathbf{x}) - f_c(\mathbf{x}). \quad (14)$$

Second, we have developed geometrical modeling tools for creation of arbitrary muscle cell models (Figure 10) [Parulek et al.

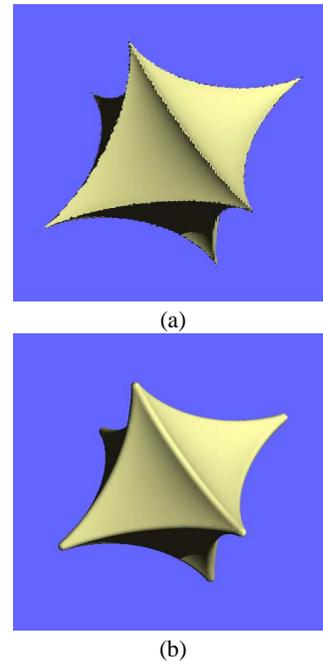


Figure 7: Voxelization of solids with sharp details. Images of volume data obtained by (a) standard voxelization (b) SDC method. Object: superellipsoid. Rendering method: raycasting of the surface reconstructed from the volume data by interpolation and thresholding.

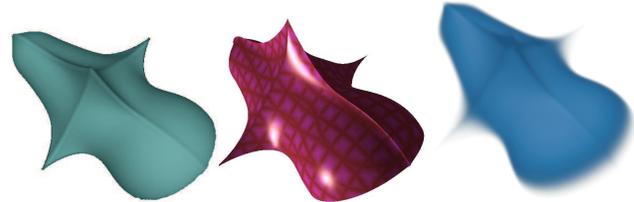


Figure 8: Rendering of the implicit surface defined in Figure 4. Left: The polygonized model preview. Middle: Direct raytracing of the implicit surface using POV-Ray. Right: The volumetric model (`f3d` format) obtained by direct voxelization. Real time rendering technique.

2004]. Due to substantially different size, shape and topology of the modeled organelles, the organelles were classified from the point of modeling as *implicit polygon based* and *skeleton based* organelles. The first class of organelles was based on the interpolation of planar implicit polygons. The second class was represented as a distance based surfaces to a set of skeletal primitives (points, lines).

5 Availability and requirements

The XISL library is provided as a free package currently working under Windows and Linux systems (<http://www.confolab.sav.sk/parulek/xisl/>). The TinyXML (<http://tinyxml.sourceforge.net/>) library is used as a basic XML parser. This library is already included as a part of the XISL package, therefore it is not required to be installed separately. The `xislViewer` tool requires the GLFW library (<http://glfw.sourceforge.net/>) to be installed. This library

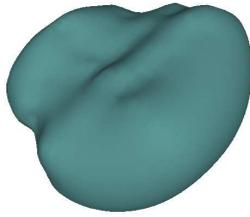


Figure 9: *The polygonized heart model.*

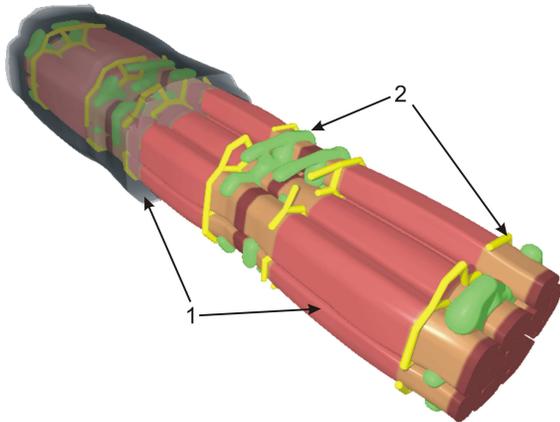


Figure 10: *The muscle cell model. Organelles are classified according to modeling process as implicit polygon based (1) and skeleton based (2) organelles.*

constitutes powerful API for handling operating system specific tasks, such as opening an OpenGL window and reading keyboard, mouse and joystick input. The f3d file format (<http://www.viskom.oeaw.ac.at/~milos/page/Download.html>) is required to store the voxelized models.

6 Conclusions and future work

We have proposed the library for handling of implicit surfaces. We have illustrated the design of a C++ class of implicit functions. Implicit surfaces were structured to n -ary hierarchical tree, where leaf stand for primitives and inner nodes correspond to operations. We have developed an object description language that is capable to work with our implicit function library. This language is based on XML that we have found to be very suitable for purposes of storing and scripting of implicits. The XISL package is provided with the tools that are capable of converting to the volumetric f3d format, polygons and to POVray format.

Because the important application area for the XISL package should be modeling of biological structures, the next goal will be development of high-level tag commands for the modeling of biological structures. Such an approach should also involve the multi-object scene support. To provide collision manager for generating of object-to-object interactions.

This work has been partly supported by a VEGA grant Real-Time Rendering and Complexity of Geometric Algorithms for Virtual Reality No. 1/3083/06 and VEGA grant No. 2/6079/26.

References

- ADZHIEV, V., CARTWRIGHT, R., FAUSETT, E., OSSIPOV, A., PASKO, A., AND SAVCHENKO, V. 1999. Hyperfun project: A framework for collaborative multidimensional f-rep modeling. In *Proc. of the Implicit Surfaces '99 EUROGRAPHICS/ACM SIGGRAPH Workshop*, 59–69.
- BÆRENTZEN, J. A., SRAMEK, M., AND CHRISTENSEN, N. J. 2000. A morphological approach to voxelization of solids. In *The 8-th International Conference in Central Europe on Computer Graphics, Visualization and Digital Interactive Media 2000*, 44–51.
- BÆRENTZEN, J. A. 2001. On the implementation of fast marching methods for 3D lattices. Tech. rep., Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby.
- BARR, A. 1981. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications* 1, 1, 11–23.
- BLINN, J. 1982. A generalization of algebraic surface drawing. *ACM Transactions on Graphics* 1, 235–256.
- BLOOMENTHAL, J., AND SHOEMAKE, K. 1991. Convolution surfaces. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, ACM Press, 251–256.
- BLOOMENTHAL, J., AND WYVILL, B. 1990. Interactive techniques for implicit modeling. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics*, ACM Press, 109–116.
- BLOOMENTHAL, J., BAJAJ, C., BLINN, J., CANI-GASCUEL, M.-P., ROCKWOOD, A., WYVILL, B., AND WYVILL, G. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufman Publisher Inc, San Francisco, California.
- BLOOMENTHAL, J. 1988. Polygonization of implicit surfaces. *Comput. Aided Geom. Des.* 5, 4, 341–355.
- CARR, J. C., BEATSON, R. K., CHERRIE, J. B., MITCHELL, T. J., FRIGHT, W. R., MCCALLUM, B. C., AND EVANS, T. R. 2001. Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 67–76.
- DEKKERS, D., VAN OVERVELD, K., AND GOLSTEIJN, R. 2004. Combining csg modeling with soft blending using lipschitz-based implicit surfaces. *Vis. Comput.* 20, 6, 380–391.
- DUCHON, J. 1977. *Splines minimizing rotation-invariant seminorms in Sobolev spaces*. Springer-Verlag, ed. Walter Schempp and Karl Zeller, Berlin-Heidelberg.
- FLOATER, M. S., AND ISKE, A. 1996. Multistep scattered data interpolation using compactly supported radial basis functions. *JCAM*; 73, 65–78.
- GIELIS, J., BEIRINCKX, B., AND BASTIAENS, E. 2003. Superquadrics with rational and irrational symmetry. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, ACM Press, New York, NY, USA, 262–265.
- HART, J. C. 1996. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer* 12, 10, 527–545.

- JIN, X., AND TAI, C.-L. 2002. Convolution surfaces for arcs and quadratic curves with a varying kernel. *The Visual Computer* 18, 8, 530–546.
- JIN, X., TAI, C.-L., FENG, J., AND PENG, Q. 2001. Convolution surfaces for line skeletons with polynomial weight distributions. *J. Graph. Tools* 6, 3, 17–28.
- LORENSEN, W., AND CLINE, H. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, vol. 21, 163–169.
- MCCORMACK, J., AND SHERSTYUK, A. 1998. Creating and rendering convolution surfaces. *Comput. Graph. Forum* 17, 2, 113–120.
- MURAKI, S. 1991. Volumetric shape description of range data using blobby model. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 227–235.
- NISHIMURA, H., HIRAI, M., KAVAI, T., KAWATA, T., SHIRAKAWA, I., AND OMURA, K. 1985. Object modeling by distribution function and a method of image generation. *Transactions of IECE J68-D*, 4, 718–725.
- NOVOTNY, P., AND SRAMEK, M. 2005. Representation of objects with sharp details in truncated distance fields. In *International Workshop on Volume Graphics*, IEEE Computer Society, Stony Brook, NY, USA, E. Groeller, I. Fujishiro, K. Mueller, and T. Ertl, Eds., 109–116, 233.
- NOVOTNY, P., DIMITROV, L. I., AND SRAMEK, M. 2004. Csg operations with voxelized solids. In *Proceedings of the Computer Graphics International 2004*, IEEE Computer Society Press, Heraklion, Crete, Greece, B. Werner, Ed., 370–373.
- PARULEK, J., AND ZIMANYI, M. 2005. Heart reconstruction from non-parallel slices using implicit surfaces. In *Conference Materials and Non-Refereed Posters*, UNION Agency - Science Press, Budmerice, Slovakia, 35–36.
- PARULEK, J., ZAHRADNIK, I., AND SRAMEK, M. 2004. A modelling tool for construction of 3d structure of muscle cells. In *Analysis of Biomedical Signals and Image proceedings*, J. Jan, J. Kozumplik, and I. Provaznik, Eds., 267–269.
- PASKO, A. A., AND SAVCHENKO, V. V. 1994. Blending operations for the functionally based constructive geometry. In *CSG 94 Set-theoretic Solid Modeling: Techniques and Applications*, 151–161.
- PASKO, A. A., ADZHIEV, V., SOURIN, A., AND SAVCHENKO, V. V. 1995. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer* 11, 8, 429–446.
- PASKO, A., SAVCHENKO, A., AND SAVCHENKO, V. 1996. Polygon-to-function conversion for sweeping. In *The Eurographics/SIGGRAPH Workshop on Implicit Surfaces*, J. Hart, K. van Overveld (Eds.), 163–171.
- POVRAY, 1996. Povray - the persistence of vision ray tracer, <http://www.povray.org/>.
- RICCI, A. 1972. A constructive geometry for computer graphics. *The Computer Journal* 16, 2, 157–160.
- SAVCHENKO, V. V., PASKO, A. A., OKUNEV, O. G., AND KUNII, T. L. 1995. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum* 14, 4, 181–188.
- SCHROEDER, W. J., MARTIN, K. M., AND LORENSEN, W. E. 1996. The design and implementation of an object-oriented toolkit for 3d graphics and visualization. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, IEEE Computer Society Press, Los Alamitos, CA, USA, 93–ff.
- SHAPIRO, V. 1991. Real functions for representation of rigid solids. Tech. rep., Ithaca, NY, USA.
- SRAMEK, M., AND DIMITROV, L. I. 2002. f3d — a file format and tools for storage and manipulation of volumetric data sets. In *1st International Symposium on 3D Data Processing, Visualization and Transmission*. to be published.
- SRAMEK, M., AND KAUFMAN, A. 1999. Alias-free voxelization of geometric objects. *IEEE Transactions on Visualization and Computer Graphics* 5, 3, 251–266.
- SRAMEK, M., AND KAUFMAN, A. 2000. vxt: a c++ class library for object voxelization. In *Volume Graphics*, M. Chen, A. E. Kaufman, and R. Yagel, Eds. Springer Verlag, London, 119–134.
- SUZUKI, R., 1999. Povray 3.0 isosurface patch, <http://www.public.usit.net/rsuzuki/e/povray/iso/>.
- TURK, G., AND O'BRIEN, J. F. 1999. Shape transformation using variational implicit functions. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 335–342.
- ČERVEŇANSKÝ, M. 2004. *Využitie komerčných grafických akceleratorov pre vizualizáciu a spracovanie objemových dát*. Master's thesis, Comenius University Faculty of Mathematics, Physics and Informatics, Bratislava, Slovakia.
- VELHO, L., DE FIGUEIREDO, L. H., AND GOMES, J. A. 1998. *Implicit Objects in Computer Graphics*. Springer-Verlag New York, Inc.
- WITKIN, A. P., AND HECKBERT, P. S. 1994. Using particles to sample and control implicit surfaces. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 269–277.
- WYVILL, B., AND VAN OVERVELD, K. 1997. Warping as a modelling tool for csg/implicit models. In *Shape Modelling Conference, University of Aizu, Japan*, IEEE Society Computer Press ISBN 0-8186-7867-4, 205–214. invited.
- WYVILL, G., MCPHEETERS, C., AND WYVILL, B. 1986. Data structure for soft objects. *The Visual Computer* 2, 4 (Aug), 227–234.
- WYVILL, B., GUY, A., AND GALIN, E. 1999. Extending the csg tree (warping, blending and boolean operations in an implicit surface modeling system). *Computer Graphics Forum* 18, 2, 149–158.