Visualization of Minkowski Operations by computer graphics techniques [*]

J.B.T.M. Roerdink & G.S.M. Blaauwgeers
Dept. of Mathematics and Computing Science, University of Groningen
P.O. Box 800, 9700 AV Groningen, The Netherlands
Tel. +31-50-3633931; Fax +31-50-3633800; Email: `roe@cs.rug.nl`

Abstract

We consider the problem of visualizing 3D objects defined as a Minkowski addition or subtraction of elementary objects. It is shown that such visualizations can be obtained by using techniques from computer graphics such as ray tracing and Constructive Solid Geometry. Applications of the method are found in solid modelling and shape description.

1 Introduction

The problem of visualizing 3D objects defined as a Minkowski addition or subtraction of elementary objects — referred to as *Minkowski objects* below — is not a trivial one. Even when the elementary objects are very simple, such as spheres, boxes, cylinders, etc., the composite objects can become very complicated higher order surfaces with not even an explicit functional or parametric representation.

To overcome these problems, we use here techniques from computer graphics to visualize these objects. The basis is the *ray tracing* technique, which is an established method in computer graphics to visualize 3D objects by simulating the physical processes of ray propagation, reflection and transmission [1,4]. To visualize composite objects, the ray tracing technique uses Constructive Solid Geometry (CSG) methods, in which it is possible to ray trace objects formed by elementary set Operations, such as union, intersection and set difference (called *CSG objects* from now on) [5].

Using this as a starting point, we establish a mapping between 3D Minkowski objects and objects defined in terms of CSG Operations, which can subsequently be visualized by the standard ray tracing technique. In particular, we use a basic set of *elementary shapes*, and derive a decomposition of a Minkowski sum of any number of objects chosen from the basic set, into CSG objects. Even for a small set of elementary shapes the number of decomposition rules can become very large, so we use a number of preprocessing steps to reduce this number. Various examples of 3D visualizations will be shown.

The method we present here may be regarded as an extension of the *sweep representations* in *solid modelling* [1,4]. A *sweep* is an object created by moving another object along a curve in space. Simple examples are *translational sweeps* defined by a 2D area swept along linear path normal to its plane, or *generalized cylinders* where the path is curved.

Applications of the technique presented here are twofold: first, it may be used as an extension of the standard CSG methods in computer graphics. Second, we mention the area of morphological image processing, where the basic image transforms — dilations and erosions — are based upon the Minkowski operations [3, 6]. Visualization of the results of such operations on 3D images might be a useful research tool. Also inn shape description and solid modeling the use of Minkowski operations has already proved to be fruitful, see e.g. Ghosh [2].

[*]In: Mathematical Morphology and its Applications to Image Processing, J. Serra and P. Soille (eds.), Kluwer, 1994, pp. 289-296. Postscript version obtainable at `http://www.cs.rug.nl/~roe/`

2 Preliminaries

In this section we first present the basics of ray tracing and Constructive Solid Geometry, followed by the definition of Minkowski Operations.

2.1 Ray tracing

Ray tracing is a technique for image synthesis: creating 2-D pictures of a 3-D world. The first step in simulating such an image will be a projection of the scene on the screen. We assume we only have one eye through which we look at the screen. The location of the eye in the 3-D world is referred to as the *eye position*. In front of the eye we place the screen, and behind it we build the scene from which we want to generate an image. The color of a pixel on the screen is determined by all light rays — emitted by one or more light sources — that strike that pixel and the average color of all these rays is assigned to it.

Since many of the rays emitted from the light source(s) never hit the screen, one uses *backward ray tracing*: for every pixel on the screen, we follow back the light ray from the eye through the given pixel, and see where it first hits an object. The intersection point thus found in principle determines the color at the pixel. From this intersection point, a line (called a *shadow ray*) is traced towards every light source in the scene. When such a line intersects an object in front of the light, the point lies in the shadow of this object, unless the object is (partially) transparent, and the rays of that light source play no role in the color of the pixel. Otherwise the amount of light (of each color component) that is reflected is calculated and averaged with the colors of the rays coming from other light sources. The amount of light that is reflected depends on the angle between the normal vector at the intersection point on the surface of the object, and the line connecting it with the light source. But this is not the only light that strikes the object. As a result of reflection it is possible that rays strike the object via an indirect path. This part must also be accounted for. The standard way to achieve this, is by pretending that the eye is moved to the intersection point, and starting the whole process from there again. In this way we can implement the ray tracing technique *recursively*. For objects which are both transparent and reflective, we also have to find the colors of the light it transmits and reflects: such light arrived along *reflection* or *transmission rays*, whose directions are determined by the laws of (specular or diffuse) reflection and transmission (Snell's law). Finally we include an *ambient* term, to account for diffuse light arriving via indirect paths.

2.1.1 Ray/Quadric intersection

A general class of objects which are relatively simple to intersect with a ray are the quadrics: cylinders, cones, ellipsoids, paraboloids, hyperboloids, spheres, planes, etc. For reasons of efficiency, such simple objects are often given their own intersection routines.

A general quadric is given by the equation

$$(x\ y\ z\ 1)\begin{pmatrix} A & B & C & D \\ B & E & F & G \\ C & F & H & I \\ D & G & I & J \end{pmatrix}\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = 0 \tag{1}$$

This expression is equivalent to the equation $F(x, y, z) = 0$, with

$$F(x, y, z) = Ax^2 + 2Bxy + 2Cxz + 2Dx + Ey^2 + 2Fyz + 2Gy + Hz^2 + 2Iz + J \tag{2}$$

We use the parametric ray representation

$$R(t) = \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix} + t\begin{pmatrix} R_x \\ R_y \\ R_z \end{pmatrix}, \quad t > 0 \tag{3}$$

Substituting (3) into (1) and solving for t yields a quadratic formula $at^2 + bt + c = 0$ with the following coefficients

$$\begin{aligned}a = \quad & AR_x^2 + ER_y^2 + HR_z^2 + 2(BR_xR_y + CR_xR_z + FR_yR_z) \\ b = \quad & 2(AX_0R_x + B(X_0R_y + R_xY_0) + C(X_0R_z + R_xZ_0) + DR_x + \\ & EY_0R_y + F(Y_0R_z + R_yZ_0) + GR_y + HZ_0R_Z + IR_z) \\ c = \quad & AX_0^2 + EY_0^2 + HZ_0^2 + 2(BX_0Y_0 + CX_0Z_0 + DX_0 + \\ & FY_0Z_0 + GY_0 + IZ_0) + J\end{aligned} \tag{4}$$

Solving for t we find

$$t_{0,1} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{5}$$

If $a \neq 0$ we have to check the discriminant ($d = b^2 - 4ac$). If $d < 0$, no intersection takes place. Otherwise calculate t_0 and possibly t_1, if needed. The smallest positive value of t is used to calculate the closest intersection point. If $a = 0$, then the solution is simply

$$t = -\frac{c}{b} \tag{6}$$

When t has been computed, we have to check that it has a value greater than 0. If not, the point of intersection lies behind our eye, and does not contribute to the image. If t has a value greater than 0 the intersection point ip is calculated by substituting t in equation (3).

2.2 Constructive Solid Geometry

In order to compute the intersection point of a ray and an object, we need a mathematical description of the ray and the object. However, we generally do not have such a description of all the objects. To solve this problem, a technique called *constructive solid geometry* (CSG) has been developed. In CSG, simple primitives (like polygons) are combined by means of set operators. A composite object can be represented as a binary tree where each node contains a set operation and two child nodes, which may also be composite solids. The leaves of this tree are primitive objects such as quadrics or polygons. Evaluation of set operations usually reduces to classification whether a point p lies on the inside or outside of the composite object. We adopt the convention that the eye is positioned outside every object. The rules for classification are given in the table below.

operator	left	right	composite
Union	IN	IN	IN
	IN	OUT	IN
	OUT	IN	IN
	OUT	OUT	OUT
Intersection	IN	IN	IN
	IN	OUT	OUT
	OUT	IN	OUT
	OUT	OUT	OUT
Difference	IN	IN	OUT
	IN	OUT	IN
	OUT	IN	OUT
	OUT	OUT	OUT

2.3 Minkowski operators

The classical Minkowski addition and subtraction for subsets X, A of $\mathbb{R}^n$ are given by

$$X \oplus A \quad = \quad \bigcup_{a \in A} X_a, \tag{7}$$

$$X \ominus A \quad = \quad = \bigcap_{a \in A} X_{-a}, \tag{8}$$

where

$$X_a = \tau_a(X) = \{x + a : x \in X\},$$

is the translate of X over the vector $a \in \mathbb{R}^n$, $x + y$ is the sum of x and y, and $-x$ the reflection of x. It can be shown that

$$X \oplus A = \{h \in \mathbb{R}^n : \check{A}_h \Uparrow X\}, \tag{9}$$

where $\check{A} = \{-a : a \in A\}$ is the *reflection* of A and $A \Uparrow B$ (A 'hits' B) is a general notation for $A \cap B \neq \emptyset$.

3 Visualization of 3D Minkowski objects

3.1 Terminology

Before describing the process of transforming 3D Minkowski objects into objects which are defined in terms of CSG operations and therefore can be visualized via standard ray tracing techniques, we shall present some terminology.

In order to define Minkowski operations, one must possess a basic set of primitive shapes, referred to as *primary Minkowski primitives.* We restrict ourselves here to a set consisting of the following elementary geometric objects: *point, xline, yline, zline, xdisc, ydisc, zdisc, sphere.* Here an `xline` is a line segment in the x-direction, and an `xdisc` is a disc with thickness zero in the plane through the origin with normal in the x-direction, etc. This type of primitives has the property of being *(Minkowski-)*irreducible, i.e. no decomposition of a primary Minkowski primitive in a finite number of shapes, different from this primitive, exists such that applying Minkowski summation to these shapes will result in this primitive. All Minkowski primitives are solid objects.

As with the ordinary boolean set operators *union*, *intersection* and *difference*, one can define terms — referred to as *Minkowski terms* — consisting of multiple operands (primary Minkowski primitives) separated by Minkowski operators ($\oplus$ or $\ominus$).

Processing Minkowski terms may lead to new Minkowski objects, which can be regarded as elementary for further calculations, although they are *(Minkowski-)reducible.* These new Minkowski objects are basic intermediate results in the reduction process; decomposing them would complicate further reduction operations quite a lot. We refer to this kind of Minkowski primitives as *secondary Minkowski primitives.* Examples of these primitives are *rectangles, blocks, cylinders, toroidal discs* (discs of which the edge consists of the outer half of a torus, a bit like a discus) and a *cyclohedron* (the Minkowski sum of two perpendicular flat discs).

Our goal is to decompose 3D Minkowski objects — defined as Minkowski terms — into elementary objects, representing the original shape by means of CSG, that can be visualized through ray tracing techniques. These objects, which can be considered elementary for the ray tracing technique, will be referred to as *ray tracing primitives.* Examples of such ray tracing primitives are: *blocks, spheres, cylinders, tori,* and *polycircles.*

It should be mentioned that it is possible to define 3D Minkowski objects that cannot be visualized through standard ray tracing techniques, simply because they constitute of complicated, higher-degree surfaces for which no intersectionroutines are provided. These surfaces will be considered *"intractable"*. Take for example the Minkowski sum of three perpendicular flat circular discs, which results in a die-shaped object. Even if an equation for its surface can be found, it will most certainly be of a degree higher than seven, so ray/surface intersection can probably not be evaluated, using existing ray tracers which usually handle at most quartic surfaces. This is for example the case with the ray tracer "POV-Ray" which we used to generate the pictures in this paper.

3.2 Decomposition of Minkowski terms

When both Minkowski plus and Minkowski minus operators are allowed in Minkowski terms, the reduction process can be very complicated. Even for our small set of Minkowski primitives, the number of decomposition rules will then become enormous. Because Minkowski plus operations have, in comparison to Minkowski minus operations, more desirable properties such as associativity, commutativity and distributivity, dealing

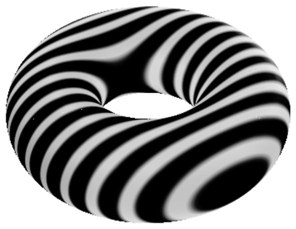

Figure 1: Visualization of the Minkowski sum of a circle and a sphere, a torus.

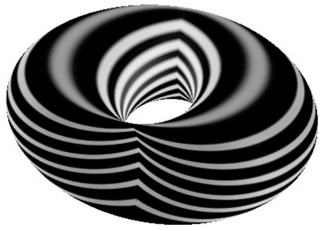

Figure 2: Visualization of the Minkowski sum of two perpendicular circles, a polycircle.

with just Minkowski sums will be the least difficult. In this paper only the decomposition of Minkowski sums will be discussed in depth.

Let us consider Minkowski sums of the form

$$A_1 \oplus A_2 \oplus \ldots \oplus A_n \tag{10}$$

where each A_j is chosen from the set of primary Minkowski primitives. Such sums may be of a simple form, such as `xline` $\oplus$ `yline`, leading to a `zrectangle`, which is a rectangle with normal in the z-direction. The Minkowski sum of a circle and a sphere leads to torus, which in fact is a *quartic* (fourth order surface), shown in Fig. 1. The Minkowski sum of two orthogonal circles leads to a somewhat unfamiliar object which we call a *polycircle*, which is also quartic, shown in Fig. 2. A derivation for the equation of this surface is given in the appendix. Notice that a polycircle, being a quartic surface, is a ray tracing primitive.

Now our goal is to decompose sums of the form (10) into a *union* of objects B_j, where each B_j is an object which can be visualized by standard ray tracing techniques. The union of the B_j's can then be handled by using CSG Operations, as explained above. For this reason we refer to the B_j as *ray tracing primitives.* In order to do so, we must eliminate the Minkowski operations in the Minkowski term. We can do this by recursively replacing every combination $A \oplus B$ within a Minkowski sum by a union of primary and secondary Minkowski primitives, a result we obtain by geometric analysis. This analysis has to be done by hand since a computer does not have knowledge of shapes at a meta level, at least not with the kind of representation we are using here. For example the Minkowski sum of a flat disc and line segment perpendicular to that disc results in a cylinder. A computer cannot come to this conclusion unless we supply this information. It is however not impossible to letting the computer figure out some of the shapes that can arise during a reduction of a Minkowski term, but this would mean that shape information would have to be included for every Minkowski primitive, which would seriously complicate implementation of objects and the process of decomposing a Minkowski term. So the former alternative is the most workable.

Thus, the purpose is to derive a list of replacement rules for every possible Minkowski sum of two Minkowski primitives. With these rules and using useful properties of the $\oplus$-operator like *commutativity*, *associativity* and *distributivity* over *union*, every Minkowski sum term can be reduced to a set of objects, which united, represent the whole Minkowski sum.

Minkowski addition of two primary Minkowski primitives of the same type will yield a primitive of the same type. For example, the Minkowski addition of two spheres results in a (bigger) sphere. This implies that the number of secondary Minkowski primitives is *finite.* Because of the commutativity of Minkowski addition operands may be exchanged. Thus given a Minkowski term of arbitrary length, we may group together operands of the same type and then, using the associativity of the Minkowski addition, reduce every group to precisely one element of the same type as the elements in the group. Doing so, the number of secondary Minkowski primitives can be at most the cardinality of the power set over the set of primary

Minkowski primitives. In fact the set of secondary Minkowski primitives that can arise with the current set of primary Minkowski primitives consists of thirteen objects: *xrectangle, yrectangle, zrectangle, block, xcylinder, ycylinder, zcylinder, xdiscus, ydiscus, zdiscus, xcyclohedron, ycyclohedron, zcyclohedron.*

3.3 Representation of Minkowski primitives

In order to understand the reduction formulas we mentioned above one must have some notion of the representation of the primary and secondary Minkowski primitives. They are represented in the following way, starting with primary Minkowski primitives:

point(x, y, z)	: a point with coordinates (x, y, z).
xline(x, y, z, lx)	: a line starting at (x, y, z) and ending at $(x + lx, y, z)$.
yline(x, y, z, lx)	: a line starting at (x, y, z) and ending at $(x, y + ly, z)$.
zline(x, y, z, lz)	: a line starting at (x, y, z) and ending at $(x, y, z + lz)$.
xdisk(rx)	: a flat disc in the YZ-plane with center at the origin and radius rx.
ydisk(ry)	: a flat disc in the XZ-plane with center at the origin and radius ry.
zdisk(rz)	: a flat disc in the XY-plane with center at the origin and radius rz.
sphere(r)	: a sphere with center at the origin and radius r.

During the process of decomposition of Minkowski terms, all kinds of secondary Minkowski primitives can arise, which can be used in consecutive reductions. These secondary Minkowski primitives are represented in a similar way, that is:

xrectangle(x, y, z, ly, lz)	: a flat rectangle parallel to the YZ-plane with lower left corner at (x, y, z) and upper right corner $(x, y + ly, z + lz)$.
yrectangle(x, y, z, lx, lz)	: a flat rectangle parallel to the XZ-plane with lower left corner at (x, y, z) and upper right corner $(x + lx, y, z + lz)$.
zrectangle(x, y, z, lx, ly)	: a flat rectangle parallel to the XY-plane with lower left corner at (x, y, z) and upper right corner $(x + lx, y + ly, z)$.
block(x, y, z, lx, ly, lz)	: a rectangular block with lower left foremost corner (x, y, z) and upper right hindmost corner at $(x + lx, y + ly, z + lz)$.
xcylinder(x, y, z, r, lx)	: the cylinder with axis **xline**(x, y, z, lx) and radius r.
ycylinder(x, y, z, r, ly)	: the cylinder with axis **yline**(x, y, z, ly) and radius r.
zcylinder(x, y, z, r, lz)	: the cylinder with axis **zline**(x, y, z, lz) and radius r.
xdiscus(x, y, z, rr, rd)	: the object obtained by the Minkowski addition of a `xdisc` with radius rd and a `sphere` with radius rr, centered at (x, y, z).
ydiscus(x, y, z, rr, rd)	: the object obtained by the Minkowski addition of a `ydisc` with radius rd and a `sphere` with radius rr, centered at (x, y, z).
zdiscus(x, y, z, rr, rd)	: the object obtained by the Minkowski addition of a `zdisc` with radius rd and a `sphere` with radius rr, centered at (x, y, z).
xcyclohedron(x, y, z, ry, rz)	: the object obtained by the Minkowski addition of a `ydisc` with radius ry and a `zdisc` with radius rz, centered at (x, y, z).
ycyclohedron(x, y, z, rx, rz)	: the object obtained by the Minkowski addition of a `xdisc` with radius rx and a `zdisc` with radius rz, centered at (x, y, z).
zcyclohedron(x, y, z, rx, ry)	: the object obtained by the Minkowski addition of a `xdisc` with radius rx and a `ydisc` with radius ry, centered at (x, y, z).

In Fig. 3 and Fig. 4 a *discus* and a *cyclohedron* are depicted.

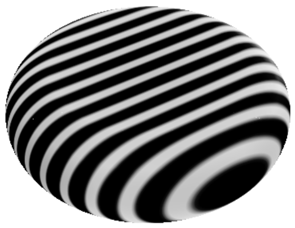

Figure 3: Visualization of the Minkowski sum of a flat disc and a sphere, (a *discus*).

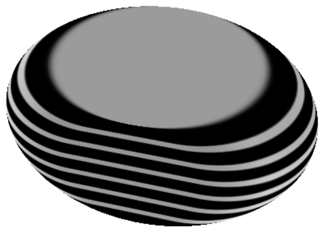

Figure 4: Visualization of the Minkowski sum of two perpendicular flat discs, (a *cyclohedron*).

3.4 Reduction to CSG objects

There are a total of 21 primary and secondary Minkowski primitives. This means that there are 21 x 21 = 441 different combinations possible for the Minkowski addition. Because of commutativity this number can immediately be reduced to 21 + (441 - 21)/2 = 231 combinations. This still is an awful lot. So we shall try to decrease this number.

One of the first things we notice is that several combinations may lead to the same result. For example,

$$\begin{array}{ll} & \texttt{xcylinder} \oplus \texttt{ycylinder} \\ = & \texttt{xdisc} \oplus \texttt{xline} \oplus \texttt{ydisc} \oplus \texttt{yline} \\ = & \texttt{xline} \oplus \texttt{yline} \oplus \texttt{xdisc} \oplus \texttt{ydisc} \\ = & \texttt{zrectangle} \oplus \texttt{zcyclohedron} \end{array}$$

This phenomenon arises in combinations where at least one of the operands is a secondary Minkowski primitive. To exclude this kind of non-uniqueness, sorting the primary Minkowski primitives in a Minkowski term of any length in a specific order before carrying out the reduction process does the trick. For example the following order is very suitable: `point`, `xline`, `yline`, `zline`, `xdisc`, `ydisc`, `zdisc`, `sphere`. This method has another advantage, namely that operands of the same type are grouped together and reduction of a Minkowski addition of primary primitives of the same type is quite simple. Doing so we end up with at most eight different primitives in the term to be reduced. Lots of combinations will be excluded this way, reducing their number by a factor of more than two.

Next, using distributivity, we can recursively reduce left-to-right the reordered Minkowski term. Yet, this way of reducing terms appears not to be the most efficient. First of all *redundant* sets may arise in the union of ray tracing primitives. For example consider,

$$\begin{array}{ll} \texttt{yline} \oplus \texttt{zline} \oplus \texttt{xdisc} \oplus \texttt{ydisc} & \longmapsto \\ \texttt{xrectangle} \oplus \texttt{xdisc} \oplus \texttt{ydisc} & \longmapsto \\ \texttt{(4 xdiscs} \cup \texttt{2 xrectangles)} \oplus \texttt{ydisc} & \longmapsto \\ \texttt{4 zcyclohedra} \cup \texttt{4 ycylinders} \cup \texttt{2 blocks} & \end{array}$$

The result consists of 4 zcyclohedra, 4 ycylinders and 2 blocks, located at a different positions of course, see Fig. 5. The result nevertheless could be composed with 4 zcyclohedra, 2 ycylinders and 2 blocks. So 4 - 2 = 2 superfluous ycylinders are being created using the method described above.

Another drawback of this method is that intermediate results can arise consisting of a union of several primitives, so that in a successive Minkowski addition the primitive at the right has to be distributed over this union of primitives. This causes a whole new series of Minkowski additions to be calculated, from which the results must be united afterwards. This way of reducing Minkowski sums would thus be attended with a lot of overhead.

To overcome these difficulties we use a divide-and-conquer strategy. First we split up the set of primary Minkowski primitives into two groups. In the first group we put the line primitives plus the point primitive. In the second group we put the disc primitives and the sphere primitive. After reordering the initial Minkowski term so primitives of the same type are put together and then reduced easily, we divide the resulting primitives over the two groups and make reductions for both groups separately. The reason for this is that reduction of a group results in at most one primitive (none, if the group is contains no primitives at all). The last step is then to reduce the outcome of the reductions of both groups.

Using this technique of *regrouping*, we have cut down the number of reduction formulas to 114, avoiding redundant sets and much overhead in the reduction process. So again we reduced the number of necessary formulas with 50%.

3.5 Reduction formulas

Minkowski terms are being reduced using a list of reduction formulas we have to determine by hand. Consider for example the following Minkowski sum:

$$\begin{gathered}\texttt{point(a0,b0,c0)} \oplus \texttt{ydisc(a1,b1,c1,r1)} \oplus \texttt{sphere(a2,b2,c2,r2)} \\ \oplus\ \texttt{xline(a3,b3,c3,l3)} \oplus \texttt{zline(a4,b4,c4,l4)}\end{gathered}$$

As mentioned before points and lines are put in a separate group as are discs and spheres. For the group of points and lines we conform to the following reduction order: `point`, `xline`, `yline`, `zline`. So first we reduce `point(x1,y1,z1)` and `xline(a3,b3,c3,l3)`. Next we perform a reduction of the previous result in combination with `zline(a4,b4,c4,l4)`. To carry out these reductions the following reduction formulas are required:

$$\begin{aligned}
\textbf{point}\,(x1,y1,z1) \oplus \textbf{xline}\,(x2,y2,z2,l) &\longmapsto \textbf{xline}\,(x1+x2,y1+y2,z1+z2,l)\\
\textbf{xline}\,(x1,y1,z1,lx) \oplus \textbf{yline}\,(x2,y2,z2,ly) &\longmapsto \textbf{zrectangle}\,(x1+x2,y1+y2,z1+z2,lx,ly)
\end{aligned}$$

For the group of discs and spheres we conform to following reduction order: `xdisc`, `ydisc`, `zdisc`, `sphere`. In this example however the reduction order is of no importance, since only one reduction has to be performed, namely that of `ydisc(a1,b1,c1,r1) and \vb`sphere(a2,b2,c2,r2)—, demanding the following reduction formula:

$$\textbf{ydisc}\,(x1,y1,z1,ry) \oplus \textbf{sphere}\,(x2,y2,z2,r) \longmapsto \textbf{ydiscus}\,(x1+x2,y1+y2,z1+z2,r,ry)$$

Next we have to reduce the results of both groups. This is the most involved part of the reduction process. We have to reduce a `yrectangle` and a `ydiscus`, which leads to an object that can be viewed as a union of several secondary Minkowski primitives, to be obtained by using the following reduction formula:

$$\begin{array}{ll}
\multicolumn{2}{l}{\textbf{yrectangle}\,(x1,y1,z1,lx,lz) \oplus \textbf{ydiscus}\,(x2,y2,z2,rr,rd) \quad \longmapsto}\\
\quad\textbf{ydiscus}\,(x1+x2,y1+y2,z1+z2,rr,rd) & \cup\\
\quad\textbf{ydiscus}\,(x1+x2+lx,y1+y2,z1+z2,rr,rd) & \cup\\
\quad\textbf{ydiscus}\,(x1+x2,y1+y2,z1+z2+lz,rr,rd) & \cup\\
\quad\textbf{ydiscus}\,(x1+x2+lx,y1+y2,z1+z2+lz,rr,rd) & \cup\\
\quad\textbf{xcylinder}(x1+x2,y1+y2,z1+z2-rd,rr,lx) & \cup\\
\quad\textbf{xcylinder}(x1+x2,y1+y2,z1+z2+lz+rd,rr,lx) & \cup\\
\quad\textbf{zcylinder}(x1+x2-rd,y1+y2,z1+z2,rr,lz) & \cup\\
\quad\textbf{zcylinder}(x1+x2+lx+rd,y1+y2,z1+z2,rr,lz) & \cup\\
\quad\textbf{block}(x1+x2-rd,y1+y2,z1+z2,lx+2*rd,2*rr,lz) & \cup\\
\quad\textbf{block}(x1+x2,y1+y2,z1+z2-rd,lx,2*rr,lz+2*rd) &
\end{array}$$

A convenient way of illustrating this formula is by drawing a sketch in outline of the object we get as a result of a Minkowski sum. Primitives that lie completely inside a resulting object are excluded. As an example we demonstrate in Fig. 5 a sketch of the Minkowski term we discussed above.

The last step is to translate the acquired list of Minkowski primitives into correct input code for a specific ray tracer, with which these primitives can be visualized. After that we can view the result of the reduced Minkowski sum, using the obtained code as input for ray tracing.

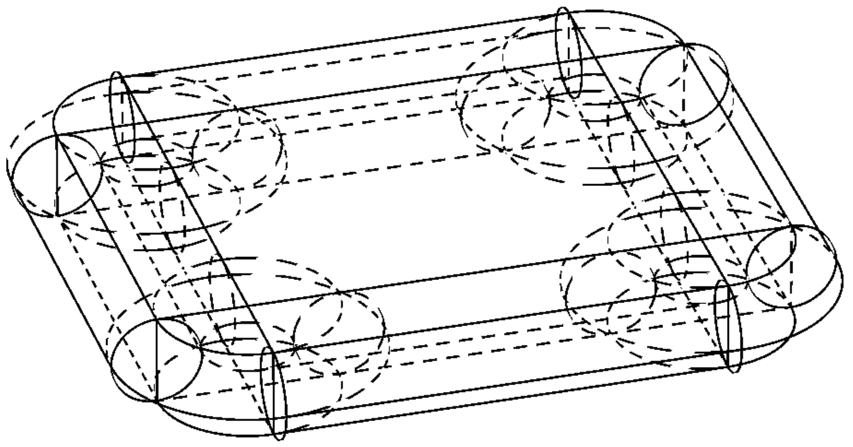

Figure 5: Sketch in outline of a Minkowski sum of a `point`, a `xline`, a `yline`, a `ydisc` and a `sphere`, from which the various primitives can be derived.

3.6 Optimizations

Up to this point we need 114 reduction formulas to cover the decomposition of all Minkowski sums we can possibly make, given the present set of primary Minkowski primitives. Although we have established a diminution of more than half the number of reduction formulas so far, we still have to cope with a large number of them. Reductions may produce several Minkowski primitives, that all have to be provided for in the eventual implementation. So we are talking about hundreds of programming rules here. It is obvious that the more we can curtail this number the more concise (and faster) the implementation will be. So let us investigate possibilities to achieve this.

The first optimization we could make is to postpone reductions involving `point` primitives. Since Minkowski addition with points comes down to shift operations we effectuate these not until the final step in the process of decomposition: the translation of the Minkowski primitives into ray tracer primitives. This modification bears several advantages. First of all, the number of required formulas is lessened by 21 (the total number of Minkowski primitives). Secondly, most ray tracers offer the possibility of translating compounded objects, which makes the implementation of the shift operation pretty simple: just adding the required translation statement satisfies. Thirdly, many occurrences of "$x1 + x2$" in the reduction formulas can be left out. As a result of this, the primary Minkowski primitives `xline`, `yline` and `zline` will always have the origin as a starting point in appearances in the optimized set of reduction formulas, so that in fact they are superfluous. This gives rise to the next optimization step: omission of the coordinates of the starting point in the representation of the line primitives.

So far the right-hand side of the reduction formulas could contain Minkowski primitives that do not belong to the set of ray tracer primitives: *disci* and *cyclohedra*. These objects should be regarded as solid, convex shapes. In order to produce ray tracer primitives for them, the following additional reductions have to be performed:

$$\begin{array}{llll}
\mathbf{xcyclohedron}\,(x,y,z,ry,rz) & \longmapsto & \mathbf{xpolycircle}\,(x,y,z,ry,rz) & \cup \\
 & & \mathbf{ycylinder}\,(x,y-rz,z,ry,2*rz) & \cup \\
 & & \mathbf{zcylinder}\,(x,y,z-ry,rz,2*ry) & \\
 & & & \\
\mathbf{ycyclohedron}\,(x,y,z,rx,rz) & \longmapsto & \mathbf{ypolycircle}\,(x,y,z,rx,rz) & \cup \\
 & & \mathbf{xcylinder}\,(x-rz,y,z,rx,2*rz) & \cup \\
 & & \mathbf{zcylinder}\,(x,y,z-rx,rz,2*rx) & \\
 & & & \\
\mathbf{zcyclohedron}\,(x,y,z,rx,ry) & \longmapsto & \mathbf{zpolycircle}\,(x,y,z,rx,ry) & \cup \\
 & & \mathbf{xcylinder}\,(x-ry,y,z,rx,2*ry) & \cup \\
 & & \mathbf{ycylinder}\,(x,y-rx,z,ry,2*rx) & \\
 & & & \\
\mathbf{xdiscus}\,(x,y,z,rr,rs) & \longmapsto & \mathbf{xtorus}\,(x,y,z,rr,rs) & \cup \\
 & & \mathbf{xcylinder}\,(x-rr,y,z,rs,2*rr) & \\
 & & & \\
\mathbf{ydiscus}\,(x,y,z,rr,rs) & \longmapsto & \mathbf{ytorus}\,(x,y,z,rr,rs) & \cup \\
 & & \mathbf{ycylinder}\,(x,y-rr,z,rs,2*rr) & \\
 & & & \\
\mathbf{zdiscus}\,(x,y,z,rr,rs) & \longmapsto & \mathbf{ztorus}\,(x,y,z,rr,rs) & \cup \\
 & & \mathbf{zcylinder}\,(x,y,z-rr,rs,2*rr) &
\end{array}$$

A third optimization is to insert unions of ray tracer primitives instead of disci and cyclohedra. This way we are able to prevent the generation of redundant cylinders. Consider for example the Minkowski addition of a `xline` and a `xdiscus`, resulting in the union of two parallel `disci` and a `xcylinder` in between. When the two `disci` are converted to ray tracer primitives, each of them will become the union of a `torus` and a `xcylinder`. Both newly created `xcylinders` share the same longitudinal axis and the cylindrical space in between lies completely interior to the embracing `xcylinder` we already had. It is preferable to replace those two `xcylinders` by a longer version that contains them both plus the space in between. This will reduce the number of cylinders to be traced by the ray tracer (time saving).

Finally we present a fourth adaptation which will reduce the number of reduction formulas drastically. Many ray tracer primitives exist in three variants (x, y, and z), that differ only in orientation. Reduction formulas involving primitives of a certain variant show much resemblance with the reduction formulas involving the other two remaining variants. For all three variants the same sort of primitives are produced, just differing from each other in orientation. We can exploit this by transforming all akin reduction formulas to one particular variant, applying an inverse transformation to the set of produced ray tracer primitives to get the right rotational forms. This entails primitives to be rotated over right angles, which will not cause problems as the set of all tractable Minkowski primitives is closed under right angle rotations. What we need at thisp[oint is the following theorem which is given without proof.

Theorem 1 (Invariance of Minkowski addition under linear transformations) *For every linear transformation R of $\mathbb{R}^3$ the following property holds:*

$$A \oplus B \;=\; R^{-1}(\; R(A) \oplus R(B)\;) \tag{11}$$

This property is most meaningful when applied to the reduction of the results of both groups of primitives (the first group being the one of all "straight" primitives and the second the one of all "curved" ones). We selected the x-variant as the standard case.

The purpose of rotating is to obtain just one basic reduction formula instead of three we had till now. When the operands of a Minkowski are turned such that one of them is always of the x-type, the other one can take on all of the three variants. So which of the groups is the most eligible for the elimination of rotated versions? Well, the group with the group with straight primitives (lines, rectangles and blocks) will be reduced from 7 to 3. The group of curved primitives (circular discs, disci, cyclohedra and spheres) on the other hand will be reduced from 10 to 4, so this one is the most appropriate ($4/10 < 3/7$!). An attendant advantage with the usage of this group for the elimination of rotation analogies is that it makes no difference in which direction a primitive is rotated: turned over right angles, irrespective of the direction, the same object is obtained. So just one direction needs to be considered, reducing the number of required rotation formulas by half. One associated effect is that this approach may cause the *reference location* of lines, rectangles and blocks to be altered as a result of rotating operations. Instead of the origin another point, of which one of its coordinates is negative, may be the new reference location. This is caused by the fact that these objects are not rotated about their centers. One way of solving this problem could be to make the reference location of these objects their center. But a major disadvantage of this approach would be that we would have to adjust many reduction formulas, causing them to be less clear. Besides that we would be obliged to make calculations with half lengths, which can be a burden. A better solution would be to simply perform a shift operation when this problem appears before executing the reduction of the Minkowski addition. After the reduction the acquired ray tracer primitives must be shifted back over the same distance (no more then one coordinate can be negative !). Here we make use of the *translation invariance* property of Minkowski addition. Eliminating rotation analogies from the group of curved primitives, our list of reduction formulas may contain only reductions on the primitives `xdisc`, `xdiscus`, `xcyclohedron` and `sphere` when any of these type of primitives is involved.

Finally one small optimization remains. Sometimes it is possible to apply elimination of rotation analogies to the set of "straight" primitives. This is the case if the necessary rotation leaves the "curved" primitive unchanged, for example in reductions of Minkowski additions with spheres. This could also be the case in combinations with `disci` and `cyclohedra`, but it appears that in this case rotated versions will already have been eliminated in the elimination process described before. So only in combinations with spheres it is required to reduce rotated versions of straight primitives. So another elimination of four reduction formulas is gained. Applying all optimizations described so far, we obtain a list of just 44 reduction formulas.

The rotation of Minkowski primitives requires extra transformations to be carried out, for which we have to contrive formulas like:

$$\textbf{block}\,(0,0,0,lx,ly,lz) \quad \overset{+\pi/2}{\longmapsto}_{y-axis} \quad \textbf{block}\,(0,0,-lx,lz,ly,0)$$

For all lines, rectangles, blocks, discs, disci and cyclohedra rotation formulas about both the y-axis and the z-axis must be derived. After reducing a Minkowski addition with a rotated primitive, the obtained ray tracer primitives are to be rotated back, requiring formulas like:

$$\textbf{block}\,(x,y,z,lx,ly,lz) \quad \overset{-\pi/2}{\longmapsto}_{y-axis} \quad \textbf{block}\,(-z-lz,y,x,lz,ly,lx)$$

These kind of rotation formulas must are needed for the following ray tracer primitives: all rectangles, block, xdisc, xtorus, all cylinders and sphere (because its center need not be the origin) about the y-axis as well as the z-axis.

3.7 Minkowski Subtraction

Minkowski subtraction lacks, in contrast with Minkowski addition, some very useful properties like associativity, commutativity and certain forms of distributivity. This make Minkowski subtraction a lot more difficult to implement, because conjuring with parentheses, permutations and distributions is not allowed. On top of that we have that Minkowski subtraction leads to many more secondary Minkowski primitives than is the case when just Minkowski sums are allowed. For all combinations of Minkowski primitives reduction formulas must be derived. When both Minkowski addition and subtraction are allowed, it is not even certain that the number of secondary primitives is finite in all cases, let alone the number of reduction formulas that would be required to cover all combinations. If no restriction are being made concerning the definition of Minkowski terms, then the decomposition will hardly be feasible.

Another problem is that many of the secondary Minkowski primitives can become so complex that the corresponding ray tracer primitives won't be tractable (read: traceable). So the use of Minkowski subtraction in Minkowski terms will only be meaningful in a limited number of cases.

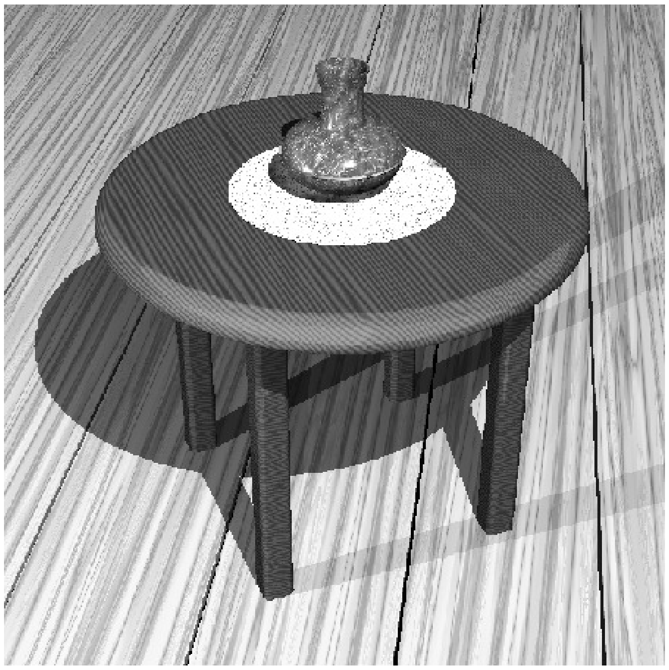

Figure 6: Legs and blade of the table are modelled by Minkowski sums.

3.8 Implementation

To implement the method a *preprocessor* routine was developed which takes a scene description as input. This description contains: positions of light sources and objects; descriptions of the composite objects in terms of unions, intersections, or Minkowski sums of elementary objects; and also, properties of the elementary objects (reflection and transmission coefficients, smoothness parameters). Also some texture may be applied to the surface of the objects to give more shape clues.

Output of the preprocessor is a file in which the Minkowski Operations have been replaced by a union of CSG objects, using the reduction process sketched above. This file is then itself used as input file for an ordinary ray tracer.

As an example Fig. 6 shows a scene of table on a floor where the legs of the table have been modelled by Minkowski sums. The legs are described by a Minkowski sum which appears in the description file in the following form:

```
$ xline(5) <+> zline(5) <+> yline(100) <+> ydisc(2) $.
```

This is in fact the Minkowski addition of a oblong block and a flat, horizontal disc. The table-top is defined by the following Minkowski sum:

```
$ ydisk(60) <+> sphere(5) <+> point(120 105 100) $,
```

resulting in a thin cylinder with a round edge, translated over a vector $(120, 105, 100)$. The `$`-signs delimit a Minkowski sum in the input file. The symbol `<+>` represents the $\oplus$-operator. Forall objects the origin is their *reference location* and in between the parentheses the accessory measurement is denoted. During preprocessing these Minkowski definitions are translated into proper ray tracer code.

Preprocessors for the ray tracers "DKBTrace v. 2.12" and POV-Ray v. 1.0 were developed. The pictures in this paper have been ray traced with "POV-Ray v. 1.0", an excellent ray tracer which offers many advanced features.

4 Summary

Though ray tracing is an ideal technique for the calculation of the well-known boolean set operations *union*, *intersection* and *difference*, the computation of Minkowski set operations using ray tracing techniques is not a trivial matter.

In this article we outlined a method of decomposing Minkowski set operations to sets of elementary objects, that can be visualized through ray tracing techniques, representing their objective by means of *Constructive Solid Geometry.* To bring about the decomposition of Minkowski terms we contrived a series of reduction formulas, endeavouring to keep their number as low as possible and keeping the result of the various reductions compact.

It appeared that the decomposition of Minkowski *sums* is quite feasible, because for Minkowski addition many pleasant properties hold, like commutativity, associativity, distributivity over *union*, translation invariance and invariance under linear transformations, thus keeping the number of reduction formulas low.

To implement the method a preprocessor routine was developed which takes a scene description as input and delivers an output file in which the Minkowski Operations have been replaced by a union of CSG objects. This file is then itself used as input file for an ordinary ray tracer.

Some remarks about extensions of this method are in order. First, when the use of Minkowski subtraction is permitted in Minkowski terms also, then the decomposition of these terms is a lot more complicated, requiring a huge number of reduction formulas and yielding a much larger variety of primitive ray tracer objects, whether tractable or not. If no conditions are prescribed concerning the form Minkowski subtractions may adopt in Minkowski terms, then implementing the decomposition process will be too laborious and practically impossible. Besides that the results can only partially be visualized by means of ray tracing.

Secondly, one may wonder to hat extent we can adapt this method in order to handle Minkowski sums of arbitrary objects. So far we have chosen to consider only a well-defined class of primitive objects, the Minkowski sums of which can be decomposed into ray tracer objects. In fact what we would need for arbitrary objects is a direct method to derive the intersection equations of a Minkowski sum from the equations of the individual objects. Perhaps similar methods used for sweep representations can be adapted to our situation [4Ch.3]. A minimum requirement still is that the opearnds of the Minkowski sum themselves are described in terms of an explicit mathematical equation.

5 References

[1] Foley, J. D., Dam, A. V. and Feiner, S. K., *Computer Graphics : Principles and Practice.* Reading, MA, Addison Wesley, 1990.

[2] Ghosh, P. K., “A mathematical model for shape description using Minkowski operators,” *Comp. Vis. Graph. Im. Proc.*, 44, pp. 239–269, 1988.

[3] Giardina, C. R. and Dougherty, E. R., *Morphological Methods in Image and Signal Processing.* Englewood Cliffs, NJ, Prentice-Hall, 1988.

[4] Glassner, A. S., Ed., *An Introduction to Ray Tracing*: Academic Press, New York, 1989.

[5] Roth, S., “Ray casting for modeling solids,” *Comp. Graph. Im. Proc.*, 18, pp. 109–144, 1982.

[6] Serra, J., *Image Analysis and Mathematical Morphology.* Academic Press, New York, 1982.

Appendix

Derivation of the surface equation of a XPOLYCIRCLE :

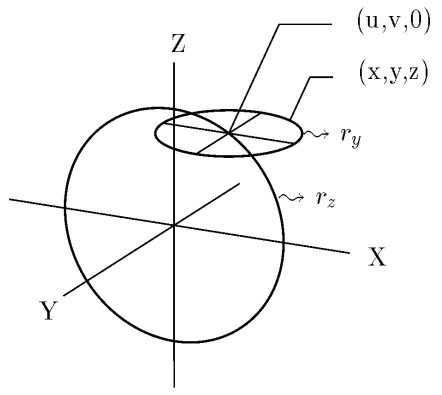

Figure 7: Sketch of two perpendicular circles.

Consider the Minkowski sum of a circle with radius r_z in the $X-Y$-plane and a circle with radius r_y in the $X-Z$-plane. To compute this sum we move the origin of the second circle to each point $(u,v,0)$ of the first circle. Let a typical point of the surface so generated be denoted by (x,y,z), see Fig. 7. Then the following relations hold:

$$\begin{aligned} u^2+v^2 &= r_z^2 \\ (x-u)^2+(y-v)^2+z^2 &= r_y^2 \\ y &= v \end{aligned}$$

Eliminating v and rewriting the second equation this becomes

$$\begin{aligned} u^2 &= r_z^2-y^2 \\ 2xu &= x^2-y^2+z^2+r_z^2-r_y^2 \end{aligned}$$

Now square the second equation and eliminate u by using the first one. Then there results the equation

$$4x^2(r_z^2-y^2)=(x^2-y^2+z^2+r_z^2-r_y^2)^2, \tag{12}$$

which in standard form of a quartic becomes

$$x^4+y^4+z^4+2x^2y^2+2x^2z^2-2y^2z^2+-2(r_y^2+r_z^2)x^2+2(r_y^2-r_z^2)y^2-2(r_y^2-r_z^2)z^2+(r_y^2-r_z^2)^2=0.$$