and surfaces were investigated before implicit methods; however, there are some good reasons to develop algorithms to visualize implicit surfaces. In this chapter we explore the implications of deriving the data from a modeling process rather than from a scanner.

Despite the computational overhead of finding the implicit surface, designing with implicit modeling techniques offers some advantages over other modeling methods. Many geometric operations are simplified using implicit methods including:

- the definition of blends;
- the standard set operations (union, intersection, difference, etc.) of constructive solid geometry (CSG);
- functional composition with other implicit functions (e.g., R-functions, Barthe blends, Ricci blends, and warping);
- inside/outside tests, (e.g., for collision detection).

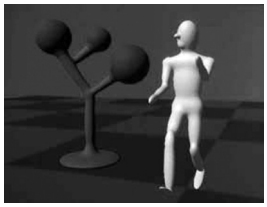


Figure 22.1. Blinn's Blobby Man 1980. *Image courtesy Jim Blinn.*

Visualizing the surfaces can be done either by direct ray tracing using an algorithm as described in (Kalra & Barr, 1989; Mitchell, 1990; Hart & Baker, 1996; deGroot & Wyvill, 2005) or by first converting to polygons (Wyvill, McPheeters, & Wyvill, 1986).

One of the first methods was proposed by Ricci as far back as 1973 (Ricci, 1973), who also introduced CSG in the same paper. Jim Blinn's algorithm for finding contours in electron density fields, known as *Blobby molecules* (J. Blinn, 1982), Nishimura's *Metaballs* (Nishimura et al., 1985) and Wyvills' *Soft Objects* (Wyvill et al., 1986) were all early examples of implicit modeling methods. Jim Blinn's *Blobby Man* (see Figure 22.1) was the first rendering of a non-algebraic implicit model.

22.1 Implicit Functions, Skeletal Primitives, and Summation Blending

In the context of modeling an *implicit* function is defined as a function f applied to a point $\mathbf{p} \in \mathbb{E}^3$ yielding a scalar value $\in \mathbb{R}$.

The implicit function $f_i(x, y, z)$ may be split into a distance function $d_i(x, y, z)$ and a *fall-off filter function*[1] $g_i(r)$, where r stands for the distance from the skeleton and the subscript refers to the ith skeletal element.

[1] These functions have been given many names by researchers in the past, e.g., *filter, potential, radial-basis, kernel*, but we use *fall-off filter* as a simple term to describe their appearance.

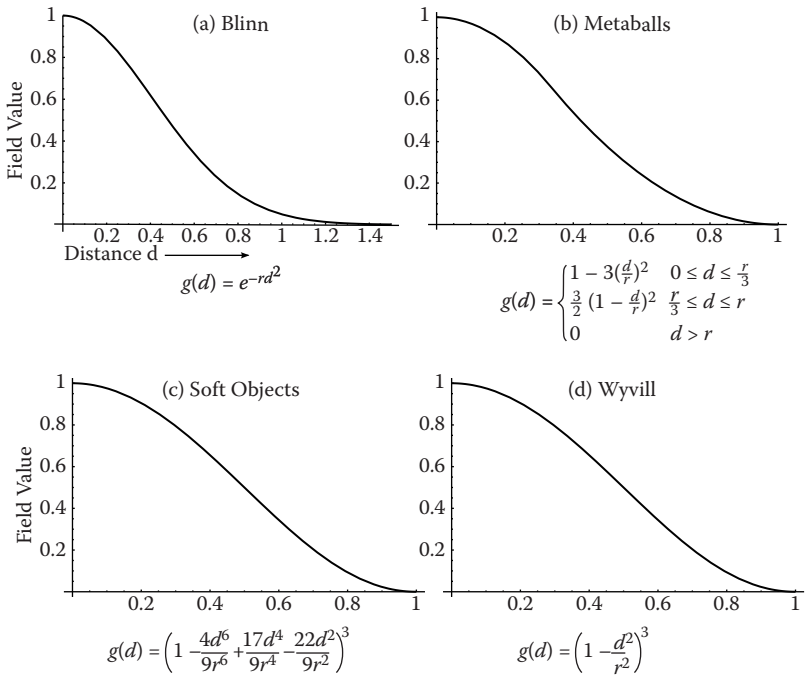


Figure 22.2. Fall-off filter functions ($0 \le r \le 1$). (a) Blinn's Gaussian or "Blobby" function; (b) Nishimura's "Metaball" function; (c) Wyvill et al.'s "soft objects" function; (d) the Wyvill function.

We will use the following notation:

$$f_i(x, y, z) = g_i \circ d_i(x, y, z) \tag{22.2}$$

A simple example is a point primitive, and we take the analogy of a star radiating heat into space. The field value (temperature in this example) may be measured at any point p and can be found by taking the distance from p to the center of the star and supplying the value to a fall-off filter function similar to one of those given in Figure 22.2. In these sample functions, the field is given a value of 1 at the center of the star; the value falls off with distance. The surface of a model may be derived from the implicit function $f(x, y, z)$ as the points of space whose values are equal to some desired *iso-value* (iso); in the star example, a spherical shell for values of iso $\in (0, 1)$.

In general, filter functions (g_i) are chosen so that the field values are maximized on the skeleton and fall off to zero at some chosen distance from the

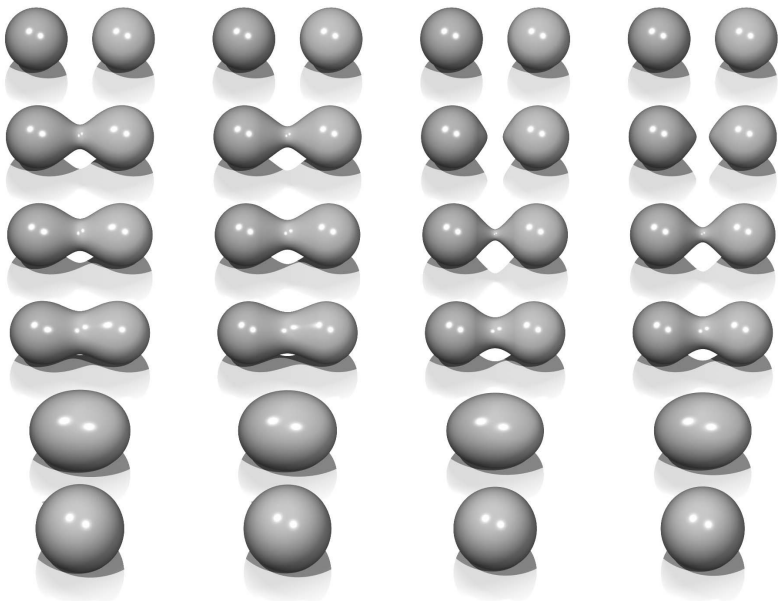


Figure 22.3. Each column shows two point primitives approaching each other. From left to right: the fall-off filter functions used are Blobby, Metaball, soft objects, and Wyvill. *Image courtesy Erwin DeGroot.*

skeleton. In the simple case where the resulting surfaces are blended together, the global field $f(x, y, z)$ of an object, the implicit function, may be defined as

$$f(x, y, z) = \sum_{i=1}^{i=n} f_i(x, y, z), \tag{22.3}$$

where n skeletal elements contribute to the resulting field value. An example is shown in Figure 22.3 in which the field at any point (x, y, z) is calculated as in Equation (22.3).

In this case, two point primitives are placed in close proximity. As the two points are brought together, the surfaces bulge and then blend together. The term *filter* function is used because the function causes the primitives to be blurred together somewhat akin to a filter function for images. The summation blend is the most compact and efficient blending operation that can be applied to implicit surfaces (see Equation (22.3)).

One advantage of using filter functions with finite support is that primitives that are far from p will have zero contribution and thus need not be considered (Wyvill et al., 1986).

22.1.1 C^1 Continuity and the Gradient

The most basic form of continuity is C^0 continuity, which ensures that there are no "jumps" in a function. Higher-order continuity is defined in terms of derivatives of functions (see Chapter 15).

In the case of a 3D scalar field f, the first derivative is a vector function known as the *gradient*, written ∇f and defined as

$$\nabla f(\mathbf{p}) = \left\{ \frac{\partial f(\mathbf{p})}{\partial x}, \frac{\partial f(\mathbf{p})}{\partial y}, \frac{\partial f(\mathbf{p})}{\partial z} \right\}.$$

If ∇f is defined at all points, and the three one-dimensional partial derivatives are each C^0, then f is C^1. Informally, C^1 surface continuity means that the *surface normal* varies smoothly over the surface. The surface normal is the unit vector perpendicular to the surface. If no unique surface normal can be defined on the edge of a cube, for example, then the surface is not C^1. For points on an implicit surface, the surface normal can be computed by normalizing the gradient vector ∇f. In the example of the circle, points inside have a negative value and those on the outside have a positive one. For many types of implicit surfaces, the sense of inside and outside is inverted, and since the normal vector must always point outward, it can be opposite to the gradient direction.

Skeletal implicit primitives are created by applying a fall-off filter function to an unsigned distance field as in Equation (22.2). Although the distance field is never C^1 at the skeleton, these discontinuities can be removed by using a suitable fall-off function (Akleman & Chen, 1999). If an operator, g, combines implicit functions, f_1 and f_2, where all points are C^1, then $g(f_1, f_2)$ is not necessarily C^1. For example, it is possible to make a sharp CSG junction using the min and max operators. The combination is *not* C^1 continuous because the min and max operators don't have that property (see Section 22.5).

The analysis of operators is complicated by the fact that it is sometimes desirable to create a C^1 discontinuity. This case occurs whenever a crease in the surface is desired. For example, a cube is not C^1 because tangent discontinuities occur at each edge. To create creases using C^1 primitives, the operator must introduce C^1 discontinuities, and hence cannot be C^1 itself.

22.1.2 Distance Fields, R-Functions, and F-Reps

The *distance field* is defined with respect to some geometric object T:

$$\mathbf{F}(\mathsf{T}, \mathbf{p}) = \min_{\mathbf{q} \in \mathsf{T}} |\mathbf{q} - \mathbf{p}| .$$

Visually, $\mathbf{F}(\mathsf{T}, \mathbf{p})$ is the shortest distance from $\mathbf{p}$ to T.Hence, when $\mathbf{p}$ lies on T,

$\mathbf{F}(\mathsf{T}, \mathbf{p}) = 0$ and the surface created by the implicit function is the object T. Outside of T, a nonzero distance is returned. The function T can be any geometric entity embedded in 3D—a point, curve, surface, or solid. Procedural modeling with distance fields started with Ricci (Ricci, 1973); *R-functions* (Rvachev, 1963) were first applied to shape modeling more than 20 years later (see (Shapiro, 1994) and (A. Pasko, Adzhiev, Sourin, & Savchenko, 1995)).

An R-function or Rvachev function is a function whose sign can change if and only if the sign of one of its arguments changes; that is, its sign is determined solely by its arguments. R-functions provide a robust theoretical framework for boolean composition of real functions, permitting the construction of C^n CSG operators (Shapiro, 1988). These CSG operators can be used to create blending operators simply by adding a fixed offset to the result (A. Pasko et al., 1995). Although these blending functions are no longer technically R-functions, they have most of the desirable properties and can be mixed freely with R-functions to create complex hierarchical models (Shapiro, 1988). These R-function-based blending and CSG operators are referred to as *R-operators* (see Section 22.4). The Hyperfun system (Adzhiev et al., 1999) is based on *F-reps* (function representation), another name for an implicit surface. The system uses a procedural C-like language to describe many types of implicit surfaces.

22.1.3 Level Sets

It is useful to represent an implicit field discretely via a regular grid (Barthe, Mora, Dodgson, & Sabin, 2002) or an adaptive grid (Frisken, Perry, Rockwood, & Jones, 2000). This is exactly what the polygonization algorithm does in the case of *level sets*; moreover, the grid can be used for various other purposes besides building polygons. Discrete representations of f are commonly obtained by sampling a continuous function at regular intervals. For example, the sampled function may be defined by other volume model representations (V. V. Savchenko, Pasko, Sourin, & Kunii, 1998). The data may also be a physical object sampled using three-dimensional imaging techniques. Discrete volume data has most often been used in conjunction with the *level sets* method (Osher & Sethian, 1988), which defines a means for dynamically modifying the data structure using curvature-dependent speed functions. Interactive modeling environments based on *level sets* have been defined (Museth, Breen, Whitaker, & Barr, 2002), although level sets are only one method employing a discrete representation of the implicit field. Methods for interactively defining discrete representations using standard implicit surfaces techniques have also been explored (Baerentzen & Christensen, 2002).

A key advantage to employing a discrete data structure is its ability to act as a unifying approach for all of the various volume models defined by potential

fields (discrete or not) (V. V. Savchenko et al., 1998). The conversion of any continuous function to a discrete representation introduces the problem of how to reconstruct a continuous function, needed for the combined purposes of additional modeling operations and visualization of the resulting potential field. A well-known solution to this problem is to apply a filter g using the convolution operator (see Chapter 9). The choice of a filter is guided by the desired properties of the reconstruction, and many filters have been explored (Marschner & Lobb, 1994). The salient point is that there is typically a tradeoff between the efficiency of the chosen filter and the smoothness of the resulting reconstruction; see also Section 22.9.

To be interactive, a discrete system must restrict the size of the grid relative to the available computing power. This, in turn, limits the ability of the modeler to include high-frequency details. Additionally, the smoothing triquadratic filter makes it impossible to include sharp edges, should they be desired. A partial solution to this problem is the use of adaptive grids, although with any discrete representation there will be limitations. A discrete grid is used in (Schmidt, Wyvill, & Galin, 2005) to act as a cache representing a *BlobTree* node. The grid in this work is used for fast prototyping and uses trilinear interpolation for position and the slower, more accurate triquadratic interpolation to calculate gradient values, because the eye is more discerning in observing gradient errors than position errors.

22.1.4 Variational Implicit Surfaces

It is often required to convert sampled data to an implicit representation. Variational implicit surfaces interpolate or approximate a set of points using a weighted sum of globally supported basis functions (V. Savchenko, Pasko, Okunev, & Kunii, 1995; Turk & O'Brien, 1999; Carr et al., 2001; Turk & O'Brien, 2002). These radially symmetric basis functions are applied at each sample point. The continuity of such a surface depends on the choice of basis function. The C^2 thin-plate spline is most commonly used (Turk & O'Brien, 2002; Carr et al., 2001). Like Blinn's exponential function (see Figure 22.2), this function is unbounded as is the resulting variational implicit surface.

If the field is is globally C^2, creases cannot be defined;[2] however, anisotropic basis functions can be used to produce fields which change more rapidly and may appear to have creases (Dinh, Slabaugh, & Turk, 2001). At the appropriate scale, the surface is still smooth. The smooth field implies that self-intersections do not

[2] Except see Section 15.2.

occur, and hence volumes are always well-defined. The thin-plate spline guarantees that global curvature is minimized (Duchon, 1977). Variational interpolation has many properties which are desirable for 3D modeling; however, controlling the resulting surfaces can be difficult.

Variational implicit surfaces can also be based on compactly supported radial basis functions (CS-RBFs) to reduce the computational cost of variational interpolation techniques (Morse, Yoo, Rheingans, Chen, & Subramanian, 2001). Each CS-RBF only influences a local region, so computing $f(\mathbf{p})$ requires only evaluation of basis functions within some small neighborhood of $\mathbf{p}$. As with the globally supported counterpart, the resulting field is C^k, creases are not supported, and self-intersections cannot occur.[3] The local support of each basis function results in a bounded global field. This also guarantees that additional iso-contours will be present, as noted by various researchers (Ohtake, Belyaev, & Pasko, 2003; Reuter, 2003).

22.1.5 Convolution Surfaces

Convolution surfaces, introduced by Bloomenthal and Shoemake (Bloomenthal & Shoemake, 1991) are produced by convolving a geometric skeleton S with a *kernel* function h. Hence, the value at any position in space is defined by an integral over the skeleton:

$$f(\mathbf{p}) = \int_{\mathsf{S}} g(\mathbf{r})\, h(\mathbf{p} - \mathbf{r})\, d\mathbf{r}.$$

Any finitely supported function can be used as h; see (Sherstyuk, 1999) for a detailed analysis of different kernels.

Like skeletal primitives, convolution surfaces have bounded fields. Blinn's "Blobby molecules" is the simplest form of a convolution surface (J. Blinn, 1982); in this case, the skeleton consists of points only. This idea was extended by Bloomenthal to include line, arc, triangle, and polygon skeletons (Bloomenthal & Shoemake, 1991). These represent $1D$ and $2D$ primitives; $3D$ primitives were later described by Bloomenthal (Bloomenthal, 1995).

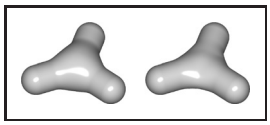


Figure 22.4. Two blended cylinders. Left: summation blend; right: convolution surface with barely discernible bulge (Bloomenthal, 1997). *Image courtesy Erwin DeGroot.*

Combination of convolution surfaces is defined by composition of the underlying geometric skeletons and has the advantage of eliminating the bulges that tend to occur when composing multiple skeletal primitives with additive blending. The surface resulting from convolution of the combined skeleton does not have bulges, as in Figure 22.4, and the field is continuous even if the combined skele-

[3] Note, $k > 0$ depending on the RBF (see Section 15.2).

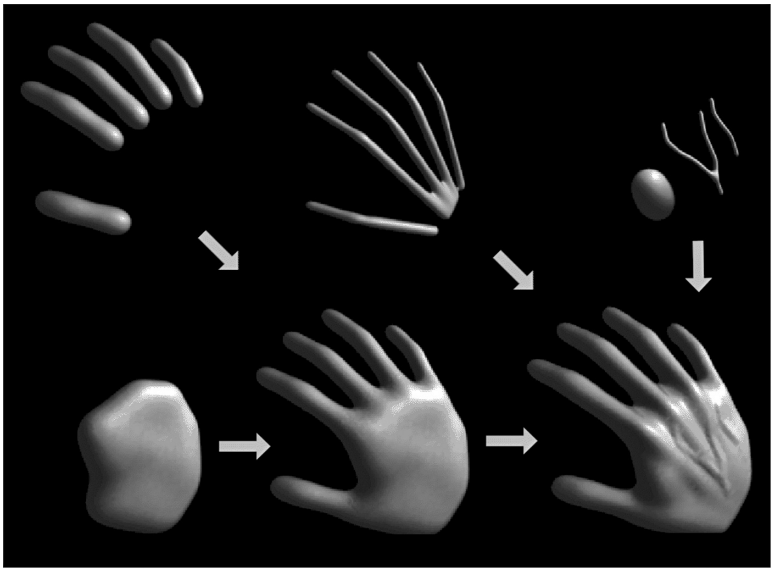


Figure 22.5. Skeletal elements convolved to build a hand model. *Image courtesy Jules Bloomenthal.*

ton is nonconvex. Convolution surfaces are offset a fixed distance from convex portions of a skeleton, but produce a fillet along concave portions of a skeleton.

An example of skeletal elements convolved to build a complex model is shown in Figure 22.5. The hand model contains fourteen primitives.

22.1.6 Defining Skeletal Primitives

As we will see in the following sections rendering the implicit models requires finding the field value and gradient for a large number of points. We need the distance to supply to Equation (22.2) and the gradient is useful for root finding as well as lighting calculations. Supplying the distance to the fall-off filter functions of Figure 22.2 is a matter of calculating the nearest distance to the skeletal primitive, simple for point primitives but a little trickier for more complex geometrical shapes. A line segment primitive (AB) can be defined as a cylinder around a line with hemispherical end caps (see Figure 22.6). Point P_0 lies on the surface where $f(P_0) = \text{iso}$ and $f(P_1) = 0$ since it lies outside of the influence of the line primitive. The distance from some P_i to the line is found by simply projecting onto the line AB and calculating the perpendicular distance, e.g., $|CP_0|$; this can be found

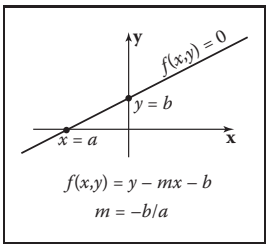


Figure 22.6. Line primitive *ab* and example points p_0, p_1, p_2 showing distance calculation.

from AC, since A, P_0, and B, are all known:

$$\vec{AC} = \vec{AB}\frac{\vec{AP_0} \cdot \vec{AB}}{\|AB\|^2}.$$

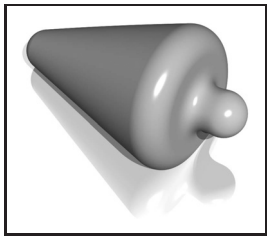


Figure 22.7. Cylinder primitive blended with a sphere. *Image courtesy Erwin DeGroot.*

In Figure 22.6, the field value of $P_2 > 0$, since P_2 is in the hemispherical endcap, which can be checked separately. Variations of this idea can define primitives with endcaps of different radii producing interesting cone shapes. An example is shown in Figure 22.7.

A great variety of geometrical skeletons have been described, and, in principle, it is simply a matter of defining the distance to the skeleton from some point $\mathbf{p}$ and also the gradient at $\mathbf{p}$. For example, an offset surface of a triangle can be defined from the vertices of the triangle and a radius r. A simple way to implement this is to use line segment primitives to describe bounding cylinders connecting the vertices (radius r). The distance from a point $\mathbf{q}$ within the triangle that does not fall within the bounding fields of one of the line segment primitives is returned as the perpendicular distance to the plane of the triangle. Other examples include an implicit disk, defined by a circle and a thickness parameter, a torus also defined by a circle and the radius of the cross section (or inner and outer circle radii), a circular cone from a disk and a height, a cube with rounded corners, etc. (see Figure 22.8).

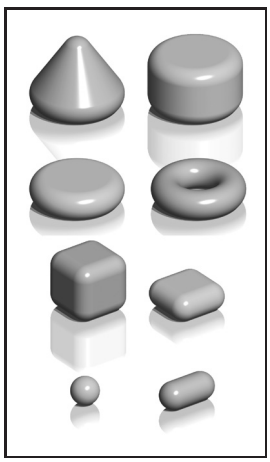


Figure 22.8. Implicit models from various skeletal primitives. *Image courtesy Erwin DeGroot.*

22.2 Rendering

Modeling methods, such as parametric surfaces, lend themselves to visualization, since it is easy to iterate over points on the surface that can be found directly from the defining equations; for example $(x, y) = (\cos\theta, \sin\theta)$, $\theta \in [0, 2\pi)$ produces a circle.

There are two techniques that are commonly used to render implicit surfaces: ray tracing and surface tiling. In practice, a designer wants to visualize an implicit surface model quickly, sacrificing quality for speed for interaction purposes. Prototyping algorithms have been concerned with producing a polygon mesh that can be rendered in real time on modern workstations. Finding the polygonal mesh which best approximates the desired surface is referred to as *polygonization* or *surface tiling*. For animation or for a final visualization, where quality is preferred over speed, ray tracing implicit surfaces directly without first polygonizing produces excellent results.

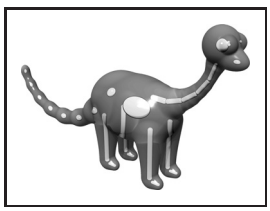


Figure 22.9. A ray-traced dinosaur model showing the underlying skeletal primitives. *Image courtesy Erwin DeGroot.*

As previously mentioned, finding an implicit surface requires searching through space to find the points that satisfy, $f(\mathbf{p}) = 0$. There are two main approaches to executing such a search: space partitioning—partitioning space into

manageable units such as cubes, and non-space partitioning, e.g., marching triangles (Hartmann, 1998; Akkouche & Galin, 2001) and the shrinkwrap algorithm (van Overveld & Wyvill, 2004).

In this chapter, we describe the original space partitioning algorithm and leave it to the reader to explore the more advanced methods. This algorithm together with postprocessing for mesh refinement (see Chapter 12) and caching provide a method for interactive viewing of implicit models on modern workstations.

22.3 Space Partitioning

22.3.1 Exhaustive Search

The basic cubic space partitioning algorithm for tiling implicit surfaces was first published in (Wyvill et al., 1986) and a similar algorithm oriented toward volume visualization, called marching cubes in (Lorensen & Cline, 1987). Since then there have been many refinements and extensions.

A first approach to finding the implicit surface might be to subdivide space uniformly into a regular lattice of cubic cells and calculate a value for every vertex. Each cell is replaced with a set of polygons that best approximates the part of the surface contained within that cell. The problem with this method is that many of the cells will be completely outside or completely inside the volume; thus, many cells that contain no part of the surface are processed. For large grids of data this can be very time consuming and memory intensive.

To avoid storing the whole grid, a hash table is used to store only the cubes that contain a piece of the surface, based on the data structures used in (Wyvill et al., 1986). Working software was published in *Graphics Gems IV* (Bloomenthal, 1990). The algorithm is based on *numerical continuation*; it starts with a seed cube that intersects part of the surface and builds neighboring cubes as necessary to follow the surface.

The algorithm has two parts. In the first part, cubic cells are found that contain the surface and in the second part, each cube is replaced by triangles. The first part of the algorithm is driven by a queue of cubes, each of which contains part of the surface; the second part of the algorithm is table-driven.

22.3.2 Algorithm Description

A fast overview of the algorithm is as follows:

- divide space into cubic voxels;
- search for surface, starting from a skeletal element;

- add voxel to queue, mark it visited;
- search neighbors;
- when done, replace voxel with polygons.

First, space is subdivided into a cubic lattice, and the next task is to find a seed cube containing part of the surface. A cube vertex v_i inside the surface will have a field value $v_i >= \text{iso}$ and a vertex outside the surface will have a field value $v_i < \text{iso}$; thus, an edge with one of each type of vertex will intersect the surface. We call this an *intersecting* edge. The field value at the nearest cube vertex to the first primitive can be evaluated by summing the contributions of the primitives as per Equation (22.3), although other operators can also be used as will be seen later. We will assume that $f(v_0) > \text{iso}$, which indicates that v_0 lies within the solid. The value of iso is chosen by the user; an example is $\text{iso} = 0.5$ when using the *soft* fall-off function, which has some symmetry properties that lead to nice blending (see Figure 22.3). The vertices along one axis are evaluated in turn until a value $v_i < \text{iso}$ is found. The cube containing the *intersecting* edge is the seed cube.

The neighbors of the seed cube are examined, and those that contain at least one *intersecting* edge are added to the queue ready for processing. To process a cube, we examine each face. If any of the bounding edges have oppositely signed vertices, the surface will pass through that face and the face neighbor must be processed. When this process has been completed for all the faces, the second phase of the algorithm is applied to the cube. If the surface is closed, eventually a cube will be revisited and no more unmarked neighbors found, and the search algorithm will terminate. Processing a cube involves marking it as processed and processing its unmarked neighbors. Those that contain *intersecting* edges are processed until the entire surface has been covered (see Figure 22.10).

Each cube is indexed by an *identifying vertex* which we define to be the lower-left far corner (i.e., the vertex with the lowest (x, y, z)-coordinate values (see Figure 22.11)). For each vertex that is inside the surface, the corresponding bit will be set to form the address in an 8-bit table (see Figure 22.11 and Section 22.3.3).

The identifying vertex is addressed by integers i, j, k, computed from the (x, y, z)-coordinate location of the cube such that $x = \text{side} * i$, etc., where side is the size of the cube. The identifying vertex of each cube may appear in as many as eight other cubes, and it would be inefficient to store these vertices more than once. Thus, the vertices are stored uniquely in a chained hash table. Since most of the space does not contain any part of the surface, only those cubes that are visited will be stored. The implicit function value is found for each vertex as it is stored in the hash table.

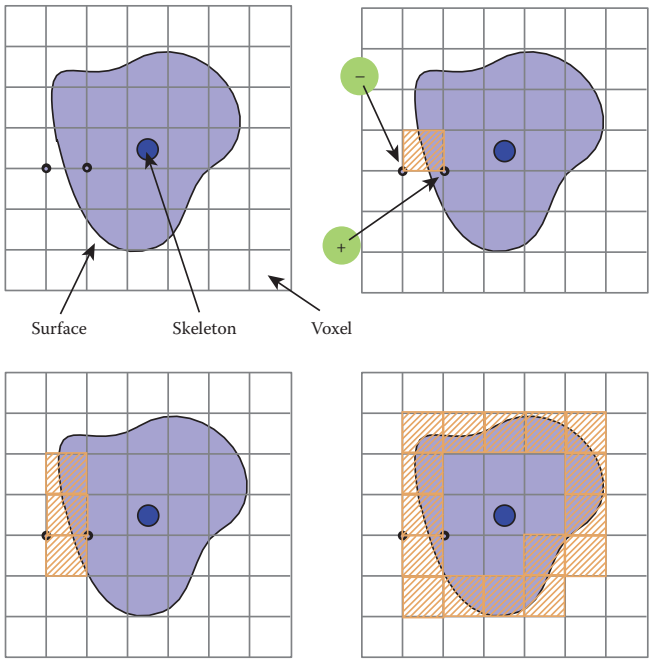
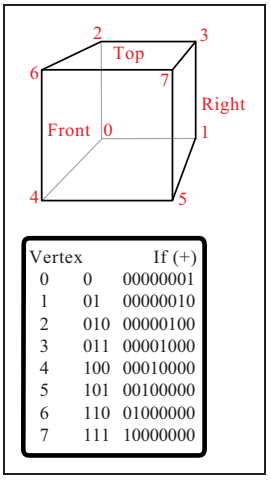


Figure 22.10. A section through the cubic lattice. The + sign indicates a vertex inside the surface ($f(v_i) \geq$ iso) and - is outside $f(v_i <$ iso).

Nothing is known about the topology of the surface so a search must be started from every primitive to avoid any disconnected parts of the surface being missed. A scalar can be used to scale the influence of a primitive. If the scalar can be less than zero, then it is possible to search along an axis without finding an intersecting edge. In this case, a more sophisticated search must be done to find a seed cube (Galin & Akkouche, 1999).



Vertex		If (+)
0	0	00000001
1	01	00000010
2	010	00000100
3	011	00001000
4	100	00010000
5	101	00100000
6	110	01000000
7	111	10000000

Figure 22.11. Vertex numbering.

Data Structures

The hash table entry holds five values:

- the i, j, k lattice indices of the identifying vertex (see Figure 22.11);
- f, the implicit function value of the identifying vertex;
- Boolean to indicate whether this cube has been visited.

The hash function computes an address in the hash table by selecting a few bits out of each of i, j, k and combining them arithmetically. For example, the five least significant bits produces a 15-bit address for a table, which must have a length

of 2^{15}. Such a hash function can be neatly implemented in the C-preprocessor as follows:

```
#define NBITS       5
#define BMASK       037
#define HASH(a,b,c) (((a&BMASK)<<NBITS|b&BMASK)
                                <<NBITS|c&BMASK)
#define HSIZE       1<<NBITS*3
```

The queue (FIFO list) is used as temporary storage to identify the neighbors for processing. The algorithm begins with a seed cube that is marked as visited and placed on the queue. The first cube on the queue is dequeued and all its unvisited neighbors are added to the queue. Each cube is processed and passed to the second phase of the algorithm if it contains part of the surface. The queue is then processed until empty.

22.3.3 Polygonization Algorithm

The second phase of the algorithm treats each cube independently. The cell is replaced by a set of triangles that best matches the shape of the part of the surface that passes through the cell. The algorithm must decide how to polygonize the cell given the implicit function values at each vertex. These values will be positive or negative (i.e., less than or greater than the iso-value), giving 256 combinations of positive or negative vertices for the eight vertices of the cube. A table of 256 entries provides the right vertices to use in each triangle (Figure 22.12). For example, entry $4(00000100)$ points to a second table that records the vertices that bound the *intersecting* edges. In this example, vertex number 2 is inside the surface ($f(V2) >= \text{iso}$) and, therefore, we wish to draw a triangle that connects the points on the surface that intersect with edges bounded by $(V2, V0)$, $(V2, V3)$, and $(V2, V6)$ as shown in Figure 22.13.

Finding Cube-Surface Intersections

Figure 22.13 shows a cube where vertex V_2 is inside the surface and all other vertices are outside. Intersections with the surface occur on three edges as shown. The surface intersects edge $V_2 - V_6$ at the point A. The fastest, but inaccurate, way to calculate A is to use linear interpolation:

$$\frac{f(A) - f(V_2)}{f(V_6) - f(V_2)} = \frac{|A - V_2|}{\text{side}}.$$

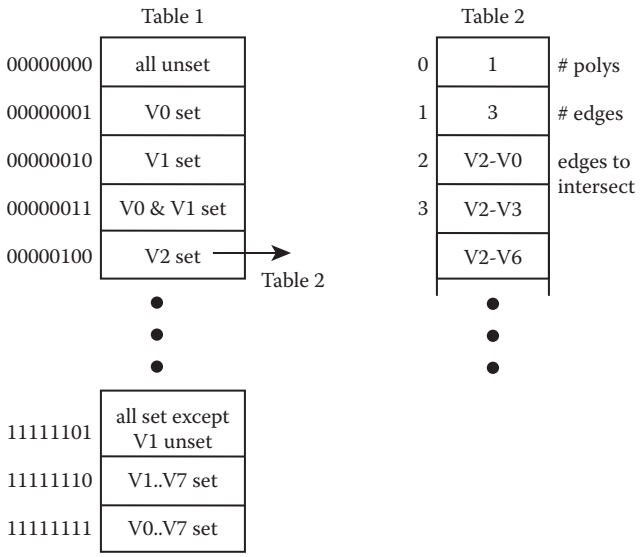


Figure 22.12. Table 2 contains the edges intersected by the surface. Table 1 points to the appropriate entry in Table 2.

If the cube side is 1 and the iso-value sought for $f(A)$ is 0.5, then

$$A = V_3 + \frac{0.5 - f(V_2)}{f(V_6) - f(V_2)}.$$

This works well for a static image, but in animation error differences between frames will be very noticeable. A root-finding method such as *regula falsi* should be employed. This becomes more computationally costly as the gradient is needed to evaluate the point of intersection. The gradient is also needed at surface points for rendering. For many types of primitives it is simpler to find a numerical approximation using sample points around p, as in

$$\nabla f(\mathbf{p}) = \left(\frac{f(\mathbf{p} + \Delta x) - f(\mathbf{p})}{\Delta x}, \frac{f(\mathbf{p} + \Delta y) - f(\mathbf{p})}{\Delta y}, \frac{f(\mathbf{p} + \Delta z) - f(\mathbf{p})}{\Delta z} \right).$$

A reasonable value for Δ has been found empirically to be $0.01 * \text{side}$ where side is the length of a cube edge.

For manufacturing a mesh, as opposed to a set of independent triangles, a second hash table can maintain a list of all the *intersecting edges*. Since each cube edge is shared by up to four neighbors, the edge hash table prevents repetition of the surface-cube edge intersection calculation. The hash address can be derived from the same hash function as for vertices (applied to the edge endpoints).

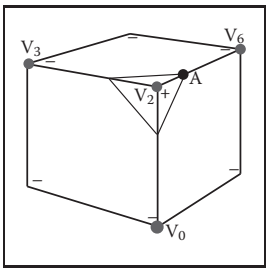


Figure 22.13. Finding the intersection of the surface with a cube edge.

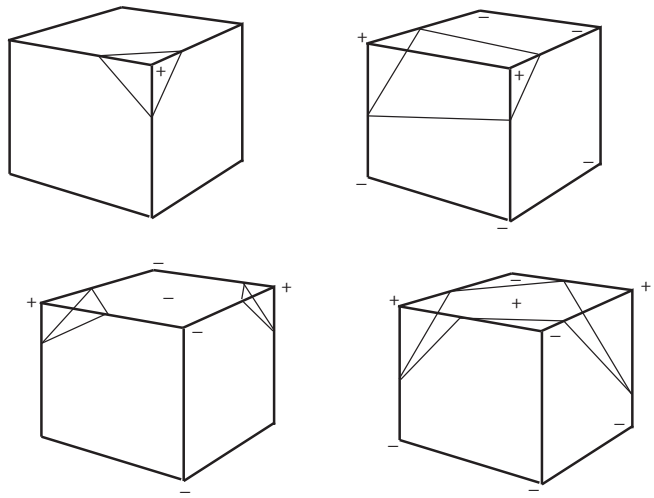


Figure 22.14. Examples of vertices inside (+) and outside (-) the surface. Note the extra sample gives a clue to avoid ambiguous cases.

22.3.4 Sampling Problems

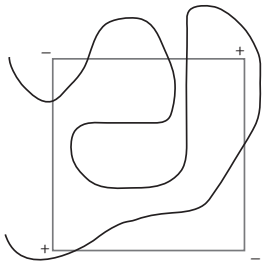


Figure 22.15. Cube too large to capture small variation in implicit function.

Ambiguities occur when opposite corners of a face (or the cube) have the same sign and the other pair of vertices on the face have the opposite sign (see Figure 22.14). A sample taken in the center of the face will give a clue as to whether the cube represents the meeting of two surfaces or a saddle. It should be made clear that a spatial grid stores a sample of the implicit function at every vertex. If the function happens to vary considerably within a cell, the polygonal representation will not show such variations (see Figure 22.15). The surface cannot be resolved by sampling alone unless something is known about the curvature of the surface. A good discussion of this topic appears in (Kalra & Barr, 1989).

This ambiguity problem (not the undersampling problem) is avoided by subdividing the cubic cell into tetrahedra. The tetrahedra can then be polygonized unambiguously. Since there are four vertices in each tetrahedron, a table of 16 entries will provide the correct triangle information. The disadvantage is that approximately twice the number of polygons will be generated.

Subdividing a Cube

Without requiring additional cell vertices, a cube may be decomposed into five or six tetrahedra as shown in Figure 22.16. These decompositions introduce diagonals on the cube faces, and to maintain a consistent diagonal direction between

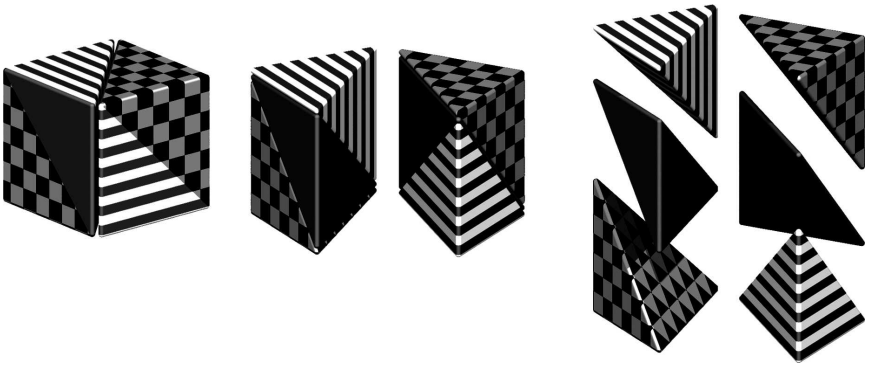


Figure 22.16. Decomposing a cube into six tetrahedra. *Image courtesy Erwin DeGroot.*

neighbors, the six decomposition is preferable. The introduction of diagonal edges produces a higher-resolution surface than replacing each cube directly with triangles. The decomposition into tetrahedra and the replacement of the tetrahedra with triangles are fast, table-driven algorithms, which produce topologically consistent meshes.

22.3.5 Cell Polygonization

Two obvious problems emerge from the use of uniform space subdivision. The size of triangles output by this algorithm do not adapt to the curvature of the surface and a further sample is required to solve the ambiguities, in which cubic cells are replaced by polygons. A space subdivision algorithm based on an octree was developed by Bloomenthal (Bloomenthal, 1988), which does adapt to the curvature of the surface. Cells are subdivided into eight octants and cracks are avoided by using a restricted octree scheme, i.e., neighboring cells cannot differ by more than one level of subdivision. This indeed reduces the number of polygons generated, but full advantage of large cells can only be taken if the flat regions of the surface happen to fall entirely within the appropriate octants. The algorithm proves in practice to be considerably slower than the uniform voxel algorithm and is more complicated to implement.

22.4 More on Blending

Section 22.1 showed that blending can be made to occur when field values are summed. Ricci, in his landmark paper (Ricci, 1973), describes super-elliptic

blending. Given two functions F_A and F_B, previously we simply found the implicit value as $F_{\text{total}} = F_A + F_B$. We can denote this more general blending operator as $A \diamond B$. The Ricci blend is defined as:

$$f_{A \diamond B} = ({f_A}^n + {f_B}^n)^{\frac{1}{n}}. \tag{22.4}$$

It is interesting to point out the following properties:

$$\lim_{n \to +\infty} ({f_A}^n + {f_B}^n)^{\frac{1}{n}} = \max(f_A, f_B),$$
$$\lim_{n \to -\infty} ({f_A}^n + {f_B}^n)^{\frac{1}{n}} = \min(f_A, f_B).$$

Moreover, this generalized blending is associative, i.e., $f_{(A \diamond B) \diamond C} = f_{A \diamond (B \diamond C)}$. The standard blending operator $+$ proves to be a special case of the super-elliptic blend with $n = 1$. When n varies from 1 to infinity, it creates a set of blends interpolating between blending $A + B$ and union $A \cup B$ (see Figure 22.17). Figure 22.27 shows the nodes to be binary or unary; in fact the binary nodes can easily be extended using the above formulation to n-ary nodes.

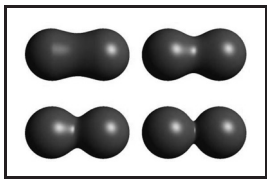


Figure 22.17. By varying n, the Ricci blend may be made to change smoothly from blend to union. *Image courtesy Erwin DeGroot.*

The power of Ricci's operators is that they are *closed* under the operations on the space of all possible implicit volumes, meaning that an application of an operator simply produces another scalar field defining another implicit volume. This new field can be composed with other fields, again using Ricci's operators. Equation (22.4) will always produce the exact union of two implicit volumes, regardless of how complex they are. Compared with the difficulties involved in applying boolean CSG operations to B-rep surfaces, solid modeling with implicit volumes is incredibly simple.

Following Pasko's functional representation (A. Pasko et al., 1995), another generalized blending function may be defined:

$$f_{A \diamond B} = \left(f_A + f_B + \alpha \sqrt{{f_A}^2 + {f_B}^2} \right) ({f_A}^2 + {f_B}^2)^{\frac{n}{2}}.$$

When $\alpha \in [-1, 1]$ varies from -1 to 1, it creates a set of blends interpolating the union and the intersection operators. However, this operator is no longer associative which is incompatible with the definition of n-ary operators.

22.5 Constructive Solid Geometry

Implicit models are frequently termed *implicit surfaces*; however, they are inherently volume models and useful for *solid modeling* operations. Ricci introduced a

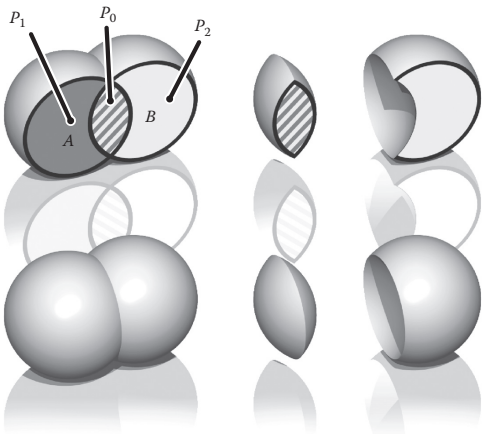


Figure 22.18. Ricci operators for CSG. *Image courtesy Erwin DeGroot.*

constructive geometry for defining complex shapes from operations such as union, intersection, difference, and blend upon primitives (Ricci, 1973). The surface was considered as the boundary between the half spaces $f(\mathbf{p}) < 1$, defining the inside, and $f(\mathbf{p}) > 1$ defining the outside. This initial approach to solid modeling evolved into *constructive solid geometry* or CSG (Ricci, 1973; Requicha, 1980). CSG is typically evaluated bottom-up according to a binary tree, with low-degree polynomial primitives as the leaf nodes and internal nodes representing Boolean set operations. These methods are readily adapted for use in implicit modeling, and in the case of skeletal implicit surfaces, the Boolean set operations union $\cup_{\max}$, intersection $\cap_{\min}$ and difference $\setminus_{\text{minmax}}$ are defined as follows (Wyvill, Galin, & Guy, 1999):

$$\begin{aligned} \cup_{\max} \quad f &= \max_{i=0}^{k-1}(f_i), \\ \cap_{\min} \quad f &= \min_{i=0}^{k-1}(f_i), \\ \setminus_{\text{minmax}} \quad f &= \min\left(f_0, 2 * \text{iso} - \max_{j=1}^{k-1}(f_j)\right). \end{aligned} \tag{22.5}$$

The Ricci operators are illustrated in Figure 22.18 for point primitives A and B. For union (bottom left) the field at all points inside the union will be the greater of $f_A()$ and $f_B()$. For intersection (center), points in the region marked as P_1 will have value $\min(f_A(P_1), f_B(P_1)) = 0$, since the contribution of B will be zero outside of its range of influence. Similarly, for the region marked as P_2, (influence of A is zero, i.e., the minimum) leaving only the intersection region with positive values. Difference works similarly using the iso-value in the three

marked regions (P_i) as follows:

$$\begin{aligned} f(P_0) &= \min\left(f_B(P_0), 2 * \text{iso} - f_A(P_0)\right) \\ &= \min([\text{iso}, 1], [2 * \text{iso} - 1, \text{iso}]) \\ &= [2 * \text{iso} - 1, \text{iso}] < \text{iso} \\ f(P_1) &= \min\left(f_B(P_1), 2 * \text{iso} - f_A(P_1)\right) \\ &= \min([0, \text{iso}], [2 * \text{iso} - 1, \text{iso}]) < \text{iso} \\ f(P_2) &= \min\left(f_B(P_2), 2 * \text{iso} - f_A(P_2)\right) \\ &= \min([\text{iso}, 1], [\text{iso}, 2 * \text{iso}]) >= \text{iso} \end{aligned}$$

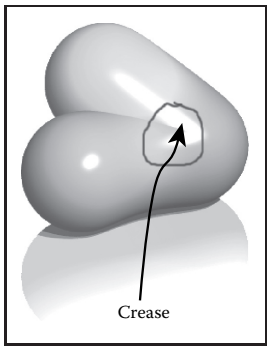


Figure 22.19. Two point primitives on the left are connected by the Ricci union. A third primitive is blended to the result, creating an unwanted crease in the field. *Image courtesy Erwin DeGroot.*

CSG operators create creases, i.e., C^1 discontinuities. For example, the $\min()$ operator (Equation (22.5)) creates C^1 discontinuities at all points where $f_1(\mathbf{p}) = f_2(\mathbf{p})$. When applied to two spheres, the discontinuities produced by this union operator result in a crease on the surface, as shown in Figure 22.18, which is the desired result. Discontinuities unfortunately extend into the field outside of the surface, which is not visible in this image. If a blend is then applied to the result of the union, the C^1-discontinuous plane in the field produces a shading discontinuity (Figure 22.19).

The problem can be avoided to an extent (G. Pasko, Pasko, Ikeda, & Kunii, 2002), and CSG operators have been developed that are C^1 at all points except those where $f_1(\mathbf{p}) = f_2(\mathbf{p}) = \text{iso}$ (Barthe, Dodgson, Sabin, Wyvill, & Gaildrat, 2003).

22.6 Warping

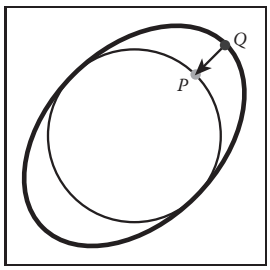


Figure 22.20. Point Q returns the field value for point P.

The ability to distort the shape of a surface by warping the space in its neighborhood is a useful modeling tool. A warp is a continuous function $w(x, y, z)$ that maps $\mathbb{R}^3$ onto $\mathbb{R}^3$. Sederberg provides a good analogy for warping when describing free form deformations (Sederberg & Parry, 1986). He suggests that the warped space can be likened to a clear, flexible, plastic parallelepiped in which the objects to be warped are embedded. A warped element may be defined by simply applying some warp function $w(\mathbf{p})$ to the implicit equation:

$$f_i(x, y, z) = g_i \circ d_i \circ w_i(x, y, z). \tag{22.6}$$

A warped element may be fully characterized by the distance to its skeleton $d_i(x, y, z)$, its fall-off filter function $g_i(r)$, and eventually its warp function $w_i(x, y, z)$. To render or perform operations on an implicit surface, the implicit

value of many points $f(P)$ must be found. First, P is transformed by the warp function to some new point Q, and $f(Q)$ is returned in place of $f(P)$. In Figure 22.20, instead of returning the implicit value of some point $f(Q)$, the value for $f(P)$ is returned. In this case, the iso-value is returned and the implicit surface (curve in 2D) passes through Q instead of P. Thus, the circle is warped into an ellipse.

Barr introduced the notion of global and local deformations using the operations of *twist*, *taper*, and *bend* applied to parametric surfaces (Barr, 1984). The deformations can be nested to produce models such as the one shown in Figure 22.27. Conceptually, these are easy to apply to an implicit surface, as indicated in Equation (22.6).

Note that the normal cannot be calculated in a similar manner to warping a point. This problem is similar to the problem outlined in Section 13.2 on instancing. In this case, the normal can most easily be approximated using Equation (22.3.3) although the use of the Jacobian, as suggested in (Barr, 1984), yields precise results. The Barr warps are described in the following sections.

22.6.1 Twist

In this example, the twist is around the z-axis by θ (see Figure 22.21) for three blended implicit cylinders with a twist warp applied to them.

The twist around z is expressed as

$$w(x,y,z) = \left\{ \begin{array}{c} x * \cos(\theta(z)) - y * \sin(\theta(z)) \\ x * \sin(\theta(z)) + y * \cos(\theta(z)) \\ z \end{array} \right\}.$$

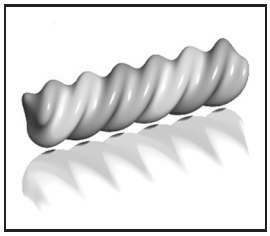


Figure 22.21. Three blended implicit cylinders twisted together. *Image courtesy Erwin DeGroot.*

22.6.2 Taper

Taper is applied along one major axis. A linear taper has proved to be the most useful although quadratic and cubic tapers are easily implemented. For example, a linear taper along the y-axis involves changing both x- and z-coordinates. (See Figure 22.22.) A linear scale is applied to y between $y_{\max}$ and $y_{\min}$:

$$s(y) = \frac{y_{\max} - y}{y_{\max} - y_{\min}} \qquad w(x,y,z) = \left\{ \begin{array}{c} s(y)x \\ y \\ s(y)z \end{array} \right\}$$

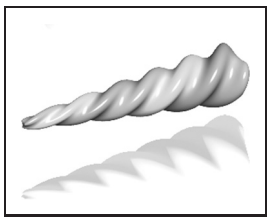


Figure 22.22. Three blended implicit cylinders, twisted then tapered. *Image courtesy Erwin DeGroot.*

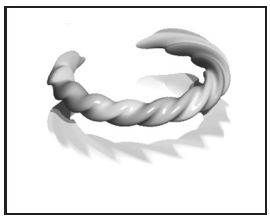


Figure 22.23. Three blended implicit cylinders, twisted together, tapered and bent. *Image courtesy Erwin DeGroot.*

22.6.3 Bend

Bend is also applied along one major axis. (See Figure 22.23.) For the bend example below, the bending rate is k measured in radians per unit length, the axis of the bend is $(x_0, 1/k)$, and the angle θ is defined as $(x - x_0) * k$. The bend around z is

$$w(x,y,z) = \left\{ \begin{array}{c} -\sin(\theta) * (y - 1/k) + x_0 \\ \cos(\theta) * (y - 1/k) + 1/k \\ z \end{array} \right\}$$

22.7 Precise Contact Modeling

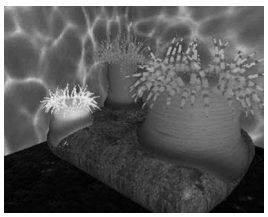


Figure 22.24. Sea anemone deforms to implicit rock. *Image courtesy Mai Nur and X. Liang.*

Precise contact modeling (PCM) is a method of deforming implicit surface primitives in contact situations while maintaining a precise contact surface with C^1 continuity (Gascuel, 1993). PCM is important in that it is a simple and automatic way of showing how a model can react to its environment. This cannot be so easily done with non-implicit methods (see Figure 22.24).

PCM is implemented by the inclusion of a deforming function $s(p)$ that modifies the field value returned for each point. For each pair of objects, collision is first detected using a bounding-box test. Once it is established that a collision is likely, PCM is applied. A local, geometric deformation term s_i is computed and added to the implicit function f_i. The volume of the colliding objects is divided into an interpenetration region and a deformation region. The result of applying s_i is that the interpenetration region is compressed so that contact is maintained without interpenetration occurring (see Figure 22.25). The effect of s_i is attenuated to zero within the propagation region so that the volume outside of the two regions is not deformed.

Given two skeletal elements generating fields $f_1(p)$ and $f_2(p)$, the surface around each one is calculated as

$$\begin{aligned} f_1(p) + s_1(p) &= 0, \\ f_2(p) + s_2(p) &= 0. \end{aligned}$$

We need to generate a surface common to both elements (dotted line in Figure 22.25), i.e., where they share a solution in the interpenetration region for some p in that region:

$$\begin{aligned} s_1(p) - f_1(p) &= \text{iso}, \\ s_2(p) - f_2(p) &= \text{iso}. \end{aligned} \tag{22.7}$$

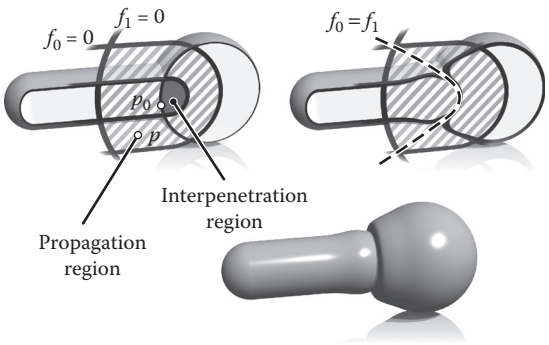


Figure 22.25. A 2D slice through objects in collision showing the various regions and PCM deformation. *Image courtesy Erwin DeGroot.*

Intuitively, the deeper within object 1 that object 2 penetrates, the higher the implicit value of object 1 and thus the more that object 2 will be compressed.

The function, s_i is defined to produce a smooth junction at the boundary of the interpenetration region, in other words where $s_i = 0$ but its derivative is greater than zero. From here to the boundary of the propagation region, s_i is used to attenuate the propagation to zero. The *nearest* point on the interpenetration region boundary p_0 is found by following the gradient.

Within the propagation region $s_i(p) = h_i(r)$, where $p = (x, y, z)$ is the point whose implicit value is being calculated and $r = \|p - p_0\|$ (see Figure 22.26). The value of r_i, set by the user, defines the size of the propagation region; no deformation occurs beyond this region. To control how much the objects inflate in the propagation region, the user provides a value for the parameter α. The maximum

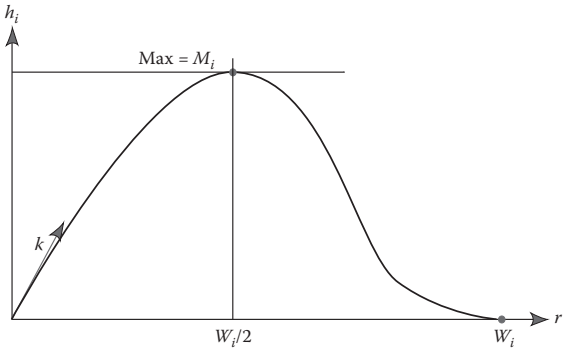


Figure 22.26. The function, $h_i(r)$ is the value of the deformation function w_i in the propagation region.

value of h_i is M_i. The current minimum of s_i is negative in the interpenetration region and is given as $s_{i\min}$, where $M_i = -\alpha_i s_{i\,\min}$. Thus an object will be compressed in the interpenetration region and will inflate in the propagation region. The equation for h_i is formed in two parts by two cubic polynomials that are designed to join at $r = r_i/2$, where the slope is zero:

$$\begin{aligned} c &= \frac{4(w_i k - 4M_i)}{w_i^3}, \\ d &= \frac{4(3M_i - w_i k)}{w_i^2}, \\ h_i(r) &= cr^3 + dr^2 + kr \quad \text{if} \quad r \in [0, w_i/2], \\ h_i(r) &= \frac{4M_i}{w_i^3}(r - w_i)^2(4r - w_i)^3 \quad \text{if} \quad r \in [w_i/2, w_i]. \end{aligned}$$

It is desirable that we have C^1-continuity as we move from the interpenetration to the propagation region. Thus, $h_i'(0) = k$ in Figure 22.26, is the directional derivative of s_i at the junction (marked as p_0 in Figure 22.25). As indicated in Equation (22.7), $s_i = -f_i$ in the interpenetration region, thus:

$$k = \|\nabla(f_i, p_0)\|$$

PCM is only an approximation to a properly deformed surface, but it is an attractive algorithm due to its simplicity.

22.8 The BlobTree

The *BlobTree* is a method that employs a tree structure that extended the CSG tree to include various blending operations using skeletal primitives (Wyvill et al., 1999). A system with similar capabilities, the *Hyperfun* project, used a specialized language to describe F-rep objects (Adzhiev et al., 1999).

In the BlobTree system, models are defined by expressions that combine implicit primitives and the operators $\cup$ (union), $\cap$ (intersection), $-$ (difference), $+$ (blend), $\diamond$ (super-elliptic blend), and w (warp). The BlobTree is not only the data structure built from these expressions but also a way of visualizing the structure of the models. The operators listed above are binary with the exception of warp, which is a unary operator. In general it is more efficient to use n-ary rather than binary operators. The BlobTree incorporates affine transformations as nodes so that it is also a scene graph and primitives (e.g., skeletons) form the leaf nodes.

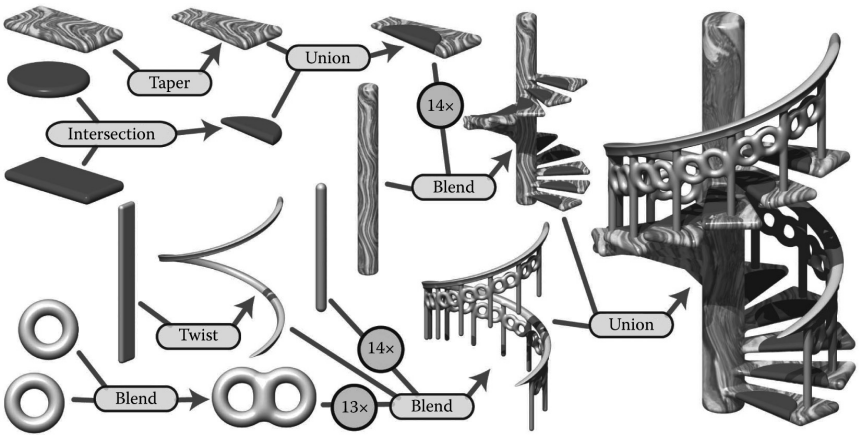


Figure 22.27. BlobTree. The spiral staircase is built from a central textured cylinder to which the stairs and the railing are blended. The railing is comprised of a series of cylinders blended with two circle (torus) primitives, blended together and further blended with a vertical cylinder. The BlobTree is also a scene graph and instancing nodes repeat the various parts transformed by the appropriate matrices. Each stair is made from a tapered polygon primitive (that becomes an offset surface); intersection and union nodes combine the inflated disk with the stair.

22.8.1 Traversing the BlobTree

An example of a BlobTree including the Barr warps and CSG operations is shown in Figure 22.27. Other nodes can include 2D texturing (Schmidt, Grimm, & Wyvill, 2006), precise contact modeling, as well as animation and other attributes. The traversal of the BlobTree is in essence very simple. All that is required to render the object either by polygonizing or ray tracing is to find the implicit value of any point (and the corresponding gradient). This can be done by traversing the tree. Polygonization and ray-tracing algorithms need to evaluate the implicit field function at a large number of points in space. The function $f(\mathcal{N}, M)$ returns the field value for the node $\mathcal{N}$ at the point M, which depends on the type of the node. The values $\mathcal{L}$ and $\mathcal{R}$ indicate that the left or right branch of the tree is explored. The algorithm below is written (for simplicity) as if the tree were binary:

function $f(\mathcal{N}, M)$:

- primitive: $f(M)$;
- warp: $f(\mathcal{L}(\mathcal{N}), w(M))$;
- blend: $f(\mathcal{L}(\mathcal{N}), M) + f(\mathcal{R}(\mathcal{N}), M))$;
- union: $\max(f(\mathcal{L}(\mathcal{N}), M), f(\mathcal{R}(\mathcal{N}), M))$;

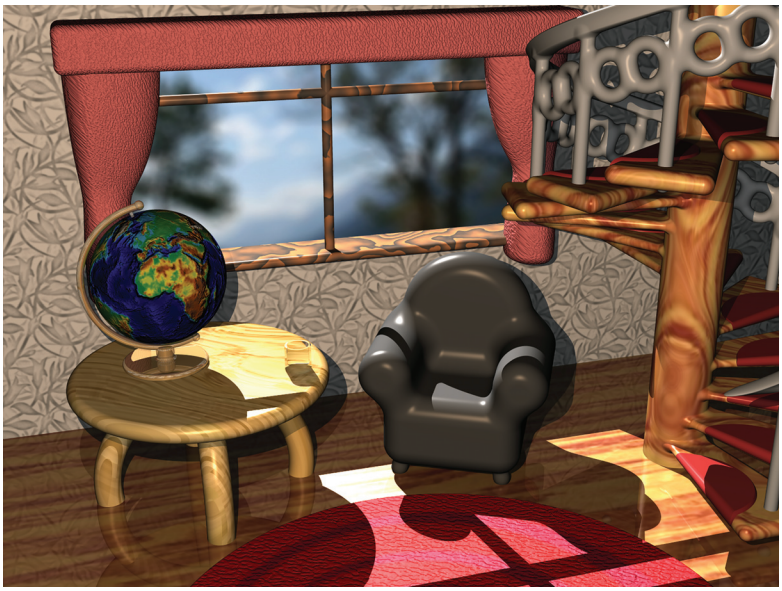


Figure 22.28. "Spiral Stairs." A complex BlobTree implicit model created in Erwin DeGroot's BlobTree.net system.

- intersection: $\min(f(\mathcal{L}(\mathcal{N}), M), f(\mathcal{R}(\mathcal{N}), M))$;
- difference: $\min(f(\mathcal{L}(\mathcal{N}), M), -f(\mathcal{R}(\mathcal{N}), M))$.

A complex BlobTree model showing many of the features that have been integrated is shown in Figure 22.28.

Figure 22.29. Outlines are inflated. *Image courtesy Erwin DeGroot.*

22.9 Interactive Implicit Modeling Systems

Early sketch-based modeling systems, such as Teddy (Igarashi, Matsuoka, & Tanaka, 1999), used a few drawn strokes from the user to infer a polygonal model in 3-space. With better hardware and improved algorithms, sketch-based implicit modeling systems are now possible. Shapeshop uses implicit sweep surfaces to manufacture 3D strokes from 2D user strokes and also preserves the hierarchy of the BlobTree unlike the early systems that produced homogeneous meshes (Schmidt, Wyvill, Sousa, & Jorge, 2005). This enables a user to produce complex models of arbitrary topology from a few simple strokes. The margin figures show a closed drawn stroke (Figure 22.29) inflated into a an implicit sweep and a second sweep (Figure 22.30) that has a smaller sweep object subtracted using CSG.

One of the improvements that made this possible is a caching system that uses a fixed 3D grid of implicit values at each node of the BlobTree representing the values found by traversing the tree below the node (Schmidt, Wyvill, & Galin, 2005). If the value of some point p is required at node N, a value may be returned without traversing the tree below N, provided that part of the tree is unaltered. Instead, an interpolation scheme (see Chapter 9) is used to find a value for p. This scheme speeds up traversal for complex BlobTrees and is one factor in enabling a system to run at interactive rates.

The next generation of implicit modeling systems will exploit hardware and software advances to be able to handle more and more complex hierarchical models interactively. A more complex Shapeshop example is shown in Figure 22.31.

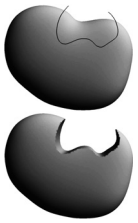


Figure 22.30. BlobTree operations can be applied, e.g., CSG difference. *Image courtesy Erwin DeGroot.*

Figure 22.31. "The Next Step." A complex BlobTree implicit model created interactively in Ryan Schmidt's Shapeshop by artist Corien Clapwijk (Andusan).

Exercises

1. In an implicit surface modeling system the fall-off filter function is defined as

$$f(r) = \begin{cases} 0, & r > R, \\ 1 - r/R, & \text{otherwise}, \end{cases}$$

   where R is a constant. A point primitive placed at $(-1, 0)$ and another at $(1, 0)$ are rendered to show the $f = 0.5$ iso-surface. The value R, the distance where the potential due to the point falls to zero in both cases, is 1.5.

   Calculate the potential at the point $(0, 0)$ and at $+0.5$ intervals until the point $(2.5, 0)$. Sketch the 0.5 contour and the contour at which the field falls to zero.

2. Why are the ambiguous cases in the polygonization algorithm considered to be a sampling problem?

3. Calculate the error involved in using linear interpolation to estimate the intersection of an implicit surface and a cubic voxel.

4. Design an implicit primitive function using the skeleton of your choice. The function must take as input a point and return an implicit value and also the gradient at that point.

23

Global Illumination

Many surfaces in the real world receive most or all of their incident light from other reflective surfaces. This is often called *indirect lighting* or *mutual illumination*. For example, the ceilings of most rooms receive little or no illumination directly from luminaires (light-emitting objects). The direct and indirect components of illumination are shown in Figure 23.1.

Although accounting for the interreflection of light between surfaces is straightforward, it is potentially costly because all surfaces may reflect any given surface, resulting in as many as $O(N^2)$ interactions for N surfaces. Because the entire global database of objects may illuminate any given object, accounting for indirect illumination is often called the *global illumination* problem.

There is a rich and complex literature on solving the global illumination problem (e.g., Appel, 1968; Goral, Torrance, Greenberg, & Battaile, 1984; Cook et al.,

Figure 23.1. In the left and middle images, the indirect and direct lighting, respectively, are separated out. On the right, the sum of both components is shown. Global illumination algorithms account for both the direct and the indirect lighting.

1984; Immel et al., 1986; Kajiya, 1986; Malley, 1988). In this chapter, we discuss two algorithms as examples: particle tracing and path tracing. The first is useful for walkthrough applications such as maze games, and as a component of batch rendering. The second is useful for realistic batch rendering. Then we discuss separating out "direct" lighting where light takes exactly once bounce between luminaire and camera.

23.1 Particle Tracing for Lambertian Scenes

Recall the transport equation from Section 18.2:

$$L_s(\mathbf{k}_o) = \int_{\text{all } \mathbf{k}_i} \rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k}_i) \cos\theta_i d\sigma_i.$$

The geometry for this equation is shown in Figure 23.2. When the illuminated point is Lambertian, this equation reduces to:

$$L_s = \frac{R}{\pi} \int_{\text{all } \mathbf{k}_i} L_f(\mathbf{k}_i) \cos\theta_i d\sigma_i,$$

where R is the diffuse reflectance. One way to approximate the solution to this equation is to use finite element methods. First, we break the scene into N surfaces each with unknown surface radiance L_i, reflectance R_i, and emitted radiance E_i. This results in the set of N simultaneous linear equations

$$L_i = E_i + \frac{R_i}{\pi} \sum_{j=1}^{N} k_{ij} L_j,$$

where k_{ij} is a constant related to the original integral representation. We then solve this set of linear equations, and we can render N constant-colored polygons. This finite element approach is often called *radiosity*.

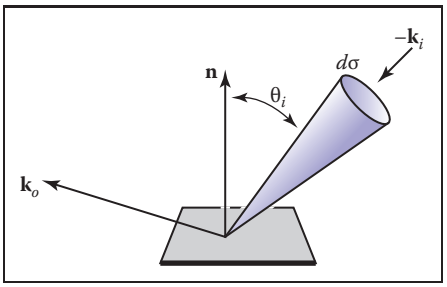


Figure 23.2. The geometry for the transport equation in its directional form.

An alternative method to radiosity is to use a statistical simulation approach by randomly following light "particles" from the luminaire through the environment. This is a type of *particle tracing*. There are many algorithms that use some form of particle tracing; we will discuss a form of particle tracing that deposits light in the textures on triangles. First, we review some basic radiometric relations. The radiance L of a Lambertian surface with area A is directly proportional to the incident power per unit area:

$$L = \frac{\Phi}{\pi A}, \tag{23.1}$$

where Φ is the outgoing power from the surface. Note that in this discussion, all radiometric quantities are either spectral or RGB, depending on the implementation. If the surface has emitted power Φ_e, incident power Φ_i, and reflectance R, then this equation becomes

$$L = \frac{\Phi_e + R\Phi_i}{\pi A}.$$

If we are given a model with Φ_e and R specified for each triangle, we can proceed luminaire by luminaire, firing power in the form of particles from each luminaire. We associate a texture map with each triangle to store accumulated radiance, with all texels initialized to

$$L = \frac{\Phi_e}{\pi A}.$$

If a given triangle has area A and n_t texels, and it is hit by a particle carrying power ϕ, then the radiance of that texel is incremented by

$$\Delta L = \frac{n_t \phi}{\pi A}.$$

Once a particle hits a surface, we increment the radiance of the texel it hits, probabilistically decide whether to reflect the particle, and if we reflect it we choose a direction and adjust its power.

Note that we want the particle to terminate at some point. For each surface we can assign a reflection probability p to each surface interaction. A natural choice would be to let $p = R$ as it is with light in nature. The particle would then scatter around the environment, not losing or gaining any energy until it is absorbed. This approach works well when the particles carry a single wavelength (Walter, Hubbard, Shirley, & Greenberg, 1997). However, when a spectrum or RGB triple is carried by the ray as is often implemented (Jensen, 2001), there is no single R and some compromise for the value of p should be chosen. The power ϕ' for reflected particles should be adjusted to account for the possible extinction of the particles:

$$\phi' = \frac{R\phi}{p}.$$

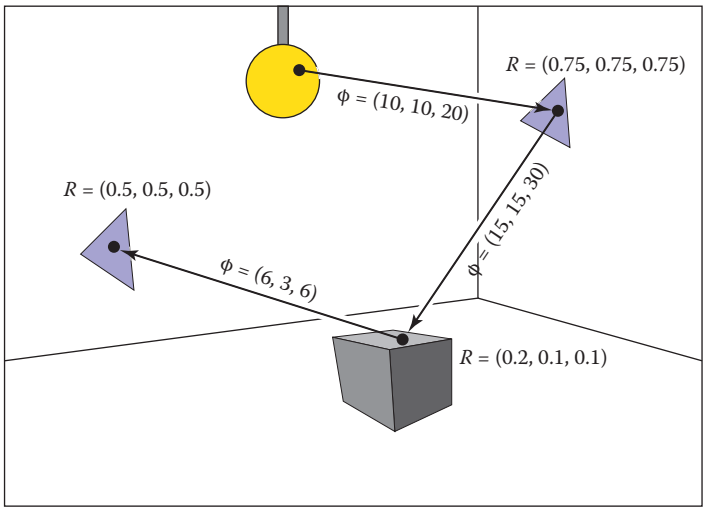


Figure 23.3. The path of a particle that survives with probability 0.5 and is absorbed at the last intersection. The RGB power is shown for each path segment.

Note that p can be set to any positive constant less than one, and that this constant can be different for each interaction. When $p > R$ for a given wavelength, the particle will gain power at that wavelength, and when $p < R$ it will lose power at that wavelength. The case where it gains power will not interfere with convergence because the particle will stop scattering and be terminated at some point as long as $p < 1$. For the remainder of this discussion we set $p = 0.5$. The path of a single particle in such a system is shown in Figure 23.3.

A key part to this algorithm is that we scatter the light with an appropriate distribution for Lambertian surfaces. As discussed in Section 14.4.1, we can find a vector with a cosine (Lambertian) distribution by transforming two canonical random numbers (ξ_1, ξ_2) as follows:

$$\mathbf{a} = \left(\cos(2\pi\xi_1)\sqrt{\xi_2}, \sin(2\pi\xi_1)\sqrt{\xi_2}, \sqrt{1-\xi_2}\right). \tag{23.2}$$

Note that this assumes the normal vector is parallel to the z-axis. For a triangle, we must establish an orthonormal basis with $\mathbf{w}$ parallel to the normal vector. We can accomplish this as follows:

$$\begin{aligned}
\mathbf{w} &= \frac{\mathbf{n}}{\|\mathbf{n}\|}, \\
\mathbf{u} &= \frac{\mathbf{p}_1 - \mathbf{p}_0}{\|\mathbf{p}_1 - \mathbf{p}_0\|}, \\
\mathbf{v} &= \mathbf{w} \times \mathbf{u},
\end{aligned}$$

where $\mathbf{p}_i$ are the vertices of the triangle. Then, by definition, our vector in the appropriate coordinates is

$$\mathbf{a} = \cos(2\pi\xi_1)\sqrt{\xi_2}\mathbf{u} + \sin(2\pi\xi_1)\sqrt{\xi_2}\mathbf{v} + \sqrt{1-\xi_2}\mathbf{w}. \tag{23.3}$$

In pseudocode our algorithm for $p = 0.5$ and one luminaire is:

```
for (Each of n particles) do
  RGB phi = Φ/n
  compute uniform random point a on luminaire
  compute random direction b with cosine density
  done = false
  while not done do
    if (ray a + tb hits at some point c ) then
      add n_t Rφ/(πA) to appropriate texel
      if (ξ_1 > 0.5) then
        φ = 2Rφ
        a = c
        b = random direction with cosine density
    else
      done = true
```

Here ξ_i are canonical random numbers. Once this code has run, the texture maps store the radiance of each triangle and can be rendered directly for any viewpoint with no additional computation.

23.2 Path Tracing

While particle tracing is well suited to precomputation of the radiances of diffuse scenes, it is problematic for creating images of scenes with general BRDFs or scenes that contain many objects. The most straightforward way to create images of such scenes is to use *path tracing* (Kajiya, 1986). This is a probabilistic method that sends rays from the eye and traces them back to the light. Often path tracing is used only to compute the indirect lighting. Here we will present it in a way that captures all lighting, which can be inefficient. This is sometimes called *brute force* path tracing. In Section 23.3, more efficient techniques for direct lighting can be added.

In path tracing, we start with the full transport equation:

$$L_s(\mathbf{k}_o) = L_e(\mathbf{k}_o) + \int_{\text{all } \mathbf{k}_i} \rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k}_i) \cos\theta_i d\sigma_i.$$

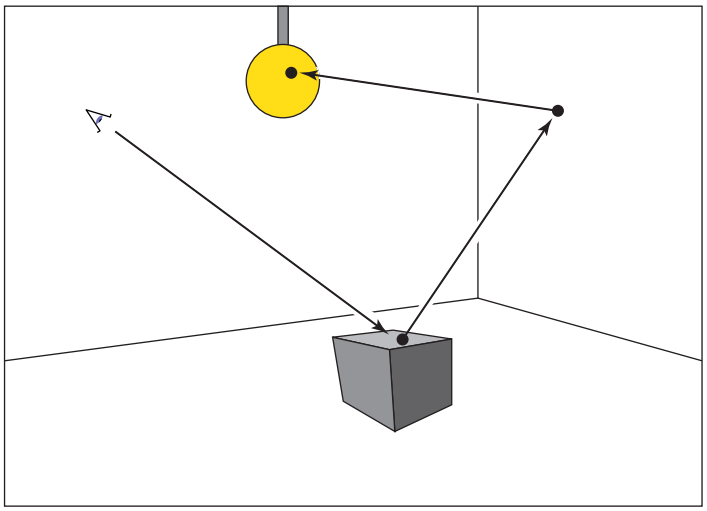


Figure 23.4. In path tracing, a ray is followed through a pixel from the eye and scattered through the scene until it hits a luminaire.

We use Monte Carlo integration to approximate the solution to this equation for each viewing ray. Recall from Section 14.3, that we can use random samples to approximate an integral:

$$\int_{x \in S} g(x) d\mu \approx \frac{1}{N} \sum_{i=1}^{N} \frac{g(x_i)}{p(x_i)},$$

where the x_i are random points with probability density function p. If we apply this directly to the transport equation with $N = 1$ we get

$$L_s(\mathbf{k}_o) \approx L_e(\mathbf{k}_o) + \frac{\rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k}_i) \cos\theta_i d\sigma_i}{p(\mathbf{k}_i)}.$$

So, if we have a way to select random directions $\mathbf{k}_i$ with a known density p, we can get an estimate. The catch is that $L_f(\mathbf{k}_i)$ is itself an unknown. Fortunately, we can apply recursion and use a statistical estimate for $L_f(\mathbf{k}_i)$ by sending a ray in that direction to find the surface seen in that direction. We end when we hit a luminaire and L_e is nonzero (Figure 23.4). This method assumes lights have zero reflectance, or we would continue to recurse.

In the case of a Lambertian BRDF ($\rho = R/\pi$), we can use a cosine density function:

$$p(\mathbf{k}_i) = \frac{\cos\theta_i}{\pi}.$$

A direction with this density can be chosen according to Equation (23.3). This allows some cancellation of cosine terms in our estimate:

$$L_s(\mathbf{k}_o) \approx L_e(\mathbf{k}_o) + RL_f(\mathbf{k}_i).$$

In pseudocode, such a path tracer for Lambertian surfaces would operate just like the ray tracers described in Chapter 4, but the *raycolor* function would be modified:

```
RGB raycolor(ray a + tb, int depth)
if (ray hits at some point c ) then
   RGB c = L_e(−b)
   if (depth < maxdepth) then
      compute random direction d
      return c + R raycolor(c + sd, depth+1)
else
   return background color
```

This will result in a very noisy image unless either large luminaires or very large numbers of samples are used. Note the color of the luminaires must be well above one (sometimes thousands or tens of thousands) to make the surfaces have final colors near one, because only those rays that hit a luminaire by chance will make a contribution, and most rays will contribute only a color near zero. To generate the random direction $\mathbf{d}$, we use the same technique as we do in particle tracing (see Equation (23.2)).

In the general case, we might want to use spectral colors or use a more general BRDF. In practice, we should have the material class contain member functions to compute a random direction as well as compute the p associated with that direction. This way materials can be added transparently to an implementation.

23.3 Accurate Direct Lighting

This section presents a more physically based method of direct lighting than Chapter 10. These methods will be useful in making global illumination algorithms more efficient. The key idea is to send shadow rays to the luminaires as described in Chapter 4, but to do so with careful bookkeeping based on the transport equation from the previous chapter. The global illumination algorithms can be adjusted to make sure they compute the direct component exactly once. For example, in particle tracing, particles coming directly from the luminaire would not be logged, so the particles would only encode indirect lighting. This makes

nice looking shadows much more efficiently than computing direct lighting in the context of global illumination.

23.3.1 Mathematical Framework

To calculate the direct light from one *luminaire* (light emitting object) onto a non-emitting surface, we solve a form of the transport equation from Section 18.2:

$$L_s(\mathbf{x}, \mathbf{k}_o) = \int_{\text{all } \mathbf{x}'} \frac{\rho(\mathbf{k}_i, \mathbf{k}_o) L_e(\mathbf{x}', -\mathbf{k}_i) v(\mathbf{x}, \mathbf{x}') \cos\theta_i \cos\theta'}{\|\mathbf{x} - \mathbf{x}'\|^2} dA' . \quad (23.4)$$

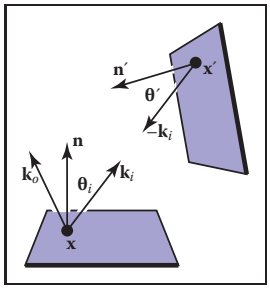


Figure 23.5. The direct lighting terms for Equation (23.4).

Recall that L_e is the emitted radiance of the source, v is a visibility function that is equal to 1 if $\mathbf{x}$ "sees" $\mathbf{x}'$ and zero otherwise, and the other variables are as illustrated in Figure 23.5.

If we are to sample Equation (23.4) using Monte Carlo integration, we need to pick a random point $\mathbf{x}'$ on the surface of the luminaire with density function p (so $\mathbf{x}' \sim p$). Just plugging into Equation (14.5) with one sample yields

$$L_s(\mathbf{x}, \mathbf{k}_o) \approx \frac{\rho(\mathbf{k}_i, \mathbf{k}_o) L_e(\mathbf{x}', -\mathbf{k}_i) v(\mathbf{x}, \mathbf{x}') \cos\theta_i \cos\theta'}{p(\mathbf{x}') \|\mathbf{x} - \mathbf{x}'\|^2} . \quad (23.5)$$

If we pick a uniform random point on the luminaire, then $p = 1/A$, where A is the area of the luminaire. This gives

$$L_s(\mathbf{x}, \mathbf{k}_o) \approx \frac{\rho(\mathbf{k}_i, \mathbf{k}_o) L_e(\mathbf{x}', -\mathbf{k}_i) v(\mathbf{x}, \mathbf{x}') A \cos\theta_i \cos\theta'}{\|\mathbf{x} - \mathbf{x}'\|^2} . \quad (23.6)$$

We can use Equation (23.6) to sample planar (e.g., rectangular) luminaires in a straightforward fashion. We simply pick a random point on each luminaire.

The code for one luminaire is:

```
color directLight( x, k_o, n)
pick random point x' with normal vector n' on light
d = x' − x
k_i = d/||d||
if (ray x + td has no hits for t < 1 − ε) then
    return ρ(k_i, k_o) L_e(x', −k_i)(n · d)(−n' · d)/||d||^4
else
    return 0
```

The above code needs some extra tests such as clamping the cosines to zero if they are negative. Note that the term $\|\mathbf{d}\|^4$ comes from the distance squared term

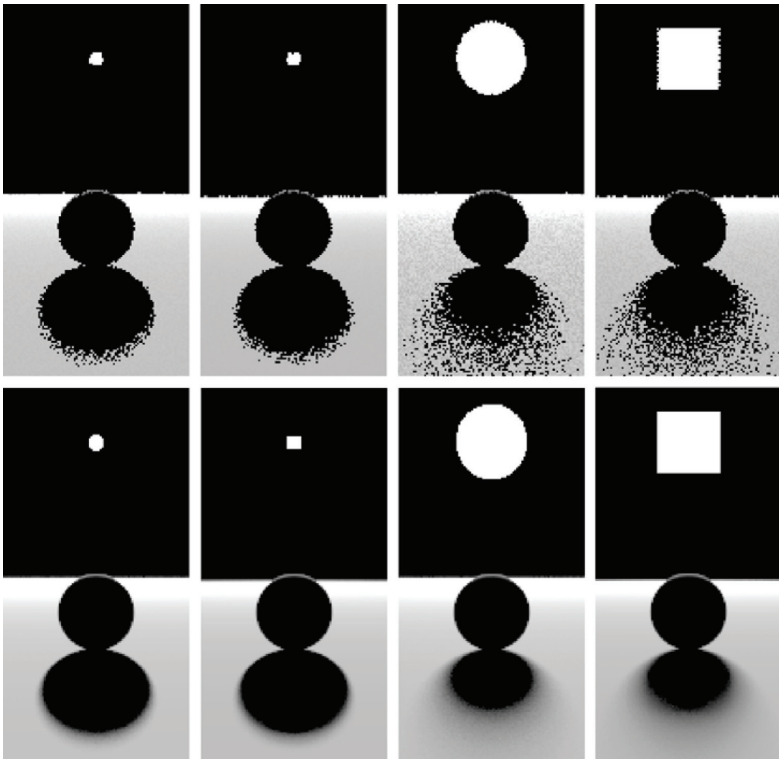


Figure 23.6. Various soft shadows on a backlit sphere with a square and an area light source. Top: 1 sample. Bottom: 100 samples. Note that the shape of the light source is less important than its size in determining shadow appearance.

and the two cosines, e.g., $\mathbf{n} \cdot \mathbf{d} = \|\mathbf{d}\| \cos\theta$ because $\mathbf{d}$ is not necessarily a unit vector.

Several examples of soft shadows are shown in Figure 23.6.

23.3.2 Sampling a Spherical Luminaire

Though a sphere with center $\mathbf{c}$ and radius R can be sampled using Equation (23.6), this sampling will yield a very noisy image because many samples will be on the back of the sphere, and the $\cos\theta'$ term varies so much. Instead, we can use a more complex $p(\mathbf{x}')$ to reduce noise.

The first nonuniform density we might try is $p(\mathbf{x}') \propto \cos\theta'$. This turns out to be just as complicated as sampling with $p(\mathbf{x}') \propto \cos\theta'/\|\mathbf{x}' - \mathbf{x}\|^2$, so we instead discuss that here. We observe that sampling on the luminaire this way is the

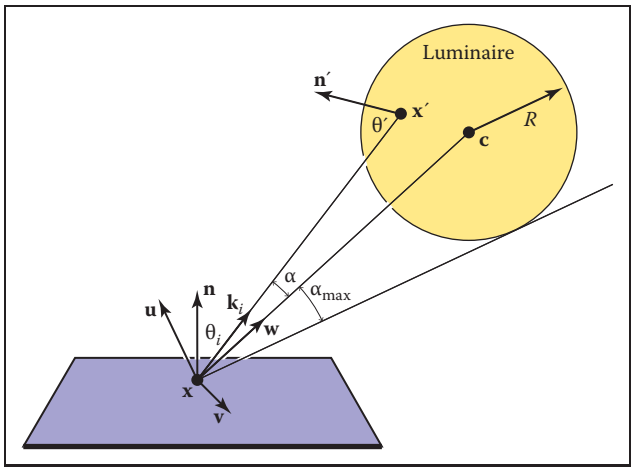


Figure 23.7. Geometry for direct lighting at point **x** from a spherical luminaire.

same as using a constant density function $q(\mathbf{k}_i) = \text{const}$ defined in the space of directions subtended by the luminaire as seen from $\mathbf{x}$. We now use a coordinate system defined with $\mathbf{x}$ at the origin, and a right-handed orthonormal basis with $\mathbf{w} = (\mathbf{c} - \mathbf{x})/\|\mathbf{c} - \mathbf{x}\|$, and $\mathbf{v} = (\mathbf{w} \times \mathbf{n})/\|(\mathbf{w} \times \mathbf{n})\|$ (see Figure 23.7). We also define (α, ϕ) to be the azimuthal and polar angles with respect to the *uvw* coordinate system.

The maximum α that includes the spherical luminaire is given by

$$\alpha_{\max} = \arcsin\left(\frac{R}{\|\mathbf{x} - \mathbf{c}\|}\right) = \arccos\sqrt{1 - \left(\frac{R}{\|\mathbf{x} - \mathbf{c}\|}\right)^2}.$$

Thus, a uniform density (with respect to solid angle) within the cone of directions subtended by the sphere is just the reciprocal of the solid angle $2\pi(1 - \cos\alpha_{\max})$ subtended by the sphere:

$$q(\mathbf{k}_i) = \frac{1}{2\pi\left(1 - \sqrt{1 - \left(\frac{R}{\|\mathbf{x}-\mathbf{c}\|}\right)^2}\right)}.$$

And we get

$$\begin{bmatrix} \cos\alpha \\ \phi \end{bmatrix} = \begin{bmatrix} 1 - \xi_1 + \xi_1\sqrt{1 - \left(\frac{R}{\|\mathbf{x}-\mathbf{c}\|}\right)^2} \\ 2\pi\xi_2 \end{bmatrix}.$$

This gives us the direction $\mathbf{k}_i$. To find the actual point, we need to find the first point on the sphere in that direction. The ray in that direction is just $(\mathbf{x} + t\mathbf{k}_i)$,

Figure 23.8. A sphere with $L_e = 1$ touching a sphere of reflectance 1. Where the two spheres touch, the reflective sphere should have $L(\mathbf{x}') = 1$. Left: 1 sample. Middle: 100 samples. Right: 100 samples, close-up.

where $\mathbf{k}_i$ is given by

$$\mathbf{k}_i = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \begin{bmatrix} \cos\phi \sin\alpha \\ \sin\phi \sin\alpha \\ \cos\alpha \end{bmatrix}.$$

We must also calculate $p(\mathbf{x}')$, the probability density function with respect to the area measure (recall that the density function q is defined in solid angle space). Since we know that q is a valid probability density function using the ω measure, and we know that $d\Omega = dA(\mathbf{x}') \cos\theta' / \|\mathbf{x}' - \mathbf{x}\|^2$, we can relate any probability density function $q(\mathbf{k}_i)$ with its associated probability density function $p(\mathbf{x}')$:

$$q(\mathbf{k}_i) = \frac{p(\mathbf{x}') \cos\theta'}{\|\mathbf{x}' - \mathbf{x}\|^2}. \tag{23.7}$$

So we can solve for $p(\mathbf{x}')$:

$$p(\mathbf{x}') = \frac{\cos\theta'}{2\pi\|\mathbf{x}' - \mathbf{x}\|^2 \left(1 - \sqrt{1 - \left(\frac{R}{\|\mathbf{x}-\mathbf{c}\|}\right)^2}\right)}.$$

A good debugging case for this is shown in Figure 23.8.

23.3.3 Nondiffuse Luminaries

There is no reason the luminance of the luminaire cannot vary with both direction and position. For example, it can vary with position if the luminaire is a television. It can vary with direction for car headlights and other directional sources. Little in our analysis need change from the previous sections, except that $L_e(\mathbf{x}')$ must change to $L_e(\mathbf{x}', -\mathbf{k}_i)$. The simplest way to vary the intensity with direction is to use a Phong-like pattern with respect to the normal vector $\mathbf{n}'$. To avoid using an exponent in the term for the total light output, we can use the form

$$L_e(\mathbf{x}', -\mathbf{k}_i) = \frac{(n+1)E(\mathbf{x}')}{2\pi} cos^{(n-1)}\theta',$$

where $E(\mathbf{x}')$ is the *radiant exitance* (power per unit area) at point $\mathbf{x}'$, and n is the Phong exponent. You get a diffuse light for $n = 1$. If the light is nonuniform across its area, e.g., as a television set is, then E will not be a constant.

Frequently Asked Questions

• My pixel values are no longer in some sensible zero-to-one range. What should I display?

You should use one of the *tone reproduction* techniques described in Chapter 21.

• What global illumination techniques are used in practice?

For batch rendering of complex scenes, path tracing with one level of reflection is often used. Path tracing is often augmented with a particle tracing preprocess as described in Jensen's book in the chapter notes. For walkthrough games, some form of world-space preprocess is often used, such as the particle tracing described in this chapter. For scenes with very complicated specular transport, an elegant but involved method, Metropolis Light Transport (Veach & Guibas, 1997) may be the best choice.

• How does the ambient component relate to global illumination?

For diffuse scenes, the radiance of a surface is proportional to the product of the irradiance at the surface and the reflectance of the surface. The ambient component is just an approximation to the irradiance scaled by the inverse of π. So although it is a crude approximation, there can be some methodology to guessing it (M. F. Cohen, Chen, Wallace, & Greenberg, 1988), and it is probably more accurate than doing nothing, i.e., using zero for the ambient term. Because the indirect irradiance can vary widely within a scene, using a different constant for each surface can be used for better results rather than using a global ambient term.

• Why do most algorithms compute direct lighting using traditional ray tracing?

Although global illumination algorithms automatically compute direct lighting, and it is, in fact, slightly more complicated to make them compute only indirect lighting, it is usually faster to compute direct lighting separately. There are three reasons for this. First, indirect lighting tends to be smooth compared to

Figure 23.9. A comparison between a rendering and a photo. *Image courtesy Sumant Pattanaik and the Cornell Program of Computer Graphics.*

direct lighting (see Figure 23.1) so coarser representations can be used, e.g., low-resolution texture maps for particle tracing. The second reason is that light sources tend to be small, and it is rare to hit them by chance in a "from the eye" method such as path tracing, while direct shadow rays are efficient. The third reason is that direct lighting allows stratified sampling, so it converges rapidly compared to unstratified sampling. The issue of stratification is the reason that shadow rays are used in Metropolis Light Transport despite the stability of its default technique for dealing with direct lighting as just one type of path to handle.

- How artificial is it to assume ideal diffuse and specular behavior?

For environments that have only matte and mirrored surfaces, the Lambertian/specular assumption works well. A comparison between a rendering using that assumption and a photograph is shown in Figure 23.9.

- How many shadow rays are needed per pixel?

Typically between 16 and 400. Using narrow penumbra, a large ambient term (or a large indirect component), and a masking texture (Ferwerda, Shirley, Pattanaik, & Greenberg, 1997) can reduce the number needed.

- How do I sample something like a filament with a metal reflector where much of the light is reflected from the filament?

Typically, the whole light is replaced by a simple source that approximates its aggregate behavior. For viewing rays, the complicated source is used. So a car

headlight would look complex to the viewer, but the lighting code might see simple disk-shaped lights.

- Isn't something like the sky a luminaire?

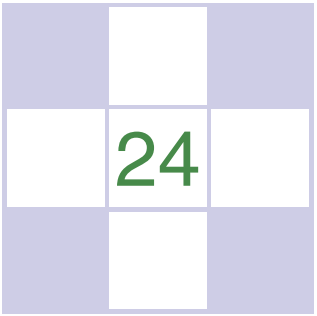
Yes, and you can treat it as one. However, such large light sources may not be helped by direct lighting; the brute-force techniques are likely to work better.

Notes

Global illumination has its roots in the fields of heat transfer and illumination engineering as documented in *Radiosity: A Programmer's Perspective* (Ashdown, 1994). Other good books related to global illumination include *Radiosity and Global Illumination* (M. F. Cohen & Wallace, 1993), *Radiosity and Realistic Image Synthesis* (Sillion & Puech, 1994), *Principles of Digital Image Synthesis* (Glassner, 1995), *Realistic Image Synthesis Using Photon Mapping* (Jensen, 2001), *Advanced Global Illumination* (Dutré, Bala, & Bekaert, 2002), and *Physically Based Rendering* (Pharr & Humphreys, 2004). The probabilistic methods discussed in this chapter are from *Monte Carlo Techniques for Direct Lighting Calculations* (Shirley, Wang, & Zimmerman, 1996).

Exercises

1. For a closed environment, where every surface is a diffuse reflector and emittor with reflectance R and emitted radiance E, what is the total radiance at each point? *Hint: for* $R = 0.5$ *and* $E = 0.25$ *the answer is* 0.5. This is an excellent debugging case.
2. Using the definitions from Chapter 18, verify Equation (23.1).
3. If we want to render a typically sized room with textures at centimeter-square resolution, approximately how many particles should we send to get an average of about 1000 hits per texel?
4. Develop a method to take random samples with uniform density from a disk.
5. Develop a method to take random samples with uniform density from a triangle.
6. Develop a method to take uniform random samples on a "sky dome" (the inside of a hemisphere).



24 Reflection Models

As we discussed in Chapter 18, the reflective properties of a surface can be summarized using the BRDF (Nicodemus, Richmond, Hsia, Ginsberg, & Limperis, 1977; Cook & Torrance, 1982). In this chapter, we discuss some of the most visually important aspects of material properties and a few fairly simple models that are useful in capturing these properties. There are many BRDF models in use in graphics, and the models presented here are meant to give just an idea of nondiffuse BRDFs.

24.1 Real-World Materials

Many real materials have a visible structure at normal viewing distances. For example, most carpets have easily visible pile that contributes to appearance. For our purposes, such structure is not part of the material property but is, instead, part of the geometric model. Structure whose details are invisible at normal viewing distances, but which do determine macroscopic material appearance, are part of the material property. For example, the fibers in paper have a complex appearance under magnification, but they are blurred together into an homogeneous appearance when viewed at arm's length. This distinction between microstructure that is folded into BRDF is somewhat arbitrary and depends on what one defines as "normal" viewing distance and visual acuity, but the distinction has proven quite useful in practice.

In this section, we define some categories of materials. Later in the chapter, we present reflection models that target each type of material. In the notes at the

end of the chapter, some models that account for more exotic materials are also discussed.

24.1.1 Smooth Dielectrics and Metals

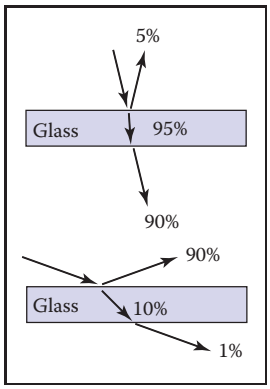


Figure 24.1. The amount of light reflected and transmitted by glass varies with the angle.

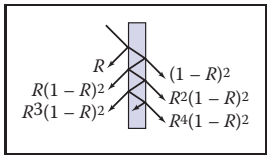


Figure 24.2. Light is repeatedly reflected and refracted by glass, with the fractions of energy shown.

Dielectrics are clear materials that refract light; their basic properties were summarized in Chapter 4. Metals reflect and refract light much like dielectrics, but they absorb light very, very quickly. Thus, only very thin metal sheets are transparent at all, e.g., the thin gold plating on some glass objects. For a smooth material, there are only two important properties:

1. How much light is reflected at each incident angle and wavelength.
2. What fraction of light is absorbed as it travels through the material for a given distance and wavelength.

The amount of light transmitted is whatever is not reflected (a result of energy conservation). For a metal, in practice, we can assume all the light is immediately absorbed. For a dielectric, the fraction is determined by the constant used in Beer's Law as discussed in Chapter 13.

The amount of light reflected is determined by the *Fresnel equations* as discussed in Chapter 4. These equations are straightforward, but cumbersome. The main effect of the Fresnel equations is to increase the reflectance as the incident angle increases, particularly near grazing angles. This effect works for transmitted light as well. These ideas are shown diagrammatically in Figure 24.1. Note that the light is repeatedly reflected and refracted as shown in Figure 24.2. Usually only one or two of the reflected images is easily visible.

24.1.2 Rough Surfaces

If a metal or dielectric is roughened to a small degree, but not so small that diffraction occurs, then we can think of it as a surface with *microfacets* (Cook & Torrance, 1982). Such surfaces behave specularly at a closer distance, but viewed at a further distance seem to spread the light out in a distribution. For a metal, an example of this rough surface might be brushed steel, or the "cloudy" side of most aluminum foil.

For dielectrics, such as a sheet of glass, scratches or other irregular surface features make the glass blur the reflected and transmitted images that we can normally see clearly. If the surface is heavily scratched, we call it *translucent* rather than transparent. This is a somewhat arbitrary distinction, but it is usually clear whether we would consider a glass translucent or transparent.

24.1.3 Diffuse Materials

A material is *diffuse* if it is matte, i.e., not shiny. Many surfaces we see are diffuse, such as most stones, paper, and unfinished wood. To a first approximation, diffuse surfaces can be approximated with a Lambertian (constant) BRDF. Real diffuse materials usually become somewhat specular for grazing angles. This is a subtle effect, but can be important for realism.

24.1.4 Translucent Materials

Many thin objects, such as leaves and paper, both transmit and reflect light diffusely. For all practical purposes no clear image is transmitted by these objects. These surfaces can add a hue shift to the transmitted light. For example, red paper is red because it filters out non-red light for light that penetrates a short distance into the paper, and then scatters back out. The paper also transmits light with a red hue because the same mechanisms apply, but the transmitted light makes it all the way through the paper. One implication of this property is that the transmitted coefficient should be the same in both directions.

24.1.5 Layered Materials

Many surfaces are composed of "layers" or are dielectrics with embedded particles that give the surface a diffuse property (Phong, 1975). The surface of such materials reflects specularly as shown in Figure 24.3, and thus obeys the Fresnel equations. The light that is transmitted is either absorbed or scattered back up to the dielectric surface where it may or may not be transmitted. That light that is transmitted, scattered, and then retransmitted in the opposite direction forms a diffuse "reflection" component.

Note that the diffuse component also is attenuated with the degree of the angle, because the Fresnel equations cause reflection back into the surface as the angle increases as shown in Figure 24.4. Thus, instead of a constant diffuse BRDF, one that vanishes near the grazing angle is more appropriate.

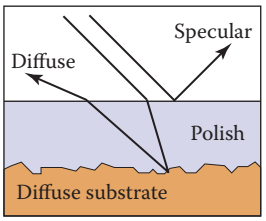


Figure 24.3. Light hitting a layered surface can be reflected specularly, or it can be transmitted and then scatter diffusely off the substrate.

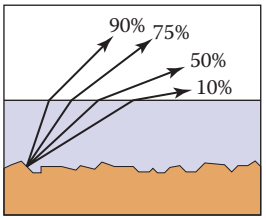


Figure 24.4. The light scattered by the substrate is less and less likely to make it out of the surface as the angle relative to the surface normal increases.

24.2 Implementing Reflection Models

A BRDF model, as described in Section 18.1.6, will produce a rendering which is more physically based than the rendering we get from point light sources and Phong-like models. Unfortunately, real BRDFs are typically quite complicated and cannot be deduced from first principles. Instead, they must either be measured

and directly approximated from raw data, or they must be crudely approximated in an empirical fashion. The latter empirical strategy is what is usually done, and the development of such approximate models is still an area of research. This section discusses several desirable properties of such empirical models.

First, physical constraints imply two properties of a BRDF model. The first constraint is energy conservation:

$$\text{for all } \mathbf{k}_i, R(\mathbf{k}_i) = \int_{\text{all } \mathbf{k}_o} \rho(\mathbf{k}_i, \mathbf{k}_o) \cos\theta_o \, d\sigma_o \leq 1.$$

If you send a beam of light at a surface from any direction $\mathbf{k}_i$, then the total amount of light reflected over all directions will be at most the incident amount. The second physical property we expect all BRDFs to have is reciprocity:

$$\text{for all } \mathbf{k}_i, \mathbf{k}_o, \rho(\mathbf{k}_i, \mathbf{k}_o) = \rho(\mathbf{k}_o, \mathbf{k}_i).$$

Second, we want a clear separation between diffuse and specular components. The reason for this is that, although there is a mathematically clean delta function formulation for ideal specular components, delta functions must be implemented as special cases in practice. Such special cases are only practical if the BRDF model clearly indicates what is specular and what is diffuse.

Third, we would like intuitive parameters. For example, one reason the Phong model has enjoyed such longevity is that its diffuse constant and exponent are both clearly related to the intuitive properties of the surface, namely surface color and highlight size.

Finally, we would like the BRDF function to be amenable to Monte Carlo sampling. Recall from Chapter 14 that an integral can be sampled by N random points $x_i \sim p$ where p is defined with the same measure as the integral:

$$\int f(x) d\mu \approx \frac{1}{N} \sum_{j=1}^{N} \frac{f(x_j)}{p(x_j)}.$$

Recall from Section 18.2 that the surface radiance in direction $\mathbf{k}_o$ is given by a transport equation:

$$L_s(\mathbf{k}_o) = \int_{\text{all } \mathbf{k}_i} \rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k}_i) \cos\theta_i d\sigma_i.$$

If we sample directions with pdf $p(\mathbf{k}_i)$ as discussed in Chapter 23, then we can approximate the surface radiance with samples:

$$L_s(\mathbf{k}_o) \approx \frac{1}{N} \sum_{j=1}^{N} \frac{\rho(\mathbf{k}_j, \mathbf{k}_o) L_f(\mathbf{k}_j) \cos\theta_j}{p(\mathbf{k}_j)}.$$

This approximation will converge for any p that is nonzero where the integrand is nonzero. However, it will only converge well if the integrand is not very large relative to p. Ideally, $p(\mathbf{k})$ should be approximately shaped like the integrand $\rho(\mathbf{k}_j, \mathbf{k}_o)L_f(\mathbf{k}_j)\cos\theta_j$. In practice, L_f is complicated, and the best we can accomplish is to have $p(\mathbf{k})$ shaped somewhat like $\rho(\mathbf{k}, \mathbf{k}_o)L_f(\mathbf{k})\cos\theta$.

For example, if the BRDF is Lambertian, then it is constant and the "ideal" $p(\mathbf{k})$ is proportional to $\cos\theta$. Because the integral of p must be one, we can deduce the leading constant:

$$\int_{\text{all } \mathbf{k} \text{ with } \theta < \pi/2} C\cos\theta d\sigma = 1.$$

This implies that $C = 1/\pi$, so we have

$$p(\mathbf{k}) = \frac{1}{\pi}\cos\theta.$$

An acceptably efficient implementation will result as long as p doesn't get too small when the integrand is nonzero. Thus, the constant pdf will also suffice:

$$p(\mathbf{k}) = \frac{1}{2\pi}.$$

This emphasizes that many pdfs may be acceptable for a given BRDF model.

24.3 Specular Reflection Models

For a metal, we typically specify the reflectance at normal incidence $R_0(\lambda)$. The reflectance should vary according to the Fresnel equations, and a good approximation is given by (Schlick, 1994a)

$$R(\theta, \lambda) = R_0(\lambda) + (1 - R_0(\lambda))(1 - \cos\theta)^5.$$

This approximation allows us to just set the normal reflectance of the metal either from data or by eye.

For a dielectric, the same formula works for reflectance. However, we can set $R_0(\lambda)$ in terms of the refractive index $n(\lambda)$:

$$R_0(\lambda) = \left(\frac{n(\lambda) - 1}{n(\lambda) + 1}\right)^2.$$

Typically, n does not vary with wavelength, but for applications where dispersion is important, n can vary. The refractive indices that are often useful include water ($n = 1.33$), glass ($n = 1.4$ to $n = 1.7$), and diamond ($n = 2.4$).

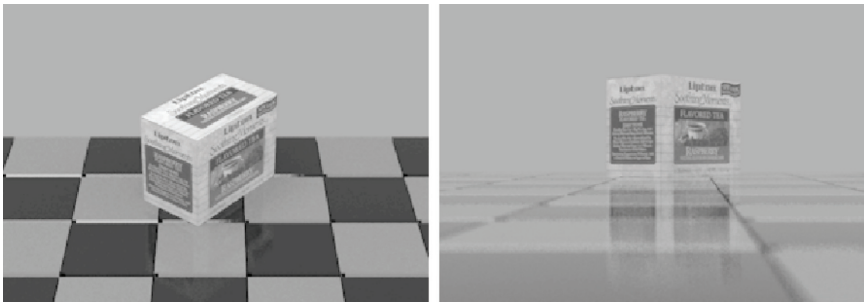


Figure 24.5. Renderings of polished tiles using coupled model. These images were produced using a Monte Carlo path tracer. The sampling distribution for the diffuse term is $\cos\theta/\pi$.

24.4 Smooth-Layered Model

Reflection in matte/specular materials, such as plastics or polished woods, is governed by Fresnel equations at the surface and by scattering within the subsurface. An example of this reflection can be seen in the tiles in the renderings in Figure 24.5. Note that the blurring in the specular reflection is mostly vertical due to the compression of apparent bump spacing in the view direction. This effect causes the vertically streaked reflections seen on lakes on windy days; it can either be modeled using explicit microgeometry and a simple smooth-surface reflection model or by a more general model that accounts for this asymmetry.

We could use the traditional Lambertian-specular model for the tiles, which linearly mixes specular and Lambertian terms. In standard radiometric terms, this can be expressed as

$$\rho(\theta, \phi, \theta', \phi' \lambda) = \frac{R_d(\lambda)}{\pi} + R_s \rho_s(\theta, \phi, \theta', \phi'),$$

where $R_d(\lambda)$ is the hemispherical reflectance of the matte term, R_s is the specular reflectance, and ρ_s is the normalized specular BRDF (a weighted Dirac delta function on the sphere). This equation is a simplified version of the BRDF where R_s is independent of wavelength. The independence of wavelength causes a highlight that is the color of the luminaire, so a polished rather than a metal appearance will be achieved. Ward (G. J. Ward, 1992) suggests to set $R_d(\lambda) + R_s \leq 1$ in order to conserve energy. However, such models with constant R_s fail to show the increase in specularity for steep viewing angles. This is the key point: in the real world the relative proportions of matte and specular appearance change with the viewing angle.

One way to simulate the change in the matte appearance is to explicitly dampen $R_d(\lambda)$ as R_s increases (Shirley, 1991):

$$\rho(\theta, \phi, \theta', \phi', \lambda) = R_f(\theta)\rho_s(\theta, \phi, \theta', \phi') + \frac{R_d(\lambda)(1 - R_f(\theta))}{\pi},$$

where $R_f(\theta)$ is the Fresnel reflectance for a polish-air interface. The problem with this equation is that it is not reciprocal, as can been seen by exchanging θ and θ'; this changes the value of the matte damping factor because of the multiplication by $(1 - R_f(\theta))$. The specular term, a scaled Dirac delta function, is reciprocal, but this does not make up for the non-reciprocity of the matte term. Although this BRDF works well, its lack of reciprocity can cause some rendering methods to have ill-defined solutions.

We now present a model that produces the matte/specular tradeoff while remaining reciprocal and energy conserving. Because the key feature of the new model is that it couples the matte and specular scaling coefficients, it is called a *coupled* model (Shirley, Smits, Hu, & Lafortune, 1997).

Surfaces which have a glossy appearance are often a clear dielectric, such as polyurethane or oil, with some subsurface structure. The specular (mirror-like) component of the reflection is caused by the smooth dielectric surface and is independent of the structure below this surface. The magnitude of this specular term is governed by the Fresnel equations.

The light that is not reflected specularly at the surface is transmitted through the surface. There, either it is absorbed by the subsurface, or it is reflected from a pigment or a subsurface and transmitted back through the surface of the polish. This transmitted light forms the matte component of reflection. Since the matte component can only consist of the light that is transmitted, it will naturally decrease in total magnitude for increasing angle.

To avoid choosing between physically plausible models and models with good qualitative behavior over a range of incident angles, note that the Fresnel equations that account for the specular term, $R_f(\theta)$, are derived directly from the physics of the dielectric-air interface. Therefore, the problem must lie in the matte term. We could use a full-blown simulation of subsurface scattering as implemented, but this technique is both costly and requires detailed knowledge of subsurface structure, which is usually neither known nor easily measurable. Instead, we can modify the matte term to be a simple approximation that captures the important qualitative angular behavior shown in Figure 24.4.

Let us assume that the matte term is not Lambertian, but instead is some other function that depends only on θ, θ' and λ: $\rho_m(\theta, \theta', \lambda)$. We discard behavior that depends on ϕ or ϕ' in the interest of simplicity. We try to keep the formulas reasonably simple because the physics of the matte term is complicated and

sometimes requires unknown parameters. We expect the matte term to be close to constant, and roughly rotationally symmetric (He et al., 1992).

An obvious candidate for the matte component $\rho_m(\theta, \theta', \lambda)$ that will be reciprocal is the *separable* form $kR_m(\lambda)f(\theta)f(\theta')$ for some constant k and matte reflectance parameter $R_m(\lambda)$. We could merge k and $R_m(\lambda)$ into a single term, but we choose to keep them separated because this makes it more intuitive to set $R_m(\lambda)$—which must be between 0 and 1 for all wavelengths. Separable BRDFs have been shown to have several computational advantages, thus we use the separable model:

$$\rho(\theta, \phi, \theta', \phi', \lambda) = R_f(\theta)\rho_s(\theta, \phi, \theta', \phi') + kR_m(\lambda)f(\theta)f(\theta').$$

We know that the matte component can only contain energy not reflected in the surface (specular) component. This means that for $R_m(\lambda) = 1$, the incident and reflected energy are the same, which suggests the following constraint on the BRDF for each incident θ and λ:

$$R_f(\theta) + 2\pi k f(\theta) \int_0^{\frac{\pi}{2}} f(\theta') \cos\theta' \sin\theta' d\theta' = 1. \tag{24.1}$$

We can see that $f(\theta)$ must be proportional to $(1 - R_f(\theta))$. If we assume that matte components that absorb some energy have the same directional pattern as this ideal, we get a BRDF of the form

$$\rho(\theta, \phi, \theta', \phi', \lambda) = R_f(\theta)\rho_s(\theta, \phi, \theta', \phi') + kR_m(\lambda)[1 - R_f(\theta)][1 - R_f(\theta')].$$

We could now insert the full form of the Fresnel equations to get $R_f(\theta)$, and then use energy conservation to solve for constraints on k. Instead, we will use the approximation discussed in Section 24.1.1 We find that

$$f(\theta) \propto (1 - (1 - \cos\theta)^5).$$

Applying Equation (24.1) yields

$$k = \frac{21}{20\pi(1 - R_0)}. \tag{24.2}$$

The full coupled BRDF is then

$$\begin{aligned}\rho(\theta, \phi, \theta', \phi', \lambda) = \\ &\left[R_0 + (1 - \cos\theta)^5(1 - R_0)\right] \rho_s(\theta, \phi, \theta', \phi') + \\ &kR_m(\lambda) \left[1 - (1 - \cos\theta)^5\right] \left[1 - (1 - \cos\theta')^5\right].\end{aligned} \tag{24.3}$$

The results of running the coupled model is shown in Figure 24.5. Note that for the high viewpoint, the specular reflection is almost invisible, but it is clearly visible in the low-angle photograph image, where the matte behavior is less obvious.

For reasonable values of refractive indices, R_0 is limited to approximately the range 0.03 to 0.06 (the value $R_0 = 0.05$ was used for Figure 24.5). The value of R_s in a traditional Phong model is harder to choose, because it typically must be tuned for viewpoint in static images and tuned for a particular camera sequence for animations. Thus, the coupled model is easier to use in a "hands-off" mode.

24.5 Rough-Layered Model

The previous model is fine if the surface is smooth. However, if the surface is not ideal, some spread is needed in the specular component. An extension of the coupled model to this case is presented here (Ashikhmin & Shirley, 2000). At a given point on a surface, the BRDF is a function of two directions, one in the direction toward the light and one in the direction toward the viewer. We would like to have a BRDF model that works for "common" surfaces, such as metal and plastic, and has the following characteristics:

1. Plausible. As defined by Lewis (R. R. Lewis, 1994), this refers to the BRDF obeying energy conservation and reciprocity.

2. Anisotropy. The material should model simple anisotropy, such as seen on brushed metals.

3. Intuitive parameters. For material, such as plastics, there should be parameters R_d for the substrate and R_s for the normal specular reflectance as well as two roughness parameters n_u and n_v.

4. Fresnel behavior. Specularity should increase as the incident angle decreases.

5. Non-Lambertian diffuse term. The material should allow for a diffuse term, but the component should be non-Lambertian to assure energy conservation in the presence of Fresnel behavior.

6. Monte Carlo friendliness. There should be some reasonable probability density function that allows straightforward Monte Carlo sample generation for the BRDF.