Interactive Modeling of Implicit Surfaces using a Direct Visualization Approach with Signed Distance Functions

Tim Reiner [a], Gregor Mückl [b], Carsten Dachsbacher [a]

[a] *Computer Graphics Group, Karlsruhe Institute of Technology, Germany*
[b] *SimTech, University of Stuttgart, Germany*

Abstract

Modeling appealing virtual scenes is an elaborate and time-consuming task, requiring not only training and experience, but also powerful modeling tools providing the desired functionality to the user. In this paper, we describe a modeling approach using signed distance functions as an underlying representation for objects, handling both conventional and complex surface manipulations. Scenes defined by signed distance functions can be stored compactly and rendered directly in real-time using sphere tracing. Hence, we are capable of providing an interactive application with immediate visual feedback for the artist, which is a crucial factor for the modeling process. Moreover, dealing with underlying mathematical operations is not necessary on the user level. We show that fundamental aspects of traditional modeling can be directly transferred to this novel kind of environment, resulting in an intuitive application behavior, and describe modeling operations which naturally benefit from implicit representations. We show modeling examples where signed distance functions are superior to explicit representations, but discuss the limitations of this approach as well.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Computational Geometry and Object Modeling—Modeling packages

Keywords: distance functions, implicit surfaces, implicit surface rendering, interactive modeling

1. Introduction

Implicit descriptions of smooth continuous surfaces are well established in computer graphics, next to discrete approximations like the ubiquitous polygonal meshes. Implicit surfaces spark interest due to their intrinsic way of representing solid objects, aptitude for constructive solid geometry, and eminent blending abilities.

However, it is inherently difficult to render implicit surfaces directly in real-time. This requires an extensive incremental search for surface points, which are neither explicitly given nor easy to compute. Consequently, indirect visualization techniques, such as marching cubes [7], became popular, which generate discrete approximations of surfaces first, making them fast to render. However, these intermediate steps introduce geometric aliasing, giving up both smoothness and high-frequency details.

Implicit surfaces are amenable to ray tracing, but it is difficult to find intersections of rays with complex surfaces analytically. Thus, ray marching is often used to render implicit surfaces directly, where possible intersections are determined by marching along a ray in small steps until a surface is hit. Sphere tracing [4] speeds up ray marching significantly, but requires distance surfaces [1], implicitly described by functions that return a measurement or bound of the geometric distance, instead of a generic algebraic distance.

It has always been tempting and desirable to use implicit surfaces for shape modeling. Providing an interactive modeling environment for this purpose is difficult, as it involves the necessity to render implicit surfaces in real-time in order to provide constant visual feedback. Indirect visualization techniques are capable of doing so, but introduce another problem: while editing a surface, continuous re-evaluations of its discrete approximation are required. Unless this is achieved in real-time, interactivity gets interrupted at this crucial point. Reducing approximation quality helps, but it removes subtle details and introduces artifacts as well.

In this paper, we make the following contributions:

- We compose complex implicit surfaces using signed distance functions. Then, we show how to translate these compositions into a fragment shader that renders defined surfaces directly on graphics hardware. As a result of using signed distance functions, we are able to apply sphere tracing to efficiently render a scene in real-time.

- We present a complete modeling environment for objects and scenes that are implicitly defined by signed distance functions. As we are able to render them in real-time, we provide full interactivity for the modeling tool. We are neither dependent on indirect visualization techniques nor restricted by spatial limitations. Likewise, explicit grids or approximations are not required.

2. Related Work

Previously, a lot of pioneer work in implicit shape modeling has been done. In this section, we focus on fundamental and most related work, ranging from the notion and rendering of implicit surfaces to editing operators and modeling environments for them.

Groundbreaking, Hypertextures [15] combined implicit shapes with textures to model various phenomena, extending the notion of shape by including surrounding volumes. Turk and O'Brien [23, 24] introduced new shape transformations and interpolation techniques for implicit surfaces.

Loop and Blinn [6] showed how to render implicit surfaces in real-time on the GPU using analytic techniques for solving for roots of polynomials. Hence, they are limited to fourth order algebraic surfaces, and form objects by assembling piecewise smooth algebraic surfaces, which were introduced by Sederberg [20].

It becomes cumbersome to evaluate functions defining implicit surfaces of increasing complexity. Consequently, distance fields attracted attention, which explicitly store distance information inside a three-dimensional grid. Again, this leads to geometric aliasing in trade for fast distance evaluations. Obviously, this representation requires a tremendous amount of memory if used naively, and therefore substantially limits the possible extent of a scene. Frisken et al. [3] proposed adaptively sampled distance fields, which significantly reduce memory consumption for detailed objects.

The level set method, first presented by Osher and Sethian [12] and thoroughly explained by Osher and Fedkiw [13], defines a surface using a time-varying scalar function. It spawned a large body of research and gained recognition in modeling surface deformations.

Museth et al. [10] contributed editing operators for implicit surfaces based on level set models. Surfaces are imported as distance fields into their level set framework, allowing arbitrary surfaces modified with a single procedure. As their framework is based on uniform grids, it suffers from technical limitations again, particularly spatial limits, memory constraints, and artifacts due to discretization.

Comprehensive approaches to interactive implicit modeling were proposed. A primary contribution is HyperFun [16], that has been extended by others in various directions. In its essential form, it provides a symbolic user interface for direct textual input in a high-level geometric language. Note that interactivity does not extend to real-time surface visualization.

Our approach closely resembles BlobTrees [25], that have been widely adapted. Its hierarchical structure stores implicit surfaces at the leaves and places composition operators at internal nodes. ShapeShop [19] is a prominent example based on the BlobTree. Interactivity is achieved by introducing spatial caching for objects.

WarpCurves [22], inspired by FiberMesh [11] and Wires [21], extends ShapeShop to allow for interactive manipulation of implicit surfaces using explicit curve-based spatial deformations. Recently, Martinez Esturo et al. [9] described a method for continuous deformations of implicit surfaces focusing on volume, continuity, and topology conservation.

Related important areas of research in interactive modeling are haptics-based [5] and sketch-based [19] user interfaces for convenient sculpting.

Note that the modeling environments mentioned before use an indirect marching cubes visualization, contrary to our real-time direct visualization of the scene.

3. Describing Scenes with SDFs

This section briefly introduces SDFs (signed distance functions), shows how to describe primitives with them, and how transformations and combinations can be applied in order to create complex objects. Eventually, whole scenes emerge by assembling several objects.

3.1. Definition

We mostly stick to the notation from [13], which provides, inter alia, a comprehensive introduction to implicit surfaces. Let Ω^- and Ω^+ be the space in and outside of an implicit surface, respectively, and $\partial\Omega$ the interface in between, i.e., the set of points composing the surface. A signed distance function $\phi(\vec{x})$ is defined as

$$\phi(\vec{x}) = \begin{cases} \min(|\vec{x} - \vec{x_I}|) & \text{if } \vec{x} \in \Omega^+, \\ 0 & \text{if } \vec{x} \in \partial\Omega, \\ -\min(|\vec{x} - \vec{x_I}|) & \text{if } \vec{x} \in \Omega^-, \text{ for all } \vec{x_I} \in \partial\Omega, \end{cases}$$

and returns the Euclidean distance from $\vec{x}$ to its closest surface point, with a negative sign if inside. Thus, SDFs are a subset of implicit functions, which rather imply algebraic distances.

3.2. Properties

SDFs retain characteristics of implicit functions, yet provide more beneficial properties, such as $|\nabla\phi| = 1$. This is comprehensible, since ϕ is Euclidean distance, and satisfies the criteria of sphere tracing precisely; see Section 4.2. Note that the equation is not true for points equidistant to at least two interface points, where the path of steepest descent is ambiguous. Fortunately, we can safely ignore this issue, as we march along given ray directions only.

We are interested in providing a comprehensive variety of flexible modeling operations and transformations. In exchange, we comply with producing distorted distance functions that do not yield the accurate Euclidean distance anymore. If $0 < |\nabla\phi| < 1$, sphere tracing decelerates but still ensures not to penetrate surfaces. Alas, this is not the case for $|\nabla\phi| > 1$, but sphere tracing is robust enough if a bound is known. Section 4.2 presents a concept to compensate for gradients too steep.

As a matter of particular interest, SDFs remain monotonic across the interface, not having a kink like (unsigned) distance functions. This allows reliable gradient constructions, e.g., using central differences, which is required for illumination and shading.

3.3. Describing Primitives

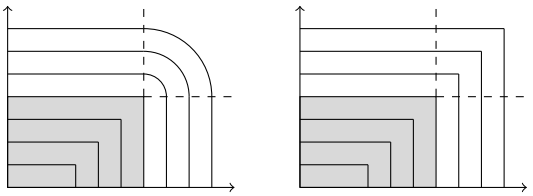
Many primitives can be described directly using an SDF. For example, $\phi(\vec{x}, r) = |\vec{x}| - r$ describes a sphere at the origin with radius r. Other basic primitives such as tori and infinite planes, cylinders, and cones are easy to describe as well.

Given the SDF of a two-dimensional curve, it can be easily extended to a solid of revolution. Distances are evaluated by projecting points in question onto the plane where the curve is defined.

Primitives featuring hard edges between surface regions require SDFs to have corresponding kinks. This can be accomplished by composing individual SDFs for each surface region, using case-by-case analysis which region is closest to a point in question. We try to avoid case-by-case analysis in our shaders, however, as branching is still an expensive operation on the GPU. Instead, we edge surfaces by switching function spaces. For instance, a sphere formed using the L_∞ metric represents an axis-aligned cube. In this way, we can state an SDF for a box with width w, height h, and depth d as

$$\phi(\vec{x}, w, h, d) = \max(|\vec{x}_1| - w/2,\ |\vec{x}_2| - h/2,\ |\vec{x}_3| - d/2).$$

This SDF returns a Chebyshev distance, which can be greater than the Euclidean distance. This is the case for a point $\vec{p}$ outside the box, whose nearest surface point is



(a) Euclidean metric, (b) L_∞ metric.

Figure 1: Box isocontours for different metrics.

$\vec{p}_I$, if the line through $\vec{p}$ and $\vec{p}_I$ is not parallel to one of the coordinate axes (see Fig. 1.) Note that this provokes more sphere tracing steps during rendering, but is still much faster and superior to branching thanks to efficient GPU implementations of min/max-functions.

3.4. Describing Transformations

In order to apply a transformation T to a surface described by $\phi(\vec{x})$, the inverse transformation is applied to the domain. Hence, $\phi(T^{-1}(\vec{x}))$ describes the transformed surface.

Any concatenation of translations, rotations, and reflections is a global isometry on Euclidean spaces, i.e., all properties of the SDF are preserved. This, however, does not hold for morphing, blending, and non-linear transformations used to model complex and appealing objects. These transformations distort the Euclidean distance, and we need to make sure that surfaces are still amenable to sphere tracing.

In the following we will discuss the application of transformations to SDFs, starting with transformations as in traditional modeling tools, followed by artistic ones that are easy to define implicitly.

3.4.1. Rigid Body Transformation

Applying translations and proper rotations preserves the Euclidean distance. Thus, to rotate and translate an object described by $\phi(\vec{x})$, we evaluate $\phi(R(\hat{e}, -\theta)\vec{x} - \vec{t})$ instead, where $R(\hat{e}, \theta)$ is a matrix describing a rotation around an axis $\hat{e}$ by angle θ, and $\vec{t}$ is the translation vector.

3.4.2. Scaling

To apply a uniform scale by factor s, we evaluate $s \cdot \phi(s^{-1}\vec{x})$. Note the final multiplication by s, since $|\nabla\phi(s^{-1}\vec{x})| = s^{-1}$ violates the Euclidean distance. Unfortunately, a compensation for non-uniform scalings by factors s_x, s_y, s_z is not as straightforward. A final multiplication by $\min(s_x, s_y, s_z)$ ensures that the gradient magnitude does not exceed 1, which is desirable for sphere tracing. However, as scaling factors drift apart,

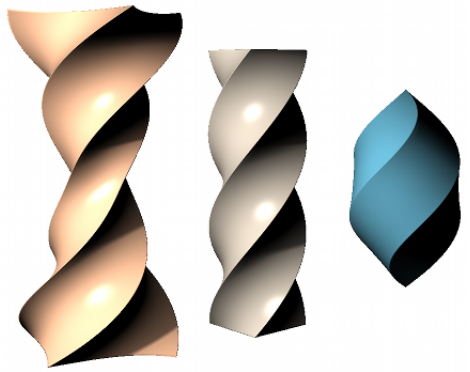


Figure 2: Blend between box and sphere with twist.

distances get overestimated more and more along the axis of the widest stretch, decelerating sphere tracing significantly.

3.4.3. Eminent Implicit Operations

Particular operations are amazingly simple to apply to implicit surfaces. A prime example is *constructive solid geometry* (CSG). Unions, intersections, and differences of two objects A and B, described by ϕ_A and ϕ_B, satisfy the following statements:

$$\begin{aligned} \phi_{A\cup B}(\vec{x}) = 0 \quad &\Leftrightarrow \quad \min(\phi_A(\vec{x}), \phi_B(\vec{x})) = 0, \\ \phi_{A\cap B}(\vec{x}) = 0 \quad &\Leftrightarrow \quad \max(\phi_A(\vec{x}), \phi_B(\vec{x})) = 0, \\ \phi_{A-B}(\vec{x}) = 0 \quad &\Leftrightarrow \quad \max(\phi_A(\vec{x}), -\phi_B(\vec{x})) = 0. \end{aligned}$$

Note that unions cause wrong distances inside objects, whereas intersections implicate the L_∞ metric outside.

Blending between two objects is trivial as well:

$$(1-\alpha)\cdot\phi_A(\vec{x}) + \alpha\cdot\phi_B(\vec{x}),$$

using a blend factor α. An alternative notion is to set A at the origin, and blend over to B while moving away:

$$\max(0, 1 - s\cdot|\vec{x}|)\cdot\phi_A(\vec{x}) + \min(1, s\cdot|\vec{x}|)\cdot\phi_B(\vec{x}),$$

with a stretch factor s defining the transition region.

A *twist* is easily described implicitly:

$$\xi_a(\vec{x}) = \begin{pmatrix} x_1\cos a(x_2) - x_3\sin a(x_2) \\ x_2 \\ x_1\sin a(x_2) + x_3\cos a(x_2) \end{pmatrix},$$

with a linear function a defining amount and offset.

Fig. 2 shows an example for a blend between a sphere and a box, where a twist is applied subsequently. On the far left, a negative α is used. A box is frequently used to demonstrate how edges remain continuous and sharp.

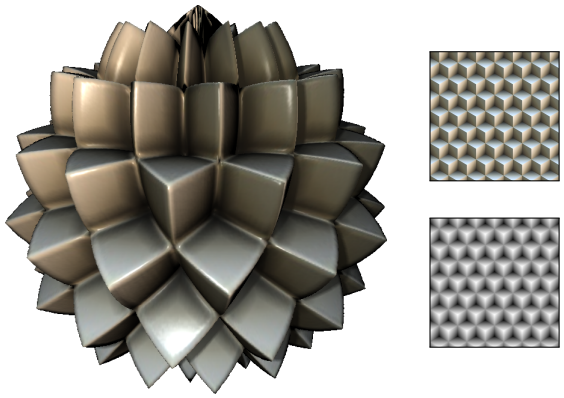


Figure 3: Displaced sphere using procedural textures.

Displacement mapping can be easily applied to surfaces by simply adding or subtracting values to the results of SDFs. Fig. 3 shows an example. Using procedural textures, surfaces remain continuous and smooth.

Whereas blending is non-trivial on meshes, twists and displacements are easy, but only look good for highly tessellated meshes. A major advantage of implicit descriptions is that meshes don't have to be tessellated and regenerated over and over again during deformations.

By applying the modulo operator to the domain, infinite replications of an object can be defined. Jittering can be added to break the visible equidistant pattern. For natural scenes, such as many trees in a forest, a Poisson distribution yields much more appealing and convincing results, yet there is no implicit topology between neighboring objects anymore. The results of this implicit replication technique are comparable to those of geometry instancing on graphics hardware.

3.5. Arranging Scenes

Individual objects, described by SDFs $\phi_i(\vec{x})$, can be easily combined to a single SDF representing the entire scene:

$$\min_{i\in S}(\phi_i(\vec{x})) = 0 \Leftrightarrow \vec{x} \in \bigcup_{i\in S}\partial\Omega_i.$$

4. Interactive Modeling using SDFs

Signed distance functions and the transformations described in the previous section form the basis of our modeling environment. Fig. 10 displays the user interface. We provide an intuitive environment that looks and behaves similar to traditional modeling software.

This is accomplished by sticking to well-known user interface design patterns that assist experienced artists in being productive right from the start.

The key feature is the scene management that has to fulfill two requirements. First, it has to provide the flexibility to reflect all aforementioned shapes and transformations. Second, it must be possible to transform the scene representation quickly into a fragment shader that is suitable for real-time rendering on graphics hardware.

In this section, we first describe our interactive rendering system for SDFs and the underlying data structures. Performance is crucial to allow for interactive scene manipulation and immediate visual feedback. Next, we discuss the modeling tools that we provide and demonstrate their flexibility by means of several examples. Eventually, we discuss combinations with traditional polygon modeling, and benefits and drawbacks of our modeling approach.

4.1. Scene Graph Traversal

We store all scene objects and transformations in a connected acyclic scene graph that is presented to the user as a hierarchical tree structure. Primitives are represented by leaf nodes, while internal nodes store transformations and operations applied to their distance functions. Altogether, the root node represents the whole scene. The user can modify the hierarchy by rearranging nodes via drag-and-drop at any time.

After modifying the scene graph, i.e., adding, altering, rearranging, or deleting nodes, a new fragment shader has to be generated. This is achieved by a depth-first traversal of the scene graph. Every visited node N prepends its required uniform variables and functions to the shader, applies its transformations, and asks its children to append their source fragments as arguments of the result statement of N. Eventually, primitives write out calls to their corresponding basic signed distance functions. To compose a scene out of individual objects, min-functions are used (see Section 3.5).

A small example follows, using the object shown in Fig. 2. It has been composed using blend and twist transformations on a box and a sphere as follows:

$$\phi\Big[\xi_a^{-1}\Big((1-\alpha)\cdot\phi_{\text{Box}}(\vec{x})+\alpha\cdot\phi_{\text{Sphere}}(\vec{x})\Big)\Big].$$

Listing 1 shows the generated fragment shader source. There, `p` corresponds to $\vec{x}$, `vec2 a` to the linear twist definition of ξ_a, and `alpha` to the blend factor α.

Listing 1: Fragment Shader Source Example

```
float sp(vec3 p, float r)
{
  return length(p) - r;
}

float box(vec3 p, vec3 dim)
{
  p = abs(p) - dim;
  return max(max(p.x, p.y), p.z);
}

uniform float alpha;
uniform vec2 a;

float twist(vec3 p)
{
  float new_x = p.x * cos(a.x*p.y + a.y)
    - p.z * sin(a.x*p.y + a.y);

  p.z = p.x * sin(a.x*p.y + a.y)
    + p.z * cos(a.x*p.y + a.y);

  p.x = new_x;

  return box(p, vec3(0.5, 1.5, 0.5))
    * (1.0-alpha) + sp(p, 1.0) * alpha;
}

float phi(vec3 p)
{
  return twist(p);
}
```

4.2. Ray Marching

For a point $\vec{x}$ on a ray, evaluating the all-embracing min-function (see Section 3.5) of the generated fragment shader yields the distance from $\vec{x}$ to its closest surface. One can safely march this distance within a single step along the ray without penetrating a surface. This technique is called sphere tracing [4], and accelerates ray marching significantly. When the distance is below a certain threshold, $\vec{x}$ is considered to be a surface point.

As discussed in the previous section, certain transformations violate the Euclidean distance. A counteracting heuristic method is to sample the space within the scene at several points and then adjust the marching step size according to the gradient. Nonetheless, this is likely to fail for extreme pathological surfaces and volatile gradients. If the surface is unintentionally penetrated, and given that the object has not been skipped at all, it is still possible to revert to the last step and proceed again conservatively.

As a last resort, we offer an interface to manually decelerate the ray marching process when approaching surfaces, however, this user interaction is required rarely and for extreme deformations only. We experienced good results by progressively decelerating sphere tracing when approaching surfaces. This allows for

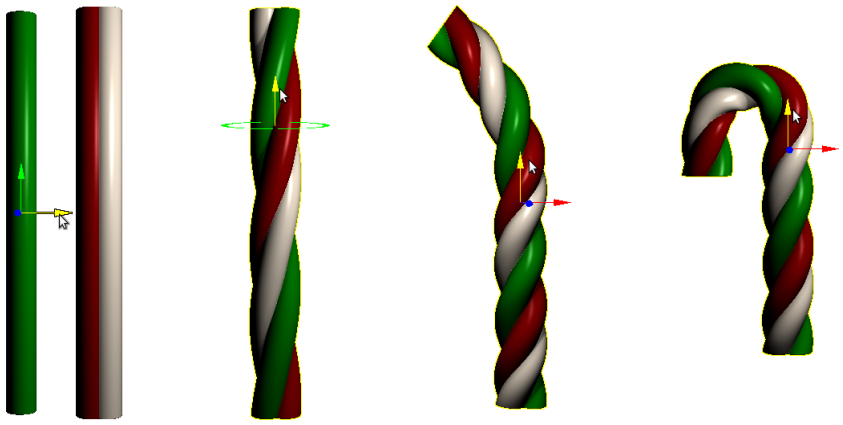


Figure 4: Modeling a candy cane using manipulators for translations, twisting, and bending.

marching with larger step sizes far away from volumes, and proceed carefully only near wrinkles in pathological surfaces, i.e., where it is really needed.

4.3. Manipulating Objects

Manipulators are convenient and powerful visual tools to interactively alter objects and control transformations. Our modeling tool offers the functionality of common and established manipulators, known from professional traditional modeling environments, to work on SDFs. This makes our environment as intuitive and convenient as possible.

Primitives and transformations possess particular, individual properties. For instance, an ordinary sphere has a radius, and a rigid body transformation holds both a translation vector and a rotation quaternion. These properties are not numerically fixed within the fragment shader; instead, they are represented by corresponding uniform variables. Thus, we can modify many scene parameters without regenerating and recompiling the shader.

We propose a three-way mapping between properties, their corresponding uniform variables, and the state of appropriate interactive manipulators. Rigid body transformations, to give an example, are mapped to arrows and disks, for translations along and rotations around particular axes. During manipulation, the state of the manipulator is interpreted and the value of its mapped property is updated. The associated uniform values in the fragment shader are updated upon changes. In this way, rendering a single frame is sufficient to reflect the current state, which is crucial for interactivity.

Fig. 4 gives a small example, where manipulators are used to model a candy cane. The user starts by placing three cylinders using translation manipulators. Then, joint twisting of these cylinders begins by dragging up a twisting manipulator. Eventually, a candy cane is formed through bending the object.

A specific characteristic of implicit surfaces is that prior transformations inside the hierarchical scene structure can still be manipulated without any further efforts. In the previous example, the user is still able to modify the twist parameter after bending the candy cane.

In addition to manipulators, our modeling environment offers a property-value editor. This interface suits well for manual adjustments and modifying properties that are not covered by manipulators.

Lastly, an experienced programmer is given the opportunity to directly edit the generated fragment shader source. This provides a wide-ranging freedom of expression to define new types of objects, e.g., fractal shapes like the Menger sponge (see Fig. 7).

5. Implementation Details

Our fully functional interactive modeling environment, based on the concepts mentioned before, is written in C++ using Qt 4.7, OpenGL, and GLSL. A few implementation details follow.

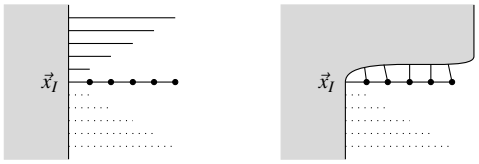
Interactive Scene Modeling. The key component for interactive modeling is a fragment shader that is generated by translating the scene graph into a composition of GLSL code fragments, representing all individual SDFs, operations, and transformations. Subsequently, the sphere tracing code is appended, which is then used to render the scene in real-time.

Hybrid Rendering. An interactive modeling environment provides more than a simple view into the scene. Users are able to interact with the application, to select objects, which appear highlighted then, and to manipulate these objects. All GUI elements in our application are rendered with standard OpenGL and overlayed after ray marching. When combining ray marching with rasterization, everything has to merge seamlessly. Thus, the OpenGL modelview-projection matrices are set according to the camera specified for ray marching. The grid is rasterized with enabled depth test against the scene. For picking and highlighting selected objects we use an additional buffer, containing unique IDs for every visible object.

Shading. Surface points are shaded using the gradient, estimated via central differences, as normal vector. Shadow rays are cast to each light source, and the shadow test is performed using sphere tracing again. We can also render soft shadow effects easily: we keep track of the minimum evaluated distance $d_{\min}$ to any object during ray marching along a shadow ray. For small distances, $0 < d_{\min} < \ell$, we assume that the surface point is located within the penumbra region. The penumbra region extends for an increasing ℓ, and $d_{\min}/\ell \in (0, 1)$ yields a shadowing factor. This is akin to [14], however, instead of having to compute auxiliary distances, we already obtained them during ray marching.

Using SDFs also allows us to increase the shading quality by approximating ambient occlusion as described by Evans [2] (see Fig. 5).

With negligible additional computational cost, we are able to produce plausible soft shadows and ambient occlusion. For modeling purposes, they provide important visual cues, supporting the user or artist in better perceiving objects and their interrelations [8, 18].



(a) Open region, (b) concave region.

Figure 5: Ambient occlusion approximation. The SDF is sampled along the normal of a surface point. For points on open convex regions, it returns the same value as the distance along the normal, whereas for points on occluded concave regions, it yields smaller values.

Figure 6: Organic structure on a displaced torus.

Optimizations. Due to entirely implicit representations, we are able to create complex, large scenes. These, however, are costly to render as many objects contribute to the distance evaluation during ray marching. Bounding volume hierarchies are well suited to reduce the rendering cost in this case. For sophisticated objects involving tedious distance evaluations, an SDF may safely return the distance to their bounding boxes first, given that points in question are outside. Moreover, ray marching is only necessary inside a bounding box.

Interface to Polygonal Meshing. Any part of the scene modeled with SDFs can be exported as a triangle mesh. Our environment uses the 3D surface mesh generator of CGAL [17], the *Computational Geometry Algorithms Library*. Vice versa, we can integrate triangle meshes into our environment. For this, we transform a mesh into a discrete distance field stored as a regular 3D grid of floating point values. Such objects seamlessly integrate into our modeling framework.

6. Results and Discussion

It is impossible to claim that implicit descriptions are generally more appropriate for surface modeling than other established approaches, such as mesh-based, point-based, or voxel-based representations. In the following, we discuss examples where the appropriateness varies from case to case. Moreover, we provide a detailed report on the performance of our system, and discuss the limitations of our approach.

Examples. Our approach is well suited to model organic structures as in Fig. 6, and natural objects, such as tree trunks, for instance. These objects feature noisy and bumpy surfaces with smooth displacements.

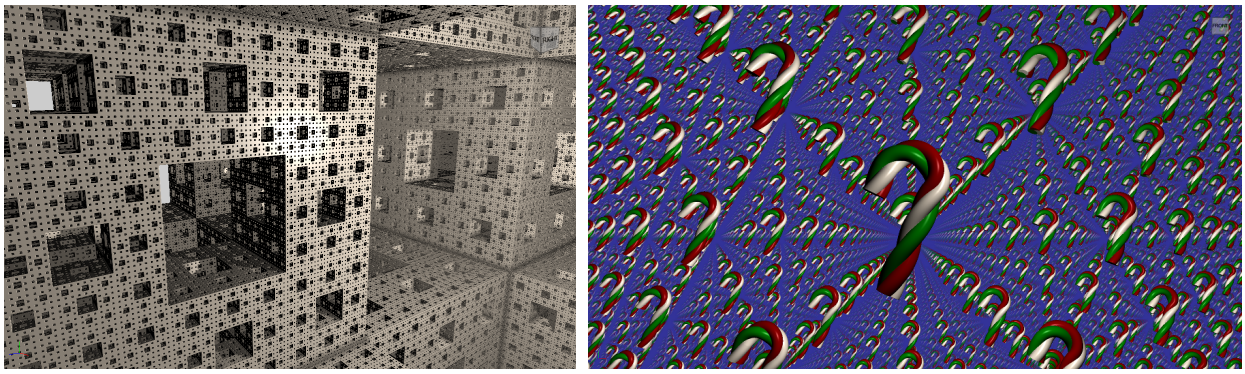


Figure 7: Two examples of exploiting the modulo operator. *Left:* implicitly defined Menger sponge at iteration 5. *Right:* infinite replications of the candy cane, spanning the entire three-dimensional space.

Modeling a coffee mug is a popular choice to demonstrate CSG abilities. For the artist, the modeling process using our environment (Fig. 8) is the very same as with traditional mesh-based tools. Internally, on the other hand, it is less effort and much easier for us to generate and maintain a proper representation. Moreover, the mug features smooth continuous surfaces naturally.

Fig. 7 shows a Menger sponge constructed using SDFs. We start with a single box and carve out an infinite pattern of rectangular cuboids using the modulo operator. This subtractive approach is completely different from constructing a mesh procedurally. Our implicit definition is easy to grasp, whereas mesh construction can be tricky and involved. Nevertheless, neither technique is substantially more practical.

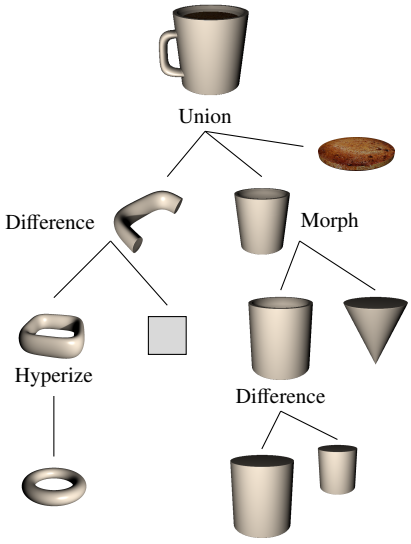


Figure 8: Coffee mug modeled using CSG and SDFs.

Performance. Next, we evaluate the performance of our environment on an Intel Core i7-860 system, equipped with an nVidia GeForce GTX 470 graphics card. The main modeling viewport has a resolution of 1280×1024, and full illumination and shading is enabled.

We obtain a frame rate higher than 400 fps for a single primitive, e.g., a sphere or a box, roughly spanning the entire viewport. We see frame rates of more than 75 fps for the torus depicted in Fig. 6, displaced by layers of procedural noise functions, and the coffee mug (Fig. 8). A single candy cane (Fig. 4) renders with 170 fps. Applying the modulo operator to its domain, as depicted in Fig. 7, produces infinite replications. With disabled shadows, we still obtain an interactive frame rate of 16 fps for the scene.

A Menger sponge at iteration 0 renders with 220 fps for viewport filling views as depicted in Fig. 7. The frame rate decreases for increasing iterations: about 83 fps for one, 42 fps for two, and eventually, 18 fps for five iterations.

Note that image order rendering, such as sphere tracing, is directly dependent on the viewport resolution; we obtain more than 60 fps in 640×480 resolution for the iteration 5 Menger sponge.

Shader Compilation. The more objects are added to the scene, the longer the compile time for the fragment shader takes. We do not meet a problem for rather plain scenes, as they compile within a few milliseconds, but as scene complexity increases, compile time does so as well. Fig. 9 shows how long it takes to compile the fragment shader for increasing numbers of different types of primitives. The plain textual generation of the shader itself is negligible.

Using `#pragma optimize(off)` speeds up GLSL shader compilation in trade for rendering performance.

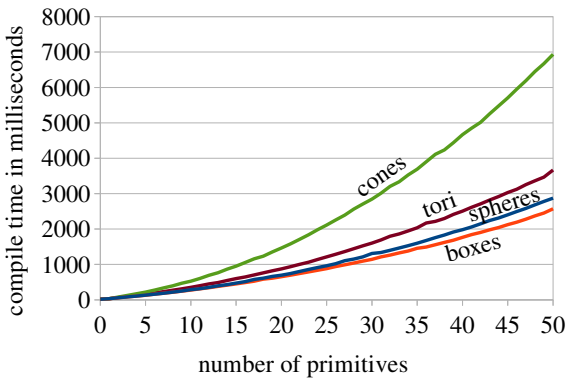


Figure 9: Compiling the fragment shader takes more time for an increasing amount of objects within a scene.

This helps to keep up interactivity and responsiveness, while an optimized version is compiled in the background as a separate CPU thread. Note that optimization results vary heavily dependent on the graphics hardware vendor, driver, and operating system.

Limitations. Rendering performance becomes poor for dense fields full of very detailed and filigree structures, such as blades of grass. Sphere tracing forfeits its efficiency when marching through these areas. However, indirect visualization techniques struggle with these cases as well, since they require unreasonably high-resolution samplings to remain artifact-free.

Currently, our modeling environment is unable to provide a convenient interface for sculpting. This is clearly the domain where flexible explicit representations, e.g., based on voxels or meshes, perform best. However, previous work discussed in Section 2 shows that it is of course possible to sculpt convincing and sophisticated objects with implicit surfaces.

7. Conclusion and Future Work

We demonstrated how compositions of signed distance functions can be constructed and used to implicitly represent complex objects and whole scenes. An intuitive and fully functional modeling environment based on SDFs was presented, enabling artists to create appealing scenes without having to deal with the novel underlying data structures.

We showed that interactive modeling and rendering based on SDFs is possible. Neither intermediate steps for indirect visualization, nor explicit discrete grids are required, avoiding geometric aliasing. Instead, the scene is directly rendered using sphere tracing at interactive frame rates.

A tempting future development is to provide a sophisticated interface for seamlessly merging meshes with implicit surfaces and explicit deformation tools. This allows for accessing the individual benefits of each modeling approach, while compensating for their particular drawbacks. If, however, it is possible to describe everything procedurally and implicitly, everything turns out to be continuous and smooth, allowing for arbitrary, detailed zooms into the scene.

We use manipulators well known from traditional mesh modeling. They fit smoothly into our environment, but it should also be investigated, whether a variety of novel, adapted manipulators exists, that are much more applicable to implicit surfaces.

References

[1] Bloomenthal, J., Shoemake, K., July 1991. Convolution surfaces. SIGGRAPH Comput. Graph. 25, 251–256.

[2] Evans, A., 2006. Fast approximations for global illumination on dynamic scenes. In: SIGGRAPH '06: ACM SIGGRAPH 2006 Courses. ACM, New York, NY, USA, pp. 153–171.

[3] Frisken, S. F., Perry, R. N., Rockwood, A. P., Jones, T. R., 2000. Adaptively sampled distance fields: a general representation of shape for computer graphics. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques. SIGGRAPH '00. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, pp. 249–254.

[4] Hart, J. C., 1996. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. The Visual Computer 12, 527–545.

[5] Hua, J., Qin, H., September 2004. Haptics-based dynamic implicit solid modeling. IEEE Transactions on Visualization and Computer Graphics 10, 574–586.

[6] Loop, C., Blinn, J., July 2006. Real-time gpu rendering of piecewise algebraic surfaces. ACM Trans. Graph. 25, 664–670.

[7] Lorensen, W. E., Cline, H. E., August 1987. Marching cubes: A high resolution 3d surface construction algorithm. SIGGRAPH Comput. Graph. 21, 163–169.

[8] Luft, T., Colditz, C., Deussen, O., 2006. Image enhancement by unsharp masking the depth buffer. ACM Transactions on Graphics 25 (3), 1206–1213.

[9] Martinez Esturo, J., Rössl, C., Theisel, H., 2010. Continuous deformations of implicit surfaces. In: Proceedings of Vision, Modeling, and Visualization (VMV 2010).

[10] Museth, K., Breen, D. E., Whitaker, R. T., Barr, A. H., July 2002. Level set surface editing operators. ACM Trans. Graph. 21, 330–338.

[11] Nealen, A., Igarashi, T., Sorkine, O., Alexa, M., 2007. FiberMesh: Designing freeform surfaces with 3D curves. ACM Transactions on Graphics 26 (3).

[12] Osher, S., Sethian, J. A., November 1988. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. J. Comput. Phys. 79, 12–49.

[13] Osher, S. J., Fedkiw, R. P., October 2002. Level Set Methods and Dynamic Implicit Surfaces, 1st Edition. Springer.

[14] Parker, S., Shirley, P., Smits, B., 1998. Single sample soft shadows. Tech. Rep. UUCS-98-019, Computer Science Department, University of Utah.

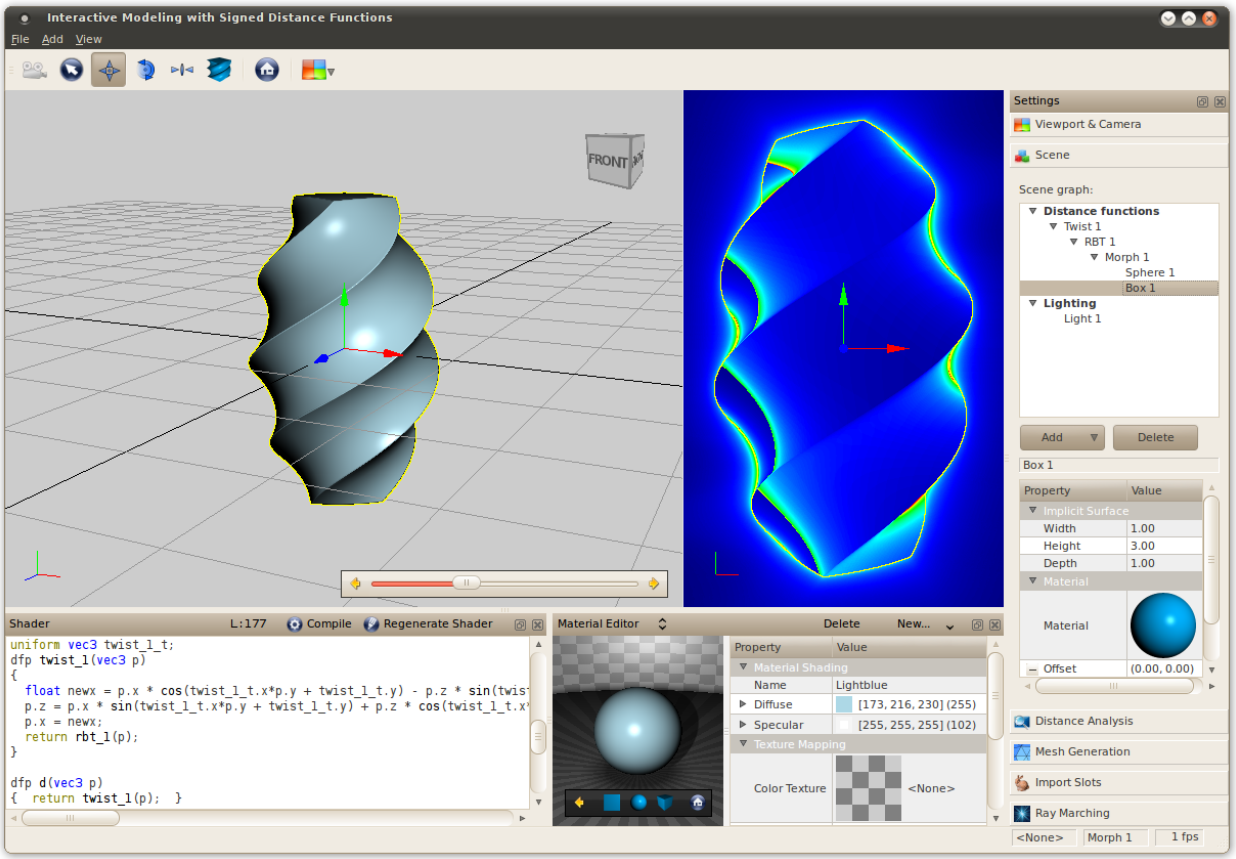


Figure 10: The user interface of our interactive modeling environment. It features a main modeling area and an interface to the scene graph on the right. A work image is displayed to visualize the amount of sphere tracing steps. In addition to modeling using visual manipulators, the user can freely edit the generated fragment shader source to define sophisticated surfaces and transformations.

[15] Perlin, K., Hoffert, E. M., July 1989. Hypertexture. SIGGRAPH Comput. Graph. 23, 253–262.

[16] Richard, V. A., Cartwright, R., Fausett, E., Ossipov, A., Pasko, E., Savchenko, V., 1999. Hyperfun project: a framework for collaborative multidimensional f-rep modeling. In: Proceedings of Implicit Surfaces '99, Eurographics/ACM SIGGRAPH Workshop. pp. 59–69.

[17] Rineau, L., Yvinec, M., 2010. 3D surface mesh generation. In: CGAL User and Reference Manual, 3.6 Edition. CGAL Editorial Board.

[18] Ritschel, T., Smith, K., Ihrke, M., Grosch, T., Myszkowski, K., Seidel, H.-P., 2008. 3D Unsharp Masking for Scene Coherent Enhancement. ACM Trans. Graph. (Proc. of SIGGRAPH 2008) 27 (3).

[19] Schmidt, R., Wyvill, B., Sousa, M. C., Jorge, J. A., 2005. Shapeshop: Sketch-based solid modeling with blobtrees. In: Proceedings of SBIM '05. pp. 53–62.

[20] Sederberg, T. W., 1985. Piecewise algebraic surface patches. Computer Aided Geometric Design 2 (1-3), 53 – 59.

[21] Singh, K., Eugene, F., 1998. Wires: a geometric deformation technique. In: Proceedings of the 25th annual conference on Computer graphics and interactive techniques. SIGGRAPH '98. ACM, New York, NY, USA, pp. 405–414.

[22] Sugihara, M., Wyvill, B., Schmidt, R., 2010. WarpCurves: A tool for explicit manipulation of implicit surfaces. Computers & Graphics 34 (3), 282–291, Shape Modeling International 2010.

[23] Turk, G., O'Brien, J. F., 1999. Shape transformation using variational implicit functions. In: Proceedings of the 26th annual conference on Computer graphics and interactive techniques. SIGGRAPH '99. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, pp. 335–342.

[24] Turk, G., O'Brien, J. F., October 2002. Modelling with implicit surfaces that interpolate. ACM Trans. Graph. 21, 855–873.

[25] Wyvill, B., Galin, E., Guy, A., June 1999. Extending The CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. Computer Graphics Forum 18 (2), 149–158.