FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS
COMENIUS UNIVERSITY
IN BRATISLAVA


# RIGOROUS THESIS

2004

Mgr. Pavol Novotný

FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS
COMENIUS UNIVERSITY IN BRATISLAVA
INSTITUTE OF APPLIED INFORMATICS

# RIGOROUS THESIS

## Volume Representation by Gradient and Distance Fields

MGR. PAVOL NOVOTNÝ

BRATISLAVA 2004

**Statutory Declaration:**
I do solemnly and sincerely declare that this thesis has been written by myself without any unauthorized help, just using the literature listed at the end.

. . . . . .
Pavol Novotný

**Abstract**

*We present a new method for representation of volumetric data sets in this thesis. We use truncated discrete distance fields supplemented by additional information about the surface normal. To reduce memory requirements, we have developed a compression based on a similar idea as the well-known run-length encoding. Furthermore, we present a new way to perform CSG operations between volumes at the voxel level. Our technique eliminates artifacts of straightforward volumetric CSG operations by taking conditions for object representability into account, according to which sharp details are not correctly representable in discrete distance fields. Finally, we are interested in other solids with sharp details and try to remove artifacts in a similar way like in the process of CSG operations. Both the problems are solved by rounding edges and other sharp details to get representable objects. The* `vxtRL` *library has been created as an implementation of these algorithms.*

# Contents

# List of Figures

# Chapter 1

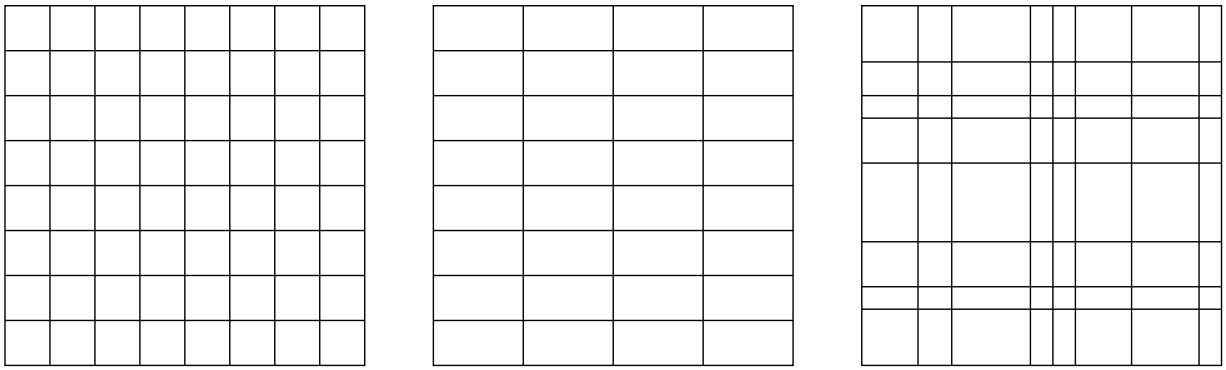
# Introduction

## 1.1 Volume Graphics and Voxelization

*Volume graphics*, introduced by authors Kaufman, Cohen and Yagel in 1993 [1], is a sub-area of computer graphics, which studies space objects in their true three dimensional essence. This approach is considerable different than the traditional one used in surface graphics. Whereas the goal of surface graphics is to create models which strive to simulate the appearance of the real world, volume graphics examines the inside substance of solids. Many years ago some artists (for example Michelangelo) had realized that to make a picture of reality it is not sufficient just to look around, but it is necessary to have a good knowledge about the inside structure of objects to understand also the surface properties. Surface graphics has accomplished admirable results in many areas of modelling, but it is still limited by a level of complexity of the surface. It is very difficult to handle objects, which do not have surface defined, for instance fire, smoke, mist, or fog. Moreover, various applications need to visualize data sets, which are three dimensional in the principle. As an instance we can mention medicine (tomography, magnetic resonance, ultrasound), biology (microscopic systems), geography (seismology), particle physics (electron density), industry (material structure, liquid and gas flow), meteorology (atmospheric phenomena), and many others. Volume graphics comes into being as a natural tool to solve these problems, but also it has a potential to compete with surface graphics in areas, where the traditional approach still dominates.

The most cited disadvantages of volume graphics are huge requirements on memory and processing time. However, with the increasing power of computer technology these drawbacks will probably soon stop to play an important role. On the other hand, we can mention benefits of volume graphics: independence of the scene complexity, uniform manipulation with different types of data (sampled or evaluated data, parametric and implicit surfaces) and simple realization of *CSG operations* [1].

The basic term of volume graphics is *voxel*, which is an abbreviation of *volume element*. Each voxel stores an information that characterizes locally given volume. The type of this information depends on the particular implementation. There is a wide range of options from a *binary voxel* (it can have just one of two values depending on whether it lies outside or inside the object) up



(a) (b) (c)

Figure 1.1: *Grid types*
*(a) Cartesian, (b) regular, (c) rectilinear*

to a voxel with many attributes (density, normal, color, material, reflection and refraction index, ...). Voxels can be arranged in various ways inside the volume, but the most common way is to use a kind of orthogonal grid (see Figure 1.1). In the case of Cartesian grid we can define a *voxel unit (VU)* as the distance between two neighbouring voxels.

The process of conversion from surface to volume representation is called *voxelization*. As first, some techniques for generation of binary data have been presented. This approach is not ideal from the point of view of subsequent rendering of such data, because it leads to undesirable aliasing. The problem is especially with the estimation of surface normal. Several methods for its computation from wider voxel neighbourhood have been proposed, but they are not precise enough for simulation of such phenomena as reflection or refraction on the surface. Better results have been achieved using *discrete raytracing* [2], where the analytical description of objects has been used. However, it is necessary to store an additional information in each voxel to identify which object it belongs to.

An other way to overcome the problem of aliasing are *filtered techniques* [3][4]. There is used a low pass filter for modification of binary data (for example *Barlet filter*, *Gaussian filter* or *oriented box*). By convolution of object with the filter an continuous function is defined and it is sampled into the grid. If the size of filter domain is less than the voxelized object then the inside of the object is represented with specific *inside density* and the outside with specific *outside density*. In the surface vicinity there is a thin *transition area* where the density changes continuously from inside to outside value. Data that is obtained during this process has properties similar to the data that is obtained by scanning of real objects (for instance using tomography), so it is possible to combine and visualize them easy in an uniform way. There are various possibilities how to use the transition area. We can define the surface exactly by thresholding at the middle value of density, or we can exploit this "blurred" surface for antialiasing directly so that the ray takes the intensity according to densities of voxels which are hit [3].

An important question is what filters to use to get the best results. Authors of paper [4] have found out that the choice of filter is closely associated with the methods we want to use for the density interpolation and the normal estimation. Using an improper combination of voxelization

filters with reconstruction techniques we get a systematic error which does not depend on the surface curvature, but on its orientation what is undesirable. We are going to discuss it later in the section *Reconstruction Errors*.

## 1.2 Representation of Objects by Distance Fields

The next approach that is able to reduce aliasing is the volume representation by *distance fields (DFs)*. In this method each voxel stores an information about the distance from the nearest surface point. To distinguish between outside and inside area, voxels on different sides of the surface have opposite signs (for example positive outside and negative inside). Firstly, DFs were implemented by filling whole the grid with values of distance function defined by the surface [5], so all the voxels stored information about the object. It is in contrast with the traditional conception of an object which can be located in the space. The goal of more sophisticated methods is to identify more precisely the set of voxels which are useful for the surface reconstruction and to store the information just in these voxels. As examples we can mention *truncated distance fields* [4] and *adaptively sampled distance fields — ADF* [6].

The main idea of truncated DFs is that we need just a thin layer of voxels in the surface vicinity for coding of the surface. Inside this thin layer (called transition area) the density changes linearly with respect to the distance from the surface and other voxels store constant inside or outside density. More formally, the density $d(x,y,z)$ is according to [4] defined by function:

$$d(x,y,z) = \begin{cases} 2T & \text{for } D(x,y,z) < -\delta \\ 0 & \text{for } D(x,y,z) > \delta \\ T\left(1 - \frac{D(x,y,z)}{\delta}\right) & \text{else} \end{cases}, \tag{1.1}$$

where $D(x,y,z)$ is the distance of the point $(x,y,z)$ from the surface (negative inside), $2\delta$ is the width of the transition area and $T$ is the threshold value defining the surface. It is important to choose the constant $\delta$ properly. Its size determines dimensions of the smallest detail that is representable. If this value is too high, small details are smoothed. On the other hand, too small $\delta$ leads to problems with surface reconstruction, in the limit case we actually get binary data. From results presented in [4] we know that the ideal value of $\delta$ is approximately 1.8 what is a little bit more than the length of the unit cube diagonal. It comes from the fact that for the surface reconstruction we need to interpolate values of nearest eight voxels lying in vertices of the cube inside which the surface point is located, so it is desirable to have all these eight voxels in the transition area. In the situation where the surface point lies near a vertex of the cube, the distance from the opposite vertex is equal just to the length of the above mentioned unit cube diagonal ($\sqrt{3}$).

Technique called adaptively sampled distance fields enables to represent efficiently especially scenes where large smooth surfaces are combined with small details. Using homogeneous sampling of the scene we need extremely high resolution for the sake of representation of some details also if they occupy only a fraction of the space. ADF solve this problem using a hierarchical subdivision of the space. In the comparison with other systems where the recursive

subdivision runs on the base of space blocks categorization into inside, outside and transition, ADF control the construction of octree by taking local curvature of the surface into account. The result is that the sampling is around details denser than in homogeneous areas, so the memory is employed reasonable.

DFs have been implemented successfully in various kinds of applications, as for example in robotics (motion planning, swept volumes), volume rendering, offset surfaces, morphing and sculpting systems [7].

## 1.3 Surface Normal Estimation

To make a high-quality visualization of volumetric data we have to reconstruct from the discrete information in the grid not only the surface, but also the surface normal for simulation of various light effects (shading, reflection, refraction, ...). The most common method for the surface normal estimation is the evaluation of density gradient using *central differences* and following normalization of this vector:

$$\begin{aligned} g^x_{i,j,k} &= d_{i+1,j,k} - d_{i-1,j,k} \\ g^y_{i,j,k} &= d_{i,j+1,k} - d_{i,j-1,k} \\ g^z_{i,j,k} &= d_{i,j,k+1} - d_{i,j,k-1} \qquad (1.2) \\ n_{i,j,k} &= \frac{g_{i,j,k}}{\|g_{i,j,k}\|} \, . \qquad (1.3) \end{aligned}$$

Some authors have referred to the fact that this filter blurs details in the picture. So, there has been proposed an *adaptive technique* [8] which takes the maximum of forward and backward difference into account:

$$g^x_{i,j,k} = \max(d_{i+1,j,k} - d_{i,j,k},\ d_{i,j,k} - d_{i-1,j,k}) \, . \qquad (1.4)$$

An other approach was chosen by Goss [9] who developed an adjustable filter based on the truncation of the ideal gradient filter $\frac{\cos \pi x}{x}$ using *Kaiser window*. The sensitivity for higher frequencies can be set by parameter $\alpha$ which modifies the width of the Kaiser window.

Reconstruction filters for the gradient estimation are usually used without specific requirements on the properties of volume data (except of the condition for bounded band). If we voxelize geometric objects, we can exploit the advantage that we have properties of our volume data under control. In particular, we can choose the voxelization filter consistent with the reconstruction one to get the best results. The advantage of the filter that uses central differences is a simple implementation and low computation demands. Šrámek and Kaufman showed in [4] that using this filter we obtain the true surface normal just in the case where the density changes linearly with respect to the distance from the surface. It means that central differences give us the best result just in the combination with DFs.

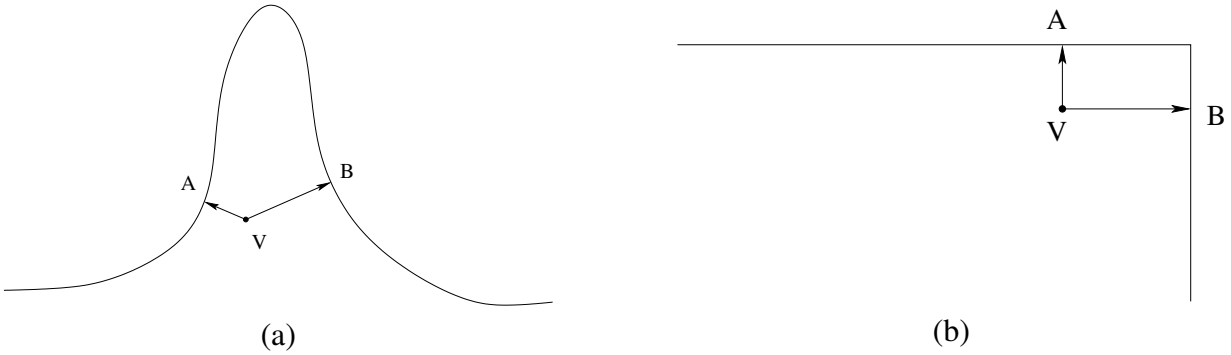



Figure 1.2: *Reconstruction problems*
*(a) detail, (b) edge*

## 1.4 Object Representability

### 1.4.1 Reconstruction Errors

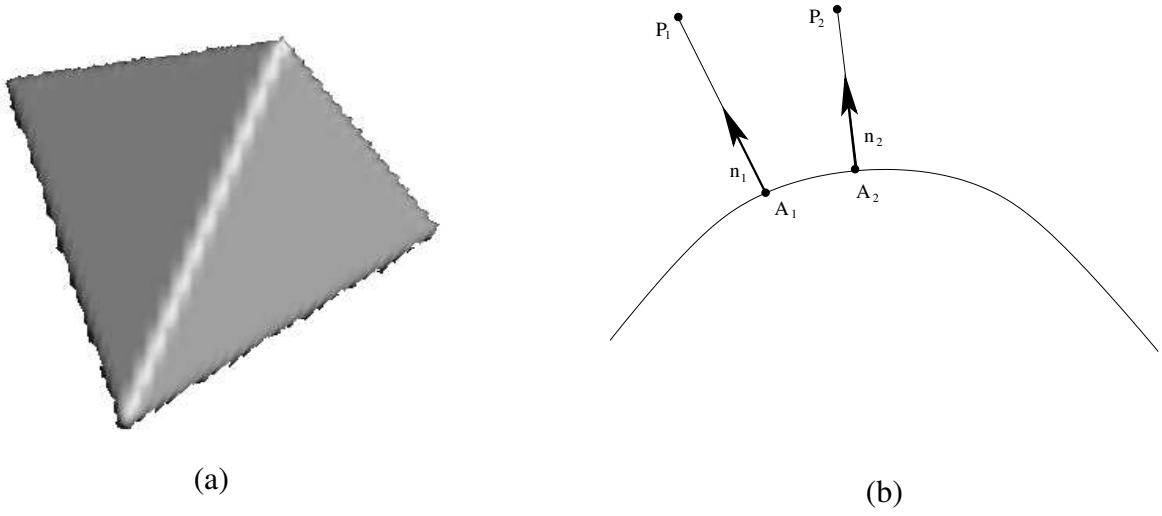
Voxelization is a conversion from continuous data to the discrete one, so it is necessarily followed by some information loss. By reconstruction there is an effort to get the original data, but in fact we are able to obtain only certain approximation of it. Quality of this approximation depends on the properties of given objects and on the sampling density. As we try to capture objects for which the representation is not suitable, some errors arise during the visualization in the form of various artifacts.

We have to realize that a voxel carries information about distance just from one (the nearest) surface. If there are more surfaces in the voxel vicinity, then we are not able to capture this fact in a voxel with just one value of density, so of course we do not avoid some reconstruction errors. The situation is illustrated in Figure 1.2. For coding of the surface around points $A$ and $B$ we need the information in the voxel $V$. However, we can store the correct information about only one of points $A$, $B$ — for instance $A$. When we use this information from voxel $V$ for reconstruction of the surface around the point $B$, it is obvious that we obtain incorrect results. In the case of a small detail coding the problem can be overcome by increase of the grid resolution (voxel $V$ would not be "in the vicinity" of both surfaces any more). Around an edge the problem is more principal — problematic voxels exist for arbitrarily high resolution (it leads to artifacts as in Figure 1.3 (a)). Although, using high enough resolution this reconstruction error can be invisible in the picture, nothing changes in the fact that angular objects are not correctly representable in this way.

The reconstruction error also arises if all voxels store correct information about the surface. The magnitude of this error depends on the interpolation method and on the surface curvature. From Figure 1.3 (b) according to [5] can be derived the dependence for density $d$ in the point $P_2$:

$$d(P_2) = (P_2 - A_1) \cdot \vec{n_1} + (P_2 - A_2) \cdot (\vec{n_2} - \vec{n_1}) - (A_2 - A_1) \cdot \vec{n_1} \ , \tag{1.5}$$

where $A_1$, $A_2$ are respectively the nearest points of the surface from points $P_1$, $P_2$ and $\vec{n_1}$, $\vec{n_2}$ are surface normals in these points. The first term in the equation constitutes a linear component




(a) (b)

Figure 1.3: *Reconstruction errors*
*(a) artifacts around edges, (b) illustration to the error estimation*

of the three dimensional DF, the remaining terms are non-linear. The bigger is the local surface curvature, the stronger is the weight of non-linear terms. In the case of locally plane surface these terms are zero, otherwise they are very small in the surface vicinity but dominate in a bigger distance from the surface. As we use linear reconstruction filters, we can derive the optimal sampling for given object from this equation, if we define the maximal acceptable error of the surface reconstruction. We have to be aware that the curvature is a quantity inversely proportional to the chosen unit of length which is in our case determined by the distance between two neighbouring voxels. By increasing resolution we get a less curvature for given surface, so we get a less reconstruction error.

Authors of the paper [4] made a set of experiments to compare various voxelization filters from the point of view of error that occurs during reconstruction of the data which was acquired by filtered voxelization. The exact position of the surface point is obtained by thresholding at the middle value of density which we get in the continuous space using trilinear interpolation of values from eight voxels in the current *cell* (cell is the smallest cube with vertices in voxels). Experiments showed that the usage of Barlet filter causes a big systematic error — the surface is moved inside the object. Distance techniques seem to be more suitable, because they do not induce such a movement.

The goal of the next experiment in [4] was to examine the error of surface normal estimation. The conclusion of the analysis says that the usage of Gaussian produces about 50 times bigger error than the usage of filter oriented box. From the observation it is also obvious that the most proper method for the normal computation are central differences.

The filtration of objects using oriented box leads to an identical representation as the technique of DFs. All the mentioned facts imply our decision to use DFs together with the method

of central differences in this thesis.

### 1.4.2 Representable Objects

As it was mentioned before, some objects are not correctly representable using DFs. The paper [10] discusses this problem more in detail. There have been set two conditions which an object has to fulfill to be suitable for voxelization:

**Condition 1:** The curvature of the surface has to be relatively small with respect to the grid resolution.

**Condition 2:** Filters for surface and normal reconstruction can not use samples located on both sides of the *medial surface*. The medial surface is defined as a set of points $P$ for which there exist at least two surface points in the distance $d$ from $P$, where $d$ is the distance of point $P$ from the surface. In other words: there exists more than one nearest surface point for the point $P$.

The reason for the first condition is obvious from the observation in previous pages — the bigger is the curvature, the bigger is the reconstruction error. Also the second condition was explained already in an informal way — it is implied by problems illustrated in Figure 1.2.

Taken these two conditions into account, authors of the paper [10] made the result that a geometric object $X$ is suitable for the voxelization with given resolution, if $X$ is both $S_r$-*open* and $S_r$-*closed*, where $r > \sqrt{6}$ is chosen in a way to make the reconstruction error acceptable for the particular application. Here, we have to be aware that the choice of $r$ determines also the resolution, because $r$ is expressed in voxel units. The criterion of openness and closeness can be said in other words as the property of a surface around which it is possible to round a sphere of radius $r$ from both its sides.

The problem of objects representability touches also the realization of CSG operations with volume data. Whereas CSG operations with binary data can be implemented very easily using block operations, the situation with DFs is more complicated. The main problem is caused by the fact that the CSG result has some edges in general, so it does not fulfill the criterion of openness and closeness. Thus, our goal is to create the CSG result as an object which does fulfill this criterion and it is as near as possible to the ideal analytical result. Authors of [11] chose an approach where the CSG operation is performed regardless of potential edges on the result and then artifacts around edges are corrected using following *revoxelization*. This method has brought some visual enhancement, but there is no guarantee that the mentioned criterion has been fulfilled. It seems to be more correct to voxelize directly a representable object. In the paper [12] there was proposed following scheme: (a) reconstruct original solids from their volume representation; (b) perform the CSG operation in the continuous space; (c) modify the result to fulfill the criterion of openness and closeness; (d) get the final volume representation by its voxelization. There was presented an algorithm that does not proceed exactly according to this scheme, but it gives the same result.

The main idea is to use morphological operators erosion and dilation to make the CSG result representable. In volumetric intersection, for example, in order to enforce the condition

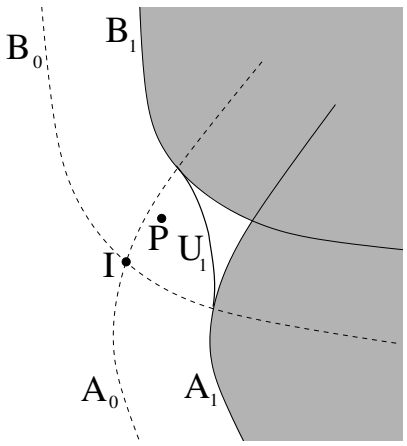



Figure 1.4: *The problematic area during construction of the union of two objects*

of representability, both objects are $S_r$-eroded first and the results is obtained as $S_r$-dilation of intersection of such eroded objects:

$$\mathsf{A} \cap \mathsf{B} = ((\mathsf{A} \ominus S_r) \cap (\mathsf{B} \ominus S_r)) \oplus S_r \ . \tag{1.6}$$

The algorithm works with one solid in the grid and one object defined analytically in the continuous space. The second object is used to modify the given volume by means of CSG operation. The process runs in two phases where we traverse all voxels in each of them. We describe this method for the realization of union of two objects (intersection and subtraction can be done analogically). As we create the DFs of the union, we need to know just the distance from both objects for the most of voxels and we get the resulting value as the maximum of these two distances (or minimum — it depends on the sign convention) — we name this approach *minmax criterion*. An other situation is in the vicinity of surfaces intersection where we need to compute the distance in a more complicated way. This is illustrated in Figure 1.4. Objects $A$, $B$ have the outside surface respectively $A_0$, $B_0$ and the inside surface $A_1$, $B_1$ (the density changes linearly between these surfaces and it is constant elsewhere). Intersection of outside surfaces is the set $I$, a space curve in general. Our aim is to create the union in a way to make the inside surface $U_1$. There is a problematic (*inconsistent*) point P laying in the vicinity of $I$. In the first phase, the set $I$ is constructed and inconsistent voxels are detected. In the second phase, values of voxels are computed where we use the distance from $I$ for inconsistent voxels and minmax criterion for the others.

One goal of this thesis is to propose a method for performing of CSG operations with two volumes (with no information about their analytical description), just using one run through all voxels without the necessity of the set $I$ construction.

# Chapter 2

# Representation of Volume Data Sets

## 2.1 RL Compression

Among the most cited drawbacks of the volume graphics belong its huge memory requirements. Although, the continual progress in the hardware development reduces this difficulty, it is obvious that we have always not enough memory, because our demands on the technique increase simultaneously with its growing power. This is a strong impulse for proposing methods to decrease memory requirements of volumetric representation.

In the introductory chapter we mentioned the technique called adaptive distance fields [6] which uses a hierarchical subdivision of the space. The general disadvantage of hierarchical stored data sets lies in the fact that manipulation with them is quite complicated. In the following pages, we present a new method which chooses an other approach. The core of this method is well-known compression *run-length (RL) encoding* that has been modified to enable effective representation of truncated distance fields.

Our implementation of the *RL compression* comes from the observation that the information about an object is stored just in voxels located in the surface vicinity. The typical row in the grid

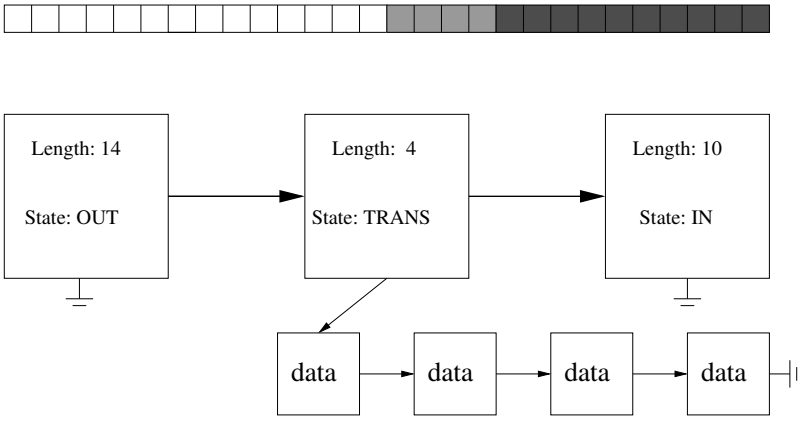



Figure 2.1: *Representation of a compressed row*

consists of several long sequences of outside or inside voxels separate with short sequences of transition voxels. Therefor, our idea is to divide the row into segments of three types: *outside*, *inside* and *transition*. For coding of an outside or inside segment we need to store just its length and flag (*OUT, IN*). In addition the transition segment also stores a pointer at a list of voxels which form the segment. The principle of this compression is illustrated by Figure 2.1. Whole the grid is stored as two dimensional array of compressed rows. A similar method for representation of a volume was used by authors of papers [13, 14], but not for distance fields.

## 2.2 Types of Voxels

The original technique of truncated distance fields makes do with voxel which stores just density given by its distance from the surface. During visualization, when we need to know the gradient of density for normal estimation, we usually evaluate it using values of densities stored in neighbouring voxels (most often by above mentioned central differences). In the case when we voxelize geometric objects where we know the gradient from their analytical description, we have the occasion to store the gradient in voxel together with the density. So we save some rendering time and we can achieve higher precision, because we use the true gradient instead of the estimated one. Of course, we need more memory, but the increase of memory requirements is not such critical due to the RL compression.

Strictly speaking, we do not need to remember the gradient, but only its direction which is obtained by normalization of given vector. The vector in three dimensional space is usually represented by three coordinates $(x, y, z)$, but for direction storage we need just two numbers $(\phi, \psi)$, which can be computed for example using sphere parametrization:

$$\begin{aligned} x &= \cos\phi \, \cos\psi \\ y &= \sin\phi \, \cos\psi \\ z &= \sin\psi \, . \end{aligned} \tag{2.1}$$

So we save some memory, but on the other hand the conversion to spherical coordinates and the inverse conversion will need some extra processing time.

The next important question is, how many bytes to use for density and gradient storage. For the testing purposes we have implemented all the combinations of one-, two-, and four-bytes density with one, two, and four bytes for each component of the gradient. There are three classes of voxels:

**vxtPlainVoxel:** Just density is stored.

**vxtGradVoxel:** Both density and gradient (three components) are stored.

**vxtSphGradVoxel:** Both density and gradient are stored. The gradient is stored using spherical coordinates (two components).

In the first case there are 3 possibilities and in the other cases there are 9 possibilities according to different precision of the density and gradient storage. Altogether we have 21 different kinds

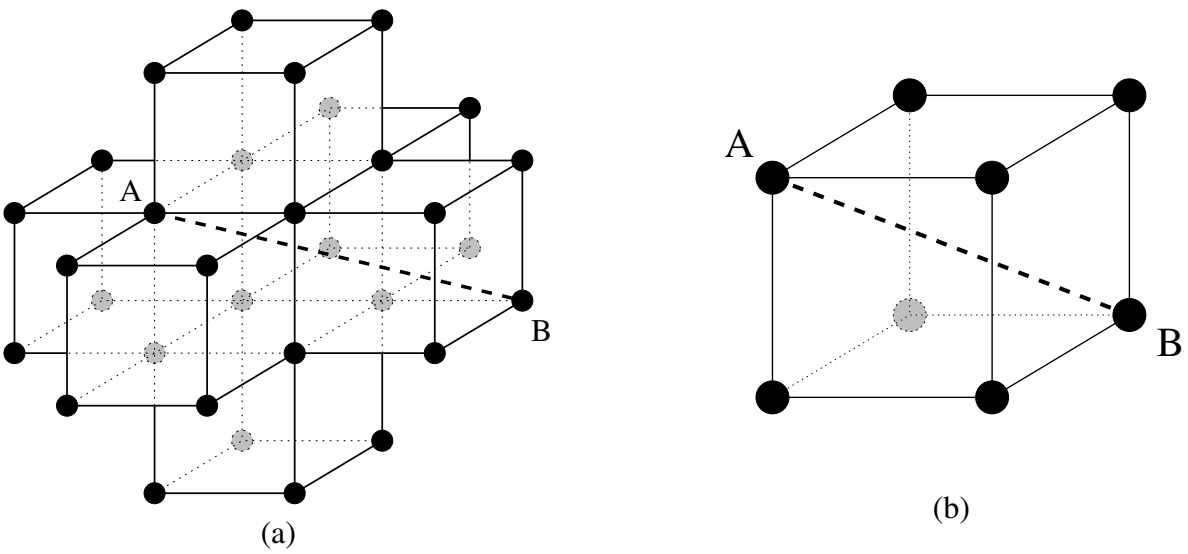



Figure 2.2: *Configuration of voxels needed for the normal estimation (a) voxel without gradient, (b) voxel with gradient*

of voxels. Indeed, some of them are not very meaningful, for instance the combination of very precise gradient with inaccurate density. For the sake of completeness we have to remark that some voxels take more memory in the real implementation than it is the theoretically evaluated value. It is because the size of complex data structures is aligned to a multiple of two or four bytes.

The selection of voxel type hangs together with the selection of the width of the transition area. For the surface normal estimation we need wider neighbourhood of a point for voxels without gradient than for voxels with gradient. The configuration of voxels needed for the gradient estimation is depicted in Figure 2.2. If the surface point is located in the vicinity of voxel $A$, then the furthermost needed voxel ($B$) lies in the distance $\sqrt{6}$ for voxels without gradient, otherwise it lies in the distance $\sqrt{3}$. These numbers determine the radius of the transition area. The value $\sqrt{6}$ is not given very strictly. If the radius is slightly shorter, the error does not grow dramatically, because voxels on the border of the transition area have low weight in the computation. Nevertheless, we use the values theoretically derived: $\sqrt{6}$ for the `vxtPlainVoxel`, and $\sqrt{3}$ for the `vxtGradVoxel` and the `vxtSphGradVoxel`.

## 2.3 Voxelization

The RL compression has been tested for implicit surfaces. We have proposed two different methods for their voxelization. The first one (taken from [15]) is based on the space subdivision using homogeneity check, the second one fills the grid slice by slice and uses an efficient run

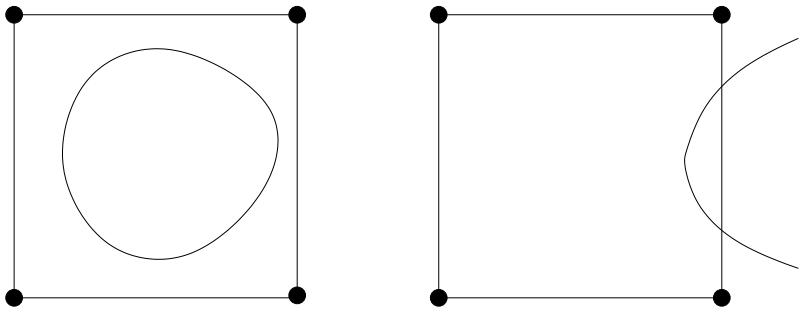


Figure 2.3: *Shortcoming of the homogeneity check*

through the row. The distance $d(P)$ of a point $P$ from the surface is computed by

$$d(P) = \frac{f(P)}{\| \bigtriangledown f(P) \|} , \tag{2.2}$$

where $f$ is the function defining the implicit surface $f(P) = 0$ and $\| \bigtriangledown f(P) \|$ is the gradient magnitude. This linear approximation is precise enough in the surface vicinity, if we assume that the gradient magnitude does not change radically.

The first step of the space subdivision method is to divide the volume into small blocks of voxels (for example $8 \times 8 \times 8$). In each vertex of the block densities are evaluated. If all the values are identical, that block does not cross the transition area, so we fill all its voxels with the same value of density. Otherwise, the block is subdivided recursively, until the homogeneity check passes or the block has dimensions $2 \times 2 \times 2$. Of course, the homogeneity check is not fully ideal. Figure 2.3 illustrates some situations, where this check fails. It is in the case, when the surface crosses just the margin of the block or when whole the object lies inside the block. So, the size of the initial block has to be chosen appropriately with respect to expected properties of objects, which are going to be voxelized.

The core of the second method is voxelization of one row. We move along the row in sequence and fill visited voxels with values of density. If current voxel lies in the transition area, the neighbouring voxel is visited in the next step. Otherwise, we try to estimate its distance from the surface and skip appropriate number of voxels. The problematic point of the algorithm is the distance estimation, because the formula (2.2) is precise enough just in the surface vicinity. Fortunately, we need to get just a crude guess of the distance. If we skip too far (the new voxel has different density than the previous one), we correct the skip to the half of its original length.

The voxelization of a row is a part of more complex process, during which the grid is filled using *sweeping planes*. The original idea was to determine during voxelization if the voxel in the transition area is really needed for the surface reconstruction (in Figure 2.4 there is an example of some useless voxels). As sweeping are called two neighbouring planes in the grid, which move step by step through whole the volume. This sweeping process consists of three phases:

1. Filling of the sweeping plane using row voxelization.

2. Marking of useful voxels.

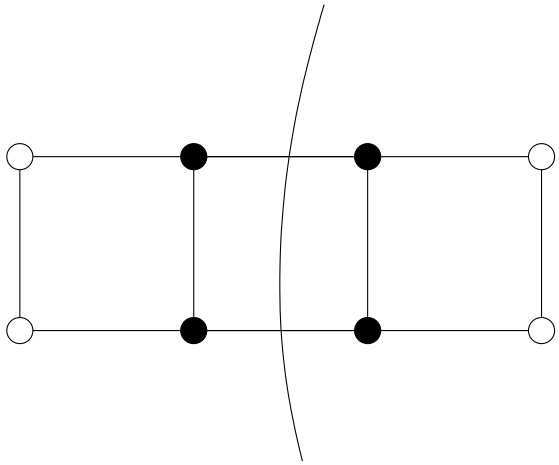


Figure 2.4: *An example of useless voxels in the transition area*
*White voxels lie approximately* 1.5 *(less than* $\sqrt{3}$*) VU from the surface, but they are not needed for its coding (in representation using* `vxtGradVoxel` *and* `vxtSphGradVoxel`*).*

3. Conversion of data from the sweeping plane to the grid.

During the second phase, each transition voxel is checked, if it is an component of at least one cell, which is crossed with the surface. That is the reason, why we need two sweeping planes.

The goal of this method was saving some memory by elimination of useless voxels. Unfortunately, as we have discovered later, to perform more sophisticated CSG operations we also need this "throwaway information". So, the second phase of this process has to be removed and we actually make do just with the efficient pass through rows.

Our observation has showed that both mentioned methods have similar time complexity. For testing purposes described in following pages, there was used the voxelization with sweeping planes.

## 2.4 Results

### 2.4.1 Reconstruction Precision

To test the precision of the surface and its normal reconstruction, we have used an experiment proposed in the paper [4]. There is a sphere voxelized in the grid. We shoot a ray from its centre and find the intersection point with the surface. In this point also the surface normal is estimated. Then we determine the difference between the true surface point and the evaluated one, and the angle between the true and the estimated surface normal. In this evaluation we use the trilinear interpolation of values from eight voxels of current cell. In the case of voxels without the gradient, the normal is estimated using central differences. We send a number of rays in different directions. We repeat this experiment for 125 ($5 \times 5 \times 5$) positions of the sphere centre distributed homogeneously inside the cell to minimize the influence of the choice of centre position on

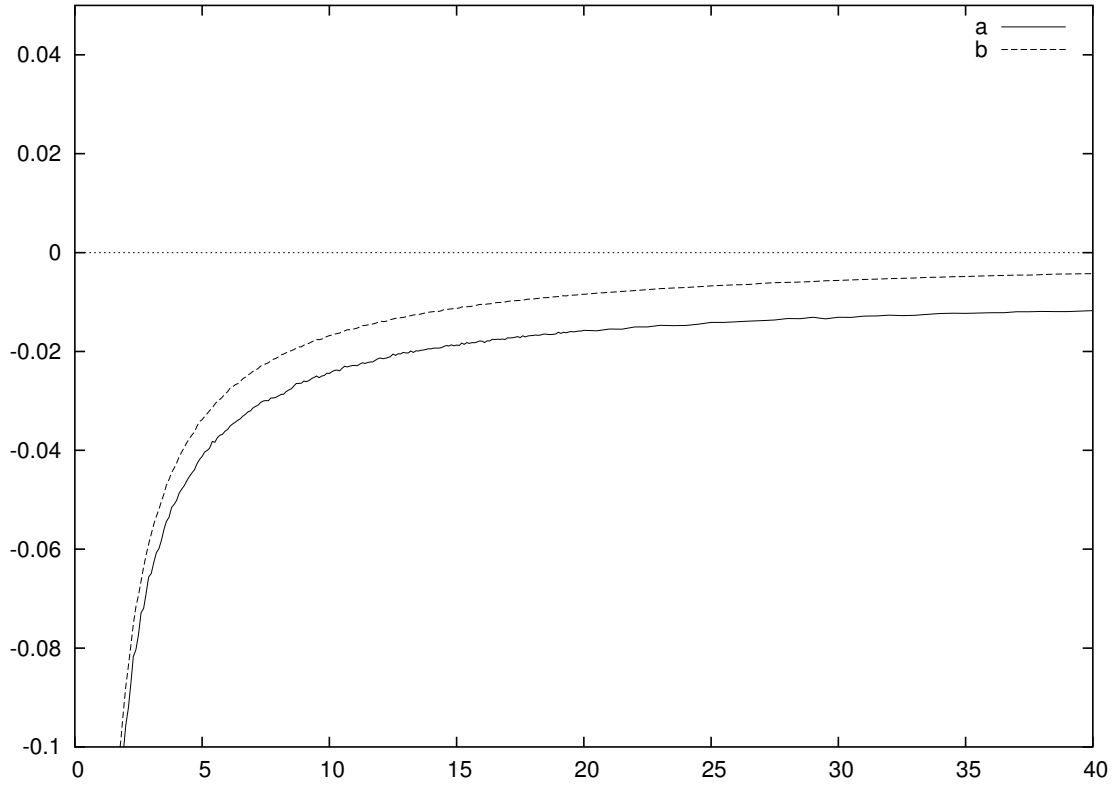



Figure 2.5: *Dependence of the error of surface point estimation on the surface curvature x-axis: radius of the sphere (in $VU$), y-axis: error of the estimation (in $VU$). Precision of voxels: (a) 1 byte for density storage, (b) 2 bytes for density storage.*

results. We examine, how the above mentioned errors depend on the surface curvature, so we change the sphere radius from $1\,VU$ to $40\,VU$.

Our experiment has showed that two bytes are fully enough for the density storage, because the four-bytes density do not bring higher reconstruction precision. Figure 2.5 illustrates comparison of errors for one- and two-bytes density. The reconstructed surface is moved inside the sphere against the true surface. For the radius of $40\,VU$, the error is approximately three times less for the two-bytes density than for the one-byte density.

Figure 2.6 illustrates the error distribution in different directions for a sphere with its centre in the middle of a cell. As we have expected the error pattern is symmetrical and gets more soft as the radius grows.

As we have mentioned above we can choose theoretically from 21 different possibilities, how to form a voxel. Although, it would be very complicated to test all of them and make results well-arranged. Fortunately, the normal estimation is not significantly influenced with the surface point estimation in our experiment. So, we can ignore the number of bytes for the density storage, when we test the dependence of the normal estimation error from the precision of the gradient storage. Results of this experiment have showed that for the gradient storage two bytes

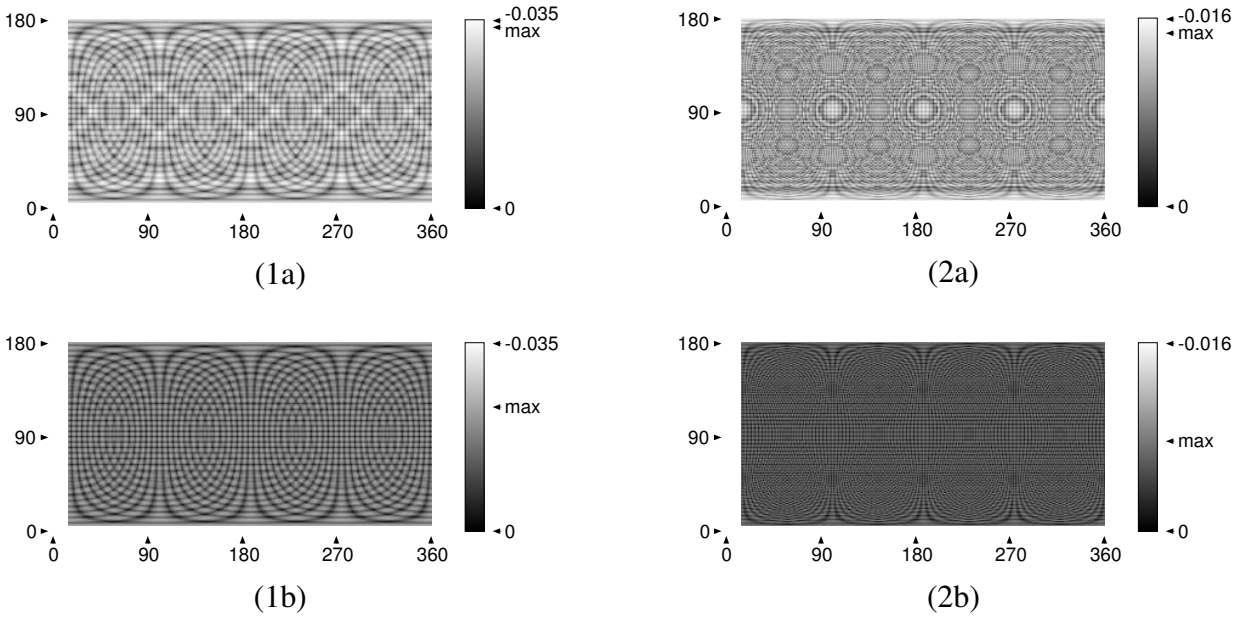
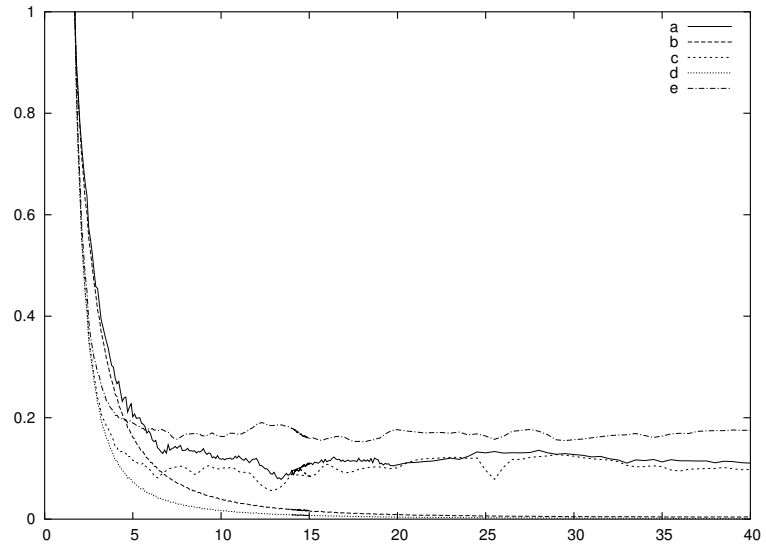



Figure 2.6: *Distribution of the error of surface point position*
*Radius of the sphere: (1) R=11VU, (2) R=33VU. Precision of voxels: (a) 1 byte for density storage, (b) 2 bytes for density storage. The intensity indicates the error of surface point position in given direction, where the direction is represented in spherical coordinates.*

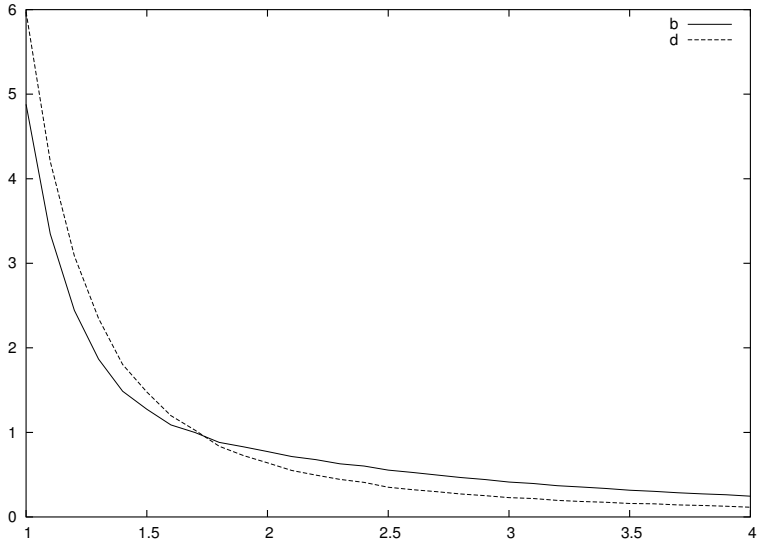
for each component are quite enough, because four-bytes gradient storage does not bring higher precision, likewise in the case of the density storage. The reason is that the error is influenced much more with the space discretization than with the variable quantization.

Figure 2.7 (1) depicts the dependence of surface normal error from the surface curvature for five chosen types of voxels. We observe that the error is significantly less for two-bytes storage of density and gradient than for the one-byte storage, for both voxels with gradient and without it. Figure 2.7 (2) shows that we get higher precision using voxels with stored gradient. The precision is two times higher for the sphere of radius $4VU$ and four times higher for the sphere of radius $40VU$. Next interesting observation concerns the voxel with one-byte compressed gradient. This voxel achieves worse results than the voxel without gradient. The explanation is obvious from Figure 2.8, where we can see that the error is concentrated around the *xy* plane, whereas it is minimal around the *z*-axis. It is caused by the spherical coordinates which do not cover the surface of a sphere homogeneously, so we can handle vertical direction more precisely than the horizontal one. This phenomenon is not presented in the case of two- or four-bytes gradient storage — there are practically the same results for compressed and uncompressed gradient.

The conclusion of this analysis is that with regard to the reconstruction precision and the memory requirements the best choice of voxels is that one with two-bytes density and two-bytes gradient represented in spherical coordinates.




(1)




(2)

Figure 2.7: *Dependence of the surface normal error on the surface curvature*
*x-axis: radius of the sphere (in VU), y-axis: angle between true and estimated normal (in degrees). Precision of voxels: voxel without gradient with one- and two-bytes precision of the density storage (a, b); voxel with uncompressed gradient (3 components) with one- and two-bytes precision of the density and gradient storage (c, d); voxel with compressed gradient (2 components) with one-byte precision (e). In the bottom (2) the same dependence is depicted in detail for voxels with two-bytes precision (b, d).*

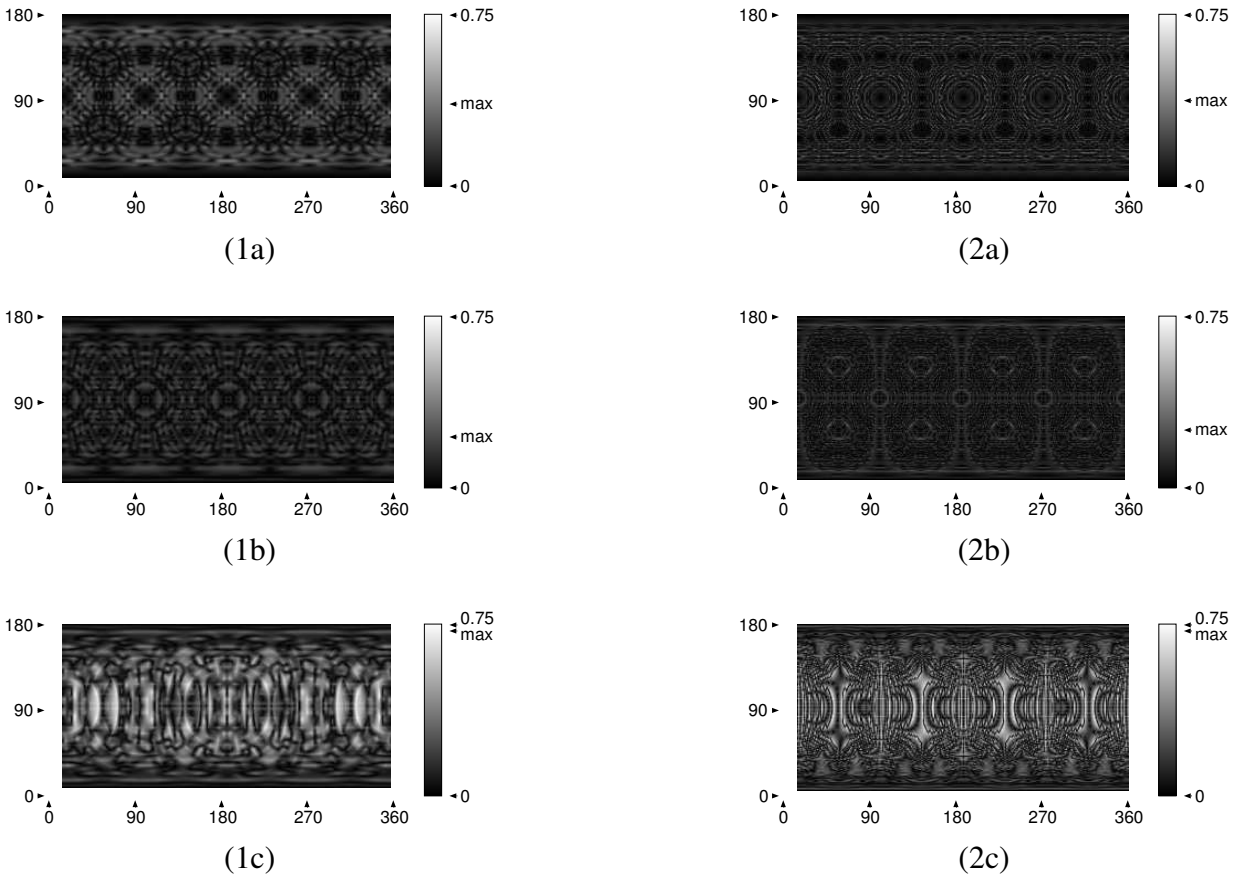



Figure 2.8: *Distribution of the error of surface normal estimation for voxels with one-byte precision of density and gradient storage*
*Sphere radius: (1) R=11VU, (2) R=33VU. Voxel types: (a) without gradient, (b) with uncompressed gradient, (c) with compressed gradient. The intensity indicates the error of surface normal estimation (in degrees) in given direction, where the direction is represented in spherical coordinates.*

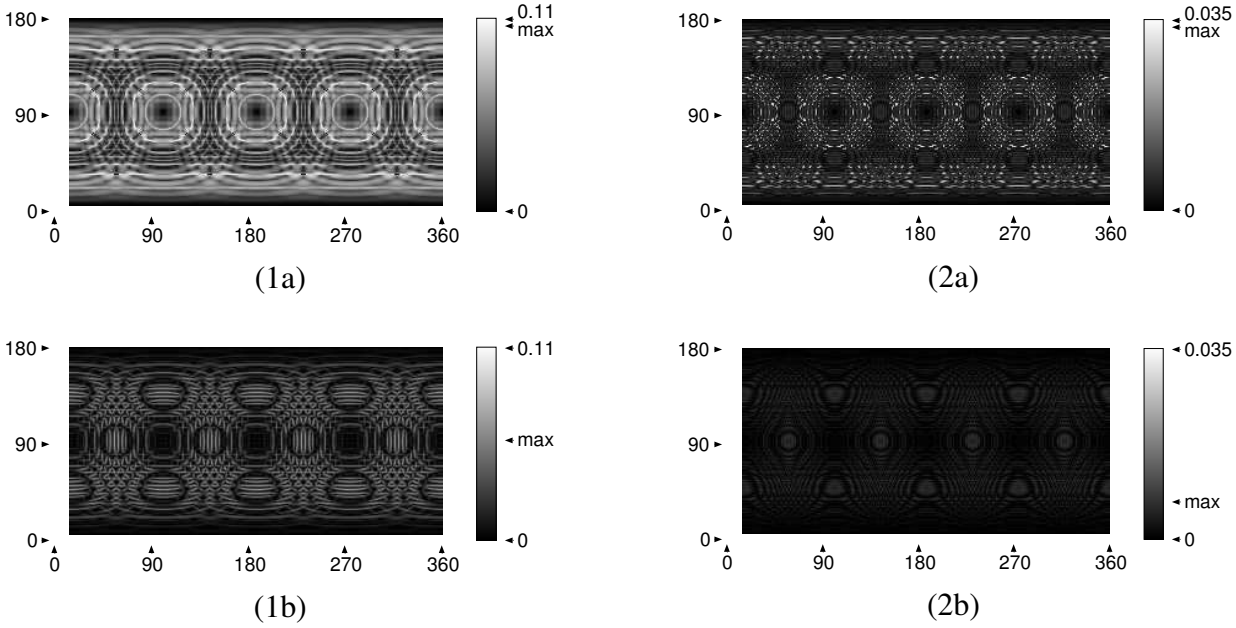



Figure 2.9: *Distribution of the error of surface normal estimation for voxels with two-byte precision of density and gradient storage*
*Sphere radius: (1) R=11VU, (2) R=33VU. Voxel types: (a) without gradient, (b) with gradient. The intensity indicates the error of surface normal estimation (in degrees) in given direction, where the direction is represented in spherical coordinates.*

### 2.4.2 Memory Requirements

The memory complexity of the RL compression can be estimated theoretically using a simple thought. It is given by the number $v$ of voxels in the transition area and the number $s$ of row segments. Number $v$ depends linearly on the surface area and $s$ can be computed as $r+2p$, where $r$ is the number of rows in the grid and $p$ is the number of intersections between the surface and rows, because each intersection adds two segments into row. Also $p$ is bounded above by the surface area and $r = n^2$, so for the memory complexity $S_{RL}$ we have

$$S_{RL}(k,n) = \theta(n^2 + kn^2) \ , \tag{2.3}$$

where factor $k$ characterizes the surface area and $n$ is resolution of the grid (we suppose dimensions $n \times n \times n$). So, for given scene the size of used memory depends quadraticly on the grid resolution in the comparison with an uncompressed volume, where this dependence is cubic. Of course, RL compression is suitable for scenes with the factor $k$ small enough.

Our experiments are consistent with the theoretical conclusions. We have tested different objects: empty volume, cube, sphere, union of a sphere with a cube, regular tetrahedron, regular octahedron, *onion* — implicit surface defined by function

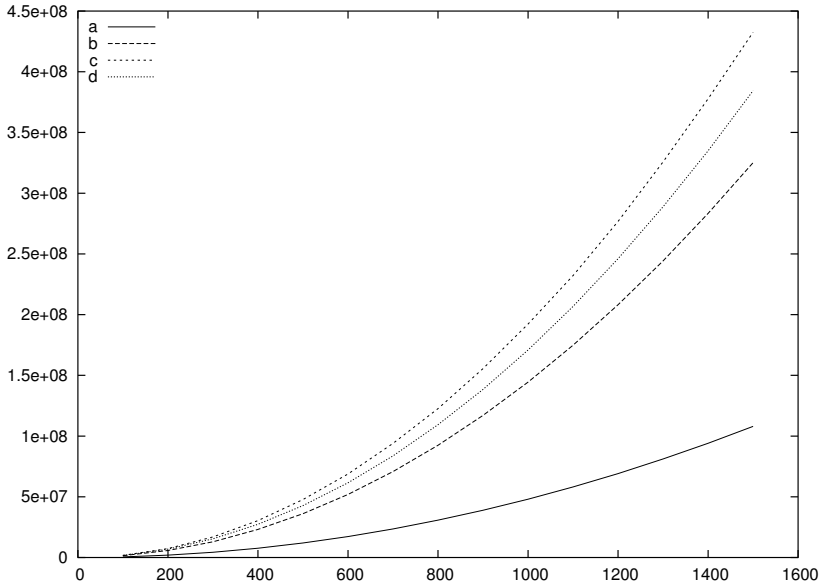
$$f(x,y,z) = \sqrt{y^2+z^2} - 0,4\ \frac{\cos 2\pi x\ +\ 1}{2} \cdot \left( 0,3 \cdot \left| \cos\left( 10\pi x + 4\ \mathrm{arctg}\ \frac{z}{y + \frac{1}{1000}} \right) \right| + 0,7 \right) , \tag{2.4}$$

*supersphere* — given by function

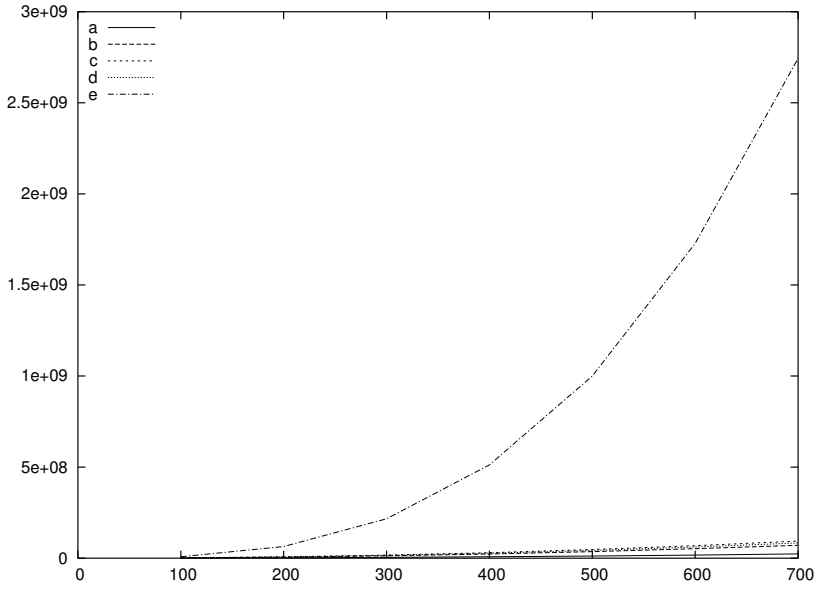
$$f(x,y,z) = \left( |x|^{\frac{2}{p}} + |y|^{\frac{2}{p}} \right)^{\frac{p}{q}} + |z|^{\frac{2}{q}} - r^{\frac{2}{q}} \tag{2.5}$$

(for $p = 0,3$; $q = 0,7$; $r = 0,5$), *sphereflake* — union of 91 spheres of different sizes (see Figure 2.12). All objects have very similar dependence of needed memory on the grid resolution, some of them are depicted in Figure 2.10 (1). The same dependence is illustrated for a better demonstration in Figure 2.10 (2) with the addition of uncompressed volume. As we can see in the next figure (2.11), for resolution $1500^3$ using RL compression, we need only up to 2% of memory used by uncompressed volume (for all the tested objects).

Figure 2.13 illustrates the dependence of memory size on the voxel type. Although, the voxel with gradient takes four times more memory than the voxel without gradient, the memory requirements for whole the volume are only up to two times higher (for tested objects with small factor $k$). The reason is that a part of memory serves for the representation of row segments and pointers between voxels — size of this memory does not depend on the voxel size. So, the difference between voxel with the uncompressed gradient and the compressed one is not very significant.




(1)




(2)

Figure 2.10: *Dependence of the memory size on the grid resolution for different objects 1: (a) empty volume, (b) sphere, (c) onion, (d) union of a sphere with a cube. The same dependence in the comparison with uncompressed volume (e). Voxel stores two-bytes density and two-bytes gradient (uncompressed).*

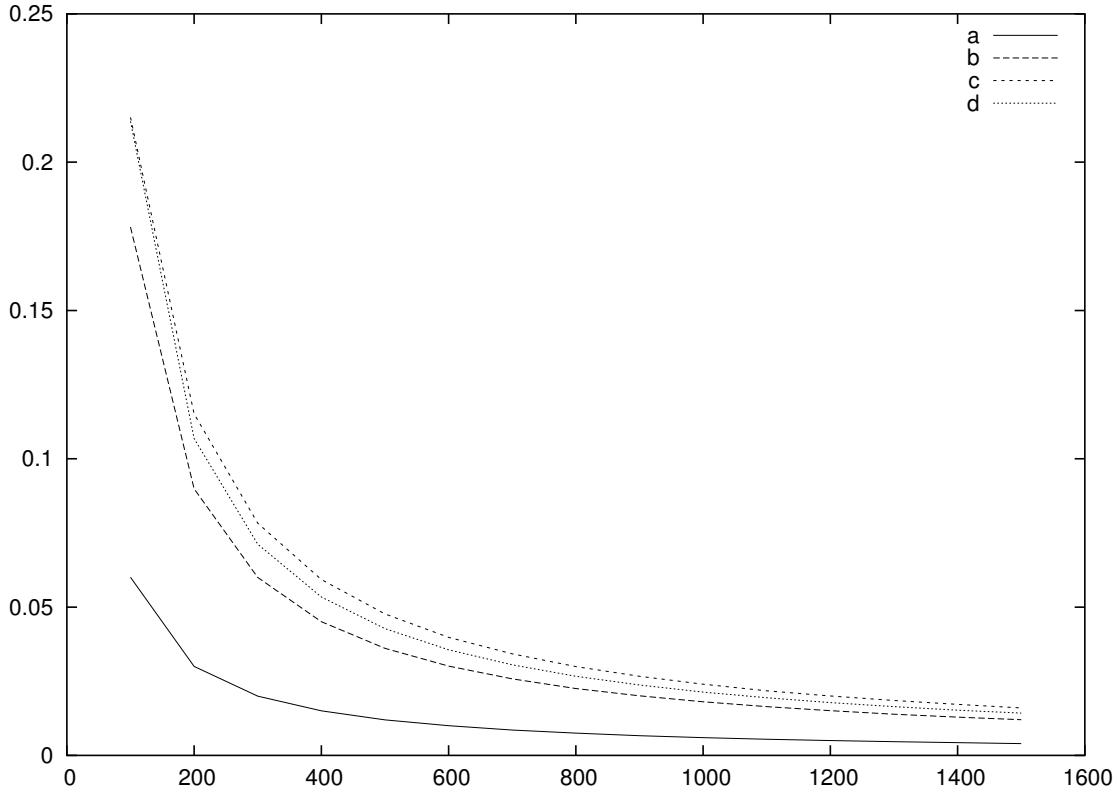
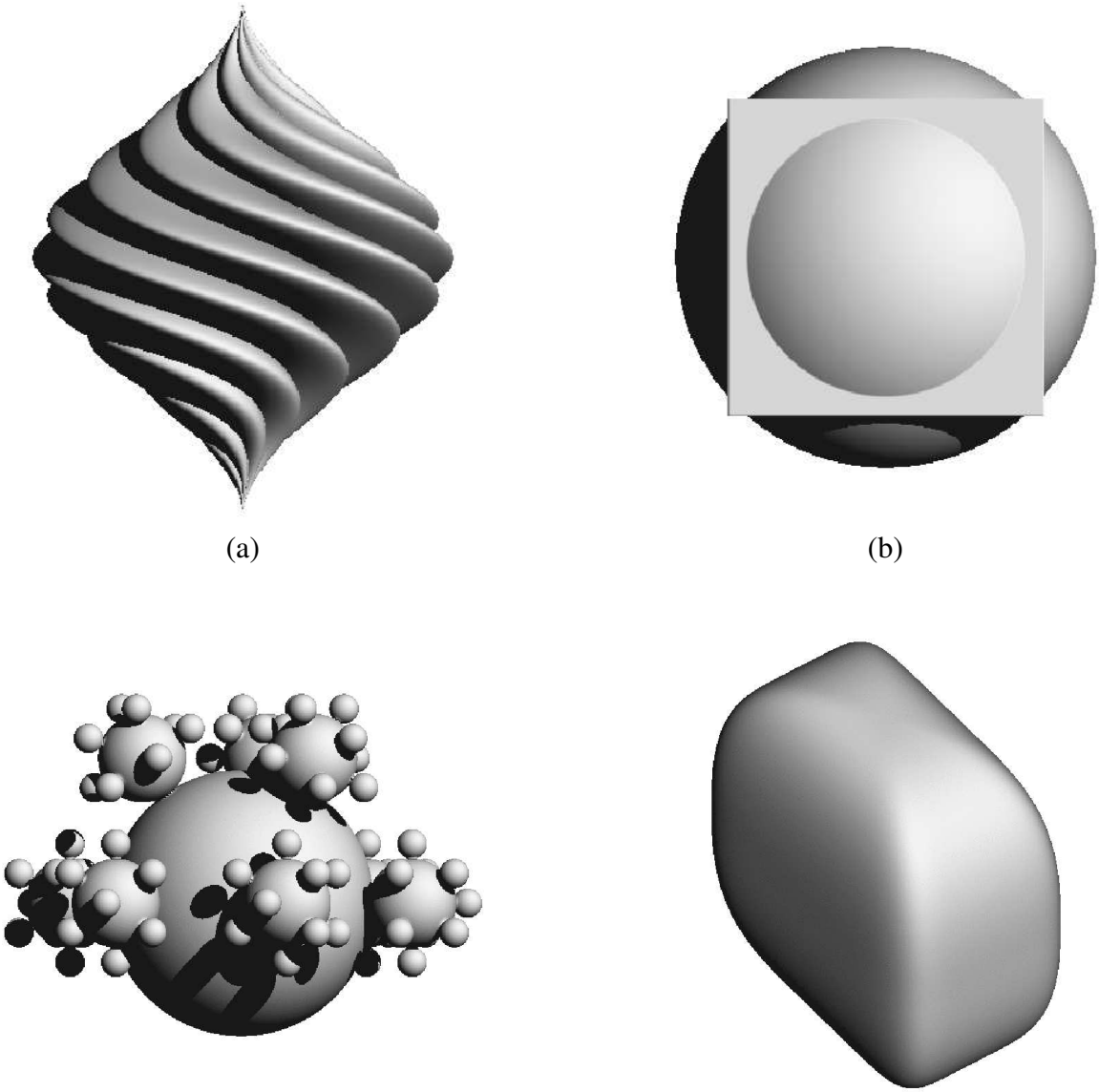


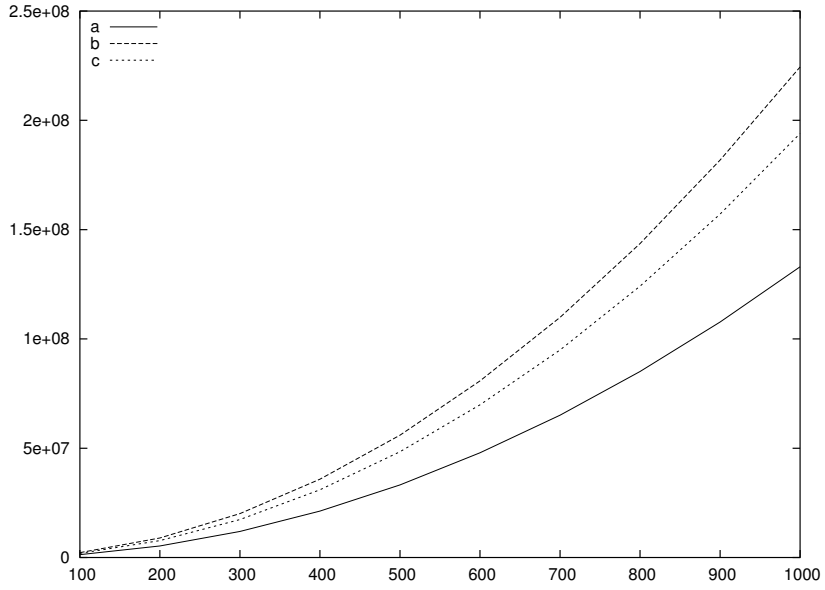

Figure 2.11: *Relative memory costs for different objects*
*x-axis: grid resolution, y-axis: ratio R between the size of compressed C and uncompressed U volume,* $R = \frac{C}{U}$*. Objects: (a) empty volume, (b) sphere, (c) onion, (d) union of a sphere with a cube.*



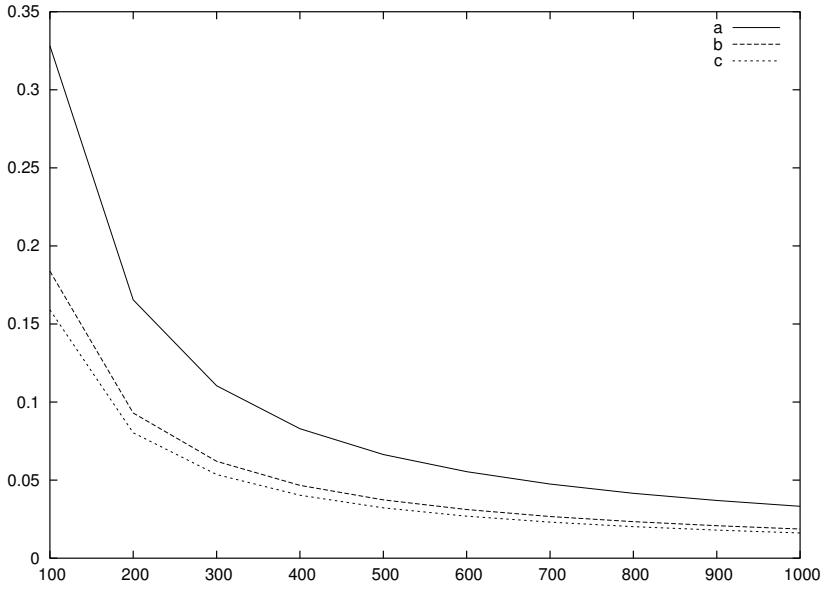
(a) (b)

(c) (d)

Figure 2.12: *Examples of some voxelized objects*
*(a) onion, (b) union of a sphere with a cube, (c) sphereflake, (d) supersphere*

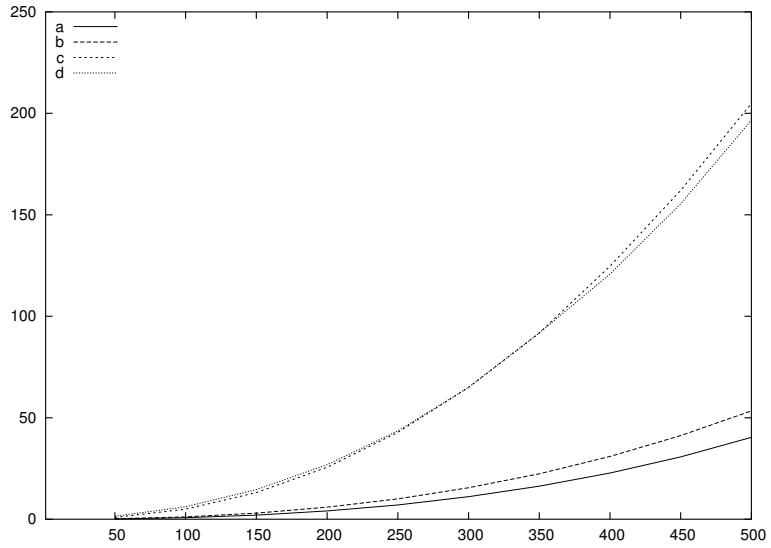



(1)




(2)

Figure 2.13: *Dependence of the memory size on the grid resolution for different voxel types (1) absolute memory costs, (2) relative memory costs in the comparison with uncompressed volume. Voxel types (four-bytes precision): (a) voxel without gradient, (b) voxel with uncompressed gradient, (c) voxel with compressed gradient. Tested object: sphere.*




(1)

Figure 2.14: *Dependence of the voxelization time on the grid resolution for different objects Time is expressed in seconds. Objects: (a) regular octahedron, (b) sphere, (c) onion, (d) supersphere.*

### 2.4.3 Voxelization Time

The voxelization time can be divided into two components:

1. time for evaluation of the implicit function
2. time for writing the information to the grid

The first component does not depend on the representation of the volume, but just on the complexity of the implicit function. As we can see in Figure 2.14, objects defined by more complex function are voxelized much more slowly than simple objects.

If we voxelize simple objects, the second component gets more relevant. The drawback of RL compression is that we do not have a direct access to voxels — the speed of access depends on the surface complexity. On the other hand we can exploit the RL compression for rapid filling of homogeneous areas — the whole outside or inside segment can be write at once. As we can see in Figure 2.15, the second effect is stronger, thanks to that the voxelization is faster with the RL compression than without it. Furthermore, we can notice that the saving of gradient causes just a small time delay if the RL compression is used.

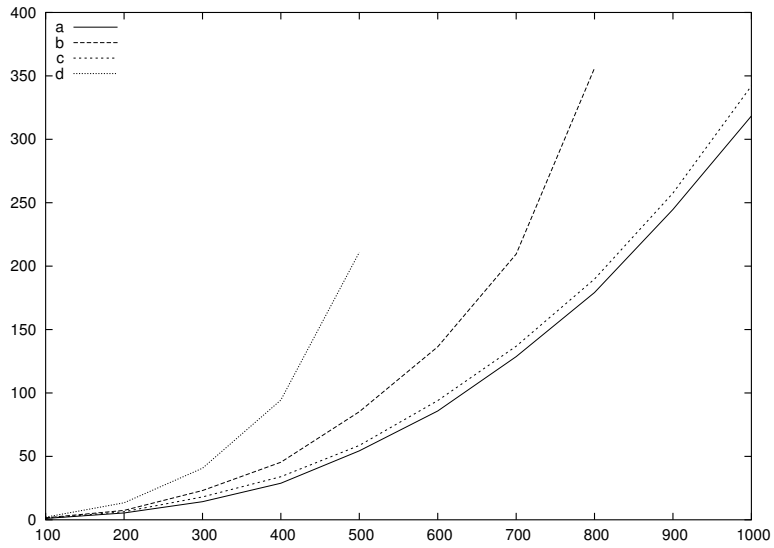



Figure 2.15: *Dependence of the voxelization time on the grid resolution for compressed and uncompressed volume*
*Time is expressed in seconds. Volume types: (a) compressed without gradient, (b) uncompressed without gradient, (c) compressed with gradient, (d) uncompressed with gradient. Tested object: sphere.*

# Chapter 3

# CSG Operations with Voxelized Solids

CSG operations between binary voxelized models can be easily performed at the voxel level as operations between the values of the corresponding voxels [1]. Unfortunately, this concept does not directly extend to filtered or DFs representations. In spite of that, the early techniques relied only on such binary operations between voxel densities. For example, the *minmax* implementation in the case of DFs models:

$$\begin{aligned} \mathsf{A}\cap\mathsf{B} &: d_{\mathsf{A}\cap\mathsf{B}} = min(d_{\mathsf{A}}, d_{\mathsf{B}}) \\ \mathsf{A}\cup\mathsf{B} &: d_{\mathsf{A}\cup\mathsf{B}} = max(d_{\mathsf{A}}, d_{\mathsf{B}}) \\ \mathsf{A}-\mathsf{B} &: d_{\mathsf{A}-\mathsf{B}} = min(d_{\mathsf{A}}, 1-d_{\mathsf{B}}) \end{aligned} \tag{3.1}$$

gained a lot of interest [16, 17, 15], since it keeps the DFs correct almost everywhere except in the edge areas which are influenced by both original solids. This insufficiency however is the source of severe problems in reconstruction sometimes (Figures 3.1, 3.2, left column), since it results in an unbounded DF error (a point arbitrarily far from the surface of the new solid may have arbitrarily low density value [12]).

The proposed implementation of CSG operations between voxelized solids represented by truncated DFs stems from a similar background as the technique proposed in [12]. The representability criterion is enforced for the intersection of the solids by means of the relation (1.6) and by similar relations for union and difference too. However, we present a more general approach in which both solids participating in the operation are voxelized. This feature renders the technique suitable also for evaluation of whole CSG trees. The assumption is that the input models fulfill the representability criterion.

Density $d$ of each voxel in the grid is from the interval $\langle 0, 1\rangle$. It is 0 outside of the object, 1 inside of it, and 0.5 on the surface. For voxels in the transition region we store also the normalized gradient of the density $\vec{n}$. In general, the algorithm works with values $d_a$, $\vec{n_a}$, $d_b$, $\vec{n_b}$ of input voxels and by means of their analysis it acquires resulting values $d$, $\vec{n}$.

Before we start to study CSG operations, it is useful to realize that there is some relationship between union, intersection and subtraction:

$$\begin{aligned} A\cap B &= (A^{\mathrm{c}}\cup B^{\mathrm{c}})^{\mathrm{c}} & \qquad A\cup B &= (A^{\mathrm{c}}\cap B^{\mathrm{c}})^{\mathrm{c}} \\ A\setminus B &= (A^{\mathrm{c}}\cup B)^{\mathrm{c}} & \qquad A\setminus B &= A\cap B^{\mathrm{c}} \end{aligned}\quad , \tag{3.2}$$

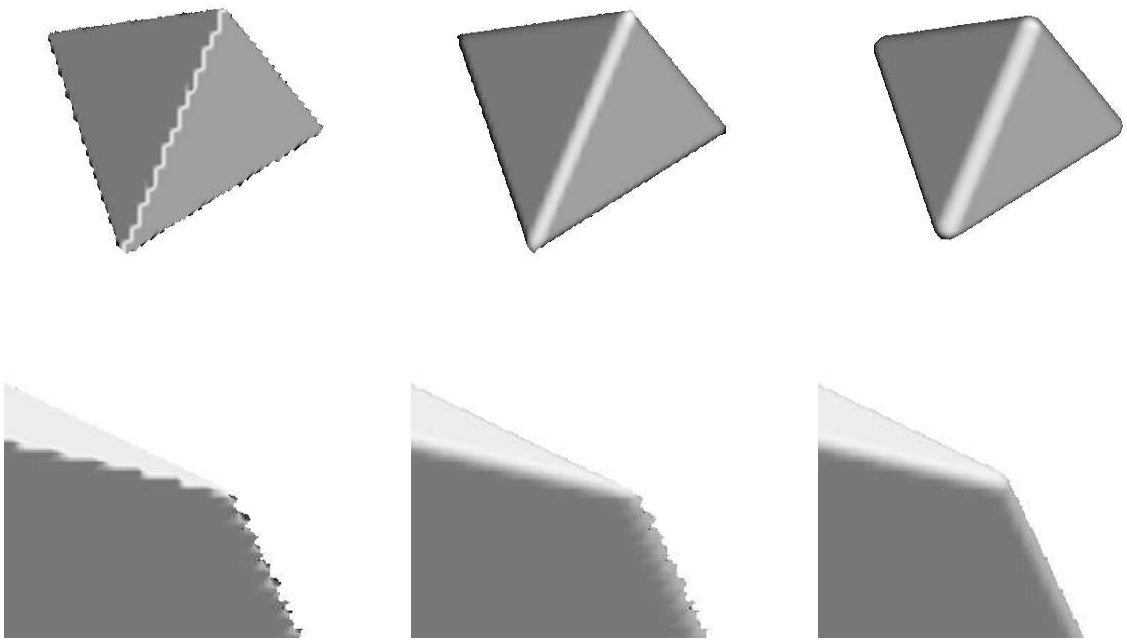


Figure 3.1: *Edge artifacts for different implementations of CSG operations I. Operations from left: simple, enhanced and advanced. Object from top: tetrahedron, wedge.*

where $A^{\mathrm{c}}$ (complement to the object $A$) is obtained at the voxel level by a simple calculation:

$$\begin{aligned} d^{\mathrm{c}} &= 1-d \\ \vec{n}^{\mathrm{c}} &= -\vec{n}\,. \end{aligned} \tag{3.3}$$

It means that all CSG operations can be performed in an uniform way. We just need to know how to make intersection (or union) and the other operators can be converted into this operator.

When object $\mathsf{A}$ is represented by a truncated DFs, each voxel of its grid belongs either to the inside $\mathcal{I}_{\mathsf{A}}$, outside $\mathcal{O}_{\mathsf{A}}$ or transition area $\mathcal{T}_{\mathsf{A}}$. We denote the boundaries of the transition area as *inner* surface $I_{\mathsf{A}}$ and *outer* surface $O_{\mathsf{A}}$ — the real object surface passes between them in the center of the transition area. Since the thickness of the transition area is $2r$, the distance from a surface point to $I_{\mathsf{A}}$ is $r$ and therefore the inside area is identical to the $S_r$-erosion of the object. Therefore, in order to compute the intersection of objects $\mathsf{A}$ and $\mathsf{B}$ according to (1.6), it is necessary to reconstruct the inner surface $I_{\mathsf{A}\cap\mathsf{B}}$, find points in the transition area of $\mathsf{A}\cap\mathsf{B}$ and correctly compute the distances of these points to $I_{\mathsf{A}\cap\mathsf{B}}$. This approach is illustrated in Figure 3.3, which shows orthogonal cuts through edges with obtuse and acute angles. The minmax criterion (3.1) is applied in such areas, where the corresponding voxels of both solids are non-transitional. The remaining space can be divided in areas **P**, **Q** and **R** according to the geometry of the newly created edge. We apply the minmax criterion also in the **P** area, while in the voxels of **Q** and

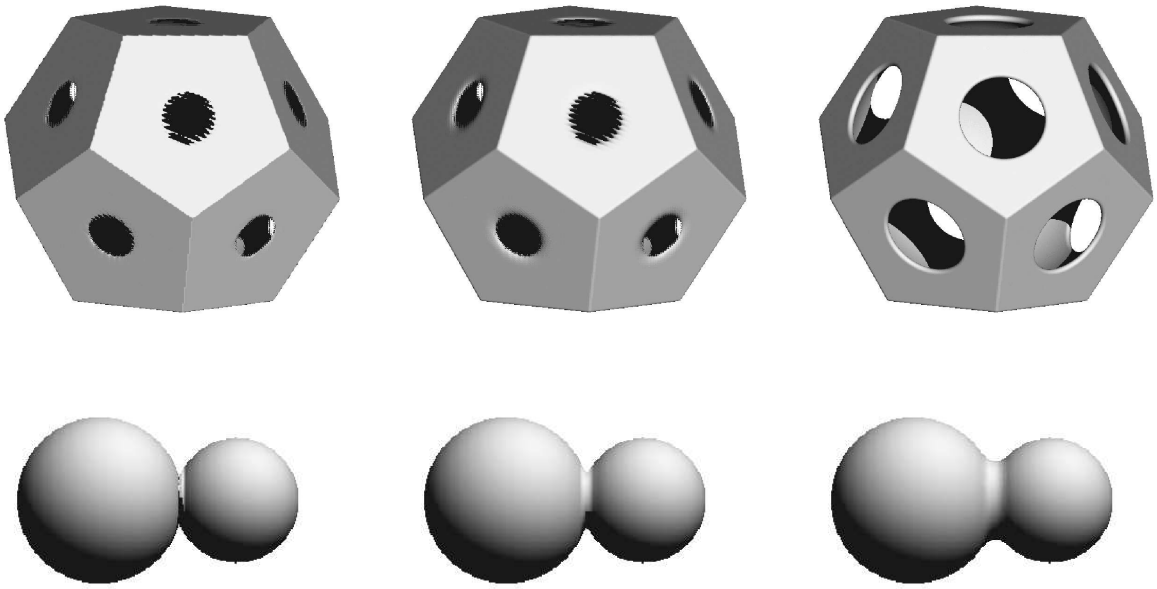


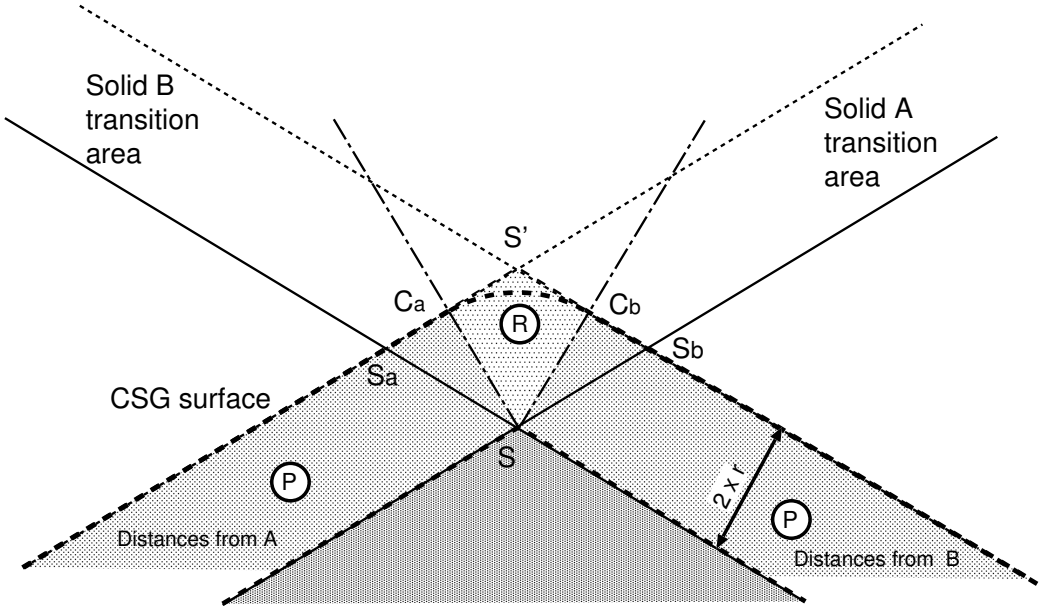
Figure 3.2: *Edge artifacts for different implementations of CSG operations II. Operations from left: simple, enhanced and advanced. Object from top: dodecahedron minus sphere, union of two touching spheres.*

**R** areas, we ideally have to compute the distance to line S instead. Line S is defined as the intersection of the inner surfaces of A and B.

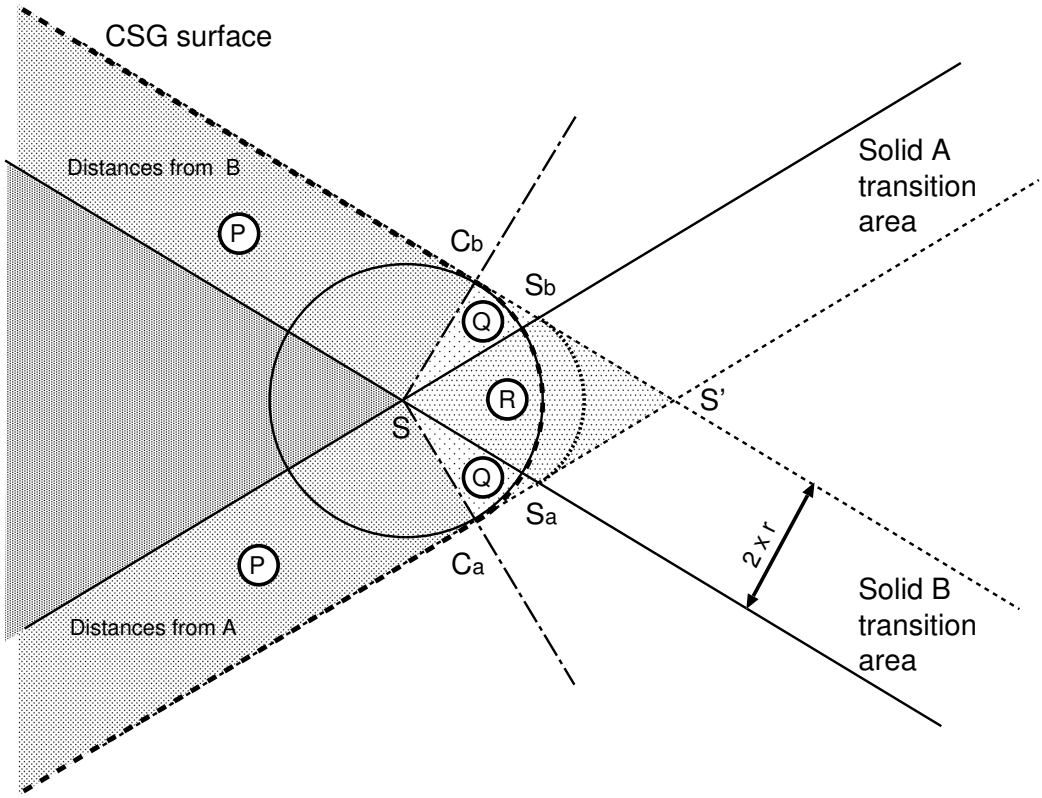
We present four variants, denoted *simple, enhanced, advanced* and *special*, in which the distance to $I_{\mathsf{A}\cap\mathsf{B}}$ is computed at various levels of correctness, thus trading quality for speed. Common features of all the variants are that they work in a single run through all voxels and that they identify voxels where the simple minmax criterion (3.1) yields correct values. For the remaining voxels the resulting densities are computed in different ways (except in the case of the simple technique, where (3.1) is used everywhere). The simple and enhanced techniques rely exclusively on point operations, while the other two compute the distance by checking a local voxel neighbourhood.

## 3.1 Simple CSG Operation

As *simple* CSG operation we denote the implementation according to (3.1). It completely ignores the sampling theory and therefore results in artifacts around edges (Figures 3.1, 3.2, left column).




(a)




(b)

Figure 3.3: *Intersection of two objects A, B*
*(a) edge with an obtuse angle, (b) edge with an acute angle*

## 3.2 Enhanced CSG Operation

The *enhanced* CSG operation is similar to the simple one in that it is based only on point operations. Here, we identify only the **R** region, $\mathbf{R} \equiv \{v : v \in \mathcal{T}_\mathsf{A} \wedge v \in \mathcal{T}_\mathsf{B}\}$ (because we cannot distinguish between regions **P** and **Q** solely by point operations) and use the minmax criterion (3.1) elsewhere. We have to distinguish between **R** regions of obtuse and sharp edges by the dot product of gradient vectors of voxels in **R**. The core of this method resides in a local approximation of the inner surface of both objects by a plane.

**Obtuse angle:** In a voxel $V$ of the region **R** we estimate density and gradient so that it corresponds to the direction and distance to the nearest point of line $S$, which is an estimate of the intersection of the inner surfaces (Figure 3.3). The outer surface of the result then forms an arc with center $S$, radius $2r$ and endpoints $C_a$, $C_b$, where $r$ is the radius of the transition area. Likewise, the real surface is formed by an arc with the same center, but with radius $r$. Thus we exactly fulfill the $S_r$-openness and $S_r$-closeness criterion for object representability.

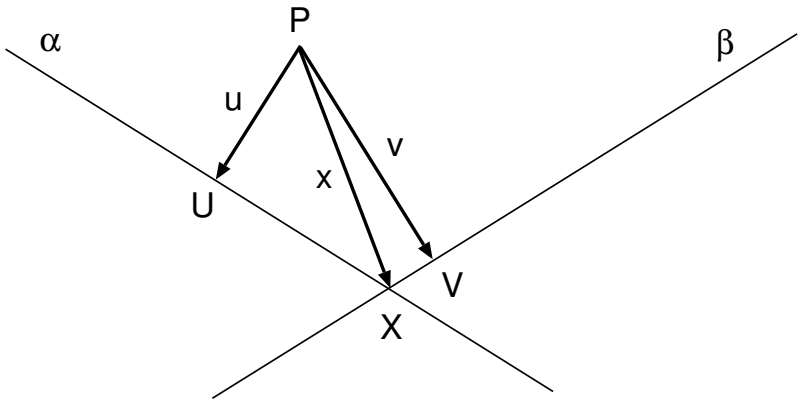



Figure 3.4: *Intersection of two planes*
*Planes* α, β *are given by point P and vectors* $\vec{u}$, $\vec{v}$. *X is to P the nearest point of the intersection of these planes. Our goal is to determine vector* $\vec{x}$.

First, we solve the problem depicted in Figure 3.4. Given is point $P$ and two linearly independent vectors $\vec{u}$, $\vec{v}$ which determine planes α, β — $\vec{u}$, $\vec{v}$ are respectively normal vectors of planes α, β and the distances between the planes and $P$ are $\|\vec{u}\|$, $\|\vec{v}\|$. Let $X$ be the nearest point of intersection of planes α, β to point $P$. Our goal is to find vector $\vec{x} = \overrightarrow{PX}$. We use three facts:

1. $(\vec{x} - \vec{u}) \perp \vec{u}$
2. $(\vec{x} - \vec{v}) \perp \vec{v}$
3. $\vec{x}$ is a linear combination of vectors $\vec{u}$, $\vec{v}$.

On the basis of this observation we arrange a system of equations, from which we get the following relation:

$$\vec{x} = \frac{(\|\vec{u}\|^2 - \vec{u}\vec{v}) \cdot \|\vec{v}\|^2}{\|\vec{u}\|^2\|\vec{v}\|^2 - (\vec{u}\vec{v})^2} \cdot \vec{u} \quad + \quad \frac{(\|\vec{v}\|^2 - \vec{u}\vec{v}) \cdot \|\vec{u}\|^2}{\|\vec{u}\|^2\|\vec{v}\|^2 - (\vec{u}\vec{v})^2} \cdot \vec{v}\,. \tag{3.4}$$

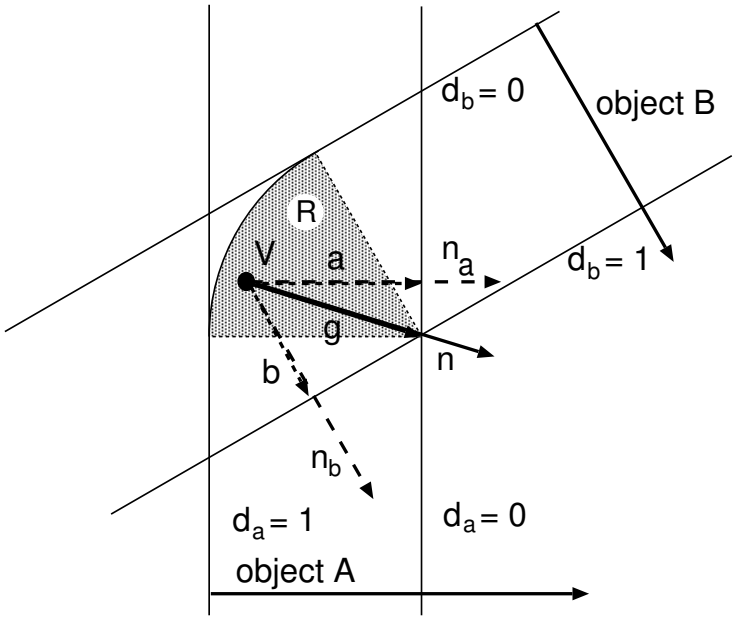



Figure 3.5: *Intersection of two objects — obtuse angle*
*Density and gradient of voxel $V \in \mathcal{T}_{A\cap B}$ are estimated through vector $\vec{g}$.*

Figure 3.5 presents a scheme that we use to evaluate density and gradient of voxels, which lie around an obtuse edge of intersection of two objects. In voxel $V$, we know the densities $d_a$, $d_b$ and the normalized gradients $\vec{n}_a$, $\vec{n}_b$ of the input volumes A, B and we want to obtain the values $d$ and $\vec{n}$ of the resultant intersection. We use the equations $\vec{a} = d_a\vec{n}_a$, $\vec{b} = d_b\vec{n}_b$ and exploiting (3.4) we get:

$$\begin{aligned} \vec{g} &= K\vec{n}_a + L\vec{n}_b \\ K &= \frac{d_a - d_b\vec{n}_a\vec{n}_b}{1-(\vec{n}_a\vec{n}_b)^2} \\ L &= \frac{d_b - d_a\vec{n}_a\vec{n}_b}{1-(\vec{n}_a\vec{n}_b)^2}\,. \end{aligned} \tag{3.5}$$

For $K > 0, L > 0$ voxel $V$ lies in region **R**, and we use vector $\vec{g}$ for the evaluation of its density and gradient:

$$\begin{aligned} d &= \|\vec{g}\| \\ \vec{n} &= \frac{\vec{g}}{\|\vec{g}\|}\,. \end{aligned} \tag{3.6}$$

**Acute angle:** In the case of an acute angle between surfaces of both solids the solution is only approximate. We construct the distance field in region **R** so that the outer surface forms an arc between points $S_a$ and $S_b$. Its radius is such that this arc touches the outer surfaces of objects A, B.

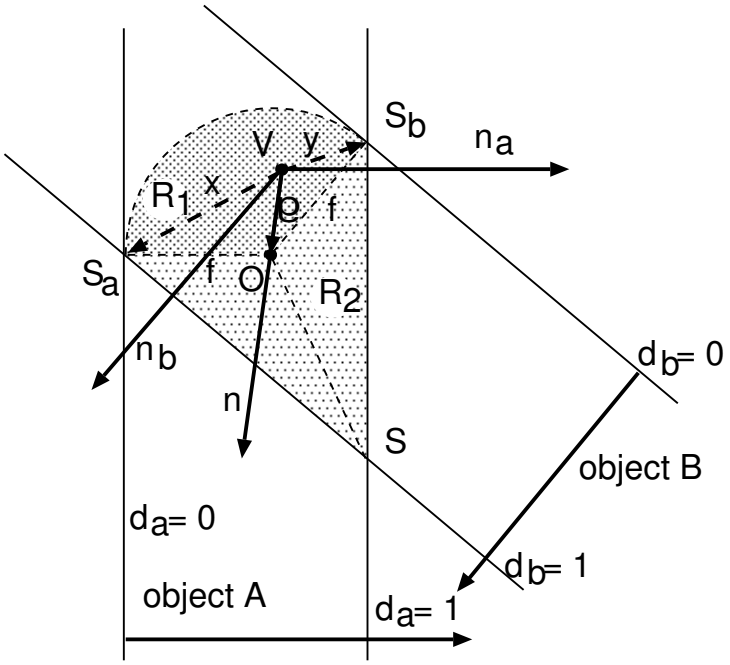



Figure 3.6: *Intersection of two objects — acute angle*
*The density and gradient of voxel $V \in \mathcal{T}_{A\cap B}$ are estimated through vector $\vec{g}$.*

Obviously, in this method we do not fulfill the $S_r$-openness and the $S_r$-closeness criterion — the smaller is the angle between surfaces the bigger is the error.

Figure 3.6 illustrates the situation. Using (3.4), vectors $\vec{x}$, $\vec{y}$ can be expressed as follows:

$$\begin{array}{rclcrcl} \vec{x} &=& -K_x\vec{n}_a + L_x\vec{n}_b & & \vec{y} &=& K_y\vec{n}_a - L_y\vec{n}_b \\ K_x &=& \frac{d_a+(1-d_b)\vec{n}_a\vec{n}_b}{1-(\vec{n}_a\vec{n}_b)^2} & & L_x &=& \frac{(1-d_b)+d_a\vec{n}_a\vec{n}_b}{1-(\vec{n}_a\vec{n}_b)^2} \\ K_y &=& \frac{(1-d_a)+d_b\vec{n}_a\vec{n}_b}{1-(\vec{n}_a\vec{n}_b)^2} & & L_y &=& \frac{d_b+(1-d_a)\vec{n}_a\vec{n}_b}{1-(\vec{n}_a\vec{n}_b)^2} \end{array} . \tag{3.7}$$

From these equations we get:

$$\begin{aligned} \vec{g} &= K_y\vec{n}_a + L_x\vec{n}_b \\ f &= K_x + K_y = L_x + L_y = \frac{1}{1-\vec{n}_a\vec{n}_b} . \end{aligned} \tag{3.8}$$

The distance of point $O$ from the inner surface is $f$. Then voxel $V$ is closer to the outer surface by $\|\vec{g}\|$, and therefore its density is:

$$d = f - \|\vec{g}\| . \tag{3.9}$$

Region $\mathbf{R}_1$ is determined by the condition $K_y > 0$, $L_x > 0$. The density of voxels which are out of this region is evaluated using the minmax criterion.

For voxels in regions $\mathbf{R}_1$, $\mathbf{R}_2$ it is not easy to decide which gradient direction they should be assigned. If we would take the direction to point $S$, the gradient would change non-continuously

across the segments $S_aS$, $S_bS$. If we would take for the gradient the direction to point $O$ in region $\mathbf{R}_1$ and use the minmax criterion elsewhere, the gradient would change non-continuously across the segment $OS$. Therefore, we evaluate the gradients in regions $\mathbf{R}_1$, $\mathbf{R}_2$ as follows:

$$\vec{n} = \frac{d_a\vec{n}_a + d_b\vec{n}_b}{\|d_a\vec{n}_a + d_b\vec{n}_b\|} . \tag{3.10}$$

This equation is derived from the intersection of two perpendicular surfaces. Although the direction of this gradient estimation is not identical to that of the real surface normal, it is good that it changes continuously, since the error of such an estimation is not as critical as the error introduced by a non-continuous gradient.

Using the enhanced CSG operation we smooth obtuse edges correctly and we get better results than with the simple one around acute edges, too. We can see visible differences between the results of the simple and enhanced implementations for the regular tetrahedron in Figure 3.1, second column, although the angle between its faces is acute (about $75,5^\circ$). However, in the case of the wedge, where the edge is sharper, both results are still jaggy.

## 3.3 Advanced CSG Operation

The goal of the *advanced* CSG operation is to implement the CSG operations fully according to the $S_r$-openness and $S_r$-closeness criterion in order to remove artifacts around all edges. The main problem is the above mentioned region **Q** in Figure 3.3b, where we have information just about one surface, but where we need information about both surfaces in order to smooth the edge correctly. If we had this information, the approach for obtuse edges could be extended for all the edges. For this purpose we have to add to our algorithm the following steps:

1. For voxels lying in the vicinity of just one surface we need to distinguish if they belong to region **P** or **Q**.

2. For voxels in the region **Q** we have to estimate the missing information about the second surface before performing the CSG operation.

A test for the classification of voxels in regions **P** and **Q** is illustrated in Figure 3.7. Using the density (distance) and gradient of voxel $V : V \in \mathcal{T}_\mathsf{A} \wedge V \notin \mathcal{T}_\mathsf{B}$ we construct point $P$ as the foot of a perpendicular from $V$ to $I_\mathsf{A}$. If $P \in \mathcal{T}_\mathsf{B}$, then voxel $V \in \mathbf{Q}$, otherwise $V \in \mathbf{P}$.

It is not quite trivial to test if point $P \in \mathcal{T}_\mathsf{B}$. $P$ as a point of the continuous space is located in one cell of the grid with 8 voxels in its corners—some of them can be inside and some of them outside the transition area of $\mathsf{B}$. We use the following rules to decide about $P$:

- If all voxels of the cell lie outside (inside) $\mathcal{T}_\mathsf{B}$, then $P$ is outside (inside), too.

- Otherwise we check all 26 neighbouring cells and from them we choose those, which have all vertices in $\mathcal{T}_B$. Using trilinear interpolation for voxels of every chosen cell we estimate the density at point $P$. Each selected cell is further assigned a weight on the basis of its distance from the point $P$. Finally, we compute a weighted average from the estimated values of the density at $P$, which determines if the voxel $V$ lies in the region **Q**.

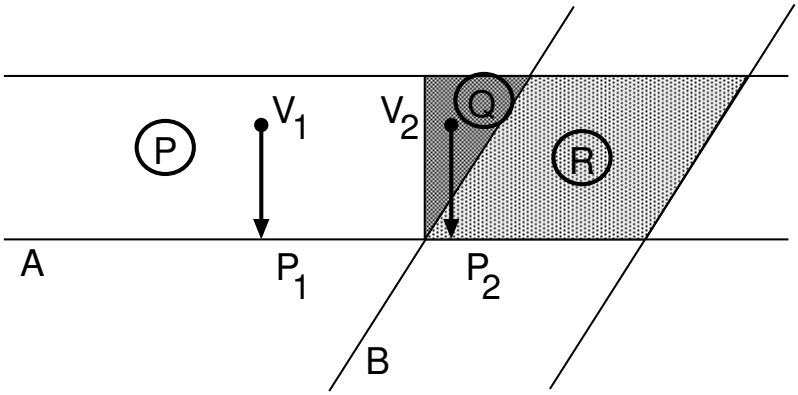



Figure 3.7: *Classification of the region $\mathcal{T}_A$ in subregions **P**, **Q** and **R***

In the second step we add the missing information about the surface B to voxel $V$. We distinguish two cases:

- If there are at least in one direction two nearest voxels to $V$ in $\mathcal{T}_B$, we use information from these voxels. For instance, if we want to evaluate density and gradient in voxel $V_{i,j,k}$ and we know density and gradient in voxels $V_{i+1,j,k}$, $V_{i+2,j,k}$, we set:

$$\begin{aligned} d_{i,j,k} &= 2d_{i+1,j,k} - d_{i+2,j,k} \\ \vec{n}_{i,j,k} &= 2\vec{n}_{i+1,j,k} - \vec{n}_{i+2,j,k} \,. \end{aligned} \tag{3.11}$$

  If we can do this estimation for pairs of voxels in more than one direction, the final values will be calculated as an average of these estimations.

- If there is no such pair of voxels in the neighbourhood of $V$, we use the information determined in the first step for point $P$. The gradient of the point $P$ is calculated in the same way as its density. We evaluate the difference $x$ of the densities in points $V$, $P$ from the known vectors $\vec{n}$, $\vec{u} = \overrightarrow{PV}$. Then voxel $V$ has values:

$$\begin{aligned} d_V &= d_P + x \\ \vec{n}_V &= \vec{n}_P \,. \end{aligned} \tag{3.12}$$

As we can see in Figures 3.1, 3.2 third column, sharp edges are now without artifacts. They are visibly smoothed, but it is the only way to represent them in a grid with the given resolution if the representability criterion is to be fulfilled. Similarly, in the union of two almost touching spheres, the thin gap between them is not representable, and therefore is filled.

We have not used any antialiasing method, so some artifacts can occur in pictures around object contours. It is not a problem of the representation but the rendering technique and can be easy solved by exploiting common routines.

We successfully solved the problem of representability of CSG solids by rounding sharp edges and small details, which would otherwise result in reconstruction artifacts. Such rounding

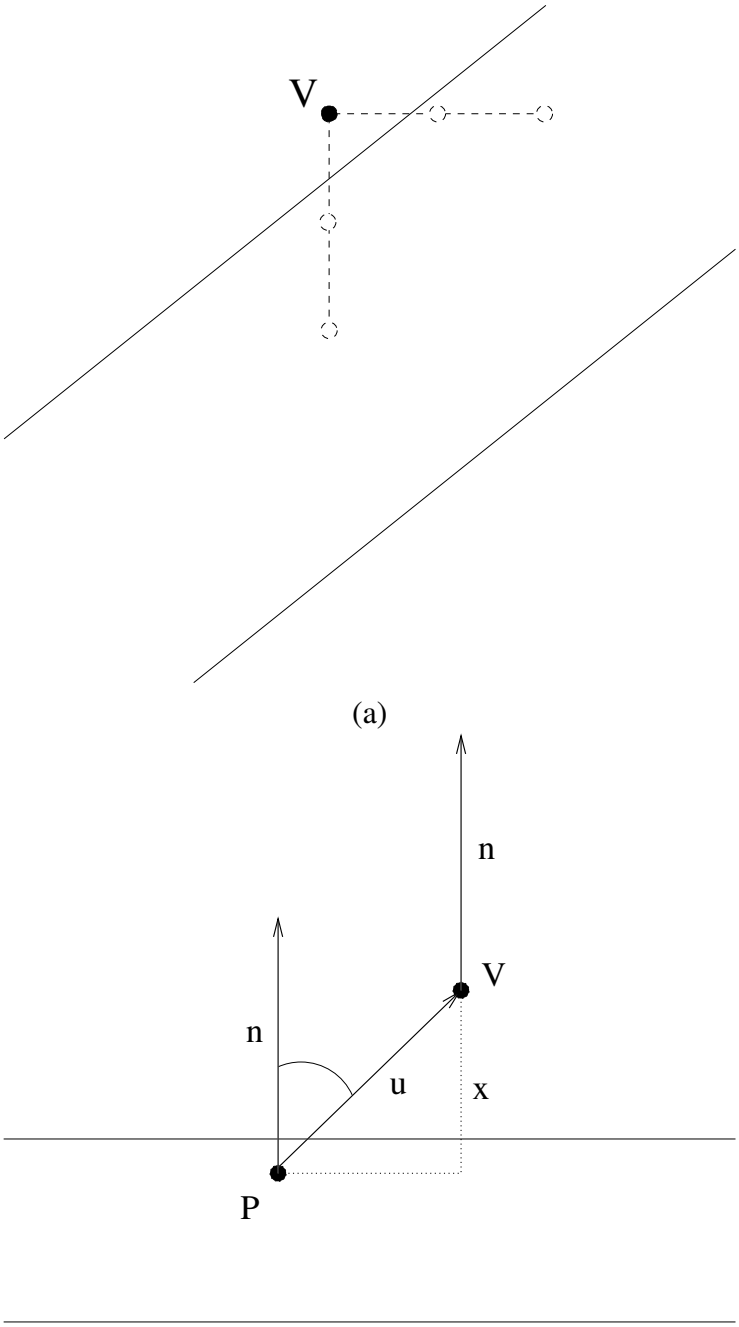



Figure 3.8: *Addition of missing information for voxel in the critical area (a) using neighbouring voxels, (b) using point P calculated previously.*

may in certain cases significantly change shape of the object, but according to sampling theory, there is no other way how to solve this problem except of local or global increase of the grid resolution (Figure 3.9).

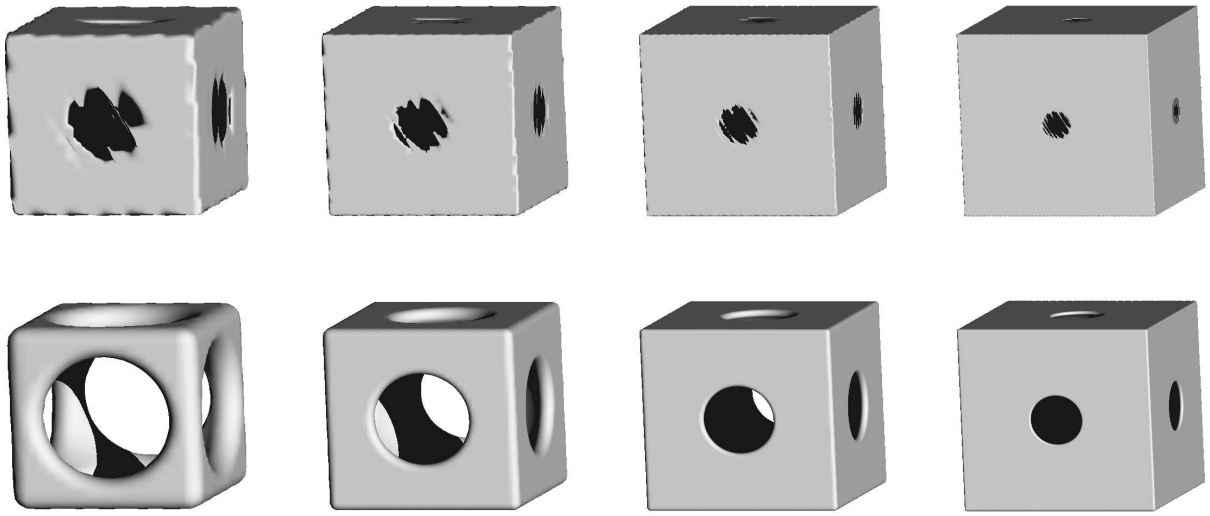


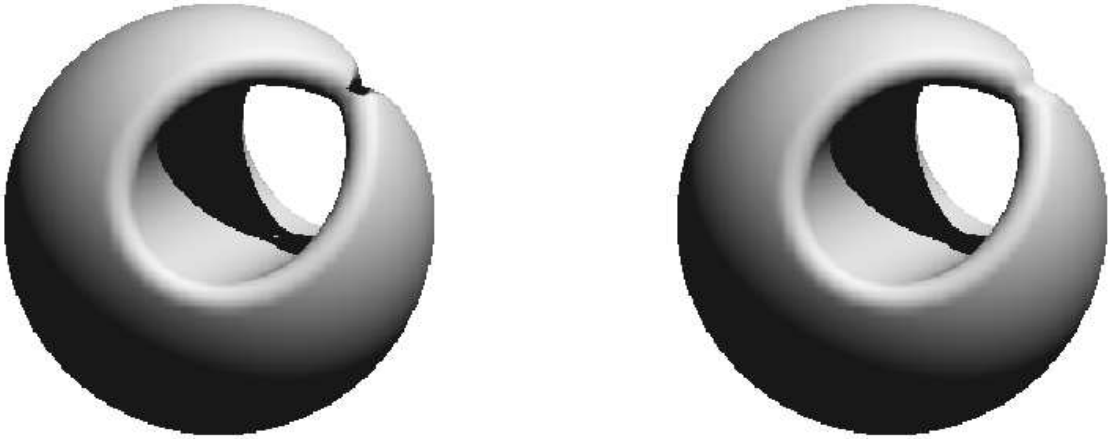
Figure 3.9: *The dependence of the object shape on the resolution*
*The cube-sphere object. Operations from top: simple, advanced. Resolution from left:* $32^3$, $64^3$, $128^3$, $256^3$.

## 3.4 Special CSG Operation

The advanced CSG operation successfully solves voxelization of representable solids. However, sometimes we want to voxelize also objects which are beyond the limits of representability. In this case artifacts may appear (Figure 3.10a). With the *special* CSG operation we propose heuristics improving such models by removing the most disturbing artifacts.

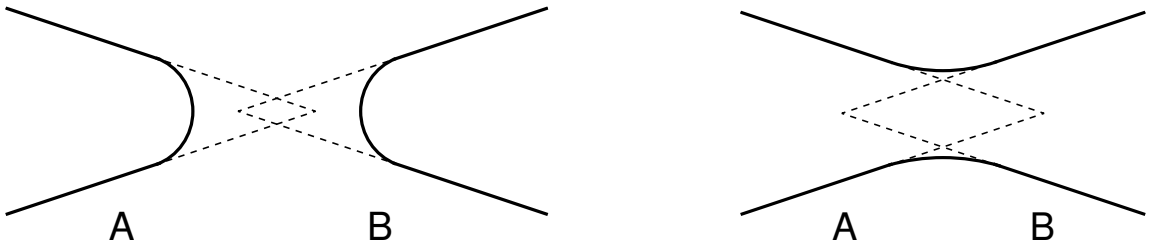
An example of such problematic configuration is shown in Figure 3.11. We have two objects A, B and we want to represent their union. If they are apart enough far, their surfaces are separated. If they intersect their surfaces are joined. It is clear that, when voxelized, there cannot exist a continuous transition between these two situations, because a thin gap or a thin bridge between the objects are not representable. This problem can be suppressed by either local or global increase of the grid resolution, but cannot be removed completely. Fortunately, the critical distance of two objects is in a narrow range of values—the advanced CSG operation gives correct results outside of this critical range.

The special CSG operation is an extension of the advanced CSG operation. Its goal is to detect regions, where the aforementioned problems can occur. It is based on the following heuristics:



(a) (b)

Figure 3.10: *Problems with representability of almost touching surfaces (a) advanced, (b) special CSG operation. Model: sphere minus cylinder.*




(a) (b)

Figure 3.11: *Problems with representability*
*A smooth transition between configurations (a) and (b) is not possible in discrete grids.*

1. During performing the advanced CSG operation we mark voxels created from input voxels with almost opposite gradients—the angle of these vectors is less than $7^\circ$ (this value was determined experimentally). These voxels are called *dangerous*.

2. After that we check all dangerous voxels. Their densities and gradients are compared with those of their neighbours (non-dangerous only) and if there are significant differences, the values of the dangerous voxels are changed. This process is quite complicated, because there is a number of different possible situations, so we do not explain it here in detail.

The special CSG operation brings some visual enhancement, although it is obvious that the result is not (and cannot be) completely correct (Figure 3.10b).

# Chapter 4

# Sharp Details Correction Method

## 4.1 Introduction

This chapter deals with implicit surfaces, which contain some sharp details. For the illustration we use various objects from the classes of *superellipsoids*, *supertoroids* and *supershapes* which are defined by the implicit equation $f(x, y, z) = 0$.

**Superellipsoids:**

$$f(x,y,z) = \left( \left|\frac{x}{a}\right|^{\frac{2}{p}} + \left|\frac{y}{b}\right|^{\frac{2}{p}} \right)^{\frac{p}{q}} + \left|\frac{z}{c}\right|^{\frac{2}{q}} - 1 \ . \tag{4.1}$$

**Supertoroids:**

$$f(x,y,z) = \left| \left( \left|\frac{x}{a}\right|^{\frac{2}{p}} + \left|\frac{y}{b}\right|^{\frac{2}{p}} \right)^{\frac{p}{2}} - \ d \ \right|^{\frac{2}{q}} + \left|\frac{z}{c}\right|^{\frac{2}{q}} - 1 \ . \tag{4.2}$$

Between $a$, $b$, $d$ and $R$ (radius of the supertoroid) there is following relationship:

$$d = \frac{R}{\sqrt{a^2 + b^2}}. \tag{4.3}$$

Superellipsoids and supertoroids belong to the class of *superquadrics*, which are described more in detail for example in [18, 19]. Parameters $a$, $b$, $c$, $d$ enable to scale the object and parameters $p$, $q$ govern its shape.

**Supershapes:** Supershapes are defined in 2D space using polar coordinates by the so called *superformula* [20, 21]:

$$r(\omega) = \left( \left| \frac{\cos\frac{m}{4}\omega}{a} \right|^{n_2} + \left| \frac{\sin\frac{m}{4}\omega}{b} \right|^{n_3} \right)^{-\frac{1}{n_1}} \ . \tag{4.4}$$

This definition can be extend easy to 3D space using spherical product:

$$\begin{aligned} x &= r_1(\phi)\cos(\phi)\cdot r_2(\psi)\cos(\psi) \\ y &= r_1(\phi)\sin(\phi)\cdot r_2(\psi)\cos(\psi) \\ z &= r_2(\psi)\sin(\psi) \\ & -\frac{\pi}{2} \le \psi \le \frac{\pi}{2} \\ & -\pi \le \phi \le \pi \end{aligned} \tag{4.5}$$

Now we multiply the first and the second equation by the factor $\frac{1}{r_1(\phi)r_2(\psi)}$ and the third one by the factor $\frac{1}{r_2(\psi)}$. Then we square all the equations and sum them together:

$$\begin{aligned} &\left(\frac{x}{r_1(\phi)r_2(\psi)}\right)^2 + \left(\frac{y}{r_1(\phi)r_2(\psi)}\right)^2 + \left(\frac{z}{r_2(\psi)}\right)^2 = \\ &\quad = \cos^2(\phi)\cos^2(\psi) + \sin^2(\phi)\cos^2(\psi) + \sin^2(\psi)\ . \end{aligned} \tag{4.6}$$

Using the formula $\cos^2(\alpha) + \sin^2(\alpha) = 1$ we get the equation:

$$\left(\frac{x}{r_1(\phi)r_2(\psi)}\right)^2 + \left(\frac{y}{r_1(\phi)r_2(\psi)}\right)^2 + \left(\frac{z}{r_2(\psi)}\right)^2 = 1\ . \tag{4.7}$$

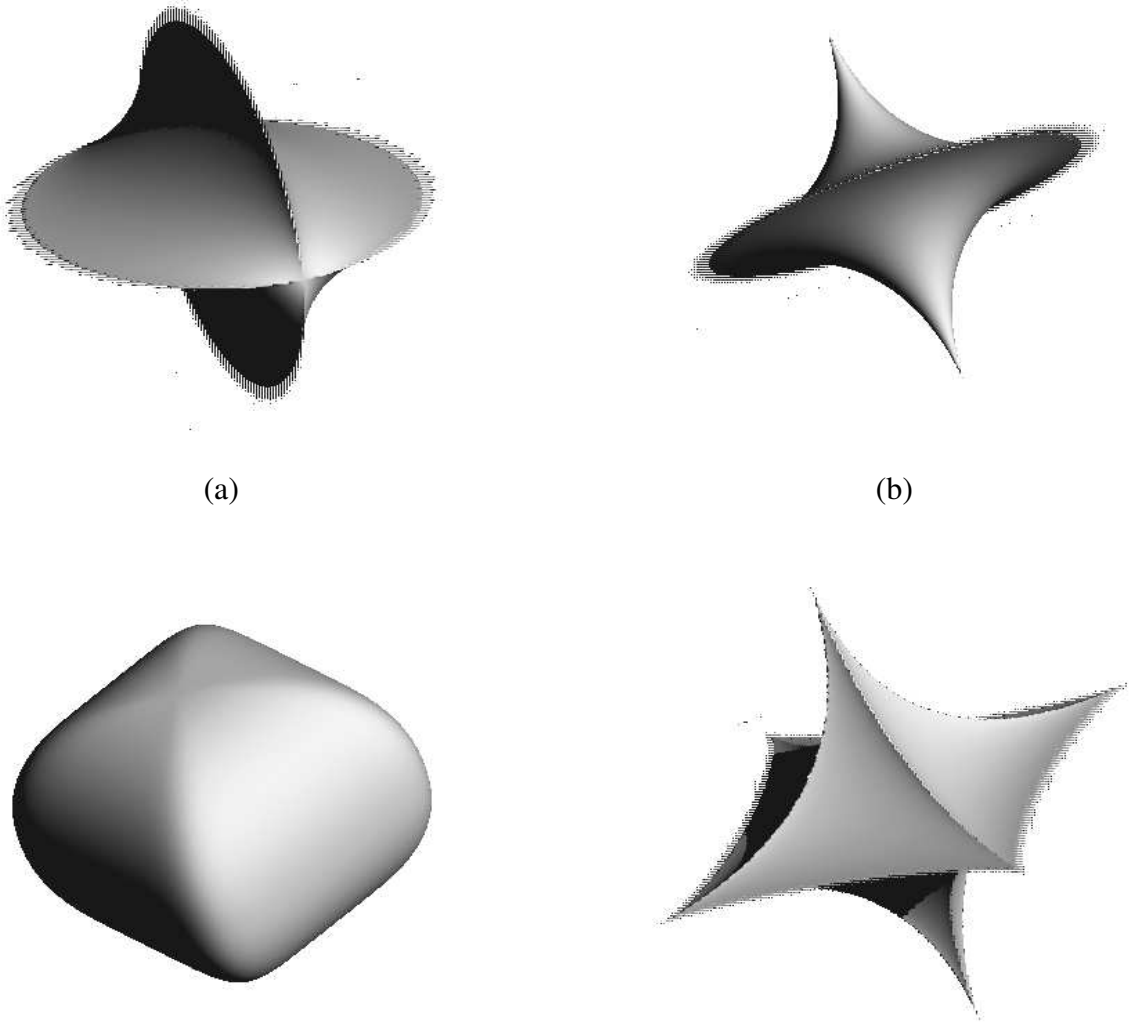
So, supershapes are defined by the implicit function $f(x, y, z)$ as follows:

$$f(x,y,z) = \left(\frac{x}{r_1(\phi)r_2(\psi)}\right)^2 + \left(\frac{y}{r_1(\phi)r_2(\psi)}\right)^2 + \left(\frac{z}{r_2(\psi)}\right)^2 - 1\ . \tag{4.8}$$

Parameters $a$, $b$, $m$, $n_1$, $n_2$, $n_3$ of the functions $r_1(\phi)$, $r_2(\psi)$ can be mutually different. We can obtain a wide scale of objects using various values of these parameters. They have been described in several papers, for instance in [20, 21]. To acquire values of angles $\phi, \psi$ we use spherical coordinates of the point $P(x, y, z)$ given by the formula (2.1).

In figures 4.1, 4.2, 4.3 we present several objects from above mentioned classes. As we can see, many of them contain some sharp details. Our goal is to voxelize these solids using DFs in a way to get correct data — it means the reconstruction error should by suitably bounded. We know from the sampling theory that in the discrete space we are able to capture details in the data just up to certain level. In other words, the highest feasible frequency is given by the sampling density. As edges are details with unbounded frequency, we are not able to represent them in discrete data. If we try to voxelize objects with edges anyway, we see in figures many disturbing artifacts after following reconstruction — edges are jaggy. The more correct approach is to modify data during the voxelization process in the way to make edges suitably rounded — their curvature should correspond with the maximal permissible frequency.

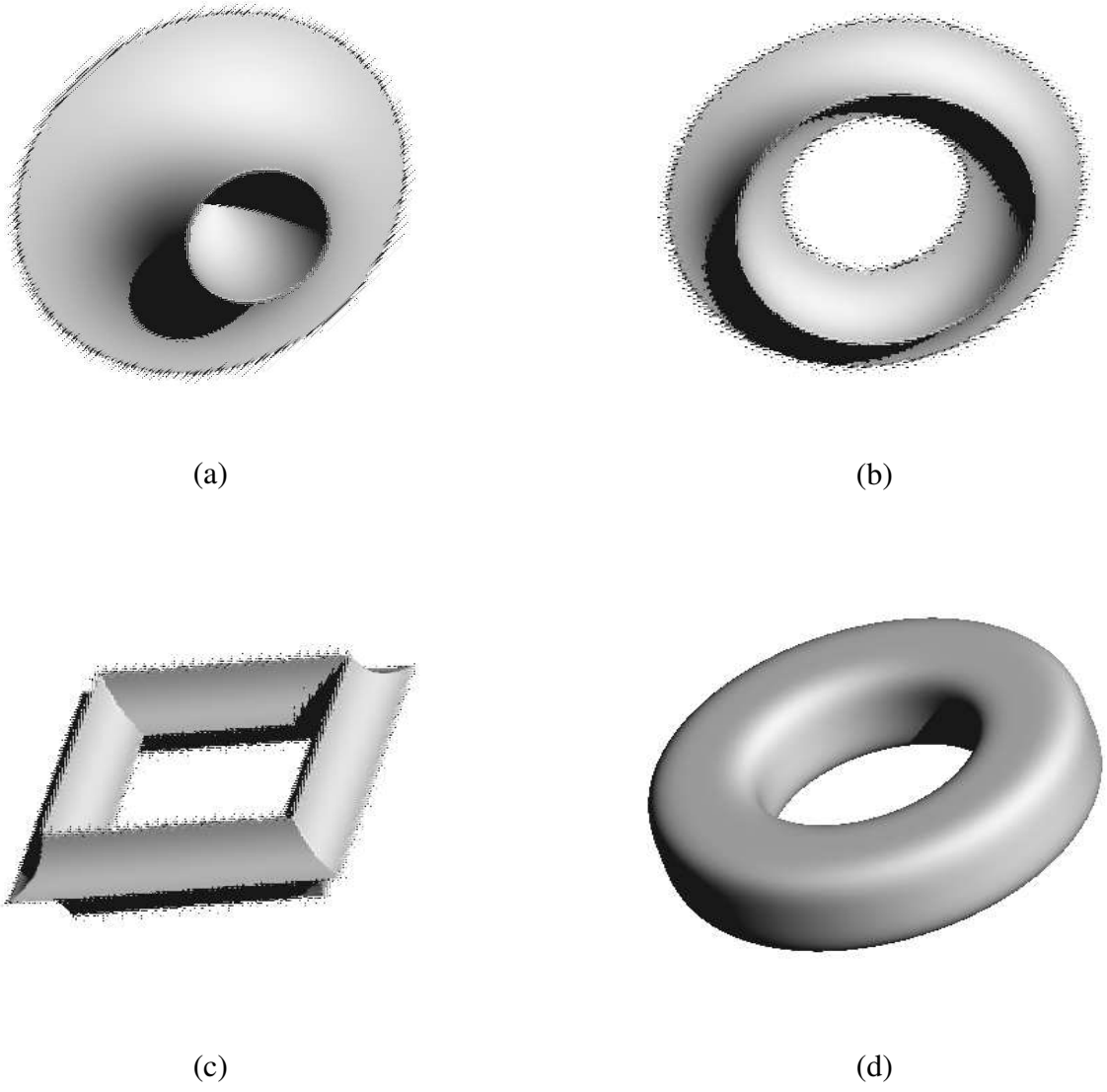
In the solution of this problem we use the same idea as in the Chapter 3, where CSG operations with voxelized models have been discussed. We take into account the representability



(a) (b)

(c) (d)

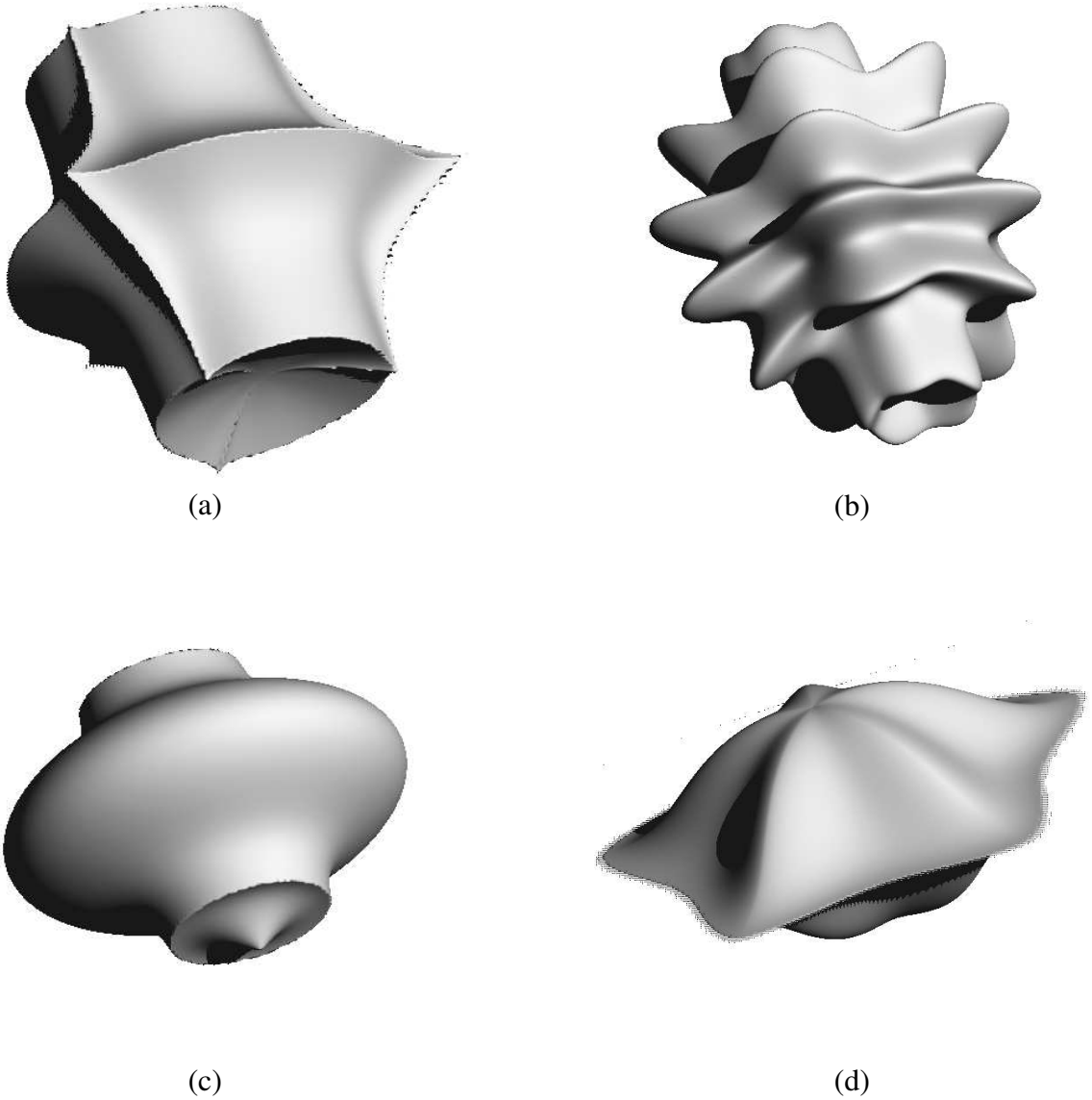
Figure 4.1: *Examples of voxelized solids - superellipsoids*
*In the figure there are superellipsoids with various parameters* $(p, q)$ *from the formula 4.1, where* $a = b = c = 1$. *Objects are displayed from different views.* *(a)*$(4, 1)$ *(b)*$(1, 4)$ *(c)*$(0.3, 0.9)$ *(d)*$(2.5, 4)$



(a) (b)

(c) (d)

Figure 4.2: *Examples of voxelized solids - supertoroids*
*In the figure there are supertoroids with various parameters* $(a, d, p, q)$ *from the formula 4.2, where* $a = b = c$. *Objects are displayed from different views.* *(a)*$(0.4, 0.5, 1, 4)$ *(b)*$(0.2, 2, 1, 4)$ *(c)*$(0.2, 2.5, 2, 4)$ *(d)*$(0.15, 3, 1, 0.5)$



(a) (b)

(c) (d)

Figure 4.3: *Examples of voxelized solids - supershapes*
*In the figure there are supershapes with various parameters a, b, $n_1$, $n_2$, $n_3$, m, from the formula 4.4 for functions $r_1(\phi)$, $r_2(\psi)$ in the equation 4.8*

criterion given in the paper [10], according to which the object $X$ is suitable for voxelization just in the case when $X$ is both $S_r$-*open* and $S_r$-*closed*. Our aim is to adjust objects during the voxelization process to make them fulfill the given criterion and to be as similar to the original objects as possible.

We solve this problem by dividing the voxelization into two stages. In the first one we compute values of voxels in the whole volume in a standard way and mark voxels, which value is problematic by reason of something (it will be explained later). We call these marked voxels *critical* — they form the *critical area*. In the second stage an other process runs, during which voxels in the critical area acquire new values. We name this new technique *Sharp Details Correction Method (SDCM)*.

## 4.2 Voxelization and Detection of the Critical Area

During the voxelization process density and the direction of its gradient is evaluated in each voxel. Density is given by the distance of the point from the surface. We compute this distance using formula (2.2). This linear approximation is correct in the surface vicinity, provided that the gradient does not change too radically. We use two tests to decide if the voxel should be marked as critical — *checking of the gradient stability* and *checking of the normal consistency*.

### 4.2.1 Checking of the Gradient Stability

Now we describe how we check the gradient stability. We move from current voxel in directions of all three axes (both forwards and backwards) using a small shift ($10^{-4}$ VU) and calculate the direction of gradient in all the six points. If there is at least one point where the gradient direction differs too much from the direction in current voxel, we deduce that the gradient is unstable. The evaluation by formula (2.2) is not meaningful, so we mark the voxel as critical.

### 4.2.2 Checking of the Normals Consistency

If the gradient of function $f$ in voxel $V$ is stable, we evaluate here the density and direction of its gradient — it is identical with the direction of $\bigtriangledown f(V)$ and it indicates the direction of the surface normal. If the voxel lies in the transition area we check the consistency of normals, which is illustrated in Figure 4.4. The idea is to find points $P$, $Q$ on the basis of information in voxel $V$ — they are feet of perpendicular from voxel $V$ to outside and inside surface. We compute the surface normals in these points. In the ordinary case they are the same as in the voxel $V$ (or their values are very close to each other). But, if the voxel $V$ lies in the edge vicinity, these directions are very different. We have to recalculate the voxel in this case, so we mark it as critical.

At the end of this stage of voxelization we have four classes of voxels in the grid:

**outside:** We know that they lie outside the object.

**inside:** We know that they lie inside the object.

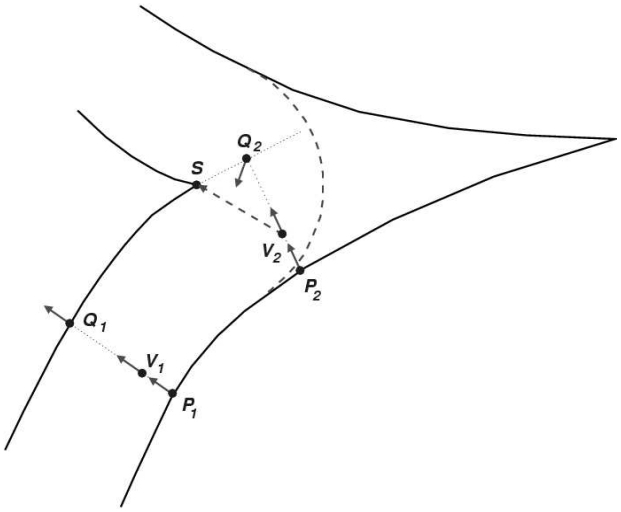



Figure 4.4: *Checking of the normals consistency*
*There should be approximately the same normals in points P, Q as in the voxel V (the first case). If it is not the case, the voxel is located near an edge and its value must be recalculate (the second case), so we mark it as critical.*

**transition:** We know that they lie in the transition area.

**critical:** We do not know where they lie. Their values have to be recalculate later.

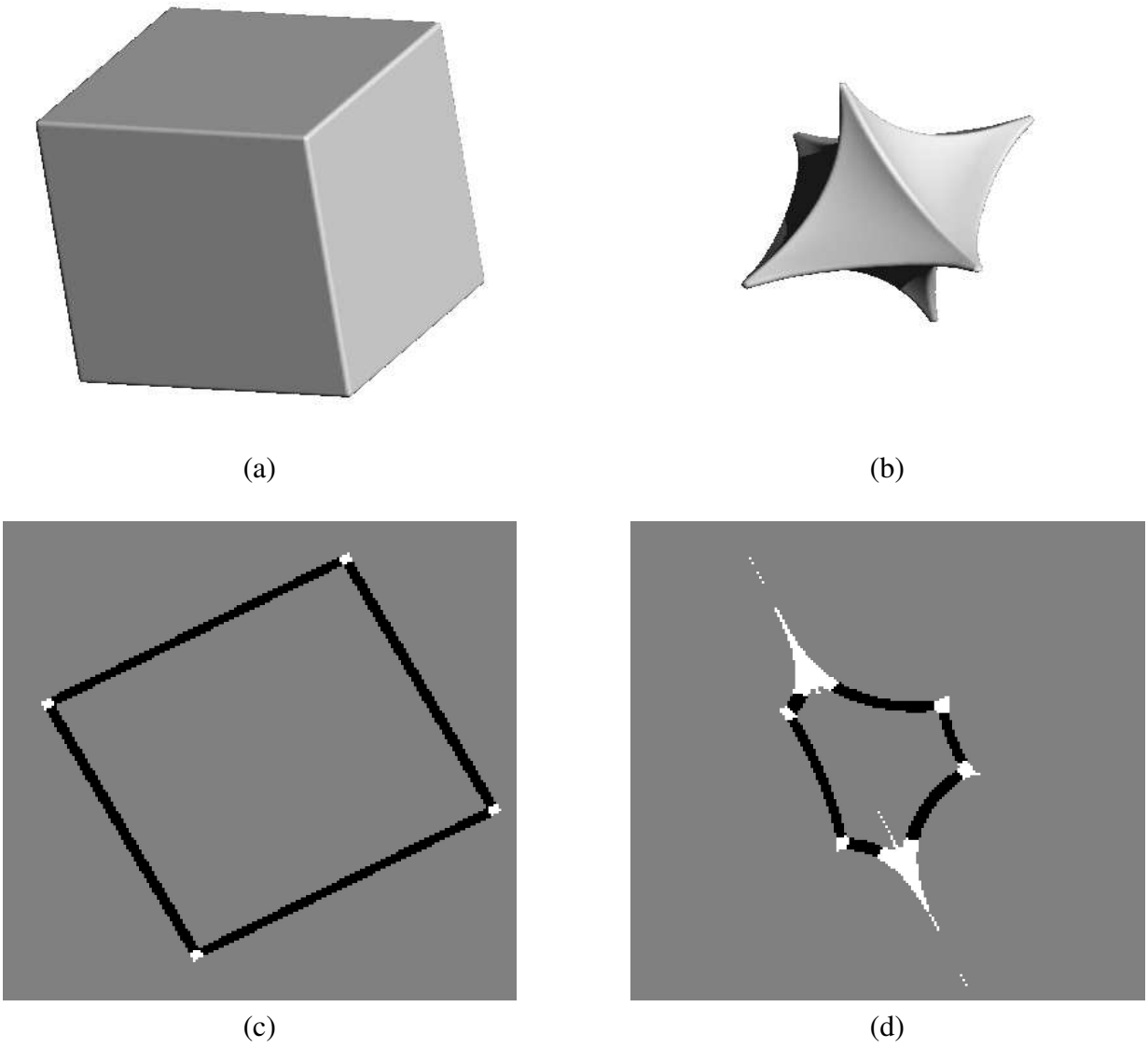
Above mentioned classes of voxels form four areas in the volume: *outside*, *inside*, *transition* and *critical*. Two examples of such voxel classification are depicted in Figure 4.5.

## 4.3 Completion of Information in the Critical Area

After the first stage of voxelization we need to evaluate correct values in all critical voxels. They lie around edges and vertices. Our goal is to transfer here information from faces in the local neighbourhood. We aim to extrapolate these faces linearly so that the information from them comes into nearest critical voxels. Of course, each voxel can obtain information from several parts of the surface — voxels around edges from two, voxels around vertices from three and more. Next, we have to analyze the information in each critical voxel and determine the resulting values. We use the same approach as in Chapter 3. We regard the faces in the local neighbourhood as halfspaces and the result is given by intersection of them. We try to solve this problem just for solids with convex edges and vertices — other sharp details require more complicated analysis.

### 4.3.1 Extrapolation of Information in the Critical Area

The propagation of information in the critical area proceeds in a series of steps. At the beginning we find all critical voxels, which neighbour with a voxel in the transition area. We create so called



(a) (b)

(c) (d)

Figure 4.5: *Voxel classification*
*Top row: voxelized solids (cube and superellipsoid). Bottom row: one slice of given volumes — black colour: transition area, white colour: critical area, grey colour: outside or inside area.*

*active entry (AE)* in each such voxel — it contains information about density and surface normal in the voxel. We acquire this data using weighted average of values taken from neighbouring transition voxels ($V_i$):

$$d = \frac{\sum_{i=0}^{k} w_i d_i^*}{\sum_{i=0}^{k} w_i} \tag{4.9}$$

$$\overrightarrow{n} = \frac{\sum_{i=0}^{k} w_i \overrightarrow{n_i}}{\| \sum_{i=0}^{k} w_i \overrightarrow{n_i} \|}, \tag{4.10}$$

where $d$ and $\overrightarrow{n}$ are resulting values of density and normal, $k$ is the number of neighbouring transition voxels, $w_i$ is the weight of $V_i$ (1 for face neighbours, $\frac{1}{\sqrt{2}}$ for edge neighbours and $\frac{1}{\sqrt{3}}$ for vertex neighbours), $\overrightarrow{n_i}$ is the surface normal of $V_i$ and $d_i^*$ is the density of $V_i$ modified in a way to correspond with the shift from the neighbouring voxel to the current one:

$$d_i^* = \frac{\overrightarrow{n_i} \cdot \overrightarrow{v_i}}{2r} + d_i \, . \tag{4.11}$$

In the last formula $d_i$ is the density of $V_i$, $r$ is the radius of transition area and $\overrightarrow{v_i}$ is the vector of movement $(V - V_i)$. It comes from the fact that $d_i$ and $\overrightarrow{n_i}$ situated in $V_i$ define a plane that we want to define using $d_i^*$ and $\overrightarrow{n_i}$ situated in $V$. One should be aware that the density $d$ in the transition area depends on the distance $D$ linearly according to the formula (1.1):

$$d = \frac{1}{2} - \frac{D}{2r} \, . \tag{4.12}$$

If the normal vectors in two neighbouring voxels are very different from each other — we call them *inconsistent* (the difference exceeds certain limit) — we consider that they describe diverse parts of the surface, so we analyze them apart. In this case we divide neighbouring voxels into groups with mutually consistent normals and we create an AE on the basis of each such group. So, one voxel can hold several AEs.

After the above described initialization step we have AEs in some critical voxels. They form so called *active front (AF)*. We move this AF in next steps until it crosses whole the critical area.

In one step of the AF movement we find for each AE all the neighbouring critical voxels, to which the information should be transfered. Those voxels have to fulfill two conditions:

1. They did not contribute to this AE in the previous step.
2. They carry no AE, which normal is consistent with this AE.

We create a *new active entry (NAE)* in each such neighbouring voxel in the same way as in the initialization step. But, now we use neighbouring AEs (with mutually consistent normals) instead of transition voxels. As NAEs are created gradually, the *new active front (NAF)* is generated — it serves as AF in the next step. The process finishes at the moment when the AF is empty. In voxels we store values of density and surface normal, as the AF runs through them. One voxel can be hit by various parts of the front (distributing information from various parts of the surface),

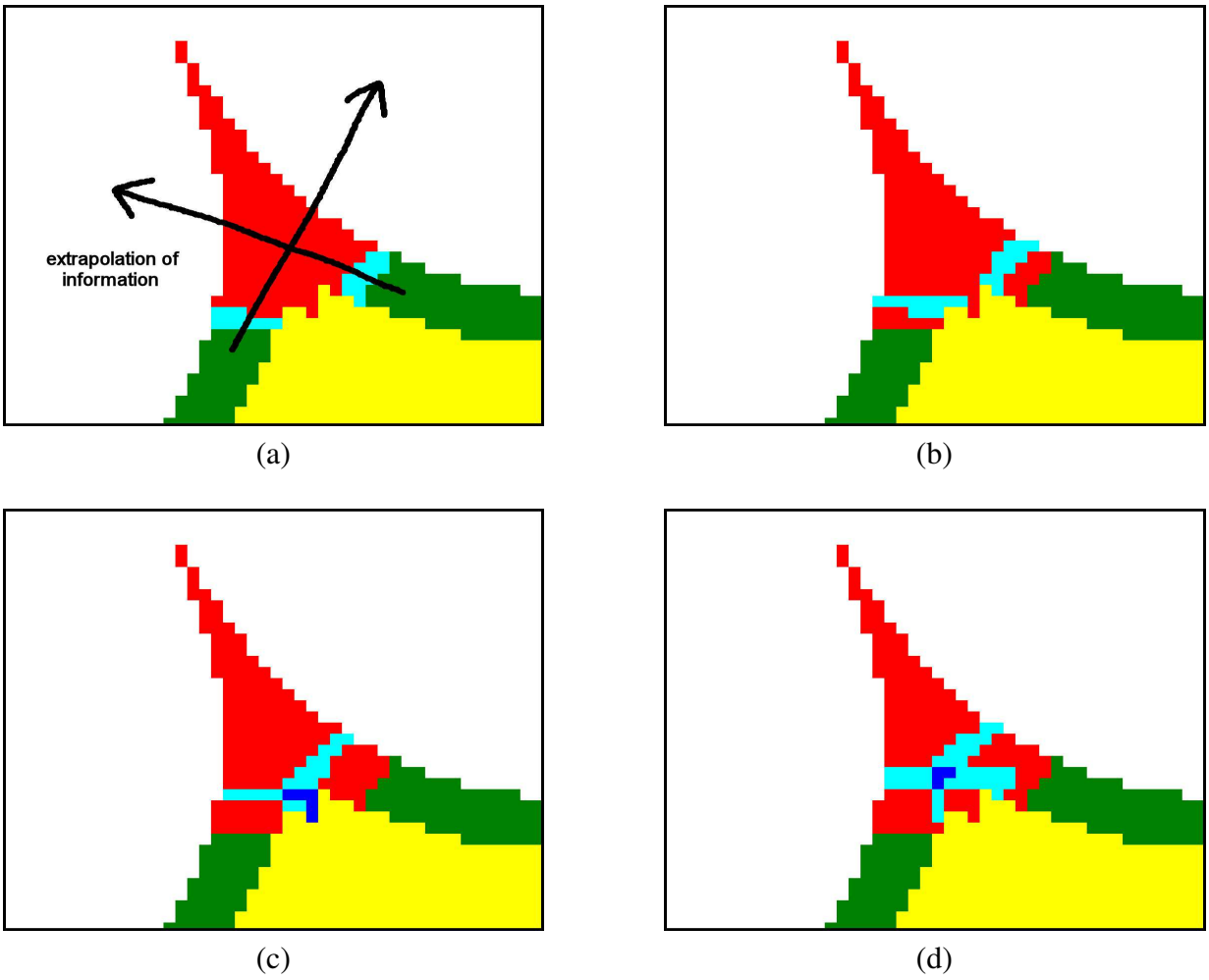



Figure 4.6: *Active front propagation*
*Initialization (a) and first three steps (b, c, d) of AF propagation. Colours of voxels: white — outside, yellow — inside, green — transition, red — critical, cyan — critical voxels with one AE, blue — critical voxels with two AEs.*

so it can store more values of density and normal. These values are used in the next phase of the process to get resulting value of the voxel.

The idea of our algorithm SDCM is similar to the well known *fast marching methods (FMM)* of front propagation. In the case of FMM the front is propagated gradually by changing the voxel from *narrow band* to *frozen* followed by an extension of the narrow band. On the other hand, in the case of SDCM, the front is propagated in several steps (see Figure 4.6). In one step all the AEs are visited and analyzed and a NAF is generated in their neighbourhood. In the next step the NAF serves as AF. Actually, the whole wave of information moves in one step and some parts of this wave can meet together and hit one voxel several times.

### 4.3.2 Evaluation of Data in the Critical Area

As we have mentioned before, we use the same idea for the evaluation of information in critical voxels as for the solution of CSG operations with voxelized solids. There are several values of density and normal stored in one voxel — they define together certain number of halfspaces. The calculation is sequential. We take the first and the second entry from the list and create a new entry using the intersection operation. This new entry is then combined with the third entry from the list — using the intersection operation again. And so on... We continue this computation, until the list of results is empty, and so we obtain the resulting value of voxel.

If the surface contains some too sharp vertices, this evaluation can be affected by certain errors. It is, because the realization of the CSG operation is derived from an intersection of two halfspaces. But the result of intersection of two halfspaces is not a halfspace. When we use this result as input for the next intersection operation (where we assume halfspaces as input), certain error can occur (it depends on the mutual positions of all halfspaces). It would be more suitable to solve the intersection of all the halfspaces at once. This approach has not been implemented yet.

There can be some critical voxels, which are never reached by the AF, because the critical area can be disconnected through the discretization (Figure 4.5d). In this case we explore the neighbourhood of given voxel and make it inside or outside according to values of nearest non-critical voxels.

## 4.4 Instability Problem

We assume in the above mentioned solution that each critical voxel obtains information exactly from those parts of the surface from where we want to transfer it. For example, voxel in the edge vicinity should be hit by information from appropriate two faces, which are joint in the edge (faces can be curved — we regard them locally as planes). Similarly, voxel in the vertex vicinity should be hit just by information from faces, which are joint in the vertex. The question is if we are able to ensure this assumption.

Our experiments have showed that sometimes the information is also distributed to voxels where it is undesirable. Of course, we evaluate incorrect values in voxels in this case. Therefor there has been a need for more detailed exploration of the front propagation and its rectification.

We have proposed several heuristics, which have managed to reduce significantly the number of errors in the process. However, we have not found an universal solution of this problem.

### 4.4.1 Delimitation of the Working Area

The first problem is that the topology of the critical area can be complex enough and the front propagation in the more distant regions can be quite complicated. This can lead to a situation where a part of front "returns" to the surface vicinity in a totally other location than it comes from. To avoid this problem we try to delimit the *working area* before running the process of AF propagation. To make this we use so called *delimitation front*. It works in the same way as the AF with an additional condition that the information is propagated just through voxels, for which the estimated density corresponds with the outside value. Also here we have to prevent the situation where the information hits some distant parts of surface. Therefor we stop the delimitation front in places where it meets a voxel visited before. We know that all the visited voxels should be outside the solid, because they belong to an outside area of at least one halfspace, from which the intersection is created. These voxels delimit the working area, in which the AF propagation proceeds in the next phase of the process.

### 4.4.2 Rectification of the Active Front Propagation

Also in the delimited working area it can sometimes happen that the information is distributed to voxels where it is undesirable. The further way how to control the front propagation is to set a restriction on the direction where the information from given voxel can transfer in the next step. To enable it we store in each AE an additional information about the direction of the front movement and about the starting point, in which the information was initialized. In next steps we acquire this data using weighted average of values from neighbouring AEs in the same way as the information about density and surface normal. As we decide where to transfer the information from current voxel we construct a space angle with the vertex in the starting point and the axis in the direction of the front movement. Then we take only voxels located inside the angle for the next examination. The open question is how to determine in voxels the direction of AF movement in the initialization step and what should be the size of the above mentioned angle.

The direction of the AF propagation $\overrightarrow{s}$ is obtained in the AE of the voxel $V$ during the initialization step on the basis of configuration of neighbouring transition voxels — we just combine directions, in which these voxels lie:

$$\overrightarrow{s} = \frac{\sum_{i=0}^{k} w_i (V - V_i)}{\| \sum_{i=0}^{k} w_i (V - V_i) \|} \,. \tag{4.13}$$

This straightforward solution is not ideal, because two neighbouring AEs can have quite different directions due to the discrete character of the problem. To "blur" these directions (to get similar directions in neighbouring AEs), we modify the direction in given entry using information in

neighbouring entries (we change all entries in several steps):

$$\overrightarrow{s^{\text{new}}} = \frac{\sum_{i=0}^{n} w_i \overrightarrow{s_i} + \overrightarrow{s^{\text{old}}}}{\| \sum_{i=0}^{n} w_i \overrightarrow{s_i} + \overrightarrow{s^{\text{old}}} \|} , \tag{4.14}$$

where $\overrightarrow{s^{\text{old}}}$ is the old value of the direction and $\overrightarrow{s^{\text{new}}}$ is the new one. Then we use the surface normal $\overrightarrow{n}$ to project the direction vector onto the tangent plane of the surface to make the information propagate in the surface direction:

$$\overrightarrow{s^{\text{new}}} = (\overrightarrow{n} \times \overrightarrow{s^{\text{old}}}) \times \overrightarrow{n} . \tag{4.15}$$

The computation of $\sum_{i=0}^{k} w_i(V - V_i)$ in the formula (4.13) is problematic, because we can get the null vector as the result. According to our experiments, this situation really occurs sometimes. The reason is that the transition area can have various shapes, so the properties of its boundary with the critical area are not fully under our control. This properties also influence the geometry of the AF and can cause its unexpected propagation. That is the reason, why we try to modify this boundary with several methods.

The first method we have proposed for this purpose is the combination of *erosion* and *dilation* of the transition area. In the process of erosion we remove from the transition area all the voxels neighbouring with the critical area, whereas in the process of dilation we add to the transition area all the neighbouring critical voxels. As a result, thin layers of transition voxels are "deleted". But, according to our experiments, this method does not bring significant enhancement. On the contrary, it leads to blurring of some details, which could be represented in given resolution.

As more suitable, it seems to be the approach, where we remove from the transition area those voxels, which have "too many" non-transition neighbouring voxels. It is not easy to constitute a criterion what voxels should be removed. We have tested several possibilities and made a result that the best solution is to remove voxels, which fulfill following condition:

- There are at least two pairs of voxels outside the transition area from three pairs of neighbouring voxels where one pair consists of two opposite face neighbours.

As we remove these voxels, we "cut" from the transition area those parts, which stick out too much. So we blur the boundary between transition and critical area thanks to that the direction estimation of the front propagation is more stable.

The next question is, what should be the size of the space angle, inside which the AF has to move from the starting point. If the angle is too small, there are some voxels, into which the needed information is not distributed. On the other hand, if the angle is too large, the AF can be propagated to undesirable areas. The problem is that this question has no universal answer, because a given angle can be too small for one configuration of voxels and too large for an other one. We just have been able to tune the right value of the angle size for given object and given resolution.

## 4.5 Algorithm Analysis

### 4.5.1 The Time Complexity of the Algorithm

Of course, the SDCM needs some extra computation time in the comparison with the common voxelization. In total, the time of the SDCM can be expressed as:

$$t_{SDCM} = t + t_a + t_{gs} + t_{nc} + t_{ci} \ . \tag{4.16}$$

In the equation, $t$ is the time of common voxelization, $t_a$ is the time needed for the construction of an auxiliary structure serving for voxel classification and realization of the AF propagation, $t_{gs}$ is the time needed for checking of the gradient stability, $t_{nc}$ is the time needed for the checking of normals consistency and $t_{ci}$ is the time needed for the completion of information in the critical area. Now we analyze particular components:

$t$ **:** Similarly, like in Chapter 2.4.3, we divide this time into two subcomponents: $t_e$ — time for the evaluation of density in voxels, $t_w$ — time for writing the computed value into memory.

$t_a$ **:** The writing into the auxiliary structure runs in parallel with the writing into the volume and takes approximately the same time, therefor $t_a \approx t_w$.

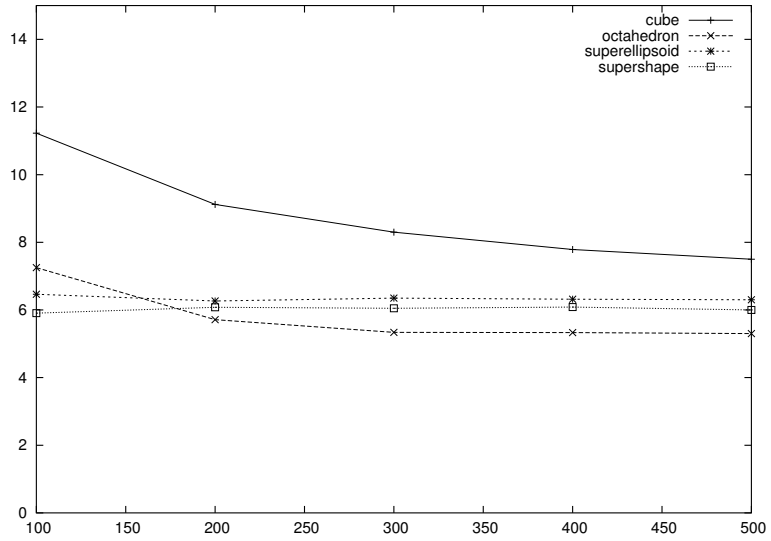
$t_{gs}$ **:** During the evaluation of function $f$ and its gradient in given voxel we also calculate the gradient in other six points around. The gradient calculation is much more costly than the function $f$ calculation, therefor $t_{gs} \approx 6t_e$.

$t_{nc}$ **:** The main part of this time is taken by the evaluation of gradient in two points $P$, $Q$ in Figure 4.4. This computation is performed just in voxels $V$, which are located in the surface vicinity, so it depends linearly on the surface area. We can make an estimation: $t_{nc} < 2t_e$.

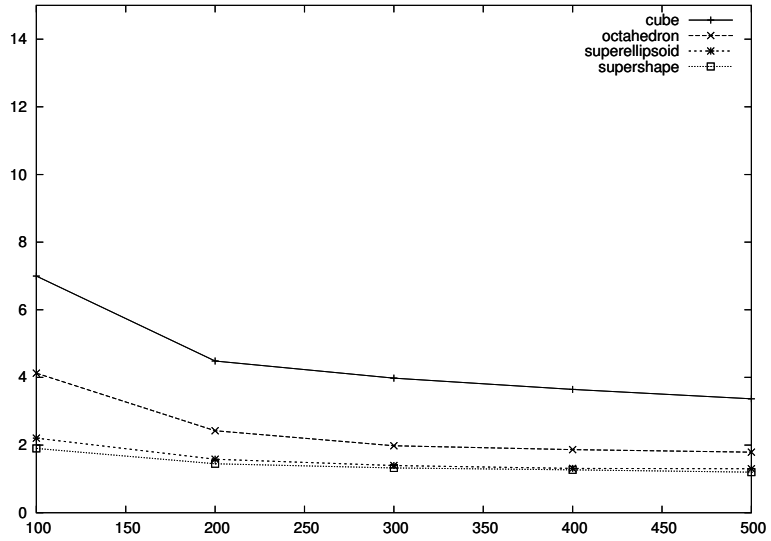
$t_{ci}$ **:** This time involves the AF propagation and the calculation of resulting values in critical voxels. Each critical voxel is visited at most $k$ times during the AF propagation where $k$ is the highest number of faces around one vertex. Similarly, we have to handle at most $k$ entries in each critical voxel during the evaluation of resulting values. This implies that $t_{ci}$ depends linearly on the number of critical voxels, which is given by the total length of object edges and by their sharpness.

The proportion of particular components in $t_{SDCM}$ can be very varied, it depends on specific object and given resolution. Naturally, it is interesting for us to compare times $t$ and $t_{SDCM}$. In Figure 4.7(a), there is depicted the dependence of the ratio $\frac{t_{SDCM}}{t}$ on the resolution for several objects. We can see that the SDCM takes approximately 5–10 times more time than the common voxelization.

As we analyze the algorithm, we can notice that a significant distribution to the time $t_{SDCM}$ is formed by the component $t_{gs}$. So we can significantly reduce the time requirements of the SDCM if we remove the checking of gradient stability (Figure 4.7(b)). According to our experiments, SDCM can function without problems on the tested objects also without above mentioned




(a)




(b)

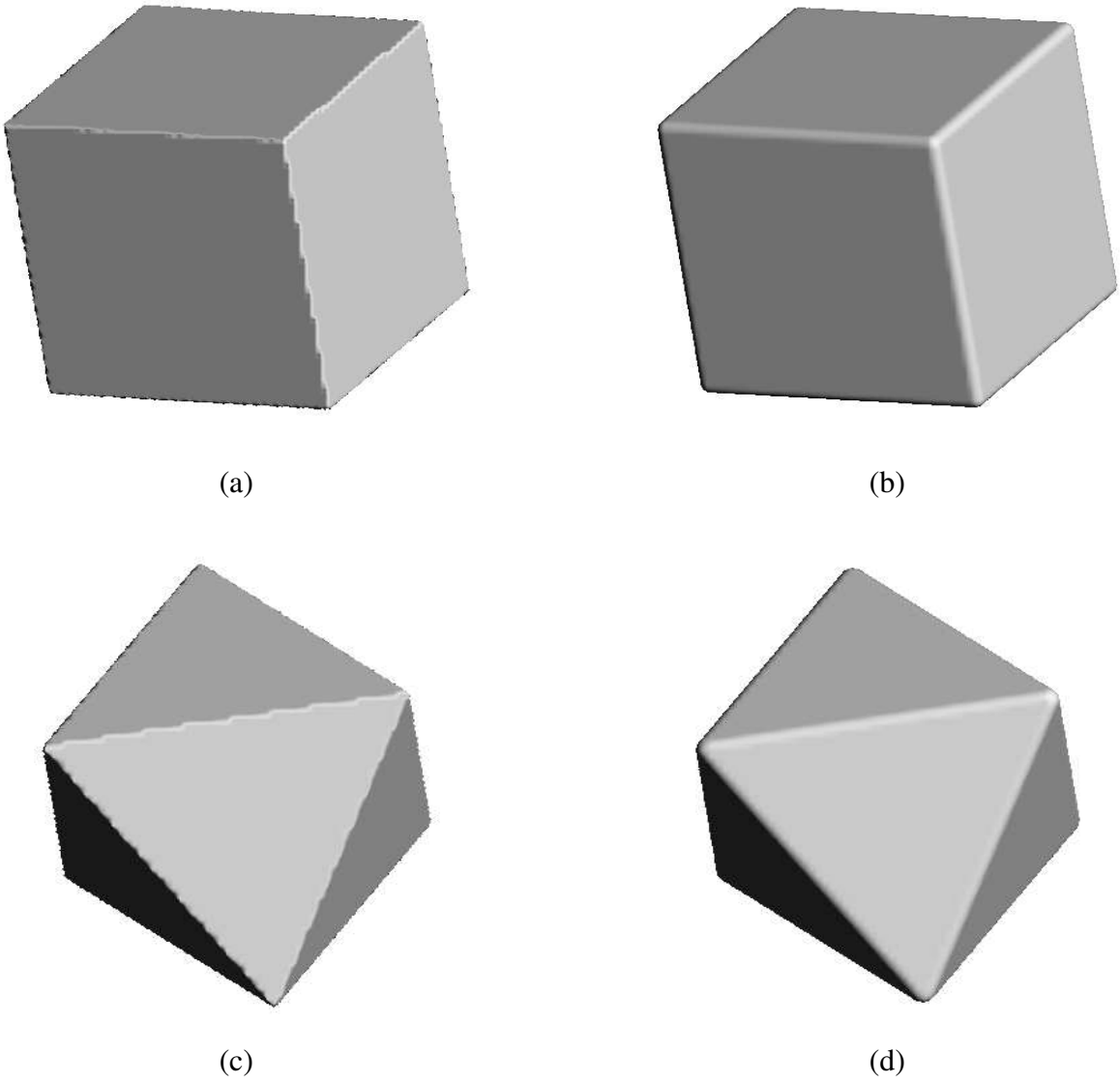
Figure 4.7: *The time complexity of the algorithm*
*There is the resolution of volume on the x-axis and the ratio* $\frac{t_{SDCM}}{t}$ *on the y-axis, where* $t_{SDCM}$ *is the time of SDCM and* $t$ *is the time of common voxelization process. The case (a) includes the checking of gradient stability, the case (b) does not.*

checking. But it is possible that there are some configurations where the voxelization process fails without this checking. So we have to decide between the speed and the accuracy of the algorithm.

### 4.5.2 Results

The SDCM has been tested on simple objects as cube and regular octahedron (Figure 4.8) and more complex objects as superellipsoids, supertoroids and supershapes with various parameters (figures 4.9, 4.10, 4.11). The basic algorithm (described in 4.2 and 4.3) functions correctly for simple objects — there is no problem with the instability of the AF propagation. But some problems occur as we voxelize more complex objects. Some heuristics (described in 4.4) extend significantly the set of objects, which are voxelized correctly, but we have not found an universal solution. Problems are caused especially by too sharp vertices and too curved edges. It is obvious from the description of SDCM that it can be not used for objects with non-convex vertices (for example object (c) in Figure 4.2).

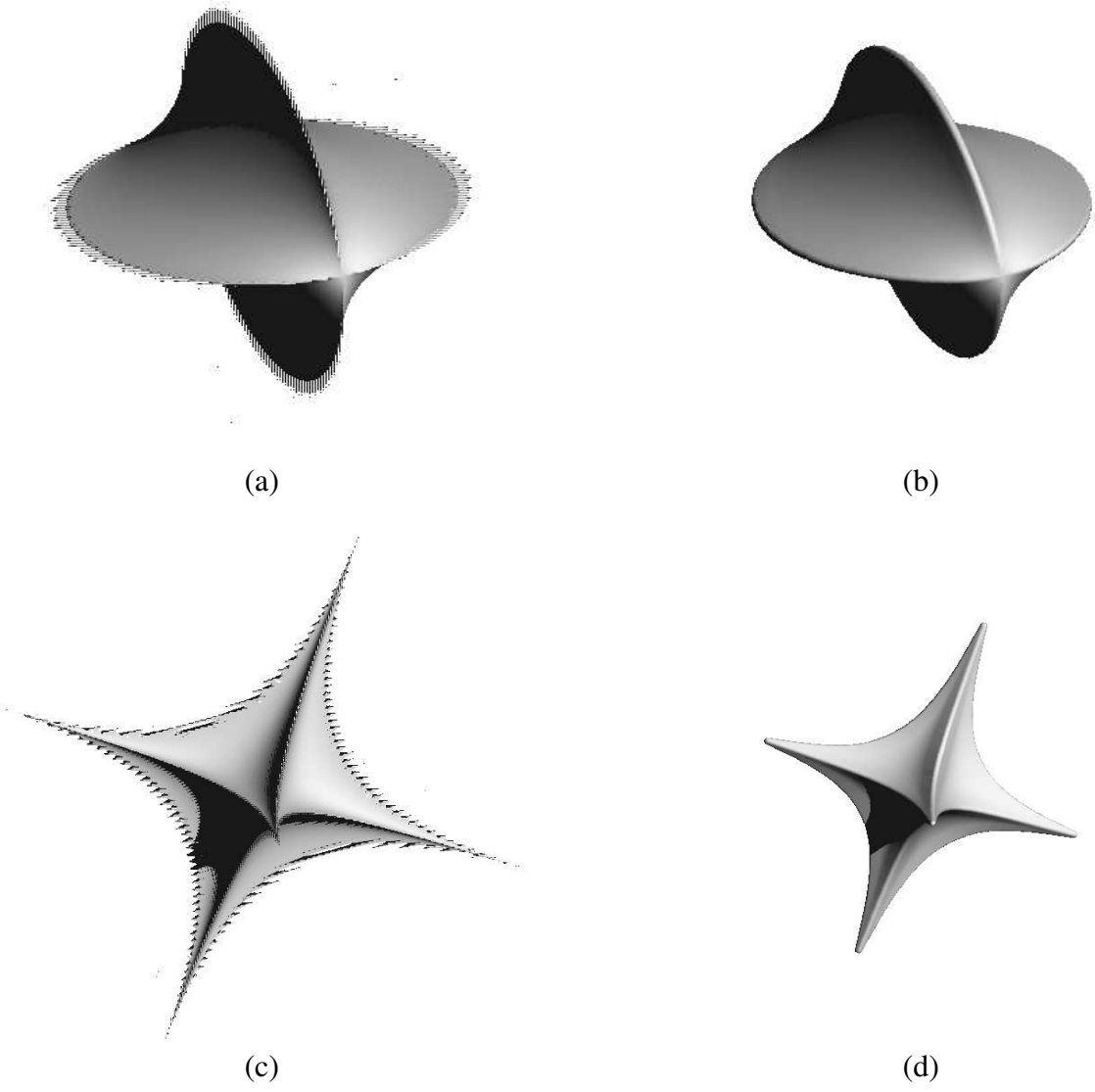
This problem calls for a next deeper research to extend more and more the set of solids, which can be handled. Presumably, it is suitable to replace the heuristics with a more reliable technique and to generalize the algorithm to make it able to process solids also with non-convex vertices.



(a) (b)

(c) (d)

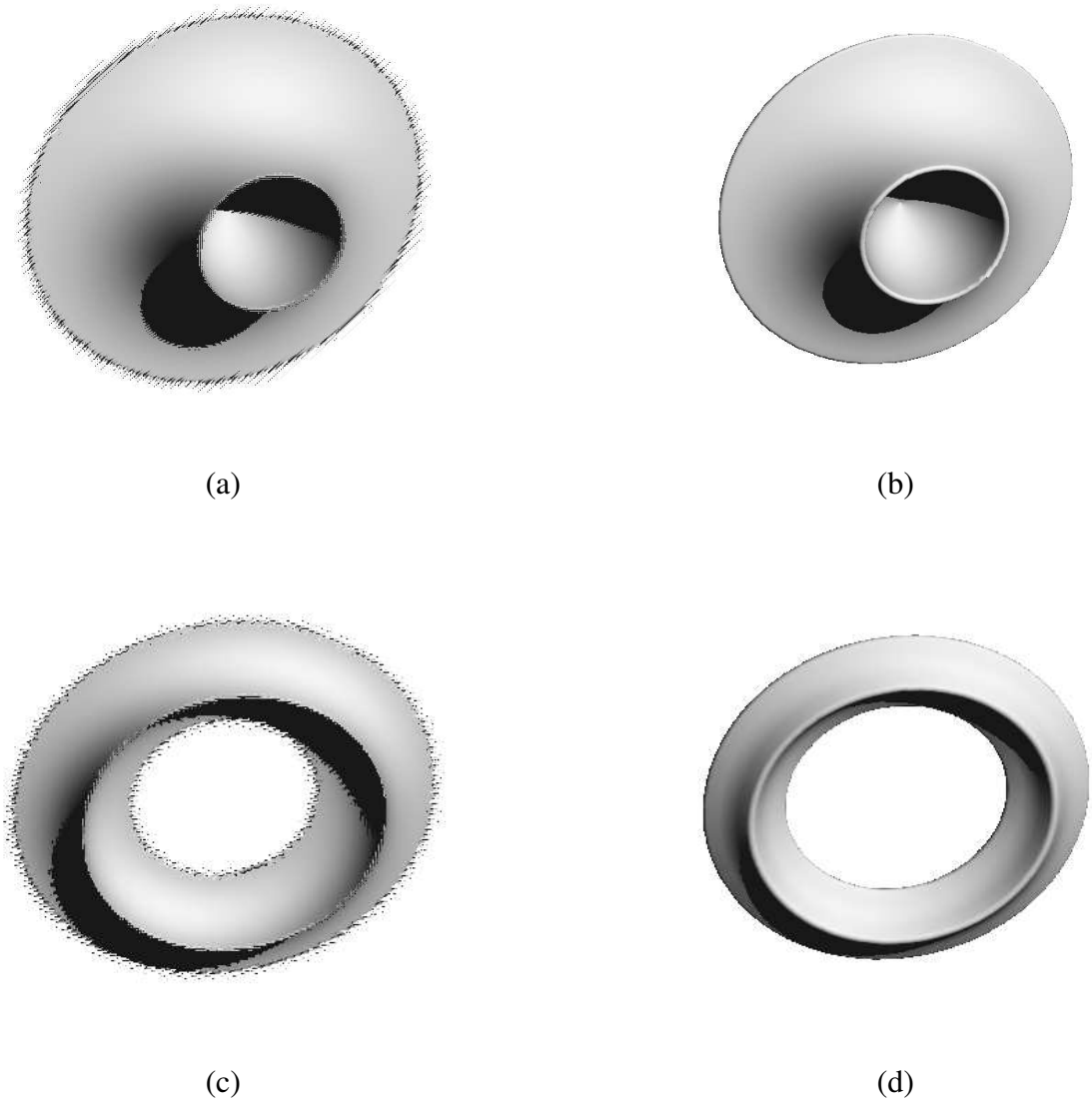
Figure 4.8: *Comparison of voxelized solids — cube and regular octahedron*
*Solids voxelized in the common way (left column) and by the SDCM (right column).*



(a) (b)

(c) (d)

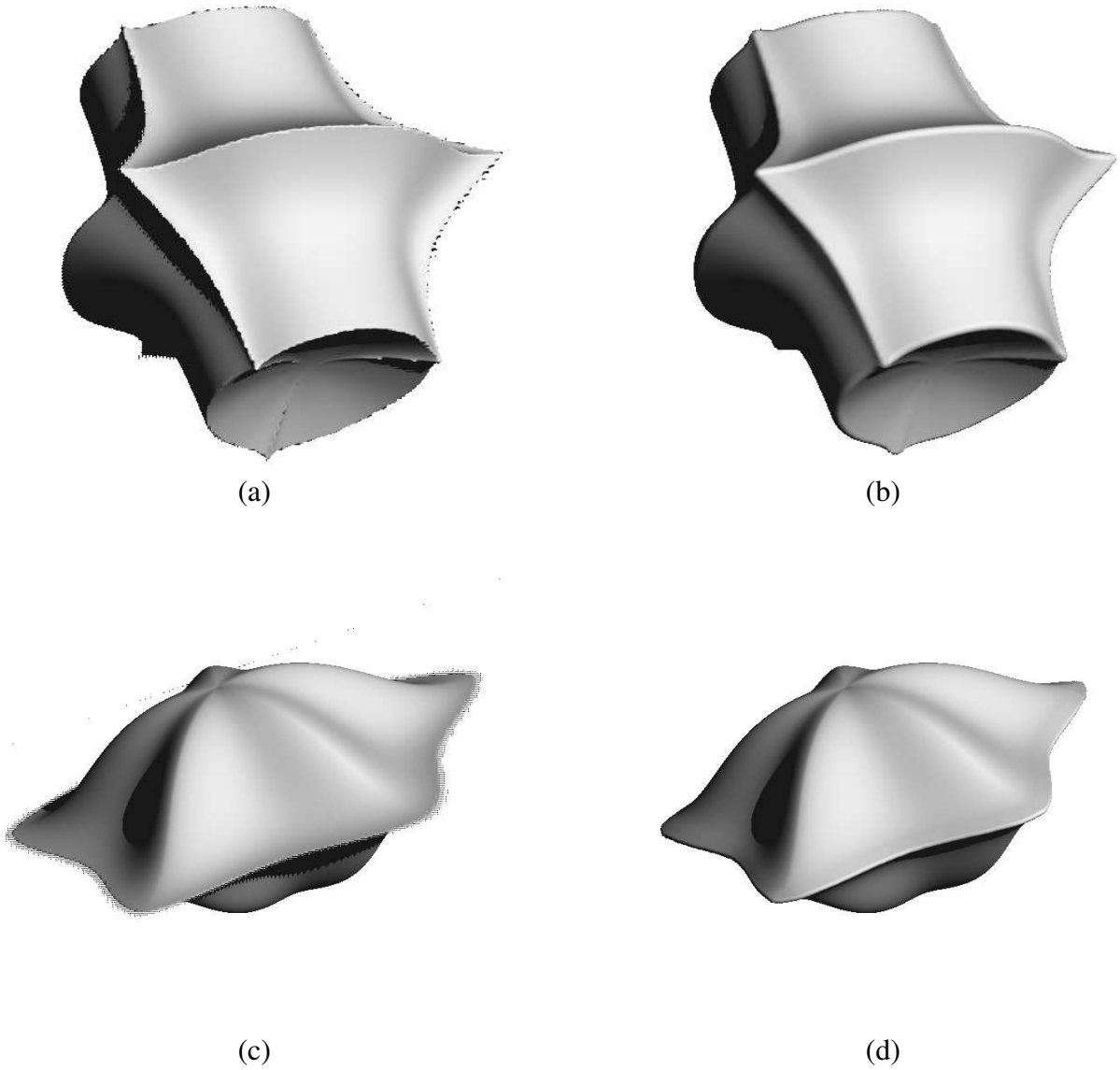
Figure 4.9: *Comparison of voxelized solids — superellipsoids*
*Solids voxelized in the common way (left column) and by the SDCM (right column).*



(a) (b)

(c) (d)

Figure 4.10: *Comparison of voxelized solids — supertoroids*
*Solids voxelized in the common way (left column) and by the SDCM (right column).*



(a) (b)

(c) (d)

Figure 4.11: *Comparison of voxelized solids — supershapes*
*Solids voxelized in the common way (left column) and by the SDCM (right column).*

# Chapter 5

# Implementation

## 5.1 The `vxt` Library

The starting point of this rigorous thesis has been the `vxt` library written in the `C++` language. Its core is formed by two hierarchies of classes derived from main classes:

**`vxtObject3D`** It represents various types of objects and voxelization techniques for them.

**`vxtGrid3D`** It represents various types of 3D grids.

The voxelization runs so that the object calls the method `Voxelize()` with a grid as the parameter. The class `vxtObject3D` is divided into two subclasses:

**`vxtPrimitive`** It represents simple objects, which play a roll of basic elements.

**`vxtAggregate`** It represents complex objects composed of the simple ones using CSG operations.

CSG operations are performed at two levels:

1. The object calls the method `Voxelize()` with a grid and a CSG operator as parameters. The information, created by combination of the original value from the grid with the new value obtained by the voxelization of given object, is written into the grid.

2. Grid *M* calls the method `Merge()` with an other grid *N* and a CSG operator as parameters. The result is written into *M*.

The class `vxtGrid3D` has subclasses `vxtVolume<T>` where `T` is a template parameter that determines the precision of the density representation.

The `vxt` library is described in [15] in more detail.

## 5.2 The vxtRL Library

The new vxtRL library has been created on the basis of the original vxt library making these modifications:

- The hierarchy under class vxtGrid3D has been radically changed to enable representation of both compressed and uncompressed volumes with various types of voxel.
- An alternative voxelization method (using sweeping planes) has been created for the class vxtImplicitSolid (a subclass of vxtPrimitive).
- There has been added three new subclasses under the class vxtImplicitSolid:
  - SolidCylinder — for representation of a solid bounded by the revolving cylindrical surface,
  - vxtSToroid — for representation of supertoroids,
  - vxtSShape — for representation of supershapes.
- The voxelization method in the class vxtAggregate has been modified to be suitable for more sophisticated CSG operations, too.
- There has been created a new class vxtSpecVolume for representation of an auxiliary structure that is used by active front propagation described in Chapter 4.

The core of the hierarchy for volume representation consists of following classes:

**vxtRLVolume<T>**

- It serves for the representation of a volume that is compressed using the modified run-length encoding (section 2.1).
- It works with an abstract voxel (template parameter T) that needs to have just operations ==, != and default inside and outside value defined.
- The access to data can be done by voxels, blocks, rows or slices.
- It exploits an auxiliary class vxtRLRow<T>.

**vxtRLRow<T>**

- It serves for the representation of a compressed row.
- T is the type of voxel.
- It contains methods for construction, updating and destruction of the row.
- The access to data can be done by voxels, segments or the whole row.
- For testing purposes, there are also implemented method for computing memory occupied by the row and method for detection of the row structure.

**vxtVoxel<T>**

- It serves for the representation of a general voxel (section 2.2). There are three subclasses derived from this class:

  **vxtPlainVoxel<T>** It stores just density.

  **vxtGradVoxel<T,G>** It stores both density and gradient (three components).

  **vxtSphGradVoxel<T,G>** It stores both density and gradient using spherical coordinates (two components).

  T, G are data types used for the representation of density and gradient. They can be one of these values:

  **f3dUChar** 1 byte

  **f3dUInt16** 2 bytes

  **f3dFloat** 4 bytes

- The class contains methods for reading and writing from/to voxels.

**vxtGrid3D**

- It is an abstract class for the representation of a grid.
- There is a class vxtVolume<V,T,G> derived from it.

**vxtVolume<V,T,G>**

- It serves for the representation of an arbitrary volume.
- Template parameters V, T, G are for voxel, density and gradient, respectively.
- It works with density from interval $\langle 0, 1\rangle$ and gradient from $\langle -1, 1\rangle^3$.
- Methods for reading and writing from/to voxels work at this level (regardless of compression, type of voxel and the precision of data).
- It contains methods for interpolation of discrete data stored inside for the need of visualization.
- There are simple methods for rendering of the volume and also for saving and loading it to/from a file (in the f3d format).
- It exploits the class vxtAnyVolume<V,T,G>.

**vxtAnyVolume<V,T,G>**

- It serves as an interface between its subclasses vxtUncompVolume<V,T,G>, vxtCompRLVolume<V,T,G> and the class vxtVolume<V,T,G>.
- The conversion of density (gradient) from $\langle 0, 1\rangle$ ($\langle -1, 1\rangle^3$) to the appropriate range according to the type T (G) is performed here.
- The CSG operation at the voxel level (Chapter 3) is implemented here.

**vxtUncompVolume<V,T,G>**

- It stores the whole volume without compression. The class `f3dgrid` from the `f3d` library is exploited.
- Methods for reading and writing from/to voxel are realized by decomposition of the voxel into particular components.
- There are methods for saving and loading of data to/from a file, too.

**vxtCompRLVolume<V,T,G>**

- It stores the volume using the class `vxtRLVolume< V<T,G> >`
- Methods of the `f3d` library are exploited to save and load the data to/from a file.

Here follows an example, how a part of program using the `vxtRL` library can look like:

```
1. vxtGrid3D *volume = createGrid2(200, 200, 200,
   VXT_RL_COMPRESS, VXT_SPH_G_VOXEL, f3dUInt16Type,
   f3dUInt16Type);

2. SolidSphere sphere(0.4);

3. sphere.Voxelize(*volume, VXT_WRITE);

4. volume->viewX("image", 3, Vector3D(1,1,1));

5. volume->save("sphere");
```

In the first step, we create a compressed volume `volume` of dimensions $200 \times 200 \times 200$, which uses a voxel with gradient represented in spherical coordinates (i.e. two components) and both density and gradient are stored in two-bytes precision. In the second step, we define an object `sphere` with the relative radius $0,4$ (i.e. $0,4 \cdot 100$ VU in the grid) and centre in the middle of the scene. In the next step we voxelize the sphere into the volume. Then we create the image `image.ppm` as a projection in the direction of $x$-axis where the scene is illuminated from the direction $(1, 1, 1)$. Finally, we save the volume to files `sphere.f3d` (density) and `sphere_G.f3d` (gradient).

As we can see from the example, the manipulation with grids of various types is uniform, the definition of particular grid type is performed through the parameters of the function `createGrid2(...)`. There is an alternative approach possible:

```
vxtVolume<vxtSphGradVoxel, f3dUInt16, f3dUInt16> volume(200, 200,
200, VXT_RL_COMPRESS);
```

The result is the same as in the first step, except of the fact that now `volume` is directly the variable of the class type, not a pointer.

# Chapter 6

# Conclusion

In the submitted thesis, we have proposed a new technique to represent voxelized solids. The basic idea is to supplement normalized gradient to the representation of truncated distance fields and exploit this extra information for more precise reconstruction of given objects and realization of more sophisticated CSG operations. Our analyses have proved that it is most suitable to use two-bytes variables for both density and gradient storage. We can spare some memory for gradient storage using spherical coordinates. We have proposed and implemented a new compression method as a modification of well-known run-length encoding. This technique reaches quite proper compression ratio for certain class of scenes. We have showed that this compression can also speed up the voxelization process.

Furthermore, we have proposed and implemented a new method for realization of CSG operations with voxelized solids. It works at a voxel level without the necessity for reconstruction of continuous object models. The technique removes artifacts of straightforward volumetric CSG operations by taking conditions for object representability into account. The idea of our solution is to round edges and other sharp details.

Finally, we have been interested in voxelization of solids, which contain some sharp details and so they are not correctly representable in discrete volume data. We have proposed and implemented a new method SDCM, which solves this problem by rounding sharp details of voxelized solids — similarly, like the above mentioned method of CSG operations. The SDCM has been successful for certain set of objects, but there is still a problem with stability.

This set of algorithms is implemented in the `vxtRL` library, which is an extension of the `vxt` package. In the future, we are planning to generalize the SDCM for a wide class of solids and reduce the instability of this process.

Results of Chapter 3 were published in the Computer Graphics International Conference 2004 with co-authors Leonid I. Dimitrov and Miloš Šrámek.

# Bibliography


[1] A. Kaufman, D. Cohen, and R. Yagel. Volume graphics. *IEEE Computer*, 26(7):51–64, July 1993.

[2] R. Yagel, D. Cohen, and A. Kaufman. Discrete ray tracing. *IEEE Computer Graphics and Applications*, 12(5):19–28, September 1992.

[3] S.W. Wang and A. Kaufman. Volume sampled voxelization of geometric primitives. In *Visualization '93*, pages 78–84, San Jose, CA, October 1993.

[4] M. Šrámek and A. Kaufman. Object voxelization by filtering. In *IEEE Symposium on Volume Visualization*, pages 111–118, 1998.

[5] S.F.F. Gibson. Using distance maps for accurate surface reconstruction in sampled volumes. In *IEEE Symposium on Volume Visualization*, pages 23–30, 1998.

[6] S. F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 249–254. ACM Press / ACM SIGGRAPH, 2000.

[7] Ronald N. Perry S. F. Frisken. Kizamu: A system for sculpting digital images. In Eugene Fiume, editor, *Siggraph 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 47–56. ACM Press / ACM SIGGRAPH, 2001.

[8] U. Tiede, K.-H. Höhne, M. Bomans, A. Pommert, M. Riemer, and G. Wiebecke. Investigation of medical 3D-rendering algorithms. *IEEE Computer Graphics and Applications*, 10(3):41–53, 1990.

[9] M.E. Goss. An adjustable gradient filter for volume visualization image enhancement. In *Proceedings of Graphics Interface '94*, pages 67–74, May 1994.

[10] J. Andreas Bærentzen, M. Šrámek, and Niels Jorgen Christensen. A morphological approach to voxelization of solids. In *The 8-th International Conference in Central Europe on Computer Graphics, Visualization and Digital Interactive Media 2000*, pages 44–51, Pilsen, Czech republic, 2000.

[11] Miloš Šrámek, Leonid I. Dimitrov, and J. Andreas Bærentzen. Correction of voxelization artifacts by revoxelization. In Klaus Mueller and Arie Kaufman, editors, *Volume Graphics'01, Proceedings of the Joint IEEE TVCG and Eurographics Workshop*, pages 265–275, Stony Brook, NY, June 21–22, 2001.

[12] J. Andreas Bærentzen and N. J. Christensen. A technique for volumentric CSG based on morphology. In Klaus Mueller, editor, *Volume Graphics'01*, pages 71–79, Stony Brook, NY, June 2002.

[13] Claudio Montani and Roberto Scopigno. Rendering volumetric data using sticks representation scheme. In *Proceedings of the 1990 workshop on Volume visualization*, pages 87–93. ACM Press, 1990.

[14] Naeem Shareef and Roni Yagel. Rapid Previewing via Volume-based Solid Modeling. In *The Third Symposium on Solid Modeling and Applications, SOLID MODELING '95*, pages 281–292, Salt Lake City, Utah, May 1995.

[15] M. Šrámek and A. Kaufman. `vxt`: a c++ class library for object voxelization. In Min Chen, Arie E. Kaufman, and Roni Yagel, editors, *Volume Graphics*, pages 119–134. Springer Verlag, London, 2000.

[16] D.E. Breen, S. Mauch, and R.T. Whitaker. 3D scan conversion of CSG models into distance volume. In *IEEE Symposium on Volume Visualization*, pages 7–14, 1998.

[17] Andreas Bærentzen. Octree based volume sculpting. In C.M. Wittenbrink and A. Varshney, editors, *LBHT Proceedings of IEEE Visualization 98*, October 1998.

[18] H. Löffelmann and E. Gröller. Parameterizing superquadrics. In V Skala, editor, *Proceedings of WSCG'95, the 3-rd International Conference in Central Europe on Computer Graphics and Visualization 1995*, pages 162–172, 1995.

[19] Aleš Jaklič, Aleš Leonardis, and Franc Solina. *Segmentation and recovery of superquadrics: computational imaging and vision*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[20] J. Gielis. A generic geometric transformation that unifies a wide range of natural and abstract shapes. *American Journal of Botany*, 90:333–338, 2003.

[21] J. Gielis, B. Beirinckx, and E. Bastiaens. Superquadrics with rational and irrational symmetry. In *Symposium on Solid Modeling and Applications*, pages 262–265. ACM, 2003.

[22] J. A. Bærentzen. On the implementation of fast marching methods for 3D lattices. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2001.