Chapter 3

Modeling with Convolution Surfaces

3.1. INTRODUCTION

Convolution surfaces were introduced to computer graphics by Bloomenthal and Shoemake [12] as a logical generalization of the classic models of implicit surfaces: blobby objects [7], metaballs [50], soft objects [73]. Being a superset of these models, convolution surfaces inherit their valuable properties, such as an ability to form smooth blends and free-form shapes. At the same time, convolution surfaces demonstrate much greater modeling flexibility, allowing a designer to create objects using skeletal-based techniques with skeletal elements of various shapes and sizes.

Such qualities make convolution surfaces particularly attractive for design of articulated objects, especially of organic origin. However, the modeling practices, developed over nearly two decades for 'classic implicits' [7, 50, 73] cannot be carried over to convolution surfaces 'as is', without doing an injustice to the convolution surface model. As a more versatile and more powerful tool, convolution surfaces need a better 'manual of operation' to fully reveal their potential.

To date, practical examples of modeling with implicit surfaces in general, and convolution surfaces in particular, mainly exercise their blending abilities. Consequently, the majority of objects, designed with implicit surfaces, demonstrate various branching structures, such as trees [8, 35], hands [13], paws [60], chromosomes [24]; or exhibit certain softness or fluidity, either in animation or as perceived from static images. Examples are: human lips [30] and faces [45], molecular shapes [7], boiling liquids [74, 75] and rubbery-looking objects [77]. Few attempts have been made to employ implicit surfaces for modeling rough objects. One of them is presented in [33].

Starting from the previous work [42, 65], we show that the modeling capabilities of convolution surfaces extend beyond traditional blends. We introduce a set of primitives and a number of techniques that allow us to sculpture objects, manipulating their shape at all levels of detail, including fine textures. We demonstrate that convolution surfaces can be successfully used to represent not only soft and pliable substances, but also objects that are hard and fragile. For that reason, most examples show marine life forms – there are plenty of creatures of all types. We also outline a system architecture for interactive design with convolution surfaces. The system allows objects to be modified at interactive rates, which helps the design to converge quickly to the final shape.

3.2. DEFINITIONS

The following are the basic concepts and equations used in modeling with implicit surfaces.

3.2.1. Implicit surfaces

An *implicit surface* S is defined as an isosurface at level T in some scalar field $F(\mathbf{p})$:

$$S = \{\mathbf{p} \in R^3 \mid f(\mathbf{p}) - T = 0\} \qquad (3.1)$$

An example of a typical early implicit surface is shown in Figure 2.1.

3.2.2. Convolution surfaces

A *convolution surface* is the implicit surface based on a field function $f(\mathbf{p})$, obtained via convolution of some kernel function h and a geometry function g:

$$f(\mathbf{p}) = g(\mathbf{p}) \star h(\mathbf{p}) = \int_{R^3} g(\mathbf{r})h(\mathbf{p}-\mathbf{r})\, d\mathbf{r} \qquad (3.2)$$

The geometry function $g(\mathbf{p})$ defines the shape of an object and its position in 3D space. The kernel function $h(\mathbf{p})$ defines the distribution of some potential that is produced by each point on the object. Convolved together, these two functions produce a tri-variate scalar function $f(\mathbf{p})$ that defines the convolution surface.

Figure 3.1 gives an example of convolution with a Gaussian-like kernel. Here the geometry function $g(\mathbf{p})$ describes a line segment aligned along the x-axis.

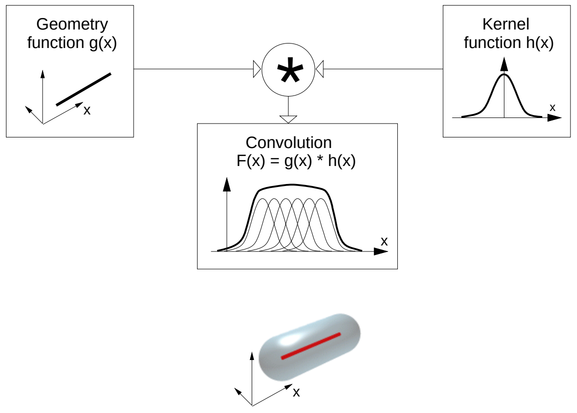

Figure 3.1: Components of a convolution surface: geometry function g (top left), kernel function h (top right), convolution field function f (center), convolution surface (bottom).

3.2.3. Implicit primitives

In the metaballs model of implicit surfaces [50], the point potential sources were introduced as "meta-primitives defined by their distribution function, that together form a meta-surface". With respect to convolution surfaces, we say that the distribution function (3.2) defines an *implicit primitive* with geometry g.

Metaballs, as defined by Nishimura et. al. [50], are essentially meta-spheres (see Figure 2.1). Implicit primitives, produced via the convolution integral (3.2), yield a variety of modeling shapes. For example, a line segment, convolved with a kernel function, produces an elongated shape as shown in Figure 3.1.

Since most modeling primitives form closed compact sets (points, line segments, triangles, etc), the integration (3.2) over 3D space may be conveniently replaced with integration over the volume $\mathbf{V}$ of the modeling primitive:

$$f(\mathbf{p}) = \int_{\mathbf{V}} h(\mathbf{p} - \mathbf{r})\, d\mathbf{r} \qquad (3.3)$$

3.2.4. Skeletons

For the purpose of this thesis, we use the following definition of a skeleton. A *skeleton* is a collection of geometric primitives that outline the inner structure of an object being modeled. With respect to the convolution surface model, a skeleton is a sum of geometry functions $\sum_{i=1}^{N} g_i(\mathbf{p})$. Visually, such a skeleton is represented by a union of corresponding primitives. Convolved with a kernel function, the skeleton yields a field function $f(\mathbf{p})$ and a convolution surface S.

Other researches used slightly different definitions of skeletons [13, 24, 80]. In general, these definitions are similar and imply that an object being modeled may be regarded as composed of distinct components, as opposed to being totally formless. However, the precedence of the skeleton and the shape of the object may be different. In most data-fitting problems [24, 45], skeletons are derived from data associated with the object that has to be visualized. In contrary, for skeletal-based modeling environments [12, 13, 30], the shape of the object follows the skeleton.

The concept of a skeletal design with convolution surfaces was introduced to computer graphics by Bloomenthal and Shoemake in [12]. They observed that the additive property of convolution allows us to build complex skeletons out of simple primitives and, most important, allows us to evaluate them individually. That makes convolution surfaces computationally practical.

Bloomenthal [13] applied convolution surfaces to generate smooth shapes, resembling various organic objects. In what follows, we will use implicit primitives obtained via convolution technique, to produce a wide variety of surfaces, including rough, prickly and wrinkled surfaces.

3.3. THE DESIGN SYSTEM

In this section, we present a set of tools and techniques for modeling with convolution surfaces.

3.3.1. New modeling primitives as skeletal elements

In principle, any geometric primitive may be used as a skeletal element for the convolution surface model by means of the generic integral (3.2). In practice, the choice of such primitives is often limited by technical difficulties of evaluating the convolution integral.

As most implementations of the convolution surface model demonstrate [12, 13, 60], such computations require point-sampling of the field in the model space and storage of intermediate results for rendering[1]. As for techniques that employ point-sampling, special care must be taken to ensure that small features of the object being modeled are not missed. This task may be especially difficult for models that contain primitives of widely varying characteristic sizes.

Closed-form solution for the convolution integral (3.2) provides such a care-free modeling environment. Based

[1]Bloomenthal and Shoemake [12] described how to evaluate the convolution integral (3.2) for polygons. They used a precomputed raster representation of a 2D convolution of a polygon and then stored results as 2D images. Sealy and Wyvill [60] developed a method for computing the convolution integral (3.2) for 3D objects. Their method involves replacing the actual integration with discrete sums over the model space. To reduce the demands on memory, Sealy and Wyvill used octrees for storage of their 3D data.

on our prior results [42], also given in Chapter 2, we suggest the following implicit primitives as skeletal elements for convolution surfaces:

- points
- line segments
- arcs
- triangles
- planes

All these modeling primitives are presented as closed-form functions, that return the amount of field generated by the primitive at an arbitrary point, calculated to a machine-size float precision. This makes it possible to visualize convolution surfaces using direct rendering algorithms, such as ray-tracing. Thus, neither preprocessing, nor intermediate storage are required to render the surface, which is particularly convenient for interactive design. Figure 3.2 shows the implicit primitives, rendered as semi-transparent surfaces with the underlying 'bare' geometric primitives inside. The actual field functions are presented in Chapter 2.

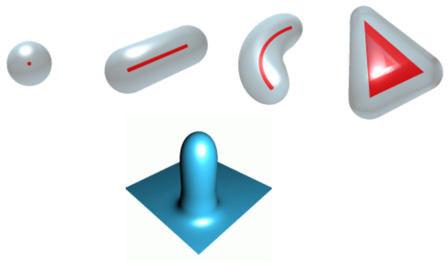
Figure 3.2: A set of implicit primitives: point, line segment, arc, triangle and plane. The plane primitive is clipped and blended with a line segment.

Although most of these primitives have been used for implicit modeling before (points in [7, 45, 50, 73], line segments in [3, 8, 60], triangles in [13, 60]), the closed form formulation of their field functions [42] is still unexplored. These functions constitute the core of our design system.

With a multitude of modeling primitives available, the concept of a skeleton becomes especially life-like, because now a designer may think of skeletons as if they consist of solid 'bones', each of unique shape and size: point, line segments, curves, triangular pieces. In addition, arcs may be combined into circles and spirals, triangles – into polygons, line segments – into polylines, etc. This allows a designer to work with the object, creating and using those elements that fit best for each particular part of the object. An example of such a multi-primitive skeleton is shown in Figure 3.3, depicting a model of a crab. Note that even this simple sketch gives a good idea of what the resulting shape of the crab is going to look like. The completed model of the crab will appear later in section 3.5.

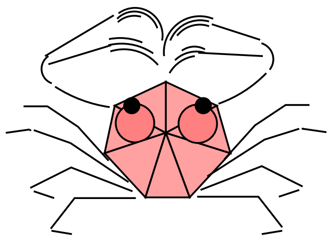
Figure 3.3: The skeleton of this coral crab is composed of 7 triangles, 24 arcs and 24 line segments.

3.3.2. Local properties of implicit primitives

As defined by equation (3.3), a convolution surface is the product of a geometry function g and a kernel function h. A geometry function $g = \sum_{i=1}^{N} g_i$ forms a skeleton, which provides a global description of the object: its general layout, location and the basic shapes of its parts. To complete the description of the object, the local properties of each element g_i of the skeleton must be specified.

Such properties may be conveniently described in terms of the radius in isolation R and blobbiness B, as was suggested by Blinn [7]. A *radius in isolation* is a characteristic distance between a geometric primitive and the implicit surface that it generates. For non-point primitives, the actual distance may vary along the surface. A *blobbiness* parameter controls blending of the object's parts. Very blobby objects, when brought together, tend to form sphere-like shapes with very little or no distinguishable detail preserved. Objects with low blobbiness look more like their skeletons. Figure 3.4 shows a series of convolution surfaces created with various values of blobbiness and radius in isolation. Blinn used these two parameters to define appearance of his point-based modeling primitives. We extend them to all the modeling primitives pictured in Figure 3.2.

The skeleton, radius in isolation and blobbiness of all skeletal elements completely define an implicit surface. The threshold T, used in the implicit surfaces equation (3.1), may be conveniently set to some canonical value, as described in [7].

As an alternative to radius in isolation, an implicit primitive may be characterized by its *region of influence*. A region of influence is the region in 3D space, where the field function f of the primitive has

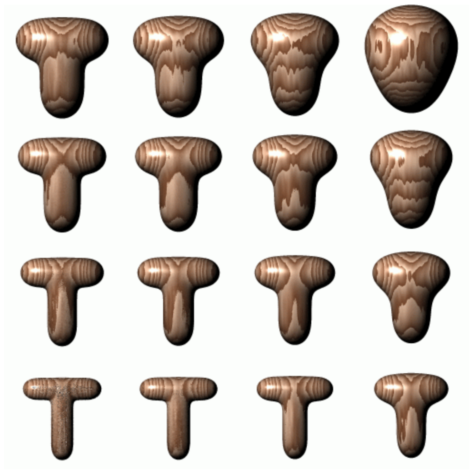

Figure 3.4: A T-shaped convolution surface, formed by two line segments. Increasing blobbiness (left to right) and radius in isolation (from bottom to top) allows us to achieve a variety of forms.

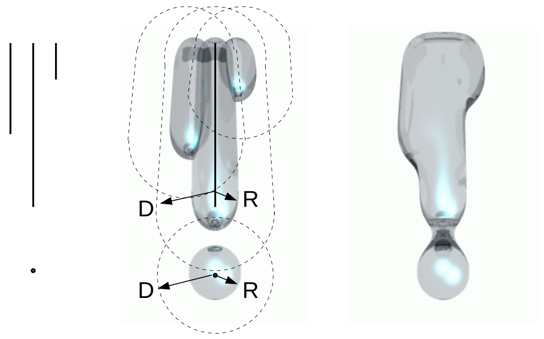

Figure 3.5: Elements on an implicit icicle, from left to right: skeleton made of 'bare' geometric primitives (three segments and one point); their implicit surfaces, rendered as stand-alone objects; the final convolution surface. In the middle image, blending between implicit surfaces is disabled to show the radius in isolation R and depth of influence D better.

non-zero values. A region of influence depends on the geometry of the primitive and the properties of the kernel function. For example, with a spherically symmetric kernel, the region of influence for a point primitive is a sphere; for a line segment — a cylinder capped by two hemispheres, etc. Regions of influence are normally used during rendering to prune out non-significant contributions from distant primitives.

An important property of regions of influence is that implicit surfaces are always located inside the boundaries of these regions. Regardless of the values of blobbiness and radius in isolation, the convolution surface of the icicle in Figure 3.5 (middle and right), always covers its 'bare' skeleton (solid lines) and is always covered by the boundaries of the regions of influence of its components (dotted lines). This property is very valuable for estimating extents of convolution surfaces.

Thus, the depth of the region of influence (or simply depth of influence) may serve as a modeling parameter, which is an alternative to radius in isolation. Figure 3.5 demonstrates the basic elements and parameters of modeling with implicit surfaces.

3.3.3. Materials and elements

Having defined the properties of skeletal elements (which are radius in isolation R and blobbiness B), we must provide a way to associate them with the skeletal elements. This may be done by

- sharing these parameters among a large number of elements;
- assigning them to each skeletal element individually.

Practice shows that both ways have their uses, so we incorporated them both into our modeling system.

3.3.3.1. Materials

The first approach is more convenient for modeling objects that have large areas with the same properties, because it allows employing the notion of a material. Normally, a description of a material includes conventional photometric characteristics, dependent on a lighting model: diffuse color, specular reflectivity, transparency, etc. Such descriptions may be easily enhanced by adding values of blobbiness and radius in isolation, which resemble softness and thickness of the material, respectively. This fits well with the idea of skeletal design: now each 'bone' in the skeleton is covered with some 'soft tissue', which is shared between skeletal elements. As an example, consider the material of an implicit icicle in Figure 3.5:

```
material Ice {
    body          SummerSky,
    ambient       1%,
    diffuse       1%,
    specular      SummerSky, shine 30,
    transparent   40%, index 1.33,
    reflective    50%,
    radius 0.25,  blobbiness -0.30
}
```

In this example, material `Ice` is defined as an almost colorless, reflective and transparent substance. Values of `radius` and `blobbiness` dictate the geometric aspects of the convolution surface, that is based upon the skeleton, defined as

```
object ICICLE
    line Ice, <0 0 0>, <0 0 4.5>
    line Ice, <0 0.5 3.0>, <0 0.5 4.5>
    line Ice, <0 -0.5 1.5>, <0 -0.5 4.5>
    dot  Ice, <0 0 -0.85>
close
```

The material-based description of convolution surfaces has two advantages.

Firstly, for design purposes, it is more convenient to think of skeletons and 'soft tissues' as of separate entities, that can be modified independently of each other. Keeping the material descriptions separate from the skeleton often allows materials of the objects to be changed interactively. This helps many surfaces parameters, both photometric and geometric, to be adjusted without reloading the model into memory for rendering.

Secondly, the local properties of the skeleton are not duplicated for each skeletal element, but are shared via description of a material. (Note the simplicity of syntax for `dot` and `line` modeling primitives in the example above.) This arrangement is memory-friendly, which may be important when the number of skeletal elements in the model is large.

3.3.3.2. Individual elements

The individual assignment of parameters R and B is needed when the model requires each primitive to be strictly different from its neighbors. Such models often come from procedural methods, especially in simulation of growth [31, 55]. For example, consider a sequence of spheres, pictured in Figure 3.6, that is characteristic for solid-based models of seashells [55]. In the domain of implicit modeling, each sphere depicts a region of influence of a point primitive, so the whole object must be described as

```
object SHELL
    sphere Plaster, r1, x1 y1 z1
    sphere Plaster, r2, x2 y2 z2
    ...
close
```

Here $r_1, r_2, ..., r_i$ denote the depth of influence of each point, triples (x, y, z) give locations of their centers. The value of blobbiness is stored in the specifications for the material `Plaster`, defined elsewhere, similarly as it was done for the material `Ice`. All seashell models that will be discussed further are modeled by setting individual depths of influence for each element in their models.

3.3.4. Profiling as a means for achieving variety

Figure 3.2 shows the basic primitives in their canonical, undeformed shapes, which may not be the best

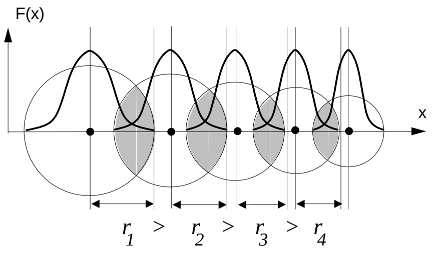

Figure 3.6: Strictly decreasing regions of influence of these implicit point primitives call for individual assignment of their values.

for a particular modeling task. Rather, these shapes provide a platform for variations and experiments.

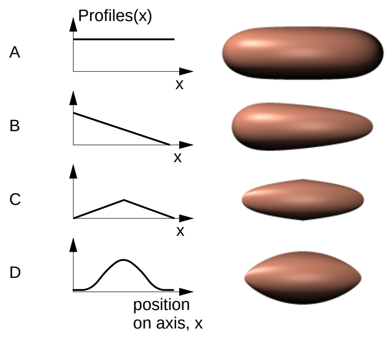

Figure 3.7: Line segments with modified field functions: constant (A), linear (B), hat (C), $sin^2(x\pi)$ (D).

As shown in Figure 3.7, an undeformed line segment (A) easily assumes various shapes (B, C, D), following the *profiling functions*. Profiling functions scale the values of the field, causing changes in the resulting implicit surface. Since line segments are axially symmetric, each profiling function, pictured in Figure 3.7, defines a surface of revolution around the axis. Profiling technique scales the value of a field function, not the final iso-surface. In terms of the modeling equation (2.2) profiling functions modify values of each constituent f_i. For example, the field function of a line segment shown in Figure 3.7 is

$$F_{modified}(x, y, z) = F_{line}(x, y, z)\, sin^2(x)$$

where $F_{line}(x, y, z)$ may be any of the functions given in Appendix B.

To continue a parallel with skeletons, profiling functions provide a mechanism for simulating changes in thickness and/or softness of the tissues along the bones, e.g., line segments. There are infinitely many ways to modify the field function of an implicit primitive. Profiling functions are used with nearly all examples of modeling with convolution surfaces.

3.3.5. Offset surfaces as visual aid and the data structures

As Figure 3.2 shows, the stand-alone implicit primitives based upon points, line segments, arcs, triangles and planes may be sufficiently approximated by an offset surface [12, 15], based upon the same primitives. They are, respectively: spheres, cylinders, arc tubes, prisms and infinite slabs. The extrusion distance may be set to radius in isolation R or depth of influence D (see Figure 3.5, middle), according to the current modeling situation.

Such similarity suggests a convenient modeling strategy: first an object is approximated by an offset surface and then refined as a convolution surface. The important feature of this approach is that most of the hard modeling work, related to building the skeleton, may be done with easy-to-render offset surfaces. To support this similarity, we use the same syntax and internal data structures both for implicit primitives and their offset counterparts. For instance, the command line

```
    sphere Plaster, r0, 0 0 0
```

creates an object in memory which may appear as an ordinary sphere, with radius `r0`, made of `Plaster` and positioned at the origin. Also, this object may appear as an implicit point with a radius in isolation `r0`. As described above, its blobbiness is stored in the description of the material `Plaster`.

Cylinders, arcs, triangles and planes also have reusable syntax and data structures. When the set of primitives is loaded into memory, it is ready for rendering in either mode: as an offset surface and as a convolution surface. In the former case, the implicit nature of all the modeling primitives is ignored and they are rendered as a union of offset surfaces. In the latter case, the primitives are treated as field functions $f_i(\mathbf{p})$ that form the composite scalar field of a convolution surface. To switch between two rendering modes, a simple on/off flag is used; the model does not have to be reloaded into memory.

Such duality of data structures has proven itself very useful. Computationally inexpensive offset drafts allow fine tuning of models at interactive rates. Since ray-tracing is used as a rendering method, all camera settings, lights and photometric properties of the objects are also adjusted in the draft mode as well.

3.3.6. Variables

For better interactivity in our design environment, the following data types are allowed to be defined as variables:

- scalars
- vectors
- colors
- textures
- materials

Variables may be created and modified ‘on the fly’ during design session; they also may be assigned to each other, observing type conversion. With the use of variables, it is possible to modify practically all elements of the model, while it is still in computer memory. They include: positions and shapes of all skeletal elements, blobbiness and thickness, all photometric characteristics. In addition, variables may be declared as auto-incremental and/or auto-multipliable, which helps specifying objects’ motion in animations.

3.4. THE MAIN MODELING LOOP

3.4.1. Datasets

In the previous section, we mentioned elements of what we call *a dataset*, which is a complete collection of parameters that defines a particular geometric model. Every dataset contains descriptions of all materials and skeletal elements that together form a convolution surface. In addition, a dataset describes all modifications of local properties of skeletal elements that are needed in order to add a desired shape to the final surface. Thus, a dataset may be considered as a program in some highly specialized language for describing convolution surfaces. The complete vocabulary of this language and several complete datasets are given in Appendix C.

With respect to design and rendering, a dataset may be regarded as an output of a design process and an input for a rendering program.

3.4.2. The main modeling strategy

We use a method of progressive refinement of the dataset as the modeling strategy. An initial dataset is acquired first, using some foreign geometric model, a program generator or a user’s inspiration with a paper-and-pencil method. Then, the dataset undergoes a series of iterations, during which a designer modifies existing elements of the dataset, adds new ones and deletes those that fail to fit the model.

The global dataflow chart of a typical design session is depicted in Figure 3.8. For this particular example, the skeletal model was produced by a tree-growing program, based upon Eric Haines’ implementation [31] of Aono and Kunii’s tree-generation method [2]. The rest of the dataset, full of question marks, has to be designed interactively.

By applying various values of thickness to the skeletal elements, several drafts have been made, pictured as a stack of images on the right-hand side of

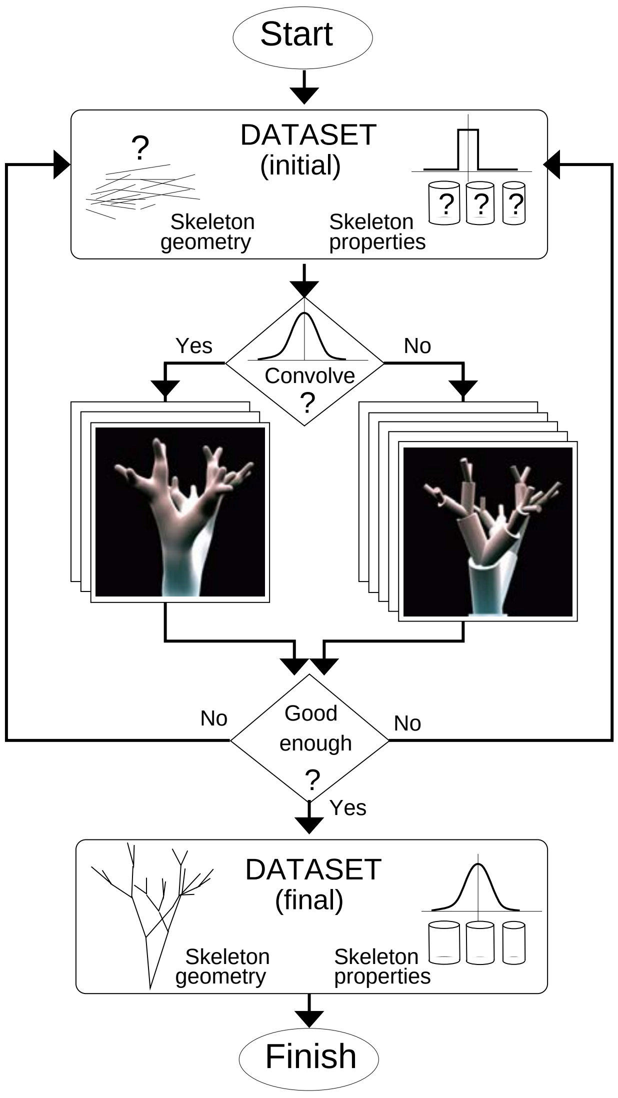

Figure 3.8: The main modeling loop.

the chart. Since offset surfaces, used for the drafts, are easy to render, the process may be easily reiterated as many times as needed to find the perfect values for thickness of all branches.

At this point, the designer may switch to using convolution surfaces and try different values of blobbiness. After several attempts, the desired shape is found – the dataset is now complete and the process terminates [2].

Note that this modeling strategy is a winning strategy by construction: the process terminates when the designer is satisfied with the result. Visual control over the quality of the current state of the model guarantees that the design process moves in the right direction. Backup copies of the dataset help to fix irreparable changes.

The modeling process bears certain resemblance to the hill-climbing programming archetype: the process enters the loop at some point and then moves towards the solution, iterating on the current state of the task.

3.5. PRACTICAL EXAMPLES OF IMPLICIT DESIGN

In this section, a number of practical modeling examples are discussed. They are grouped by methods that have been used for their design. With respect to the main modeling loop, pictured in Figure 3.8, these methods are roughly divided into those that deal with the skeletons (right-hand side of the chart) and methods that refine the local properties of the convolved surfaces (left-hand side).

3.5.1. Global skeleton manipulations

The modeling system allows, in principle, us to build a new skeleton from scratch interactively. However, it is more convenient to prepare an initial skeleton, using one of the methods listed below.

3.5.1.1. Procedural methods

Two skeletons have been produced using purely procedural methods: the flat spiral shell in Figure 3.9 and the seaweed in Figure 3.13. Both are generated using modified utilities `shell` and `tree`, respectively, from Standard Procedural Database by Eric Haines [31]. Their skeletons are fairly simple. The `tree`-skeleton was also used in the model of a coral tree, shown in Figure 3.14. Additional 512 spikes were added to the model procedurally, using random placement.

[2]Or, the designer may also try to enhance the resulting shape by applying different profiling functions. The result of such experiments are shown in Figure 3.13.

3.5.1.2. Hand-crafting

A skeleton of a seahorse in Figure 3.15 is 100% hand-crafted. The initial paper-and-pencil drawing was created first. Then it was duplicated with the *xfig* drawing tool (Figure 3.15, left), which produced the coordinates of the skeletal elements, in this case, arcs. In most cases, the joints between the arcs in the line drawing are C^1 and C^2 continuous; the cracks in the offset surface (Figure 3.15, middle) are introduced intentionally to show the skeletal elements better. The final convolved surface (Figure 3.15, right) seals these cracks. Note, that the thickness of each skeletal element is set individually. The central arc in the body of the seahorse is made especially thick. The resulting inflation fills the empty space inside the body and, perhaps, indicates the presence of internal organs.

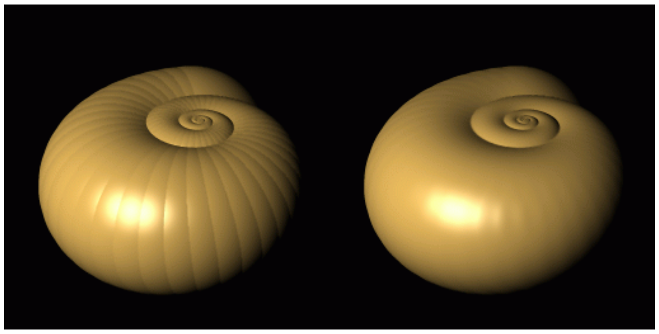
Figure 3.9: Shell I. The point-based offset surface (left) and the convolved surface (right) are very similar. This dataset served as a base model for Shell II and Shell III. 361 elements.

3.5.1.3. Mixed methods

Procedurally generated models may be significantly improved by manual editing. For instance, the oval seashell, pictured in Figure 3.10, is derived from the previous flat sphere-based model (Figure 3.9) by a simple addition of one point, which was then connected with all points in the initial dataset. Thus, all spheres become cylinders with a common base. By moving this common base, the shape of a new shell may be controlled easily. This is similar to rubber-band techniques with the wire-frame representation of solid object. Using a vector variable for holding coordinates of the common base point makes such manipulations especially convenient.

The skeleton of the next shell was also created procedurally and then updated manually. This time, two skeletons were created as described above. Then they were superimposed to produce a combined skeleton for a spiked shell, shown in Figure 3.11.

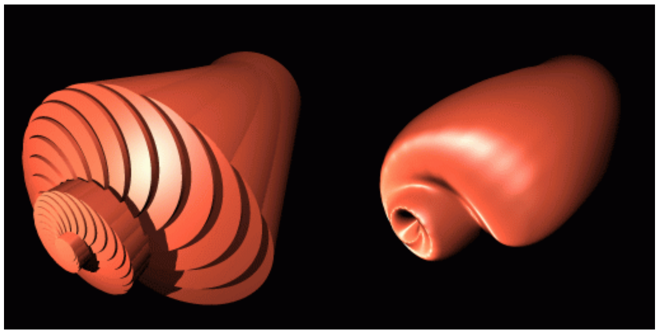
Figure 3.10: Shell II. The offset surface of this mollusk is pictured as a union of simple cylinders (instead of sphere-capped cylinders) to emphasize the spiral structure better. 42 elements.

While manual intervention often improves procedural skeletons, the reciprocal is also true: hand-crafted skeletons can be made much more interesting and visually appealing with the help of special purpose program generators. Examples of such skeletons are: a spindle cowrie shell (Figure 3.12) and coral crab (Figure 3.16). Both skeletons were hand-copied from the actual photographs [46], using *xfig* drawing tool. Then, small details were added procedurally.

Usually, skeletons of mixed origin yield the most interesting shapes.

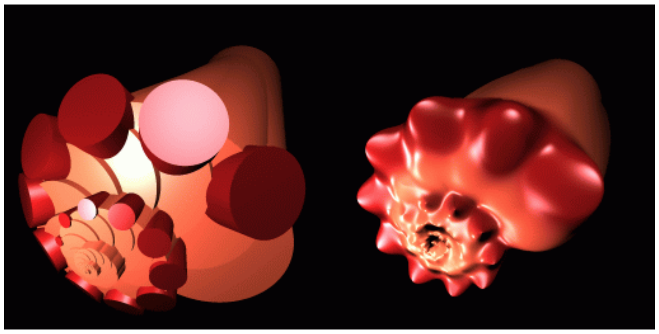
Figure 3.11: Shell III. Superposition of two skeletons: the orange base (55 segments) blends with the spiral row of red horns (15 segments) yielding a smooth convolved shape of a new seashell. Both skeletons are made procedurally.

3.5.1.4. Imported models

Two fully imported polygonal models of Yoda and Nefertiti are shown in Figure 3.19 and Figure 3.18, respectively. These models have been used as polygonal skeletons for the convolution surfaces without

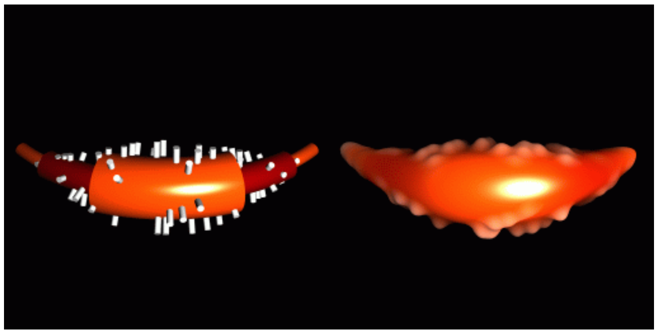

Figure 3.12: This very simple model of Spindle Cowrie is based on a croissant-like combination of 3 arcs. Additional 100 cylinders make the surface look more interesting.

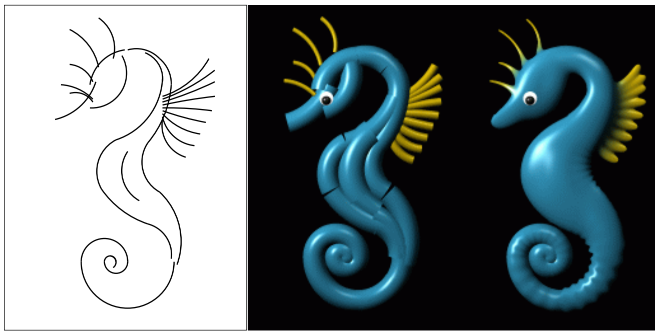

Figure 3.15: Seahorse. The hand drawing (left), the offset surface (middle) and the convolution surface (right). Note how the wrinkles in the back and the tail indicate softness of the skin. Total number of elements 45 (43 arcs and 2 spheres).

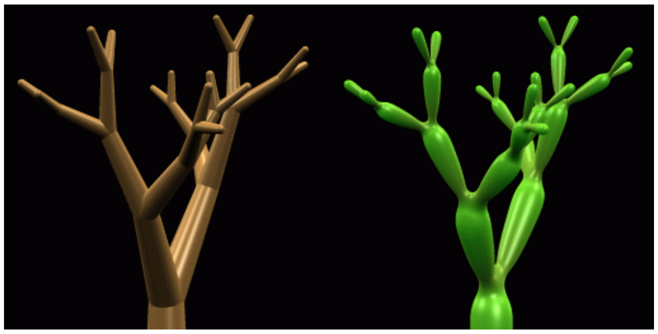

Figure 3.13: Seaweed. The offset surface is made of cones and spheres. The conical shapes of the tree branches suggested applying profiling functions in the convolution surface.

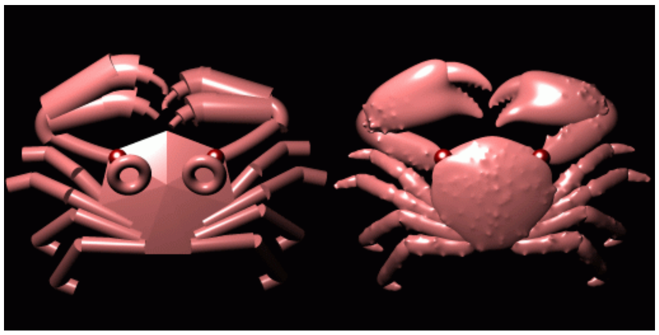

Figure 3.16: Coral crab. The simplified draft surface consists of 7 polygons (body), 26 arcs (claws, eye sockets), 28 cylinders (legs, claws) and 2 spheres (eyes). The convolved version has additional 608 spikes, scattered along the surface.

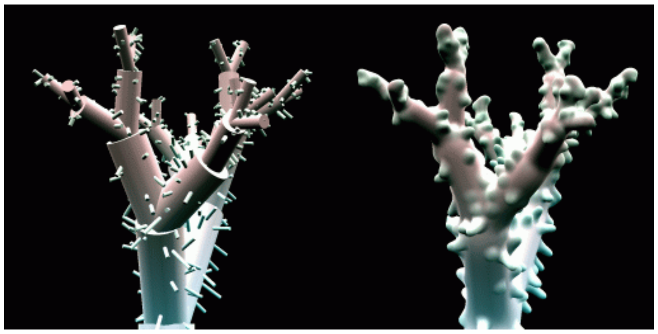

Figure 3.14: Coral trees. Note that the base skeleton (31 cylindrical branches) is identical to the seaweed model. Additional 512 cylindrical spikes of various lengths are scattered along the surface randomly.

any modification done to their initial meshes. The origin of both models is unclear.

3.5.2. Local modeling techniques

After the global skeleton is complete, a designer may switch to more subtle modeling techniques and refine the model to their taste. With respect to the modeling loop (Figure 3.8), these operations happen on the left-hand side of the loop, because most techniques that are described below apply to convolved versions of the object.

3.5.2.1. Shaping techniques

Shaping is a straight-forward application of profiling functions, which creates local variations of radius in isolation of modeling primitives (or the thickness of the material). Examples of shaping are presented in Figure 3.8 and Figure 3.13, where normal tree is turned into a seaweed by applying a linear shaping function to its branches. Less obvious examples of shaping may be found in the model of a coral crab (Figure 3.16). The end segments of the crab's legs are also linearly shaped to make them look sharper.

3.5.2.2. Carving techniques

Carving is a combination of using depth of influence as a modeling parameter with some profiling functions. The depth of influence around modeling primitives defines a volume of 'matter' to carve from; the profiling function defines the shape of the 'cutting tool'.

Technically, profiling functions operate on each modeling primitive individually, as shown in Figure 3.7. However, when the modeling primitives interpenetrate, which is true for the spiral seashells (Figures 3.9, 3.10, 3.11), the result of carving is as if the cutting tool has been revolved in the model space following the spiral curve of the shell. In this way, profiling functions are similar to the *generating curves* used for modeling seashells in [27]. The carving technique was used with all the models of spiral shells, pictured in Figures 3.9, 3.10 and detailed in Table 3.1.

Fig.	Primitives	Profiles
3.9	361 points	constant
3.10	42 line segments	linear
3.11	70 line segments	$sin^2(x\pi)$

Table 3.1: Modeling of spiral seashells.

3.5.3. Volumetric detail

Defined as isosurfaces in a scalar field, convolution surfaces are sensitive to variations of that field. Large-scale variations cause global changes in the appearance of the object. Small-scale variations modify the geometry of the surface locally, adding more detail to the surface. Such detail enhances the appearance of the object and produce various texturing effects. Since these details are based upon implicit functions, which are volumetric by their nature, we call this *volumetric detail.*

Most models presented in the paper contain volumetric details, which may be grouped in three major ways: Structural, Functional and Procedural.

3.5.3.1. Structural detail

This method involves adding more modeling primitives, which may be done manually or by using some auxiliary utilities. Although it seems like a brute-force solution, structural detail may be very effective in modeling various irregular and rough-looking surfaces, such as the body, claws and pincer grip of a coral crab, shown in Figure 3.16 or a complex shape of a coral tree (Figure 3.13). Similarly, the shell of a spindle cowrie (Figure 3.12) is enhanced by cylindrical spikes. Other examples of using spikes with convolution surfaces are presented in [13, 15].

One convenient feature of structural details is that, unlike other methods of adding detail, most iterations on the dataset may be done with the offset representation of the surfaces. This corresponds to the right-hand side of the main modeling loop (Figure 3.8), where a designer may afford a large number of takes.

3.5.3.2. Functional detail

Functional detail is produced by applying profiling functions on small areas of the surfaces. As an example, consider wrinkles on the back and the tail of a seahorse (Figure 3.15).

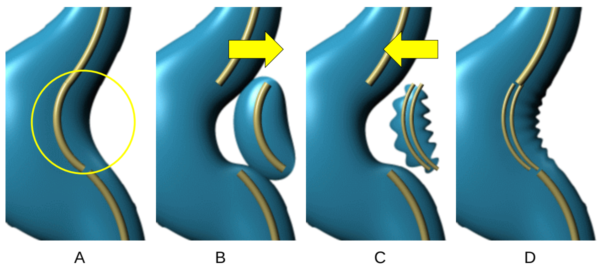

Figure 3.17: How to make volumetric wrinkles: (A) The place for wrinkles is chosen. (B) The closest arc is pulled out. (C) The arc is split into halves and one half is modulated by a sine wave. Together, they form a wrinkled volumetric implant, ready for re-insertion. (D) The wrinkled implant is put back in place.

Figure 3.17 illustrates how the wrinkles were implanted in the originally smooth convolution surface.

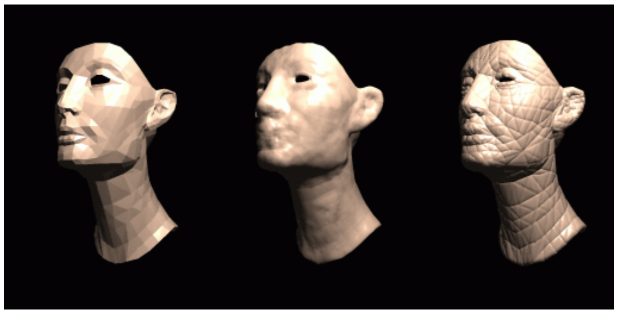

Figure 3.18: Nefertiti, before (left) and after (middle, right) convolution. Restricted blending near the edges creates the wrinkles (right). Total 1,242 implicit triangles.

This figure is an exact close-up of Figure 3.15 with more anatomical parts shown (only participating skeletal elements and close neighbors are displayed).

Note that the field function of the wrinkled volumetric implant, as described in Figure 3.15, could have been represented by a single arc, modified by an elevated and scaled sine wave $a_1(1 + a_2\sin(x))$, instead of two, as shown in the picture. In this case, it is a matter of choice whether to add another primitive into the dataset, or define one more profiling function in the modeling system.

A similar use of high-frequency functions for adding volumetric details is reported in [4]. The difference of our approach is that we use it in the skeletal-based modeling environment, which allows selective placement of such details.

3.5.3.3. Procedural detail

Perhaps, the most unusual examples of volumetric detail are given in Figure 3.18. and Figure 3.19. These images present the results of the experiments with triangular meshes used as skeletons for convolution surfaces. Triangular meshes are normally employed to approximate surfaces, as shown in Figure 3.18, left and Figure 3.19, top. Convolving triangles with a potential kernel function introduces remarkable changes into appearance of the object being modeled, as illustrated in Figure 3.18, middle. However, simple convolution makes the skin of Nefertiti look swollen. All fine features are missed, as high frequencies are removed by a low-pass convolution filter.

To reduce the swelling effect, a restricted mechanism is used, which artificially reduces the amount of field generated by triangular primitives near the edges. Thus, the surface exhibits hills over the central parts of each triangle and valleys along their edges. The maximal difference in elevation of these hills and valleys depends on the thickness of the material of the face (i.e., skin) and the value of the restriction factor. The former tends to elevate the convolution surface

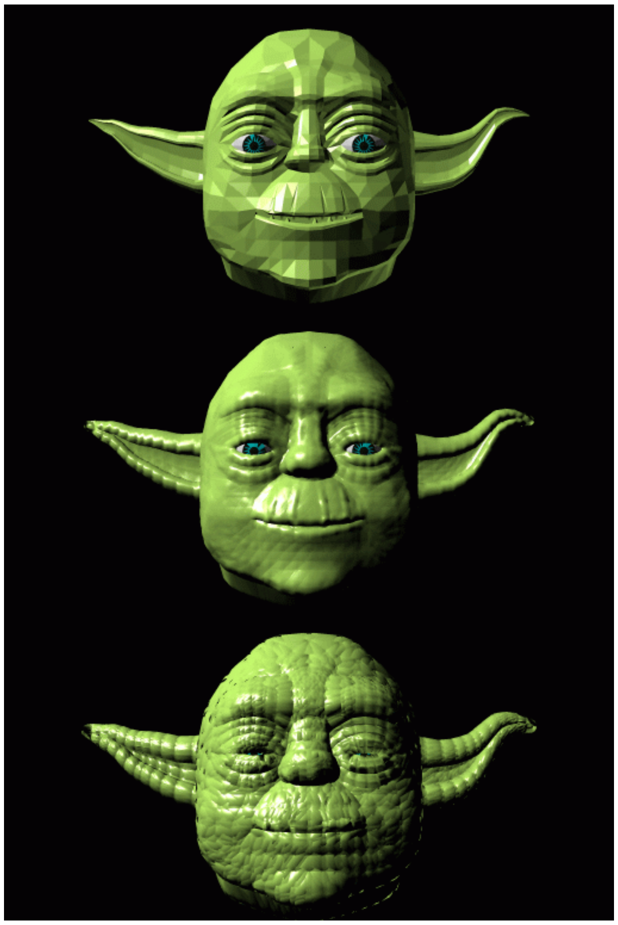

Figure 3.19: Aging Yoda: polygonal mesh (top), convolution surfaces (middle and bottom). Total 3,422 implicit triangles.

Model	**Skeleton**	**Techniques**
Shell I	Procedural	Carving
Shell II	Procedural + hand-made	Carving + + shaping
Shell III	Procedural + hand-made	Carving + + shaping
Seaweed	Procedural	Shaping
Seahorse	Hand-made	Functional wrinkles
Cowrie	Hand-made + procedural	Structural detail
Crab	Hand-made + procedural	Structural detail + + shaping
Nefertiti	Imported	Procedural wrinkles
Yoda	Imported	Procedural wrinkles

Table 3.2: Summary of modeling techniques used with all datasets.

above the 'bare' polygonal skeleton, especially in the central areas of each polygon. The latter pulls the surface closer to edges of triangles, producing wrinkles along the edges. The roughness of the resulting implicit surface may be effectively controlled by varying both parameters: thickness and restriction scaling factor.

The visible outcome of such controlled convolution is the perceived age of the character. The wrinkled and dried-up skin makes the face of Nefertiti arguably more interesting and definitely more realistic — the prototype of this model is 5,500 years old. Similarly, convolved skin gives Yoda a much more mature look.

The restriction mechanism may be combined with an additional UV-mapping to produce even more wrinkles inside each triangle. Alternatively, procedural or noisy functions may be used to modify the values of the field functions. In all cases, the visible complexity of the resulting convolution surface will be increased without increasing the number of elements in the skeleton.

Note, that all types of volumetric detail presented above, have truly geometric nature; they are not shading tricks. They participate in a full range of graphics manipulations, such as visibility calculations, shadows, transformations, etc. By modifying the local geometry of the surface, volumetric detail creates additional features that may be very informative. For instance, spikes and horns imply firm and rigid surfaces; wrinkles often suggest that the object is flexible, allows deformations, and is currently deformed from its usual state; wrinkled skin suggests old age. These details enhance the visual realism of the model and have a strong potential for complex modeling with implicit surfaces.

3.6. IMPLEMENTATION DETAILS AND TIMING RESULTS

All models discussed above were designed and ray-traced using the in-house modeling/rendering system *RATS Version 7.31* running under the Linux OS on a 90 MHz Pentium processor. The algorithm for ray-tracing convolution surfaces, used in the *RATS* system, is described in Chapter 4. Offset surfaces are also ray-traced; the relevant algorithms may be found in [29] and [71]. Table 3.3 gives details about each dataset and the rendering times. Image resolution: 512 x 512. Anti-aliasing method: shooting at most 16 primary rays per pixel (Of course, during interactive design sessions, much smaller images were used, typically 128 x 128. Shooting 1 ray per pixel allowed to render all offset surfaces under 10 seconds per iteration.)

Model	**Contains**		**Off. surf.** t_1	**Conv. surf.** t_2	t_2/t_1
Shell I	361	points	2:52	22:46	7.81
Shell II	42	lines	20:45	32:51	1.58
Shell III	70	lines	12:54	41:16	3.20
Cowrie	103	total	7:34	24:40	3.26
Seaweed	31	lines	3:34	4:46	1.34
Coral	543	lines	11:18	38:50	3.43
Seahorse	43	arcs	5:25	11:52	2.19
Crab	641	total	14:03	46:13	3.29

Table 3.3: Rendering time (min:sec) for offset surfaces and convolution surfaces.

These timing data give a somewhat distorted picture of the actual situation. For instance, some offset surfaces contain a large number of edges, that had to be anti-aliased by shooting more rays. Convolved shapes tend to be smoother, thus fewer pixels had to be supersampled. This explains an unusually low time increase of 1.58 for rendering the second seashell as a convolution surface: it has a very sharp offset-surface counterpart.

The seahorse dataset presents another 'special case' of low time ratio. Since arc tubes, used as offset surfaces, are already computationally expensive (they require solving quartic polynomials for each intersection test), switching to convolution surfaces does not introduce dramatic changes in rendering time.

The fastest implicit primitive, according to Table 3.3, is a line segment (see the seaweed model in Figure 3.8). It required only 34% more rendering time than its explicit counterpart. Such efficiency is due to a better kernel that was used for this particular dataset, that is $h(x) = b\ exp(-a^2x^2)$. The field function for line segments, produced with this kernel, turned out to be very efficient for this particular model [3]. Exponential kernel function was first used by Blinn [7].

The average time penalty for using convolution surfaces compared to offset surfaces is about 3.2. The crab model, which contains all types of new implicit primitives, shows a similar slow-down of 3.29.

3.7. CONCLUSION

In this chapter, we have presented a variety of primitives and modeling techniques for implicit design. Points, lines, arcs and triangles, when used as elements of convolution surfaces, spawn a rich palette of unusual variations of this modeling concept. Methods of adding details are discussed, which enhance the visual realism of the surfaces and expand the application base of implicit modeling.

We would like to point out that most images presented in this chapter could not have been produced by conventional techniques developed for implicit modeling to date. These images are products of the concept of convolution surfaces and a number of additional modeling methods described above, and rendering methods discussed in the next chapter.

[3]Note, that the field function for a line segment produced with a Gaussian kernel looses to the Cauchy line segment in the 'bare' speed test, as shown in Tables (2.3, 2.4) (see Chapter 2). However, in different modeling situations with different values of blobbiness and radius in isolation, the Gaussian line segment may be faster (for instance, due to tighter bounding volumes), as it apparently happened in this case.

A certain boy once drew a beautiful, if somewhat abstract picture. He was asked: "And what did you mean to say by this?" To which he replied, "I meant exactly what I said."

A boy (perhaps the same one) was taken to a museum and shown an abstract picture. The guide explained: "This is meant to be a horse." To which the boy replied, "If it is meant to be a horse, why isn't it a horse?"

–*Two complementary stories*