Examensarbete
LITH-ITN-MT-EX--06/049--SE

Blending of implicit models by means of anisotropic diffusion

Henrik Wrangel

2006-11-15

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Science and Technology
Linköpings Universitet
SE-601 74 Norrköping, Sweden

Institutionen för teknik och naturvetenskap
Linköpings Universitet
601 74 Norrköping

LITH-ITN-MT-EX--06/049--SE

Blending of implicit models by means of anisotropic diffusion

Examensarbete utfört i medieteknik
vid Linköpings Tekniska Högskola, Campus Norrköping

Henrik Wrangel

Handledare Ken Museth
Examinator Ken Museth

Norrköping 2006-11-15

Avdelning, Institution
Division, Department

Institutionen för teknik och naturvetenskap

Department of Science and Technology

Datum
Date

2006-11-15

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English

☐ _____

Rapporttyp
Report category

Examensarbete
☐ B-uppsats
☐ C-uppsats
☒ D-uppsats

☐ _____

ISBN

ISRN **LITH-ITN-MT-EX--06/049--SE**

Serietitel och serienummer
Title of series, numbering

ISSN

URL för elektronisk version

Titel
Title

Blending of implicit models by means of anisotropic diffusion

Författare
Author

Henrik Wrangel

Sammanfattning
Abstract

This thesis presents a novel approach for blending of level set models. The proposed method blends the intersection area of two models by means of anisotropic diffusion, i.e. anisotropic Gaussian low pass filtering. Combing different models to build new ones is a common and intuitive way of modeling, however merging two models tends to lead to C1 discontinuities and some times even aliasing artifacts along the intersection of the two models. This thesis will show that blending by means of anisotropic diffusion solves theses issues and highly reduces blending execution times compared to mean curvature flow based blending.

Nyckelord
Keyword

level sets, implicit surfaces, blending, anisotropic Gaussian filter

Abstract

This thesis presents a novel approach for blending of level set models. The proposed method blends the intersection area of two models by means of anisotropic diffusion, i.e. anisotropic Gaussian low pass filtering. Combing different models to build new ones is a common and intuitive way of modeling, however merging two models tends to lead to C^1 discontinuities and some times even aliasing artifacts along the intersection of the two models. This thesis will show that blending by means of anisotropic diffusion solves theses issues and highly reduces blending execution times compared to mean curvature flow based blending.

Blending of Implicit Models by Means of Anisotropic Diffusion

Master's Thesis in Media Technology

Henrik Wrangel

Supervisor:
Prof. Ken Museth

Department of Science and Technology
Linköping University, Sweden
2006

Acknowledgements

I would like to thank Prof. Ken Museth for supervising this thesis, Anders Brodersen for the geo texture bunny volumes and Ola Nilsson for helping out with the pbrt level set plug-in.

Table of Contents

1 Introduction

This thesis is the result of the work carried out at the Graphics Group of the Department of Science and Technology at Linköping Institute of Technology. It serves as a fulfillment of a Master of Science degree in Media Technology and Engineering.

1.1 Purpose and Motivation

The main purpose for this thesis has been to develop a method to blend 3D models represented as level set by means of anisotropic diffusion and to do it faster than previously known methods. It has also been an aim to incorporate the blending algorithm with the compact and efficient level set data structure, DT-Grid [5]. Secondary and as an extension of the blending algorithm it has been desired to evaluate the anisotropic Gaussian diffusion algorithm for global surface smoothing.

There exists many algorithms today for editing polygonal meshes and parametric surfaces. Lately the level set representation of 3D models has gained popularity since more and more surface editing algorithms [1] has been developed. Level set models are deformable implicit surfaces uniformly sampled on a volumetric grid [2]. These level set models offer several benefits compared to polygonal meshes and parametric surfaces. 1) The surfaces are guaranteed to be closed and non-self- intersecting, which makes them physically realizable. 2) Level set models can very easily change topological genus and 3) they are free of edge connectivity and mesh quality problems.

An easy and intuitive way to create models is to use the cut and paste operation. The basic idea is to create a new model by merging parts from different models. These operations are easy to implement for level set models. However, to make the new model look like a homogenous object and not a collection of objects, the transitions need to be smooth. Another area of interest is geometric texturing [3], where smooth transitions are needed to make the textures look integrated with the original model. A method for automatic blending of level set do already exists [1] but it is slow since it requires that one solve partial differential equations.

The creation of detailed complex 3D models can be a cumbersome and tedious project. One way of creating 3D models of real world objects is 3D photography [4], i.e. 3D scanning. The digital representations can be converted to polygonal meshes, parametric surfaces or iso surfaces, i.e. level sets. The scanning process is rarely perfect and errors and artifacts are introduced in digital representation. Some of theses errors may be removed by surface smoothing. It would be interesting to see how the surface might be enhanced by the anisotropic Gaussian diffusion.

1.2 Problem Description

The following problems have been identified.

- **Artifacts**. When merging two level set surfaces unwanted artifacts might arise along the intersection of the two input surfaces. These artifacts should be removed.
- **Sharp intersections**. The CSG union operation of two level set generates sharp intersections between the two input surfaces. This is often an unwanted feature. Therefore a blending function that integrates the two

surfaces smoothly is sought-after. It is also desirable that the user can choose the amount of blending.
- **Speed**. There already exist methods for automatic local blending of level sets. Therefore it is important that a new blending function is faster than any previously developed methods.

1.3 Objectives

From the problems identified in the previous section, the following general objectives have been taken upon.

- Develop a method for blending surfaces, represented as level sets, by means of anisotropic diffusion.
- Evaluate the use of anisotropic diffusion as means for blending of level set surfaces.
- Evaluate the use of anisotropic diffusion as a mean for global surface smoothing.

The following requirement has been established for the proposed level set blending system.

- Full integration with the efficient level set data structure, DT-Grid [5].

1.4 Method

The method for blending was decided from the start and this thesis thereby also serves as an evaluation of anisotropic diffusion as means for blending level set surfaces. Two types of level set data structures have been used during the development of this thesis' blending function; full grid level sets [2] and the very compact and efficient DT-Grid level set data structure [5]. Initially all methods have been implemented and tested using the full grid data structure. Finally they have been ported to the DT-Grid data structure.

The development process has been of the incremental kind. First a blending function using isotropic diffusion was implemented. The implementation of the isotropic Gaussian diffusion blending showed that blending by means of anisotropic diffusion might be a satisfying solution to the blending problem, which led to an implementation of an anisotropic Gaussian convolution filter. Finally anisotropic diffusion was implemented and evaluated for global smoothing of level set surfaces.

1.5 Thesis Outline

The rest of this thesis is laid out as follows. In chapter 2, the concept of level sets and different level set data structures will be explained. Chapter 2 also addresses related surface editing work, mainly with implicit modeling in mind. Chapter 3 will give a technical description of the ideas behind Gaussian diffusion as a mean for blending and smoothing of level set surfaces. The implementation of anisotropic Gaussian diffusion for level set surfaces is explained in chapter 4. The results will be presented in chapter 5 and finally discussed in chapter 6. It is assumed that the reader has a basic knowledge of computer graphics, calculus and programming.

2 Background and Related Work

This chapter will describe the general concept of level sets and the very compact and efficient DT-Grid level set data structure. Further, previous work within the field of surface smoothing and blending will be discussed.

2.1 Level sets

A level set is an iso surface, or an iso contour in 2D, of an implicit function ϕ for a specified iso value C. Given a function ϕ: $\Re^n \rightarrow \Re$, the level set surface S is defined as

$$S \equiv \left\{ \vec{x} \middle| \phi(\vec{x}) = C \right\}, \qquad \vec{x} \in \Re^n \tag{2.1}$$

In other words, the level set is the set of points in $\Re^n$ that satisfies the equation

$$\phi(\boldsymbol{x}) = C \tag{2.2}$$

This means that the level set is defined as all the points in $\Re^n$ for which the implicit function ϕ is equal to the specified iso value. This is best illustrated with the level set $\phi(\boldsymbol{x}) = x^2 + y^2 = C$, which defines a circle centered in (0, 0). The set of points that make up the level set is then obtained by solving $x^2 + y^2 = C$, which yields in a circle with a radius, $r = sqrt(C)$. When evaluating all points within the area containing the level set three different cases will arise.

- $\phi(\boldsymbol{x}) > C$. The point $\boldsymbol{x}$ is outside of the surface.
- $\phi(\boldsymbol{x}) < C$. The point $\boldsymbol{x}$ is inside of the surface.
- $\phi(\boldsymbol{x}) = C$. The point $\boldsymbol{x}$ is on the surface.

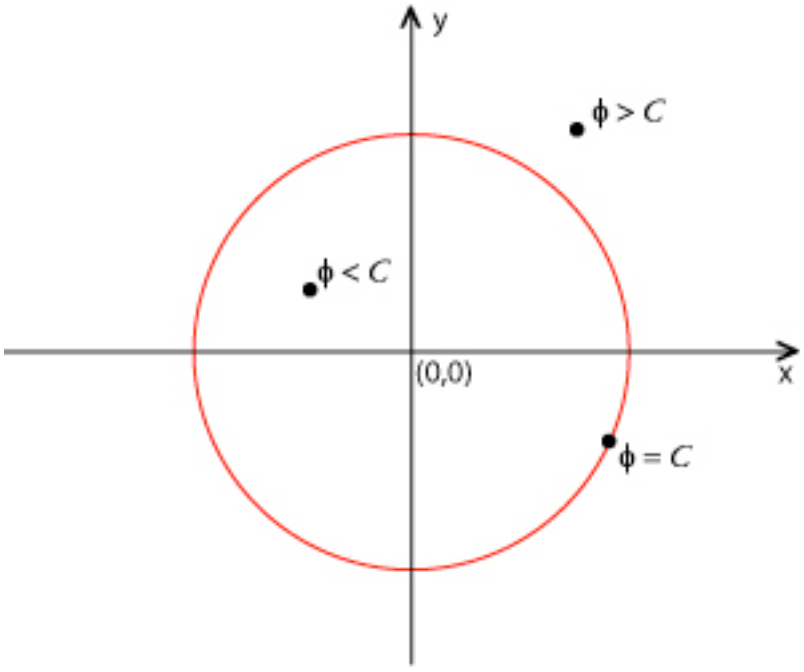


Fig 2.1. Circle level set. Points where ϕ is larger than C are outside the level set, points where ϕ is smaller than C are inside the level set and only the point where ϕ=C defines the level set.

This is an important property of level sets that makes it easy to classify points in space as being inside, outside or on the interface. It is common practice to define the level

set as the iso surface $\phi = 0$. Now all points inside the surface have negative values and points outside the surface have positive values.

It is often very important to calculate the gradient of a level set. For an n dimensional level set it is defined as

$$\nabla\phi \equiv \left(\frac{\partial\phi}{\partial x_1}, \frac{\partial\phi}{\partial x_2}, \ldots, \frac{\partial\phi}{\partial x_n}\right) \tag{2.3}$$

The gradient of a level set is always perpendicular to the iso surfaces and points in the direction of maximum increasing ϕ, which yield in the following expression for the surface normal.

$$n = \pm\frac{\nabla\phi}{|\nabla\phi|} \tag{2.4}$$

The specific level set representation used for this thesis is a so-called signed Euclidian distance function. It is an implicit function that always returns the shortest Euclidian distance to the interface. It is defined as

$$\phi(\vec{x}) = \min(|\vec{x} - \vec{x}_s|), \text{ where } \mathbf{x}_s \text{ is a point on the interface.} \tag{2.5}$$

$$|\nabla\phi| = 1 \tag{2.6}$$

$$x \in \Re^n$$

To be more specific, the level sets of interest for this thesis are signed Euclidian distance functions. Since the interface is defined as the zero level set, the level sets get the following property: all points inside the interface, $\phi < 0$, have a negative distance to the interface, points outside the interface, $\phi > 0$, have a positive distance to the interface. Points on the interface of course have zero distance to the interface.

2.1.1 Dynamic level sets

Deformation of level set models is the most important part of the level set concept and is truly one of its strongholds. In fact it is the main reason why it is a popular surface representation. To be able to deform level sets, a time parameter is added to the previous definition of the interface.

$$S \equiv \left\{\vec{x}(t) \middle| \phi(\vec{x}(t), t) = C\right\} \tag{2.7}$$

By differentiating the above definition of the interface one will see how the interface may be deformed in its local normal direction.

$$\frac{d}{dt}\phi(\vec{x}(t),t) = \frac{d}{dt}C$$

$$\frac{\partial\phi}{\partial t} + \frac{\partial\phi}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial\phi}{\partial x_2}\frac{\partial x_2}{\partial t} + \ldots + \frac{\partial\phi}{\partial x_n}\frac{\partial x_n}{\partial t} = 0$$

$$\frac{\partial\phi}{\partial t} = -\nabla\phi \bullet \frac{d\vec{x}}{dt}$$

$$\frac{\partial\phi}{\partial t} = -\left|\nabla\phi\right| \cdot \frac{\nabla\phi}{\left|\nabla\phi\right|} \bullet \frac{d\vec{x}}{dt}$$

$$\frac{\partial\phi}{\partial t} = -\left|\nabla\phi\right| \cdot \vec{n} \bullet \frac{d\vec{x}}{dt} \qquad (2.8)$$

$$\frac{\partial\phi}{\partial t} = -\left|\nabla\phi\right| \cdot F(x,\vec{n},\ldots) \qquad (2.9)$$

The *F* term in eq. 2.9 is called *the speed function* and controls the movement of the interface in the direction of the normal at each grid point. The type of speed function to use depends on what kind of surface deformation that is sought after and is user defined. The most basic case for propagating or moving a surface is when the speed function is equal to one. This will make dϕ/*dt* equal to minus one, which corresponds to erosion of the level set interface. A more complex example would be to morph one shape to another, which would correspond to a speed function equal to the final shape level set [6]. By defining a proper speed function it is also possible to smoothly blend two level set models, which is briefly explained in section 2.3.1. However, it will be shown that blending level sets by anisotropic diffusion does not require a definition of a speed function in contrast to all other surface deformation methods.

2.1.2 Level set model representations

Level sets are stored as a sampling of an implicit function. This means that in 2D a level set contour is stored in a two-dimensional uniform grid and in 3D a surface is stored as a volume. This makes the memory footprint of level sets a lot larger than for parametric surfaces.

A lot of work has been devoted to making level set methods and level set storage requirements more efficient. The main problem for level set methods is that since it adds one extra dimension it is computationally very heavy. It requires that level set computations for a surface involves all voxels within the volume containing the zero level set interface. One is actually tracking all the level sets not just the one of interest, the zero level set. To solve this problem, the narrow band concept was introduced in [7, 8]. The concept of the narrow band technique is that level set computations are restricted to a narrow band of voxels immediately surrounding the interface. This reduces the time complexity of level set computations for a surface from $O(n^3)$ to $O(n^2)$.

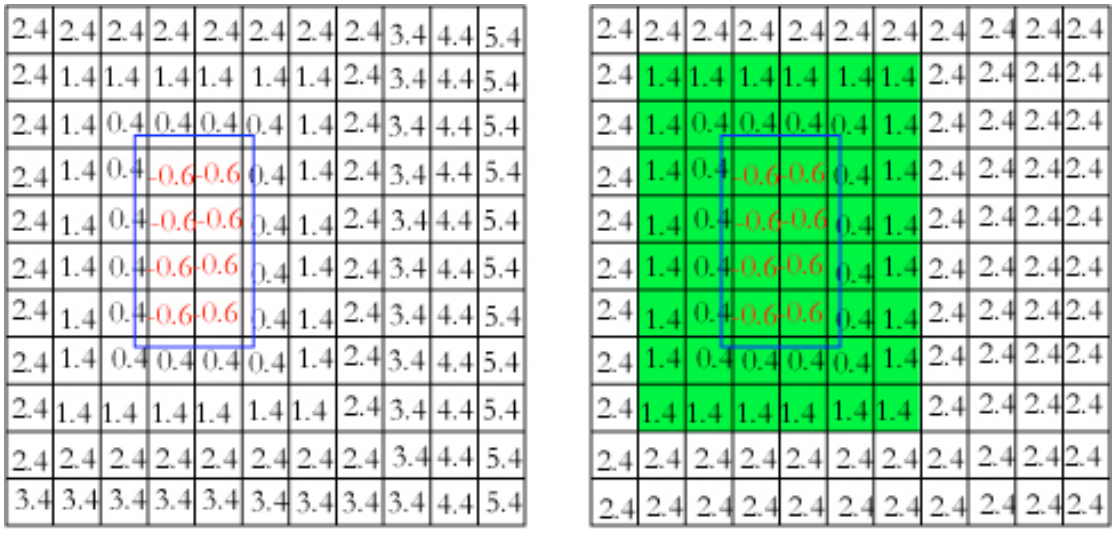


Fig 2.2. 2D Uniform sampling of level sets. A) Euclidian distance field sampled on uniform grid b) Narrowband representation of a Euclidian distance filed. Voxels that are parts of the narrow band are marked with green. The blue curve represents the interface.

One problem still remains, the level set still makes use of a full grid, which means that the memory footprint of the level set surface is still $O(n^3)$. A solution to the storage problem is the tree implementation, which stores the level set surface in an octree data structure [9, 10], allowing higher resolution around the interface. This reduces the storage requirements to $O(n^2)$. Unfortunately, due to the hierarchical data structure, the time complexity of the access operation is reduces to O(nlog n), which in turn effects the performance of the level set computations. Another drawback of the tree data structure is that the non-uniform sampling makes impossible to use higher order finite difference upwind schemes.

2.1.3 The DT-Grid

As mentioned in the previous section there exists level set data structures that *either* improves the performance of level set methods or decrease the memory footprint of level set models. The DT-Grid (Dynamic Tubular Grid) [5] brings one solution to this dilemma by offering a very compact and efficient data structure that reduces the memory footprint *and* is efficient for level set methods.

The DT-Grid *only* stores values within a narrow band of the propagating surface, which makes the memory usage proportional to the surface itself instead of the containing volume. At the same time the level set surface is sampled on a dynamic uniform grid. This means that the grid is free from any boundary restrictions on surface expansion. In other words, the DT-Grid combines the best of two worlds, the compactness of the hierarchical tree implementations and the uniform grid from the narrow band implementations. A uniform grid is important since it allows the use of all important finite difference schemes already developed for uniform full grids. Moreover, uniform grids do not suffer from Lipschitz discontinuities that may arise from interpolation over non-uniform grids.

The nature of the DT-Grid does not allow constant time for random access of grid points. Even though random access is still very fast, this fact has lead to the introduction of the iterator concept. Iterators are a fundamental part of the DT-Grid data structure. The iterator is a construct that provides constant access time for grid points when they are accessed sequentially. Constant access time may also be

obtained for the iterator's neighbors by using something called a stencil iterator. Constant neighbor access time is important for different kinds of finite difference schemes. The stencil iterator allows for constant neighbor access time on average when iterating over the entire tubular grid. The stencil iterator is in fact not a single iterator but a collection of iterators, one iterator for each grid point within the specified stencil. The DT-Grid allows user-defined stencils and tubes, narrow band of a certain with around the interface, for the iterator, which has been of most importance for this thesis.

2.2 Constructive Solid Geometry

One major advantage of level set models is that they support straightforward Constructive Solid Geometry (CSG) modeling, i.e. copy, cut and paste operations. This has been of fundamental importance for this thesis. CSG modeling is a very intuitive and simple way to create new and interesting models or for renovating pre-existing models. For signed distance fields CSG operations are defined as a set of Boolean operators, more exactly as max and min operations. These operations are described in table 2.1.

Table 2.1 CSG operations. Positive outside sign and negative inside sign are assumed for the level set *A* and *B*.

Union,	$F_{A \cup B}$	*Min* (F_A, F_B)
Intersection,	$F_{A \cap B}$	*Max* (F_A, F_B)
Difference,	F_{A-B}	*Max* (F_A, $-F_B$)

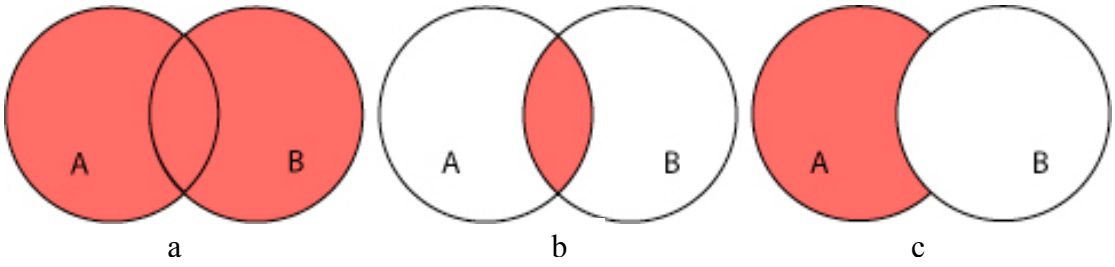


Fig 2.3. CSG operations a) Union b) Intersection c) Difference

2.3 Blending implicit surfaces

There exists a lot of related work for blending of implicit functions. As always all methods have their drawbacks and advantages. When it comes to modeling of implicit surfaces they can be categorized in to two groups. Analytical implicit functions that are continuous functions, for example the function describing a sphere.

$$r^2 = x^2 + y^2 + z^2 \tag{2.10}$$

The other type of implicit functions is called numerical implicit functions and is discretely sampled functions, for example level sets. Blending of implicit functions differ depending on the type of the implicit function. This section will first discuss related work for blending of level sets and secondary discuss the topic of blending analytical implicit functions.

2.3.1 Blending of level sets

The only previous work known on local blending of level set is the mean curvature flow blending method introduced in 2002 by K. Museth et al [1] The mean curvature flow method produces blending of two level sets by means of moving the surface until the mean curvature around the intersection region reaches a specified value. It is done by constructing a specialized speed function and solving the level set equation. The blending speed function takes several parameters into account and is defined as:

$$F_{blend}(\mathbf{x}, n, \phi) = \alpha Dp(d)C(K)K \tag{2.11}$$

Where α is a user defined positive scalar that controls the rate of convergence of the level set surface. *Dp(d)* is a distance based cut-off function dependent on the distance *d* from the level set surface to the intersection curve. *C(K)* is another cut-off function that lets the user determine the upper and lower band of the curvature value. *K* is a curvature measure of the level set surface. The use of cut-off functions to control the amount of smoothing is a feature shared by the anisotropic Gaussian diffusion blending operator proposed in this thesis. The curvature parameter is found by calculating the eigenvalues of the *shape matrix* [11]. For implicit surfaces, the shape matrix is defined as the derivative of the surface normals projected onto the tangent plane of the surface. The mean curvature can be written as:

$$K = \frac{1}{2}\nabla \bullet n = \frac{1}{2}\nabla \bullet \frac{\nabla \phi}{|\nabla \phi|} \tag{2.12}$$

The mean curvature flow blending operator also lets the user constrain the direction of the surface's motion, i.e. controlling if material is added or removed from the model. This is accomplished by clamping positive motion to zero for removing material and vice versa for adding material. Even though the mean curvature flow based blending operator produces great results it has the disadvantage of being slow. This is due to the fact that it is required to solve the level set equation and the required propagation and reinitializing of the level set that follows. As will be shown, this is not required for blending by means of anisotropic diffusion, which allows it to be a lot faster.

2.3.2 Blending of analytical implicit functions

When it comes to blending of implicit functions there exists a wide range of blending functions. This chapter will highlight some common blending methods that use density functions and R-functions.

One way of blending implicit functions is to blend so called density functions. By converting an implicit function to a density function it becomes easier to blend the implicit models with different blending functions. The density function is defined as:

$D(\mathbf{x}) > 1$ if $\mathbf{x}$ is inside the surface
$D(\mathbf{x}) = 1$ if $\mathbf{x}$ is on the surface
$D(\mathbf{x}) \in [0,1[$ if $\mathbf{x}$ is outside the surface

A signed distance function can be turned into a density function by the following transfer function,

$$D_i = exp\{-F_i(\mathbf{x})\} \tag{2.13}$$

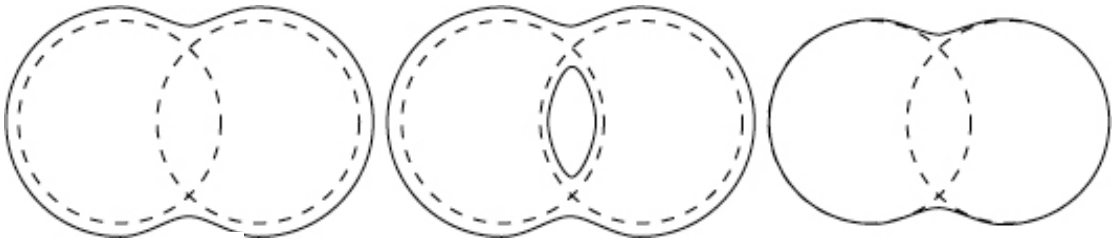
if the implicit function *i* is said to have a negative inside,

$$i = \left\{ \vec{x} \in \Re^3 \middle| F_i(\vec{x}) \le 0 \right\}. \tag{2.14}$$

Three common blending functions for density functions are the linear blending, the hyperbolic blending and the super-elliptic blending functions. They are defined as in table 2.2.

Table 2.2 Blending density functions

Linear blending	$B_L(D_1(x)\ldots D_k(x)) = \sum_{i=1}^{k} D_i(x) - 1$
Hyperbolic blending	$B_H(D_1(x)\ldots D_k(x)) = \prod_{i=1}^{k} D_i(x) - 1$
Super-elliptic blending	$B_S(D_1(x)\ldots D_k(x)) = 1 - (\sum_{i=1}^{k} Max(0,1-\alpha D_i(x))^{\beta})^{1/\beta}$



a b c

Fig. 2.4 Blending density functions a) Linear blend b) Hyperbolic blend c) Super-Elliptic blend

Another way to blend implicit models is to use R-functions. R-functions are real-valued functions, $f(x_1, x_2, \ldots, x_n)$, whose sign is completely determined by the sign of its arguments x_i. The R-function may be viewed as a logic switch. If negative values correspond to false and positive values correspond to true it may be used as a logic switch for Boolean operations of implicit surfaces with a positive inside. Take the intersection case for example that corresponds to a logic AND switch, if the R-function takes two implicit surfaces as input it will return true (inside) only if its two input surfaces are true, i.e. inside both surfaces. For the solid object A defined as,

$$A = \left\{ \vec{x} \in \Re^3 \middle| F_A(\vec{x}) \ge 0 \right\}, \tag{2.15}$$

a popular R-functions can be described as:

$$F_A \cup F_B = \frac{1}{1+\alpha}(F_A + F_B + \sqrt{F_A^2 + F_B^2 - 2\alpha F_A F_B}) \tag{2.16}$$

$$F_A \cup F_B = \frac{1}{1+\alpha}(F_A + F_B - \sqrt{F_A^2 + F_B^2 - 2\alpha F_A F_B}). \tag{2.17}$$

Where α is a continuous function $\alpha = \beta(F_A, F_B)$ that satisfies the following conditions:

$$-1 < \beta(F_A, F_B) \leq 1, \tag{2.18}$$

$$\beta(F_A, F_B) = \beta(F_B, F_A) = \beta(-F_A, F_B) = \beta(-F_B, F_A). \tag{2.19}$$

One simple case of the above R-functions are when $\alpha = 1$, which yields in simple Min and Max operations.

$$R^{\pm}(F_A F_B) = \frac{1}{2}(F_A + F_B \pm \sqrt{F_A^2 + F_B^2 - 2F_A F_B}) = \frac{1}{2}(F_A + F_B \pm \sqrt{(F_A - F_B)^2})$$
$$= \frac{1}{2}(F_A + F_B \pm |F_A - F_B|) \Rightarrow \tag{2.20}$$
$$R^{+}(F_A F_B) = Max(F_A F_B),$$
$$R^{-}(F_A F_B) = Min(F_A F_B)$$

These Min and Max operations still have the problem of having C^1 discontinuity where $F_A = F_B$. However there are other types of R-functions that provide C^n continuity along the entire boundary.

$$F_A \cup F_B = (F_A + F_B + \sqrt{F_A^2 + F_B^2})(F_A^2 + F_B^2)^{n/2} \tag{2.21}$$

$$F_A \cap F_B = (F_A + F_B - \sqrt{F_A^2 + F_B^2})(F_A^2 + F_B^2)^{n/2} \tag{2.22}$$

The blending functions described in this section could be applied, with some modifications, to level sets. The main draw back is that these blending functions are global. This means that they act on the entire surface and may blend in areas where no blending is desired, e.g. where two surfaces are close to each other. Their blending parameters are also not very intuitive, which makes it hard to find a desired blending.

2.4 Surface smoothing

Mesh smoothing or in more general terms, surface smoothing, is an important topic and a lot of research has been done in the field. The reason for its importance is that computer graphics models constructed from real world data, e.g. 3D photography, contains undesirable noise. It is desirable to remove the noise and at the same time cause minimal damage to the underlying geometry of the object. This section will focus on level set smoothing methods but also briefly discuss smoothing of meshes.

2.4.1 Smoothing of level sets

There are two main methods for smoothing of level sets, curvature flow [1] and morphological operations [1]. The curvature based smoothing method is better suited for local smoothing since it its more computationally demanding than morphological operations.

The curvature based smoothing and sharpening operator makes use of the surface curvature, as implied by its name, to determine how to move the surface in order to make it smoother or sharper. It is the same idea as for curvature flow based blending, only with a different ROI. By specifying a maximum curvature value, the surface is moved until the curvature of the specified area is below the specified value for the entire region. It is done by defining a proper speed function that moves the level set surface in the direction of the local normal with a speed that is proportional to the local curvature. The method may also be constrained to only move the surface

inwards or outwards. It is constrained locally by defining the ROI with a quadric. Now only the surface within the quadric will be affected.

For global smoothing of level sets, the morphological opening and closing operations are probably more convenient. Opening and closing are different combinations of the morphological operations, dilation and erosion. A morphological opening is the combination of a dilation followed by erosion and the closing operation is the vice versa. Opening tends to remove fine pieces or thin appendages while closing fills small gaps or holes. Morphological operations adapt well to level sets as dilation can be seen as an offset of the level set to an iso surface outside of the zero level set. Erosion can in turn be described as moving the surface to an iso surface inside of the zero level set. The morphological opening process may be seen as a four-step process 1) offset the surface inwards 2) reinitialize the level set to a signed distance function with respect to the new surface 3) offset the surface outwards with the same amount as the inward offset 4) reinitialize the level set to a signed distance function. The procedure for morphological closing is the same but reversed. The morphological operations may be implemented for level set by solving a special form of the level set equation, the Ekonial equation,

$$\frac{d\phi}{dt} = \pm|\nabla\phi| \tag{2.23}$$

2.4.2 Mesh smoothing

When it comes to smoothing of meshes, an extensive amount of work has been done. Some of the most common approaches are the Laplacian smoothing method[13, 14], the bilaplacian smoothing flow [14], the diffusion and curvature flow method [15] and the Taubin λ|μ scheme [13]. A more interesting method for smoothing polygonal meshes from this thesis' point of view is the adaptive and anisotropic Gaussian filtering method [16]. It is a three-step process where first an optimal scale of the anisotropic Gaussian filter is calculated for each vertex normal, the vertex normals are then smoothed with the appropriate filter kernel. In the final step the position of all vertices are updated to fit the mesh to the field of smoothed mesh normals. The design of the adaptive anisotropic Gaussian filter is based on the method described in [17], where they propose an adaptive filter that enhance edges and corners. The main idea is to scale the size of the filter kernel depending on the magnitude of the local gradient. The larger the magnitude of the gradient is the less the image is diffused at that point and vice versa. The idea to use this technique for mesh smoothing has shown to produce visually very pleasing results.

3 Blending and smoothing by means of Gaussian diffusion

This chapter will describe how Gaussian diffusion can be used for smoothing surfaces, both globally and locally. First there will be a description of how to use Gaussian diffusion for blending and surface smoothing. Thereafter a detailed description of Gaussian low pass filters with extra emphasis on non-orthogonal separation of the anisotropic Gaussian convolution filter will follow.

3.1 Blending

Level set models produced by combining two level sets by means of the CSG union operation tend to create sharp creases in the intersection of the two surfaces. Not only is the intersection sharp, but it may also suffer from artifacts due to aliasing effects from re-sampling. By performing a local blending operation around the intersection area, the surface will be drastically improved. This can be accomplished by means of mean curvature-based flow, but it will be shown that a more efficient method is anisotropic diffusion.

The anisotropic diffusion blending operation is an anisotropic Gaussian low pass filtering operation of a specified area around the intersection, the region of interest, ROI. What the Gaussian low pass filter will reduce the high frequency components of the surface within the ROI, which will reduce the surface's curvature. A geometric interpretation is that one is spreading the normal information at each point to its neighbors. The operation can be divided into two main parts, specifying the region of interest and calculating the size and orientation of the Gaussian filter kernel.

3.1.1 ROI

Before blending the two surfaces the region of interest must be specified. The naïve and simple way is to define all areas where the two input surfaces are within a specific distance from each other. This may however lead to blending in areas where blending is undesirable, since surfaces may be close to each other but not intersect. The way to do it is to sample the intersection curve, the curve where the two input surfaces intersect, and define the blending ROI as a narrow band of a certain radius around the intersection curve. The sample of the intersection curve is the set the voxels that contain a zero-distance value to both input surfaces. For sampling of the intersection curve, a voxel value of 0.7 has been found to produce satisfying results and for the ROI a narrow band with a radius of four voxels has shown to be suitable.

3.1.2 Shape & Orientation of the Gaussian kernel

An anisotropic Gaussian filter kernel is a filter kernel with two or more free variance parameters. In 3D that means that there are at most three and at least two free variance parameters of the filter kernel. The variance parameters describe the standard deviation along the filters main axes, u, v and w in 3D. If a Gaussian filter kernel in 3D only has two free variance parameters it means that two of its three parameters are the same. For the blending operation a filter kernel with two free variance parameters is used. The second and third variance parameters, σ_2 and σ_3, are set to be the same, $\sigma_2 = \sigma_3$. When σ_2 is set to be larger than σ_1, the filter kernel will take the shape of an oblate rotational spheroid. The filter kernel will then be oriented in such a way that its first axis, the axis along σ_1, is parallel with the local gradient at each voxel. This can be interpreted geometrically as if the oblate shaped filter kernel, for each voxel, is

lying on the voxel's tangent plane. Since the filter kernel is of larger extent along the surface's tangent plane, the smoothing effect will therefore also be larger along this plane. This allows for control over the amount of smoothing by controlling the size of the Gaussian filter kernel. The larger the two variance parameters are, the greater the blending effect will be.

3.1.3 Boundaries

One side effect of only filtering the intersection ROI is that there will be a sharp transition from filtered voxels to non-filtered voxels. There is no in-between, only filtered or non-filtered voxels. This may cause some irregularities or unwanted artifacts on the boundaries of the blended areas. To avoid this, the size of the Gaussian filter kernel can be made adaptive. By regulating the size of the filter kernel, i.e. the size of the variance parameters, smooth borders around the blended areas may be achieved. The following piece vice polynomial [1] has been found to produce good results.

$$p(\beta) = \begin{cases} 0 & for \quad \beta \le 0 \\ 2\beta^2 & for \quad 0 < \beta \le 0.5 \\ 1 - 2(\beta - 1)^2 & for \quad 0.5 < \beta < 1 \\ 1 & for \quad \beta \ge 1 \end{cases} \tag{3.1}$$

Where β is a function of the distance to the zero crossing (ZC), which is given by the voxel value at each point.

$$\beta = \frac{dist \quad to \quad ZC}{radius \quad of \quad ROI} \tag{3.2}$$

The new variance parameter where then calculated as:
$\sigma_{new} = \sigma_{start} - decayMax \cdot p(\beta)$

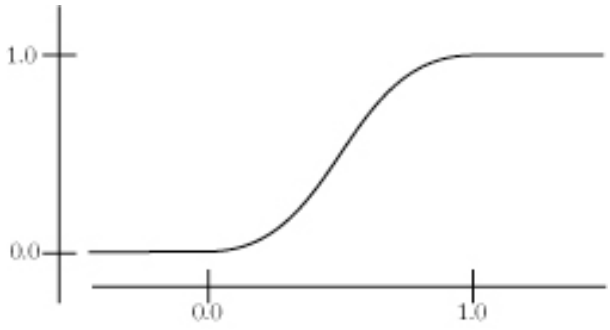


Fig 3.1. The piece wise polynomial function for scaling the size of the kernel

3.2 Surface smoothing

This section aims to describe how the idea behind the use of anisotropic Gaussian diffusion for blending of level set models may be reused and modified to suit global level set surface smoothing. A Gaussian low pass filter is a blurring filter, i.e. it regionally distributes the high frequency components of an image or 3D image. This is a fundamental fact, which blending by means of anisotropic Gaussian diffusion

relies on. But why only use it for blending when one may use it for surface smoothing in general? The features of the surface smoothing Gaussian filter kernel correspond to the features of the filter kernel used for blending. It is desired to construct a kernel that does most of the smoothing along the surface. An oblate spheroid shaped kernel oriented to lie on a tangent plane to the surface suits this purpose well, since it has a larger extent along this plane than out of the tangent plane. This is the same type of Gaussian kernel that is used for blending of two level set surfaces. Moreover, it is also oriented in the same fashion as in the blending case, since it shall be lying on the tangent plane to the surface as well. This is achieved by orienting the *u* axis of the kernel along the local gradient of the surface. However, two things differ between smoothing and blending. The most obvious difference is that the region of interest differs. Now the entire surface is of interest and not just the intersection area of two joined surfaces. Still, there is no need to filter the entire volume encapsulating the surface. A set of voxels within a narrow band of some radius, *r*, is satisfactory. Second, there is no need to have an adaptive size of the Gaussian kernel since there are no areas that shall remain unaffected. The surface-smoothing algorithm may be summarized as a filtering of the set of voxels within a narrow band around the interface with an anisotropic Gaussian convolution filter.

3.3 Gaussian diffusion

Smoothing a surface by means of Gaussian diffusion corresponds to diffusing the surface's normals. High frequency components of a surface are defined as the fine details of the surface. One can say that the more curvature there is on the surface the more high frequency components there are. When smoothing a surface the objective is to reduce the amount of high frequency components, trying to make the surface as flat as possible. This can be accomplished by a simple low pass filter.

The Gaussian low pass filter has the properties of being exponentially decaying and at the same time strict positive, which makes it suitable for smoothing and blending level sets. The surface is low pass filtered by means of convolution of the level set volume in 3D and the convolution kernel is given by the Gaussian distribution. In 1D the Gaussian distribution takes the following form:

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{x^2}{2\sigma^2}\right\} \tag{3.3}$$

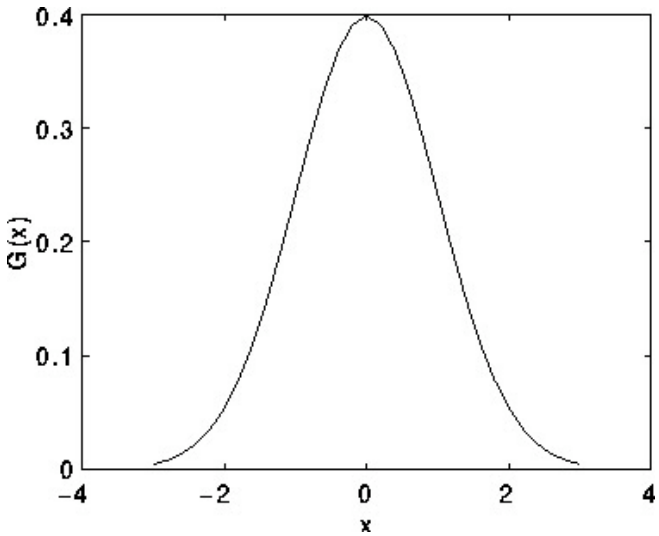


Fig 3.2. Gaussian distribution. The Gaussian distribution in 1D given by eq. 3.3 with standard deviation 1.0.

For a discrete signal $F[x]$, as the in the level set case, the Gaussian kernel can be implemented as a finite sum. Capital letters are use to accentuate that use of discrete signals instead of continuous signals.

$$F[x] = \sum_{k=-K}^{K} F[x-k]G[k] \tag{3.4}$$

In theory the Gaussian distribution never reaches zero and would thereby require an infinitely large convolution kernel. In practice the Gaussian distribution is effectively zero more than about three standard deviations from the mean, which means that the kernel can be truncated to zero beyond this point. This type of filter is called a *finite impulse response filter* (FIR).

The Gaussian distribution can easily be extended from one-dimension to *n*-dimensions and take different forms depending on the variance parameters along the coordinate axes.

3.3.1 Isotropic Gaussian low pass filtering

A Gaussian filter has as many free variance parameters as dimensions. When all variance parameters are the same, i.e. when there is only one free variance parameter, it is said to be isotropic. In 2D the isotropic Gaussian filter would have the shape of a circle and in 3D it would have the shape of a sphere.

Extending the Gaussian distribution from 1D to 3D for the isotropic case would look like this:

$$\begin{aligned} g(x,y,z) &= g(x)\cdot g(y)\cdot g(z) = \\ &= \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{x^2}{2\sigma^2}\right\}\cdot\frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{y^2}{2\sigma^2}\right\}\cdot\frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{z^2}{2\sigma^2}\right\} = \\ &= \frac{1}{2\pi\sqrt{2\pi}\sigma^3}\exp\left\{-\frac{x^2+y^2+z^2}{2\sigma^2}\right\} \end{aligned} \tag{3.5}$$

As seen in equation 3.5, the isotropic Gaussian filter in 3D can be described as the product of the 1D Gaussian along the x-axis, the 1D Gaussian along the y-axis and the 1D Gaussian along the z-axis. This means that the n-dimensional isotropic Gaussian convolution filter can be separated into n numbers of sequential 1D convolutions along its coordinate axes. The one-dimensional kernel is described by the function *g(x)*, which is the same as *g(y)* and *g(z)*. This feature can be used to enhance the performance of the filter. For example, in the two-dimensional case with a filter of size *n*n*, n^2 operations would be needed for each pixel, but when separating the filter only 2**n* operations are needed. This is because when the filter is separated the image is first filtered with a stencil of size n along the x-axis; the result is then filtered with another stencil of the same size along the y-axis. Each grid point is thereby filtered two times with a stencil of size n instead of filtered one time with a stencil of size n^2. The fact that isotropic Gaussian filters only have one free variance parameter makes them easy to handle analytically and also easy to implement. This has made them very popular but their anisotropic sibling is much more interesting for image processing since they also encode information about orientation.

3.3.2 Anisotropic Gaussian Low pass filtering

Since the purpose of the filtering is to smooth surfaces, a filter that takes this fact into account is preferred. The anisotropic Gaussian filter suits this purpose. An anisotropic Gaussian filter is a Gaussian filer with more than one free variance parameter. In three-dimensions where *u*, *v* and *w* are orthogonal, the anisotropic Gaussian filter can be described as

$$g(u,v,w,\sigma_u,\sigma_v,\sigma_w) = g(u,\sigma_u)\cdot g(v,\sigma_v)\cdot g(w,\sigma_w) =$$
$$= \frac{1}{\sqrt{2\pi}\sigma_u}\exp\left\{-\frac{u^2}{2\sigma_u^2}\right\}\cdot\frac{1}{\sqrt{2\pi}\sigma_v}\exp\left\{-\frac{v^2}{2\sigma_v^2}\right\}\cdot\frac{1}{\sqrt{2\pi}\sigma_w}\exp\left\{-\frac{w^2}{2\sigma_w^2}\right\} \qquad (3.6)$$

Where *u, v* and *w* are defined as

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}\begin{bmatrix} \cos\beta\cos\gamma & \sin\alpha\sin\beta\cos\gamma+\cos\alpha\sin\gamma & -\cos\alpha\sin\beta\cos\gamma+\sin\alpha\sin\gamma \\ -\cos\beta\sin\gamma & -\sin\alpha\sin\beta\sin\gamma+\cos\alpha\cos\gamma & \cos\alpha\sin\beta\sin\gamma+\sin\alpha\cos\gamma \\ \sin\beta & -\sin\alpha\cos\beta & \cos\alpha\cos\beta \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \qquad (3.7)$$

As in the case with the isotropic Gaussian filter the anisotropic filter can also be separated. This however is not as simple as in the isotropic case. It is standard procedure to separate the anisotropic Gaussian along its main orthogonal axes. The problem is that the direction of integration has to be rotated with respect to the coordinate grid. This means that interpolation must be used in all n dimensions for all n integration steps, which will make it cumbersome to implement and slow. Numerical errors may also occur and accumulate.

3.3.3 Non-orthogonal separation of the anisotropic Gaussian filter

It is possible to decompose the anisotropic Gaussian in $\Re^n$ in an efficient way [12]. Since this subject is of such great importance for the performance of the smoothing operation, a description of how to efficiently separate the anisotropic Gaussian will be given. This section about the non-orthogonal separation of the anisotropic Gaussian filter is based on the paper by Lampert and Wirjadi [12].

The way an efficient separation of the anisotropic Gaussian is done is not to separate it along its main axes but to try to separate it along arbitrary and possibly non-orthogonal axes, with the original coordinate axes as a base. In this way the cumbersome rotation of grid points is no longer needed. These directions, which the kernel is decomposed along, may easily and efficiently be described as vectors.

"*For any decomposition of the $\sum = VDV^t$ covariance matrix $\sum$ into square matrices D and V, where D is diagonal and positive, and V has determinant 1, there is a separation of the nD-Gaussian into 1D-Gaussians, where the separation directions are given by the column vectors of V.*" [12]

The matrix Σ is defined in eq 3.25. By finding the matrices *V* and *D*, an efficient separation of the anisotropic Gaussian kernel may be obtained. A description of this procedure follows.

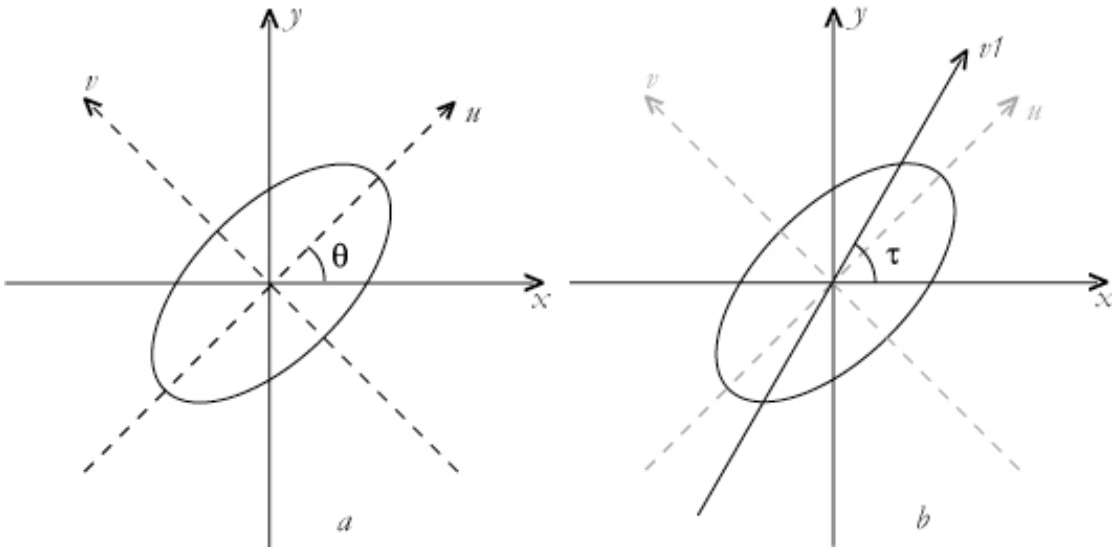


Fig 3.3. Gauss kernel axes. A 2D schematic view of the axis of which a 2D Gaussian kernel is separated along. a) The kernel's principal axes are *uv*-aligned. b) The kernel's principal axes are uv-aligned but the kernel is defined in the *x, v1*system. [18]

3.3.3.1 Factorization

Below follows how the n-dimensional Gaussian filter kernel, $g(\boldsymbol{x})$, can be factorized to a series of one-dimensional Gaussians, $g(\boldsymbol{x}) = g_1(x_1)\cdot \ldots \cdot g_n(x_n)$.

The anisotropic Gaussian kernel, $g(\boldsymbol{x})$, can be written in the following form:

$$g(x) = \frac{1}{(2\pi)^{n/2}\left|\Sigma\right|^{1/2}} \exp\left\{-\frac{1}{2}x^t\Sigma^{-1}x\right\} \tag{3.8}$$

Where $\mathbf{x} = (x1, \ldots, x_n)$, $\Sigma \in \Re^{n\times n}$ is the covariance matrix (see eq 3.25) and $\left|\Sigma\right|$ is the determinant of Σ. In $\sum = VDV^t$, Assume that D is diagonal and positive, V has the determinant 1, $|V|= 1$, then $\sum^{-1} = V^tD^{-1}V^{-1}$ and the Gaussian can be written as

$$g(x) = \frac{1}{(2\pi)^{n72}\left|D\right|^{1/2}} \exp\left\{-\frac{1}{2}x^t(V^{-t}D^{-1}V^{-1})x\right\} \tag{3.9}$$

Changing the $\boldsymbol{x}$ linearly to $\mathbf{v} = (v_1, \ldots, v_n)$ with $\mathbf{v}:=V^{-1}\mathbf{x}$. The Gaussian kernel becomes

$$g(v) = \frac{1}{(2\pi)^{n/2}\left|D\right|^{1/2}} \exp\left\{-\frac{1}{2}\vec{v}^tD^{-1}\vec{v}\right\} \tag{3.10}$$

Since D is a diagonal matrix and is assumed to have positive entries denoted by $d_1^2,\ldots,d_n^2$ and $\left|D\right|^{1/2} = d_1\cdot\ldots\cdot d_n$. It follows that D^{-1} is diagonal and with only positive entries as well and that the matrix product is a weighted sum of squares.

$$g(v) = \frac{1}{(2\pi)^{n/2}d_1\cdot\ldots\cdot d_n} \exp\left\{-\frac{1}{2}\sum_{i=1}^{n}\frac{v_i^2}{d_i^2}\right\} \tag{3.11}$$

As one can see this form of the Gaussian kernel can be can be factorized by means of laws of the exponent.

$$g(v) = \frac{1}{\sqrt{2\pi}d_1}\exp\left\{-\frac{1}{2}\frac{v_1^2}{d_1^2}\right\}\cdot\ldots\cdot\frac{1}{\sqrt{2\pi}d_n}\exp\left\{-\frac{1}{2}\frac{v_n^2}{d_n^2}\right\} \tag{3.12}$$

$$= g_1(v_1)\cdot\ldots\cdot g_n(v_n)$$

Each g_i (v_i) is now an ordinary one-dimensional Gaussian distribution with mean zero and standard deviation d_i along v_i.

$$g(v_i) := \frac{1}{\sqrt{2\pi}d_i}\exp\left\{-\frac{1}{2}\frac{v_i^2}{d_i^2}\right\} \tag{3.13}$$

3.3.3.2 Separation of the convolution integral

By changing the coordinates of the convolution integral from **x** and **y** to **u** and **v**, it will be shown how the above factorization gives rise to a separation of the convolution integral. Convolution of the two functions *f* and *g*, in this case corresponding to the level set (*f*) and the Gaussian (*g*), is given by:

$$(f * g)(\vec{x}) \equiv \int_{\Re}\int_{\Re}\ldots\int_{\Re} f(\vec{y})g(\vec{x}-\vec{y})dy_1\ldots dy_n \tag{3.14}$$

Switching from the from **x** and **y** to **u** and **v** coordinates, yields in the following integral

$$(f * g)(\vec{x}) \equiv \int_{\Re}\int_{\Re}\ldots\int_{\Re} f(V\vec{v})g(V\vec{u}-V\vec{v})\left|V^{-1}\right|dv_1\ldots dv_n \tag{3.15}$$

Where **u** and ***v*** are defined as:

$$\vec{u} \equiv V^{-1}\vec{x} \qquad \vec{v} \equiv V^{-1}\vec{y}$$

$|V^{-1}|$ has been added to the integral since there has been a change of coordinates. Since the determinant of *V* is assumed to be equal to one it follows that $|V^{-1}| = 1$. The separated integral now looks like:

$$(f * g)(\vec{x}) \equiv \int_{\Re} g_n(u_n - v_n)\int_{\Re} g_{n-1}(u_{n-1} - v_{n-1})\ldots\int_{\Re} g_1(u_n - v_n)f(V\vec{v})dv_1\ldots dv_n \tag{3.16}$$

The matrix-vector product, $V\mathbf{v}$, in the last term of the integral *f(V***v***)* may be split up into a sum, $V\vec{v} = \sum_i v_i\vec{v}^{\,i}$, where $\mathbf{v}^i$ are the column vectors of the matrix *V*. This means that when integrating over v_i one is convoluting along the direction $\mathbf{v}^i$. The notation of directional convolutions offer a very compact and neat notation for this.

$$(f * g)(\vec{x}) = g_n *_{v^n} \ldots g_1 *_{v^1} f \tag{3.17}$$

***v** is the directional convolution operator and is defined as:

$$(g *_v f)(\bar{x}) \equiv \int_{-\infty}^{\infty} g(\lambda) f(\bar{x} - \lambda\bar{x}) d\lambda \tag{3.18}$$

Now the convolution integral of dimension *n* is separated into *n* one-dimensional integrals, each one convoluting along their respective direction $\mathbf{v}^i$.

3.3.3.3 Parameterization in 3D

Next, we need to find the values for *D* and *V*. Only the three-dimensional case will be discussed here since it's the interesting case for surface smoothing purpose. To find *D* and *V* triangular factorization of Cholesky type is used. The triangular factorization has the advantage over the single value decomposition factorization that it will generate directions that will require fewer interpolation steps for the convolution step. The triangular factorization is computed by writing

$$\sum = \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{bmatrix}, \qquad V = \begin{bmatrix} 1 & v_{12} & v_{13} \\ 0 & 1 & v_{23} \\ 0 & 0 & 1 \end{bmatrix}, \qquad D = \begin{bmatrix} d_1^2 & 0 & 0 \\ 0 & d_2^2 & 0 \\ 0 & 0 & d_3^2 \end{bmatrix}$$

Which leads to

$$d_1^2 = s_{11} - \frac{(s_{12}s_{33} - s_{13}s_{23})^2}{s_{33}(s_{22}s_{33} - s_{23}^2)} - \frac{s_{13}}{s_{33}}, \tag{3.19}$$

$$d_2^2 = s_{22} - \frac{s_{23}^2}{s_{33}}, \tag{3.20}$$

$$d_3^2 = s_{33} \tag{3.21}$$

$$v_{12} = \frac{s_{12}s_{33} - s_{13}s_{23}}{s_{22}s_{33} - s_{23}^2}, \tag{3.22}$$

$$v_{13} = \frac{s_{13}}{s_{33}}, \tag{3.23}$$

$$v_{23} = \frac{s_{12}}{s_{33}} \tag{3.24}$$

The covariance matrix $\sum$ can also be described as

$$\Sigma = R^t S R, \tag{3.25}$$

S is a diagonal matrix containing the variance values. *R* is a three by three matrix describing the rotation. R_{x1} denote rotation around the x_1 axis and R_{x3} denotes rotation around the x_3 axis.

$$R(\psi, \theta, \varphi) = R_{x1}(\psi) R_{x3}(\theta) R_{x1}(\varphi) \tag{3.26}$$

where $\psi \in [0, \pi[$, $\theta \in \left[0, \frac{\pi}{2}\right]$, $\varphi \in [0, \pi[$

$$R_{x1}(\psi)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos(\psi) & -\sin(\psi)\\ 0 & \sin(\psi) & \cos(\psi)\end{bmatrix} R_{x3}(\theta)=\begin{bmatrix}\cos(\theta) & -\sin(\theta) & 0\\ \sin(\theta) & \cos(\theta) & 0\\ 0 & 0 & 1\end{bmatrix}$$

$$R_{x1}(\varphi)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos(\varphi) & -\sin(\varphi)\\ 0 & \sin(\varphi) & \cos(\varphi)\end{bmatrix} S=\begin{bmatrix}\sigma_1 & 0 & 0\\ 0 & \sigma_2 & 0\\ 0 & 0 & \sigma_3\end{bmatrix}$$

If $\sigma_2 = \sigma_3$, which is the most common case for filtering of 3D image data and for the blending method proposed in this thesis, the anisotropic Gaussian is rotationally invariant around its first axis. Because the base used is Euler angles this means that the first rotation in the x_1 direction can be removed from the equation. R is now only dependant on the two angles θ and φ.

$$R(\theta,\varphi) = R_{x3}(\theta)R_{x1}(\varphi) \tag{3.27}$$

$$=\begin{bmatrix}\cos(\theta) & -\sin(\theta) & 0\\ \sin(\theta) & \cos(\theta) & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & \cos(\varphi) & -\sin(\varphi)\\ 0 & \sin(\varphi) & \cos(\varphi)\end{bmatrix}=\begin{bmatrix}\cos(\theta) & -\sin(\theta)\cos(\varphi) & 0\\ \sin(\theta) & \cos(\theta)\cos(\varphi) & \cos(\theta)\sin(\varphi)\\ 0 & \sin(\varphi) & \cos(\varphi)\end{bmatrix}$$

Now the parameterization of ∑ can be obtained by solving $R^t SR = VDV^t$ and it becomes

$$d_1^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_2^2\cos^2\theta + \sigma_1^2\sin^2\theta} \tag{3.28}$$

$$d_2^2 = \frac{\sigma_2^2(\sigma_2^2\cos^2\theta + \sigma_1^2\sin^2\theta)}{\sigma_2^2 - (\sigma_2^2 - \sigma_1^2)\sin^2\varphi\sin^2\theta} \tag{3.29}$$

$$d_3^2 = \sigma_2^2 - (\sigma_2^2 - \sigma_1^2)\sin^2\varphi\sin^2\theta \tag{3.30}$$

$$v_{12} = \frac{(\sigma_2^2 - \sigma_1^2)\cos^2\varphi\cos\theta\sin\theta}{\sigma_2^2 - (\sigma_2^2 - \sigma_1^2)\sin^2\theta} \tag{3.31}$$

$$v_{13} = \frac{-(\sigma_2^2 - \sigma_1^2)\sin\varphi\cos\theta\sin\theta}{\sigma_2^2 - (\sigma_2^2 - \sigma_1^2)\sin^2\varphi\sin^2\theta} \tag{3.32}$$

$$v_{23} = \frac{(\sigma_2^2 - \sigma_1^2)\cos\varphi\cos\theta\sin\theta}{\sigma_2^2 - (\sigma_2^2 - \sigma_1^2)\sin^2\varphi\sin^2\theta} \tag{3.33}$$

3.3.3.4 Discretization and interpolation

The next and final step is to obtain a discrete convolution operator, which may be described as a finite sum. The convolution filter for the discrete case described with directional convolution notation in the three-dimensional case looks like

$$G * F = G_3 *_{v^3} G_2 *_{v^2} G_1 *_{v^1} F\,, \tag{3.34}$$

and the convolution integral turns into the following sum

$$(G_i *_v F)(\bar{x}) = \sum_{k=-K}^{K} G_i(k) F(\bar{x} - k\bar{v})\,. \tag{3.35}$$

Since the convolution filter is separated it means that the input signal will first be filtered along v1, the resulting signal will then be filtered along v2 and finally the output signal of the second filter pass will be filtered along v3.

$$F_2 = G_1 *_{v^1} F$$
$$F_3 = G_2 *_{v^2} F_2$$
$$F_{final} = G_3 *_{v^3} F_3$$

Remember, v^i are the column vectors of the matrix V, giving

v1 = (1, 0, 0)t
v2 = (v12, 1, 0)t
v3 = (v13, v23, 1)t

This yields in three sums, one for each filter pass

$$F_2 = G_1 *_{v^1} F = \sum_{k=-K}^{K} G_1(k) F(\vec{x} - k(1,0,0)^t) = \sum_{k=-K}^{K} G_1(k) F(x_1 - k, x_2, x_3)\,, \tag{3.36}$$

$$F_3 = G_2 *_{v^2} F_2 = \sum_{k=-K}^{K} G_2(k) F_2(\vec{x} - k(v_{12},1,0)^t) = \sum_{k=-K}^{K} G_2(k) F_2(x_1 - kv_{12}, x_2 - k, x_3)\,, \tag{3.37}$$

$$F_{finall} = G_3 *_{v^3} F_3 = \sum_{k=-K}^{K} G_3(k) F_3(\vec{x} - k(v_{13}, v_{23}, 1)^t)$$
$$= \sum_{k=-K}^{K} G_3(k) F_3(x_1 - kv_{13}, x_2 - kv_{23}, x_3 - k) \tag{3.38}$$

x is the current voxel to be filtered and G_i is given by

$$G_i(k) := \frac{1}{\sqrt{2\pi} d_i} \exp\left\{ -\frac{1}{2} \frac{k^2}{d_i^2} \right\}. \tag{3.39}$$

Now all the pieces for an efficient anisotropic Gaussian convolution filter have been given. As seen from the convolution sums, the reason that this separation scheme is so efficient is that the directions of integration do not need to be rotated with respect to the coordinate grid. The separation scheme also has the advantage of being very effective in terms of interpolation. Thanks to the factorization, no interpolation is needed for convolution along the x_1 direction. For the following convolutions only one additional interpolation direction is added per convolution. In the three-dimensional cases this means that for the second pass interpolation is only needed along the x-axis and for the third pass it is only needed within the xy-plane. This is a

major advantage compared to regular anisotropic Gauss filtering where interpolation always must be performed along all three axes if the kernel isn't grid aligned.

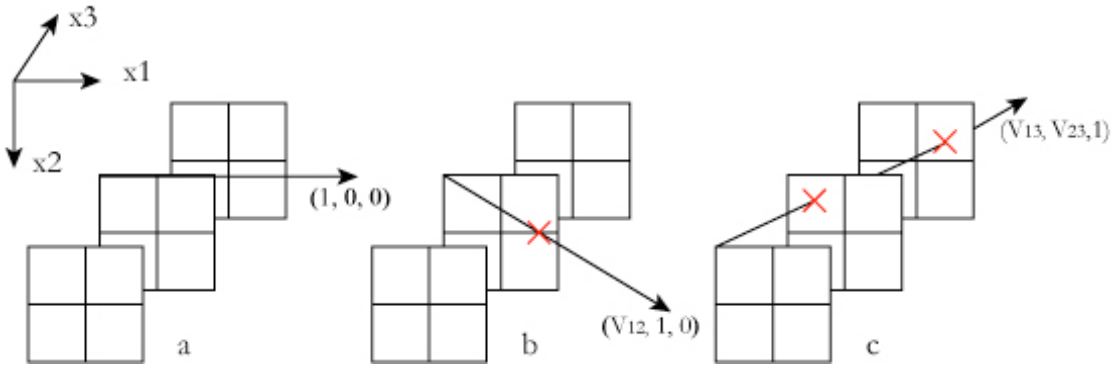


Fig 3.4. Convolution directions. Schematic view of convolution directions and interpolation. In a) no interpolations is needed, in b) interpolations is needed along x_1 and in c) interpolations is needed in the x_1x_2-plane.

4 Implementation

This chapter intends to describe the implementation part of the thesis. The emphasis of this chapter will be on how to incorporated the level set blending operation with the DT-Grid level set data structure [5]. It will also be explained how to extend the blending operator for global surface smoothing.

The level set blending and smoothing operations have been implemented in C++.

4.1 Blending – Anisotropic Gauss filtering

Blending of two level set surfaces by means of anisotropic diffusion has been implemented as convolution of a specified region of interest of the *union* level set with an anisotropic Gaussian filter kernel. The convolution filter has been implemented as a finite impulse response filter (FIR).

4.1.1 Finite impulse response filter

Since the anisotropic Gaussian diffusion is implemented as a separated FIR convolution filter in three dimensions three different convolution stencils have to be calculated for each voxel. New stencil values need to be calculated for each voxel since the stencils are dependent on the local orientation of the kernel, which in turn is dependent on the local normal of the surface. The filter process has been a three-step process. A copy of the original union volume has been used as a buffer volume. The way it works is that while filtering one volume the results are stored in the other volume and for the following filter pass the second volume is filter and the results are stored back in the first volume. It continues in this way until all convolution passes have been processed.

The Gaussian distributions given in the equations 3.36 - 3.38, describe each one of the three stencils. The variance parameters of the stencils are given by the parameters in the diagonal matrix D of the decomposition of the covariance matrix $\sum$. d_1^2 corresponds to the standard deviation of first stencil, d_2^2 corresponds to the second stencil and d_3^2 corresponds to stencil number three. The sizes of the stencils are dependent on its variance parameter and are usually of the size 5σ, rounded upward to the closes odd integer. Beyond this point the value of the Gaussian distribution is so close to zero that it can be truncated to zero.

The directions of the stencils are described as the column vectors of the V matrix of the decomposition of the covariance matrix $\sum$. The first column vector of V corresponds to the first stencil, the second column vector to the second stencil and the third column vector to the third stencil.

To obtain the variance parameters d_1^2, d_2^2 and d_3^2 and the direction parameters v_{12}, v_{13}, v_{23} the rotation angles θ and φ need to be found. These angles may be calculated with the help of the local gradient (n_x, n_y, n_z) at each voxel.

$$\theta = \tan^{-1}\frac{n_z}{n_x} \tag{4.1}$$

$$\varphi = \tan^{-1}\frac{n_y}{\sqrt{n_x^2 + n_z^2}} \tag{4.2}$$

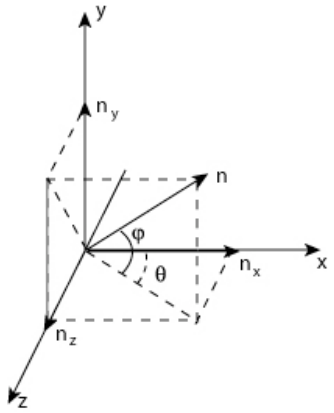


Fig 4.1 Angles. Schematic view of how to find the angles θ and φ with help from the local normal n.

Where n_x, n_y, n_z may be calculated by using finite difference and *dx*, *dy* and *dz* are the distances between two voxels along the x-, y- and z-axis respectively.

$$n_x = ((x_{+1}, y, z) - (x_{-1}, y, z))\frac{1}{2dx} \tag{4.3}$$

$$n_x = ((x, y_{+1}, z) - (x, y_{-1}, z))\frac{1}{2dy} \tag{4.4}$$

$$n_x = ((x, y, z_{+1}) - (x, y, z_{-1}))\frac{1}{2dz} \tag{4.4}$$

4.1.2 Where to blend

The tracking of the designated blending area may be performed in different ways. Which method to choose depends on which kind of data structure that is used. For this thesis the blending operator has been implemented both for the simple and intuitive dens level set representation and the advanced and compact DT-Grid level set representation. Different methods for tracking the intersection regions of the two input surfaces have been used for the two different data structures. The simplicity of the dens level set data structure lead to a very straightforward implementation where a sampling of the intersection curve is stored during the CSG union operation. The indices to these voxels are stored in an array and all neighbor voxels within a specified distance to the zero crossing are added to the intersection ROI. The intersection curve is defined as the set of voxels that contain both input surfaces.

For the DT-Grid, the blending operator has been implemented as a part of a large framework, the Graphic Group Library. This in combination with the data structure it self has lead to the use of an alternate approach for tracking the intersection ROI. The

intersection ROI is defined as the regions where the beta band of intersection volume intersects with the beta band of the union volume.

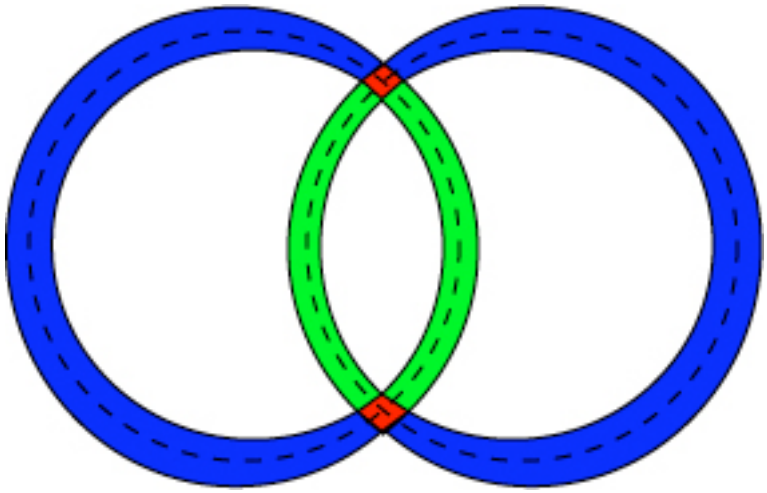


Fig. 4.2. ROI. The image shows how to find the intersection area of two curves with the help of the intersection volume. The two original shapes are marked with dotted lines. The blue band corresponds to the beta band of the union shape and the green band corresponds to the beta band of the intersection shape. The intersection areas are marked with red.

The DT-Grid relies heavily on iterators for accessing and manipulating values in the level set. To be able to set new values in the grid, an iterator must point to the specified voxel. So in order to blend the level set surfaces an iterator that iterates over the entire beta tube of the union surface is defined. While iterating over the beta tube one has to check if the current voxel is within the beta tube of the intersection surface. If inside, it means that the current voxel is within the intersection ROI and needs to be filtered.

4.1.3 DT-Grid grid access – Random access vs. stencil access

The DT-Grid only stores values in a narrow band around the interface in contrast to a simple dens level set which stores all values within a box-shaped volume. For a full grid, random access is a simple mapping from the grid point to the corresponding array index and is performed in constant time.

$$\text{Array index} = i \cdot \text{rows} \cdot \text{columns} + j \cdot \text{columns} + k \qquad (4.5)$$

Where i = row index, j = column index and k = z index.

For the DT-Grid random access may be performed in logarithmic time in the number of connected components within p-columns [5]. Since the numbers of connected components are very small in relation to the number of grid points the random access operation becomes almost as fast as for a full grid in practice. However, using random access with DT-Grid is easy since it has full support for random access. One merely has to specify the index, i, j, k, to the desired grid point. Constant time access can be obtained if sequential access with iterators is used instead. When combining an iterator with an iterator stencil constant time is obtained for neighbor access as well. This is applies when iterating over an entire tubular grid. It is implemented by specifying a stencil of iterators. These iterators are then stored in an array accessible from the main iterator. One then has to create a function in the iterator class that by accessing the values of the neighbor iterators can calculate the desired task.

For the blending operation it seems like an iterator stencil that matches the Gaussian convolution filter stencil would be the optimal way for incorporating the anisotropic diffusion blending with the DT-Grid. However, random access has been proven to be more efficient. This is because when using stencil iterator access additional iterators for all grid points within the stencil needs to be incremented as well. In other words, it may be faster to do a slow operation a few times than doing a fast operation a lot of times, see fig 4.2 below. The number of grid points that belong to the intersection ROI are very few in relation to the total number of grid points. Random access is therefore faster than stencil iterator access for accessing neighbors of an iterator when it comes to blending by means of anisotropic diffusion.

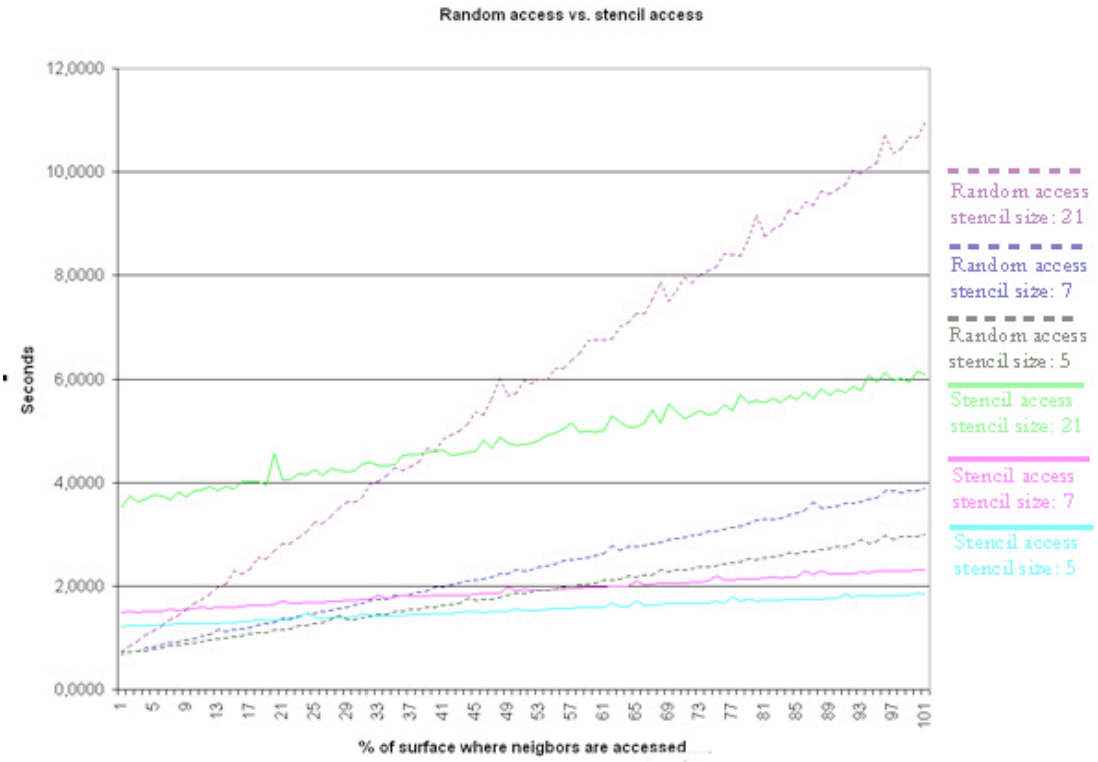


Fig 4.3. Random neighbor access vs. stencil neighbor access. This chart compares stencil iterator access (continuous lines) and random access (dotted lines) for accessing neighbors of an iterator. The y-axis is the time it takes to iterate over a surface and access the iterator's neighbors for a certain percentage of the total numbers of visited voxels. The test has been performed for three different stencil sizes of 5, 7 and 21 points.

4.2 Global smoothing

Since Gaussian diffusion is a smoothing operator it has been a secondary goal to implement it as an operator for global surface smoothing. The anisotropic Gaussian diffusion filter has been implemented straightforward as a FIR filter and is very similar to the blending case. There is however some differences that need to be addressed. Obviously the region of interest is different. The smoothing operator has been implemented to affect an entire surface and not only a sub part of it. There is also no need to dynamically change the size of the Gaussian kernel since all surface voxels are being convoluted. The fact that the entire surface is affected allows for a more efficient use of DT-Grid data structure than for the blending case. To achieve the best performance of the smoothing operator one wishes to constrain the Gaussian convolution filter to only operate within a narrow band, as thin as possible around the interface, but which still generates a satisfactory result. A narrow band with a radius

of 1.5 has proven to be sufficient. When it comes to implementation issues, there are two major differences between the implementation of the smoothing and blending operator. The issues arise when incorporating the operator with the DT-Grid level set data structure. They may be pinpointed to 1) defining stencil formats for stencil iterator access 2) defining a tube which corresponds to the region of interest.

4.2.1 Defining stencils and tubes

Since the smoothing operator has to access the entire surface it will use sequential access with a stencil iterator instead of random access, which is used for the blending operator. Stencil iterator access allows constant time access of grid points when they are accessed sequentially, compared to logarithmic time for random access. Two different stencils need to be defined; one for each of the two first filter passes. These stencils shall correspond to the stencils of the Gaussian convolution filters. The stencils must however be extended with extra grid points to support fast calculation of the local gradient. The first stencil must be extended with y+1, y-1, z+1, z-1, and the second stencil with z+1, z-1, with respect to the center of the stencil. The reason that the third filter pass uses random access and not stencil iterator access is that it requires a very large stencil. A stencil for the third filter pass would require a minimum stencil of 3x3x7 iterator, which is more expensive than accessing the few elements of the convolution filter stencil by random access. Unfortunately, the structure of the DT-Grid does not allow dynamic creation of stencil formats for the stencil iterator. This means that variance parameters of the Gaussian kernel is somewhat limited.

The DT-Grid has some predefined tubes of certain widths but these are either too narrow or too wide, therefore a custom tube has been implemented for the smoothing operator. Defining a tube is merely to define the set of voxels that are within a certain radius from the zero level set. It is an easy task, which follows a given structure of the DT-Grid.

5 Results

The intension of this chapter is to present the results of the blending and global smoothing functions implemented for this thesis. Statistics from a comparison between Gaussian diffusion blending and mean curvature flow based blending will be presented. Limitations of the developed algorithm will also be highlighted. The performance of the blending algorithms will be given first followed by the results from implementation of global smoothing.

Over all, the implementation process has been found successful as well as the integration with the DT-Grid and the Graphics Group Library.

5.1 Blending performance

The Performance of a blending function may be measured in two different ways, namely the speed in which the blending is performed and the smoothness of the blended area. The aesthetic results of the blending will be presented first.

5.1.1 Visual appearance

In computer graphic there is a saying, *if it looks good it is good*. Therefore the aesthetic results of the blending algorithm have been measured by its visual appearance in the eye of the beholder. Since it is all about the pictures, a set of images comparing Gaussian diffusion blending with mean curvature flow based blending follow below. They show that the presented algorithm produces a visual result at least equivalent to one produced by the curvature based blending method. The images also show how the amount of blending is controlled by the Gaussian filter kernel's variance parameters. To enhance the blending effects, the models have been rendered with flat shading.

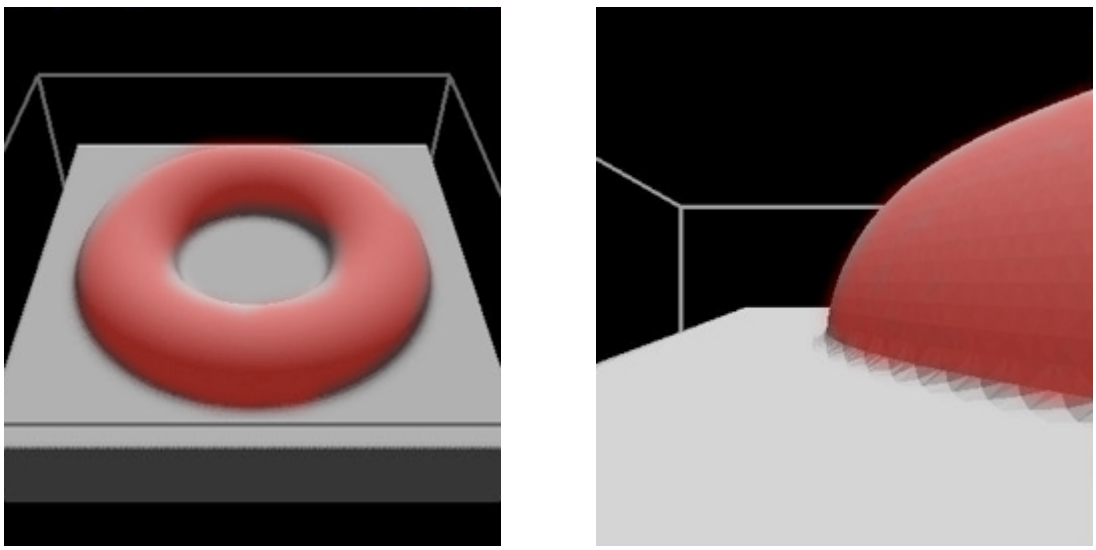
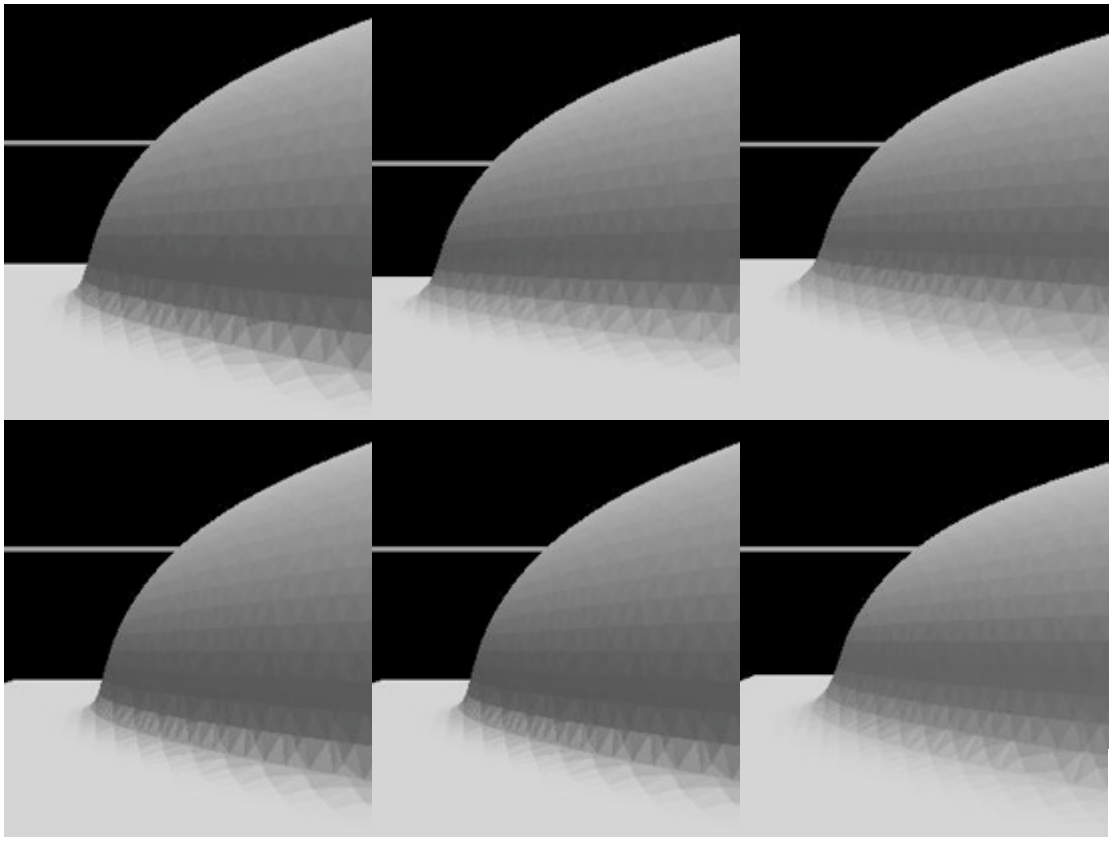


Fig 5.1 Union of a torus and a box. No blending has been performed. See fig 5.2 for blend results.



a b c
d e f

Fig. 5.2 Union of a torus and a box. Anisotropic diffusion of different variances compared with curvature flow based blending. a) diffusion, σ1 = 1.0 & σ2 = 1.5 b) diffusion, σ1 = 1.5 & σ2 = 2.25 c) diffusion, σ1 = 2.0 & σ2 = 3.0 d) curvature, 10 iterations e) curvature 15 iterations f) curvature 30 iterations. For figures of the unblended model see fig 5.1.

	1	2	3
Variance	σ1=1.0; σ2 = 1.5	σ1=1.5; σ2 = 2.25	σ1= 2.0; σ2=3.0
Time	63s	67 s	72
Filter pass 1	4,1 s	4,5 s	5,3 s
Filter pass 2	5,8 s	6,8 s	8,4 s
Filter pass 3	8,7 s	11,1 s	14,4 s

Table 5.1. Anisotropic diffusion. Execution times for blending a box (180 * 60 * 180) and a torus (138 * 42 * 138) by means of anisotropic diffusion. The ROI consists of131 072 voxels. The DT-Grid has been used as the level set data structure.

	1	2	3
Iterations	10	15	30
Time	576 s	944 s	1851 s

Table 5.2. Mean curvature flow. Execution times for blending a box (180 * 60 * 180) and a torus (138 * 42 * 138) by means of Mean curvature flow. The ROI consists of 131 072 voxels**.** The DT-Grid has been used as the level set data structure-

5.1.2 Speed

The test case that has been used for speed measurements of the blending function is as follows: a box with 180 * 60 * 180 grid points and a torus of the dimensions 138 * 42 * 138. The resulting intersection ROI consists of 131 072 voxels. The test environment has been a window workstation with an AMD Sempron CPU on 1.4 GHZ and 512 MB of RAM. This has not been an optimal environment since the computer is not performing as well as one can expect from workstation with this kind of setup. Even though the timings are not representative for this kind of setup, it stills show a proper relation between anisotropic diffusion based blending and mean curvature flow based blending. As shown in table 5.1 and 5.2, the blending proposed in this thesis out performs curvature based blending with a minimum factor of nine. When extensive blending is demanded it outperforms the curvature based blending with a factor larger than 25 and at the same time performs a heavier blending. As one can see from table 1, the actual filtering takes about 30 % of the blending time. The rest of the blending execution time is spent on initializing a buffer level set and rebuilding the resulting model.

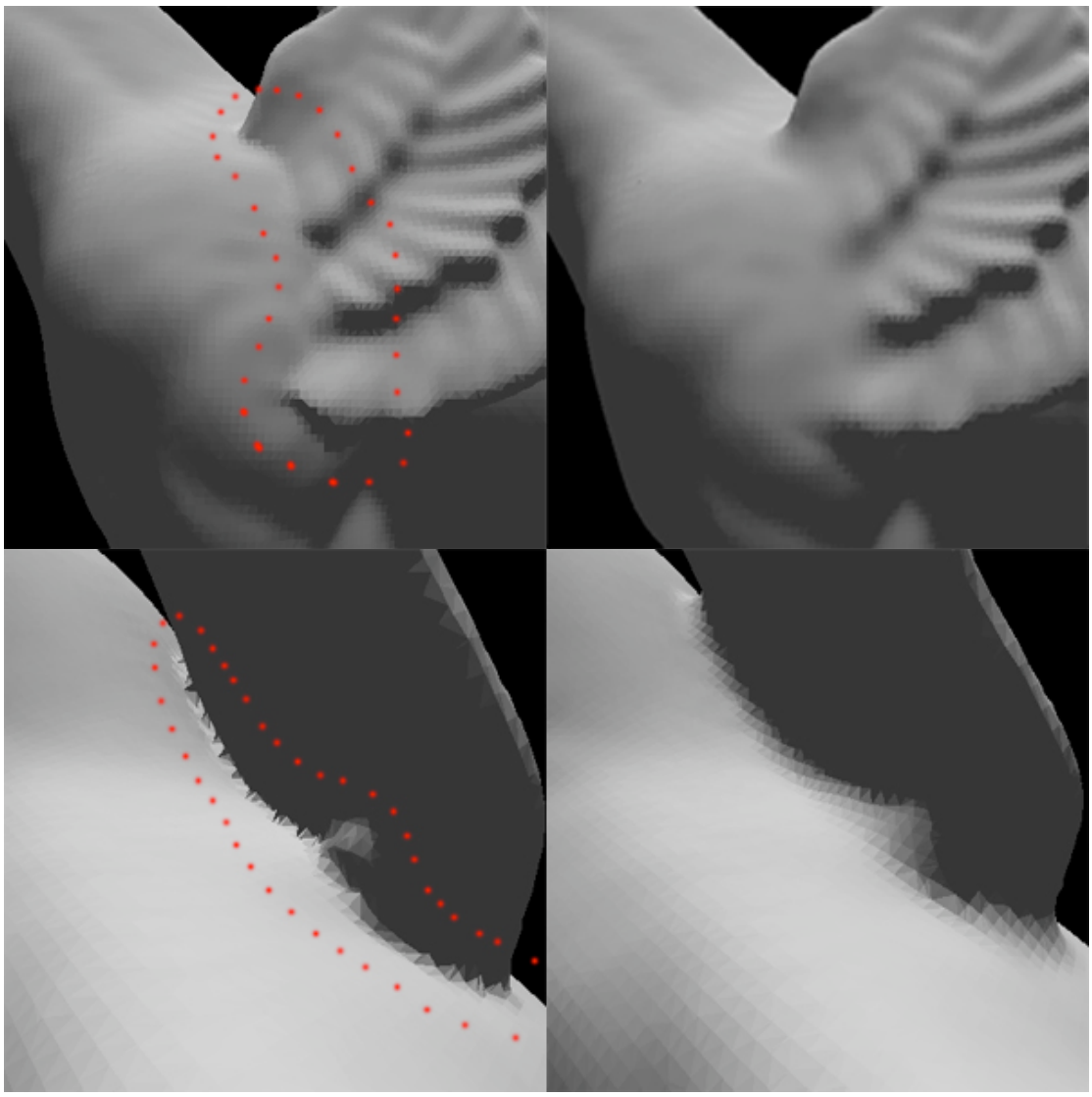


Fig 5.3. Winged horse. Detailed shots of the intersection between a wing and a horse. The images on the left have not been blended while the images on the right have been blended by means of anisotropic diffusion. The intersections are marked with a dotted red line. An overview of the model may be seen in figure 6.2.

5.2 Smoothing

The anisotropic diffusion algorithm proposed for blending has been extended to perform global smoothing of level set surfaces. The level set surface is smoothed with a constant anisotropic Gaussian filter kernel oriented with respect to the local gradient. The smoothing operator is effective in terms of visual smoothing but since the size of the filter kernel is constant, the surface is smoothed uniformly. Anisotropic diffusion smoothing has been compared with morphological opening since the opening method is the most efficient global smoothing operator for level sets. When comparing the two methods it is concluded that morphological opening is better at removing thin elements or spikes from the level set surface, see fig 5.4. Beside the spikes the anisotropic Gaussian diffusion method offers smoothed surfaces equivalent to the resulting surface of the opening operation. However, the method proposed in this thesis is a lot faster than morphological smoothing, see table 5.3.

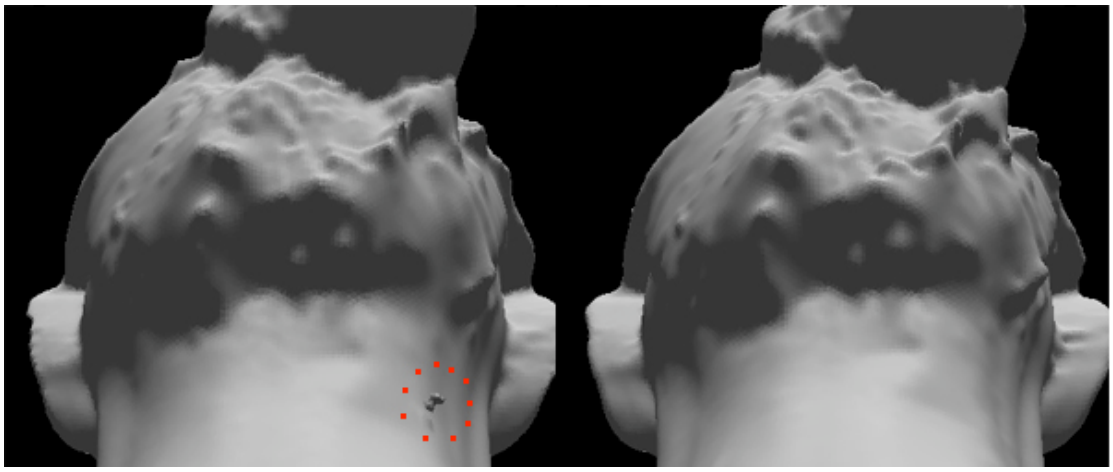


Fig 5.4 Diffusion vs. morphological opening. The model to the left is smoothed with an anisotropic kernel with standard deviations of 1.5 and 2.2. The right model is smoothed with morphological opening where the surface have been eroded three units three times then dilated three units three times. As indicated by red, the anisotropic smoothing operator has not been able to remove the thin element in the neck of the model.

Smoothing	Execution time
Morphological opening	754 s
Anisotropic diffusion, $\sigma 1=1.5$; $\sigma 2 = 2.2$	115 s

Table 5.3 global smoothing. Comparison between morphological opening and anisotropic diffusion. The morphological opening has moved the surface inward three units three times then outwards three units three times. The resolution of the model is 251*242*168 grid points.

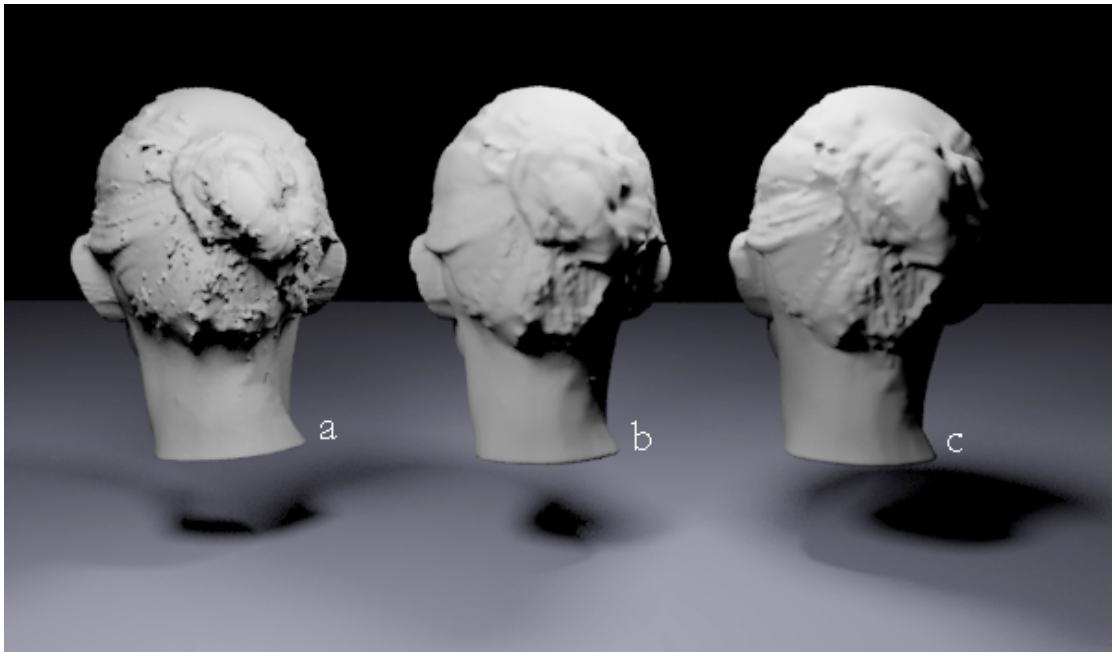


Fig 5.5. Smoothing comparison. Comparison of smoothing by anisotropic diffusion, b, and morphological opening, c. The original model is shown in a. Model c has been morphologically smoothed in the fashion described in table 5.3 above.

5.3 Limitations

There are some limitations in this project's implementation of the blending and smoothing algorithms. These issues will be addressed in this section.

- **Controlling material**. One drawback with blending and smoothing surfaces as proposed in this thesis is that the user cannot control if material is added or removed from the model. This is a feature that would give the user increased control of the final model.
- **Arbitrary local smoothing**. Only global smoothing has been implemented for this thesis. However it is often desired to constrain a smoothing operator locally. For example, there may be some damaged areas of a model that would be improved by smoothing, but global smoothing would lead to reduction of important details in other areas.
- **Shape preservation**. The global smoothing operator uses a constant size of the filter kernel, which makes it less observant of surface structure.

6 Discussion

The task of implementing blending of implicit models by means of anisotropic Gaussian diffusion has been interesting and challenging. The main issue has of course been to fully grasp the efficient separation of the anisotropic Gaussian convolution filter. Another challenge has been to understand the DT-Grid level set data structure.

The performance of the anisotropic Gaussian diffusion blending method is satisfying and compared to the previous known method, mean curvature flow based blending, it is superior in terms of execution times. The fact that the convolution part is the part of the blending system that requires the least effort shows the strength of the algorithm. Parts of the blending system that represents a significant part of the execution time are the initializing and rebuilding of the level set. There are methods for improving the performances of these processes, which are proposed in future work section. The fact that no PDEs need to be solved is the key to the great speed of the anisotropic diffusion method. It also makes it easier to implement since it does not require numerical solvers.

The actual filtering process may also be broken down into a core process and a support process. The support process is the process of iterating over the entire surface, visiting an excess of grid points that are meant to be unaffected. Consequently, the core blending function is faster than it first appears to be. This is pointed out since the issues mentioned in this section mainly refer to the use of the DT-Grid data structure, i.e. the performance of the blending algorithm may vary due to the level set data structure at hand.

As for the visual appearance the proposed method delivers a satisfying result. It offers just as smooth transitions between level set surfaces as the mean curvature flow alternative. The Gaussian diffusion blending method also offers an intuitive way for the user to control the amount of blending, i.e. the smoothness of the transition between two level sets. It is controlled by the size of the filter kernel, i.e. the size of the two variance parameters, larger variance gives smoother transitions. The proposed method has also proven to be less sensitive to the quality of the input models, which has been discovered during the implementation process.

The use of anisotropic Gaussian diffusions as a mean for surface smoothing was mainly an extension of the idea behind the blending algorithm and has therefore not been investigated as thoroughly as the blending. Gaussian diffusion would be a good candidate for local smoothing as well since it is faster than curvature based flow, which has been shown for blending. For global blending of level set surfaces morphological opening and closing operations produce a visually more satisfying result but there is still use for smoothing by anisotropic diffusion thanks to its great advantage in speed.

6.1 Conclusion

This thesis has shown that anisotropic Gaussian diffusion is a very well suited method for blending of implicit models and level sets in particular. The thesis has also shown that the blending idea can be extended for global surface smoothing with very little effort. The developed method has proved to be very advantageous in terms of speed, compared to its alternative, the mean curvature flow blending method. It has at the same time also shown to produce a visual result that is as good as previous methods.

The general conclusion is that blending by means of anisotropic Gaussian diffusion is a very good alternative to the mean curvature flow based blending method.

6.2 Future Work

This section will propose ideas on how to extend the use of anisotropic diffusion and improve its over all performance.

6.2.1 Parallelization

One way to drastically improve the speed of level set methods in general is to distribute the computations on a cluster of processing units, i.e. parallelization of level set methods. While setting up a cluster of workstations may seem like a cumbersome task, the central processing units (CPU) of today are increasingly using a dual core technology. This new technology will allow parallelization on a single desktop or laptop computer. Even more, today there exists workstations with two or more dual core CPUs and apple is soon to release their Mac Pro with two quad core CPUs. This will allow for numerous parallel processes for computations of level set methods. The blending algorithm proposed in this thesis can easily adapt to this new technology. It can be done by dividing a filter pass into several simultaneous processes, each one filtering different parts of the surface. However, the greatest benefit would come from the speed up of the CSG operations and the initialization and rebuilding of level sets, since they can also be divided into different regions distributed on separate processing units.

6.2.2 Localization with quadrics

Since the Gaussian blending operator is a smoothing operator it seems like a good idea to use the technique for local surface smoothing as well. One way to localize the smoothing operation would be regionally constrain the region of influence for the smoothing operator by using quadrics or super quadrics. This method has been proven successful for curvature based smoothing and sharpening [1].

6.2.3 Surface smoothing with shape preservation

A drawback of global surface smoothing, both morphological and by means of anisotropic Gaussian diffusion, is that it tends to also smooth away sharp structural features of the surface, not only the noise on the surface. By smoothing the surface with an adaptive Gaussian kernel this may possibly be achieved as it has been for polygonal meshes [16].

Fig. 6.1. Greek bust. A 3d scanning of a Greek bust that have been repaired by copying a nose from another model and copying the right cheek, the copied parts have then been pasted with anisotropic diffusion blending to their respective place.

Fig 6.2. Winged horse. Wings have been copied from a griffin and then pasted on a horse. The left wing is a mirrored duplication of the right wing. The wings and the body have been blended by means of anisotropic diffusion. Close ups of the blending is shown in fig 5.3.

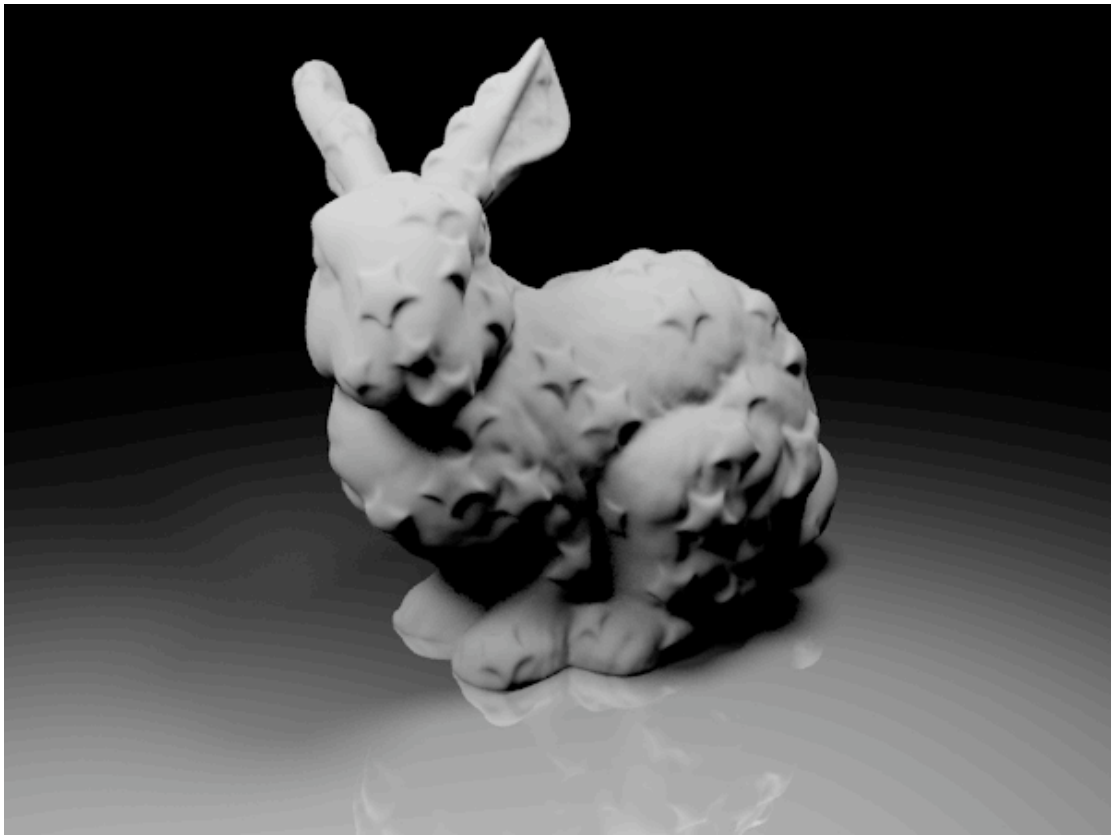


Fig 5.6 Geo textured bunny. The Stanford bunny has been textured with 56 stars in different sizes that then have been blended with the bunny.

7 References

[1] K. Museth, D. Breen, R. Whitaker and A. Barr, "Level Set Surface Editing Operators", *ACM Transactions on Graphics*, Vol. 21(3), *ACM SIGGRAPH '02* (San Antonio, TX) pp. 330-338, 2002.

[2] S. Osher and J. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of ComputationalPhysics*, 79, pp. 12–49, 1988.

[3] A. Brodersen, K. Museth, S. Porumbescu and B. Budge, "Robust Geometric Texturing Using Level Sets", *IEEE Transaction On Visualization and Computer Graphics*, submitted August 2006.

[4] BOUGUET, J.-Y., AND PERONA, P. 1999. 3D photography using shadow in dual space geometry. *International Journal of Computer Vision 35*, 2 (Nov/Dev), 129–149.

[5] M. Nielsen and K. Museth, "Dynamic Tubular Grid: An Efficient Data Structure and Algorithms for High Resolution Level Sets", *Journal of Scientific Computing* Vol. 26, No. 3, pp. 261-299, March 2006

[6] D. Breen and R. Whitaker. A level set approach for the metamorphosis of solid models. *IEEE Trans. Visualization and Computer Graphics,* 7(2):173–192, 2001.

[7] Adalsteinsson, D. & Sethian, J. A. 1995. "A fast level set method for propagating interfaces." Journal of Computational Physics. 118(2)269–277.

[8] Whitaker, R. T. 1998. "A level-set approach to 3d reconstruction from range data." International Journal of Computer Vision. 29(3)203–231

[9] Strain, J. 1999. "Tree methods for moving interfaces." Journal of Computational Physics. 151(2)616–648.

[10] Losasso, F., Gibou, F., & Fedkiw, R. 2004. Simulating water and smoke with an octree data structure. ACM Transactions on Graphics. 23(3)457–462.

[11] M. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, NJ, 1976.

[12] C.H. Lampert and O. Wirjadi. *An optimal non-orthogonal separation of the anisotropic gaussian convolution filter*. IEEE Trans. Image Processing, accepted for publication, 2006

[13] G. Taubin. A signal processing approach to fair surface design. In *Proceedings of ACM SIGGRAPH 95*, pages 351–358, 1995.

[14] Taubin, L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multiresolution modeling on arbitrary meshes. In *Proceedings of ACM SIGGRAPH 98*, pages 105–114, 1998.

[15] M. Desbrun, M. Meyer, P. Schr¨oder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. *Proceedings of ACM SIGGRAPH 99*, pages 317–324, 1999.

[16] Y. Otake, A. Belyaev, H. Seidel "Mesh Smoothing by Adaptive and Anisotropic Gaussian Filter Applied to Mesh Normals". In *Vision, Modeling and Visualization 2002*, pages 203-210, Erlangen, Germany, November 2002

[17] M. Nitzberg and T. Shiota. Nonlinear image filtering with edge and corner enhancement. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14:826–833, 1992.

[18] j.-M. Guesebroek, A. W. M. Smeulders and J. van de Wiejer, "Fast anisotropic gauss filtering" in ECCV *'02: Proceedings of the 7th European Conference on Computer Vision-Part 1*. London, UK: Springer-Verlag, 2002, pp. 99-112