I N F O R M A T I K

Geometric Modeling Based on Polygonal Meshes

Leif P. Kobbelt

Stephan Bischoff   Mario Botsch   Kolja Kähler

Christian Rössl   Robert Schneider   Jens Vorsatz

MPI–I–2000–4-002   July 2000

FORSCHUNGSBERICHT   RESEARCH REPORT

M A X - P L A N C K - I N S T I T U T
F Ü R
I N F O R M A T I K

Stuhlsatzenhausweg 85   66123 Saarbrücken   Germany

Author's Address

Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany
{kobbelt,bischoff,botsch,kaehler,
roessl,rtschnei,vorsatz}@mpi-sb.mpg.de

Contents

1. Introduction and overview

The use of polygonal meshes for the representation of highly complex geometric objects has become the de facto standard in most computer graphics applications. Especially triangle meshes are preferred due to their algorithmic simplicity, numerical robustness, and efficient display. Flexible and effective algorithms have been developed which combine results from approximation theory, numerical analysis and differential geometry and apply them to the discrete setting of polygonal meshes.

To a certain extend most of these techniques were already available for NURBS–based surface representations and have recently been generalized to unstructured polygonal meshes such that today splines can be substituted by polygonal meshes in many applications. The advantage of switching to this representation is mainly due to the fact that algorithms for polygonal meshes usually work for shapes with arbitrary topology and do not suffer from the severe restrictions which stem from the rigid algebraic structure of polynomial patches. Another advantage of triangle meshes is that they can be used for many stages of the typical processing pipeline in geometric design applications without the need for inter–stage data conversion. This accelerates the overall processing time and reduces the potential for round–off errors.

The motivation for using polygons to describe freeform geometry is quite obvious: while simple shapes can be characterized by manageable functional expressions, the complexity of those expressions explodes if the shapes are becoming more complicated. Hence *piecewise* representations are preferred since higher complexity can be obtained by simply using *more* segments (with constant complexity). The extreme case of piecewise representations are polygons: All we need are sample points on the given curve or surface and the corresponding surface description results from connecting these samples by lines or triangles.

Representing a given (real or virtual) surface geometry by a polygonal mesh is usually an approximation process. Hence there is no unique polygonal 3D–model but the density and distribution of sample points and the specific way how these samples are connected by triangles provide many degrees of freedom. For efficient storage and modeling with polygonal meshes, we have to choose a specific instance among those many possible models.

The most important criteria according to which the different polygonal approximations to a freeform surface can be rated, are: *smoothness* and *complexity*. Here the smoothness of a triangle mesh is measured by some discrete reformulation of concepts known from differential geometry. One simple example would be to use the angle between the normal vectors of adjacent triangles to measure the discrete curvature. More sophisticated measures will be presented in Section 3.

The complexity of a triangle mesh is measured by the number of vertices or the number of faces. It characterizes in some sense the computational costs that are required for displaying or modifying the mesh. Hence for computationally more expensive operations (or on low–performance computers) a mesh with less triangles would be preferred while cheap operations and high–performance computers allow for higher complexities.

In a typical application the choice of a polygonal model is constrained by a minimum smoothness requirement and a maximum complexity requirement. Inbetween these lower and upper bounds an optimal trade–off has to be found since higher smoothness increases the complexity and lower complexity decreases smoothness. The major techniques that are used to adjust both properties are *refinement* to increase smoothness and *decimation* to decrease complexity.

Both techniques together enable the generation of *hierarchical models* where different levels of the hierarchy are characterized by a varying level of detail. Here it is important to understand the conceptual difference between topological hierarchies (coarse/fine) and geometrical hierarchies (smooth/non–smooth) which refer to the above quality criteria respectively. While decimation reduces complexity and hence always removes detail information, the refinement can be used to either re–insert detail information or to increase smoothness without adding detail (cf. Fig. 1).

The mesh processing pipeline

This tutorial is organized according to a (virtual) mesh processing pipeline. The first step in such a pipeline is usually the generation of a geometric model based on measured point data. The raw data is obtained by mechanical or optical scanning of an object's surface (Section 2.1) and the "point cloud" is subsequently converted into a triangle mesh by connecting nearby samples (Section 2.2). Other sources for surface samples are volume data sets or the results of numerical simulations.

Once the mesh models exist, we can analyze their surface quality. This is done by generalizing quality criteria such as curvatures from continuous smooth surfaces to discrete meshes (Section 3).

As mentioned above, the raw triangle mesh data might not be appropriate for a given application in terms of smoothness or complexity and hence we have to apply refinement or decimation techniques respectively. In any case we enrich the plain polygonal mesh data by some hierarchical semantics in terms of increasing smoothness or decreasing complexity.

Refinement can be considered as building up the hierarchy *from coarse to fine* where so–called *subdivision schemes* insert new vertices into the mesh without introducing geometric detail (so that smooth meshes emerge, Section 4.1). Another approach to generate a coarse–to–fine hierarchy are

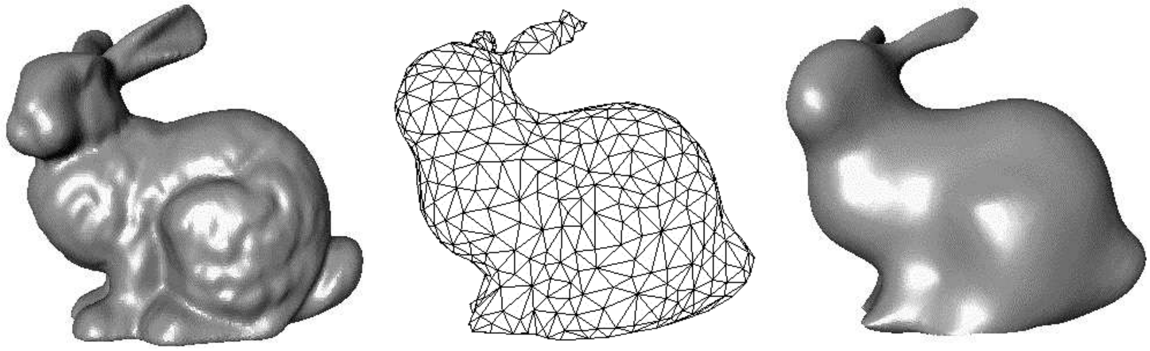


Figure 1: *Mesh decimation takes the original data (left) and computes a coarse approximation with less detail (center). Refinement can either re–insert the removed data to reconstruct the original model (topological hierarchy) or it can insert new points to generate a smooth mesh with high complexity but low geometric detail (right, geometrical hierarchy).*

remeshing techniques where the newly inserted vertices during refinement are sampled from some original, highly detailed surface (Section 4.2). In a remeshing algorithm the refinement also adds geometric detail such that the resulting surface is not necessarily smooth but a resampled version of the original surface.

Mesh decimation builds up the hierarchy *from fine to coarse* since vertices are removed from a detailed mesh such that coarser and coarser approximations are generated (Section 5.1). Starting from the coarsest level of detail, the original mesh can be reconstructed incrementally by re–inserting the removed vertices in reverse order (*progressive meshes*). Alternatively the removed vertices can be re–inserted but with their position altered in order to increase smoothness while suppressing any geometric detail. Here, the vertex positions which guarantee optimal smoothness can be computed by *discrete fairing* techniques (Section 5.2). The following table depicts the relation between the different techniques presented in the respective sections.

	smooth	non–smooth
coarse–to–fine	Subdivision	Remeshing
fine–to–coarse	Discrete fairing	Mesh decimation

Finally, the preprocessed mesh models can be modified by sophisticated editing operations. Freeform modeling can be implemented based on subdivision schemes or based on discrete fairing techniques (Section 6). The hierarchical structure further enables multiresolution modeling where global deformations with preservation of the detail information are possible.

2. Data acquisition and mesh generation

In creating a computer model of a real–world object, the first task is to measure the relevant properties of the object (its geometry, surface texture, volumetric density information, etc.). This raw data then serves as a base for further processing, for example reconstruction of the object surface.

We define *data acquisition* as the process starting with actually capturing the data up to the construction of a consistent model from these samples. This is described in detail in Section 2.1. Techniques to create a surface mesh from the output of the data acquisition stage will then be discussed in Section 2.2.

2.1. Data acquisition

2.1.1. Introduction

There is a multitude of applications that need to acquire real–world data, with varying requirements and demands on the measurement process. The following is a collection of some popular application domains:

Reverse engineering: The geometry of a real part is scanned to create a CAD/CAM representation; this model can then be used for further editing and integration with other modeled parts, as well as for simulation purposes. E.g. in the car manufacturing industry, virtual crash tests can be performed in a non–destructive fashion.

Interrogation: A real part is scanned for comparison with its specifications; measured deviations can be used for recalibration of the manufacturing process.

Pre–surgical planning: To simulate the implantation of an artificial limb, both the patients body and the part to be implanted are scanned and the operation can then be simulated in a virtual environment. For brain surgery, visualization of a volume scan of the patients head enables the surgeon to plan the path of a probe in advance, in order to avoid damage to highly sensitive tissue.

Diagnostics: Visualization of volume scans is an important tool in medical diagnostics today. Shape and size of internal organs can be segmented from the volume data; this and additional information derived from the scans (mass density distribution, speed of blood flow, etc.) can provide much more insight into body–internal structures and processes than conventional examination methods.

Surface reconstruction: Using photogrammetric methods, three–dimensional terrain can be reconstructed from satellite images; this also applies to creation of architectural models from large buildings.

Custom fitting: Computer models of generic products (prosthetics, clothes, car seats) are built and manipulated by software to fit the costumer's needs.

E–commerce: With the rise of Internet shopping, models of real objects need to be transmitted to potential buyers.

Animation: Models of characters and props can be used in film production for creation of special effects.

The generation of a CAD model of an industrial part or a prosthetic requires very precise samples of the object surface; in medical diagnostics one is interested in distinguishing specific types of body tissues; e–commerce applications don't need high precision as much as catching visual appearance, i.e. color and texture. To satisfy these different needs, a number of scanning techniques exist today.

Apart from mechanical probes, which sample a surface through physical contact, non–intrusive methods are more popular today. They do not disturb the physical environment, it is possible to get reliable data even from soft materials (e.g. human skin), and the scan process is generally faster.

A popular way to acquire surface data is *range scanning*; here essentially the distance from the scanning device to sample points on the object *surface* is estimated. There are a number of ways to measure distances, depending on range and application: optical triangulation, interferometric techniques using coherent laser light or "time of flight" (radar principle).

Especially in the medicine sector, *volumetric* data is measured by a number of methods, depending on the specific type of matter (tissue, bone, etc.) that is of interest. Conventional *x–ray imaging* can be used to reconstruct the arrangement of blood vessels (angiography). *Computer Tomography* (CT) also relies on x–rays, where slices of the human body are scanned using a rotating tube. From the variations in the measured x–ray absorption the spatial distribution of mass density can be reconstructed. In *Magnetic Resonance Tomography* (MRT) high frequency electromagnetic pulses are generated that alter the spin of protons in the body. When the protons re–align, a signal is emitted that is measured as an electric current in a detector coil.

The raw output from these and other methods typically consists of a three–dimensional voxel grid. For further processing, surfaces need to be extracted (*segmentation*). This type of volume data is very limited in spatial resolution, and so more accurate 3D surface data is often acquired using different techniques (e.g. by range scanning). To get more complete information for diagnostic purposes, datasets from different scans need to be merged and registered with each other.

Though in this section we will concentrate on range scanning devices, the reconstruction of surfaces from volumetric datasets will also be a topic in Section 2.2.

3D positions can also be reconstructed from photographic images, which has been studied already in 1913[24]; this has led to photogrammetric modeling methods, facilitating reconstruction of geometry from a photographed scene[15].

In the context of creating polygonal representations of complex models, range scanning devices based on the triangulation principle are the most popular ones for their flexibility (scanning devices come in all sizes from pens, portable constructions up to permanently installed whole–body scanners) and wide range of applications.

We are now going to discuss the process of data acquisition for range scanners based on the triangulation principle.

2.1.2. Range scanning process overview

The scan process is divided into three basic steps:

1. **Calibration:** The system's parameters are estimated according to hardware and environmental characteristics; this is a prerequisite for obtaining accurate measurements.
2. **Scanning:** The object surface is sampled from one view, resulting in a dense range map. Multiple passes have to be performed to get a set of samples covering the complete surface.
3. **Registration:** The acquired range scans reside within their own local coordinate system. Thus, they have to be aligned with each other to express them in a global frame.

The result is a dense set of sample points representing the object surface. These can then be processed to e.g. produce a triangulated model (see 2.2).

We will now explain these stages in some detail.

2.1.3. Calibration

The image of an object on the scanner's sensor array is dependent on physical parameters of the system and the environment. To be able to compute exact distance measurements for surface samples, proper system calibration is crucial. This can be an arduous task, if there are a lot of parameters[18].

Object characteristics like shape, color, texture and spectral characteristics of the surface determine the way in which light is reflected. Laser light of a certain wavelength may be absorbed, subsurface scattering can distort the laser reflection, or strong reflections may lead to wrong readings[18].

The *external parameters* of the sensor system are its pose (rotation and translation of the optical center), the *internal parameters* are e.g. size of the CCD array, pixel aspect ratio, focal length and spectral characteristics of the lenses. These are of course entirely dependent on the specific scanner setup.

In addition, the lighting conditions of the *environment* have to be considered.

It is practically impossible to exactly determine all of the abovementioned parameters (or even more), so the calibration process works with a *mathematical model* of the system approximating its behavior. The behavior of the real system is measured to adjust the parameters of the model, which might be a simple standard camera model as used e.g. in OpenGL[39], but can get arbitrarily complex, reflecting rigid transformations, geometric distortions in the samples as well as image space color distortions[20, 18].

A common procedure is to use a *calibration target*, typically a white board with a regular pattern. Since the object geometry and surface properties are known, intrinsic and external camera parameters can be corrected for by analyzing the captured images (color, geometry distortion) and scans of the object (camera pose)[35].

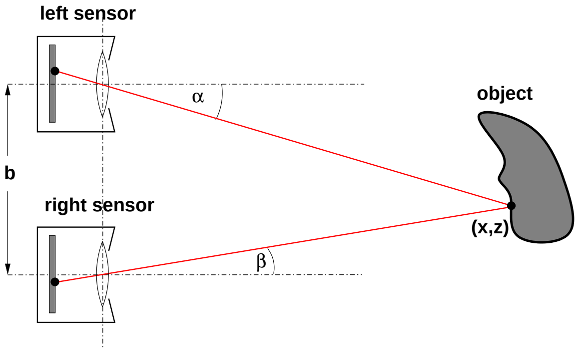
The calibration process also yields information about the usable *scan volume*, i.e. the range in which samples can actually be taken within reasonable bounds of sampling resolution and error. This volume is often quite small, e.g. $14cm^3$ ([18]).

Multiple iterations of the procedure are often necessary, manually adjusting system parameters within the degrees of freedom, to achieve a reasonable balance between resolution, accuracy and the size of the scan volume — e.g. by varying the sensor setup, or the distance from the object to the scanner.

2.1.4. Scanning: range from stereo

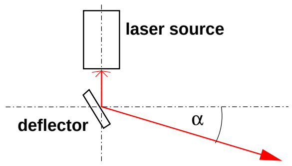
Systems based on optical triangulation work by generating distance values from stereo information. They exist in two flavors: active and passive.

In a *passive* system two or more images are taken by (usually two or more) cameras which are mounted in a fixed spatial relation to each other. The images are either already captured in digital form (CCD camera) or have to be digitized. Pixels in one camera image are now matched to pixels in the image of another camera[5]. Assuming matched pixels really correspond to the same surface point, one can calculate an estimate of its 3D position. The following picture shows the basic principle in 2D:



Light is reflected off a surface point and projected onto each camera's CCD array. Given the length b of the *base line* (the line between the two cameras) and the angles α and β related to the rays from the point through the optical center of each camera to the sensor, the intersection point (x,z) of these rays can be computed. By assigning depth values to each pixel in the sensor array, a range map is created as the result of a single scan.

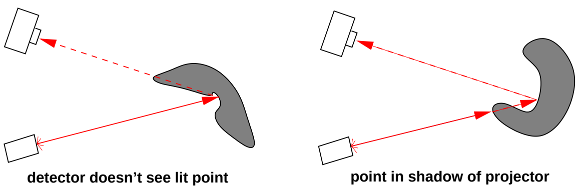
Successful matching of pixels in a passive stereo system relies on distinguishable intensity variations of the surface. This led to development of *active systems*, where the principle is altered by replacing one detector with a *controlled light source* such as a laser or a projector casting a grid of structured light[22]:



The sensor now detects the reflection of the point(s) on the object illuminated by the light source. A laser scanner traces the surface along a grid, while other systems use a succession of different stripe patterns or just a single vertical stripe moving across the surface[31].

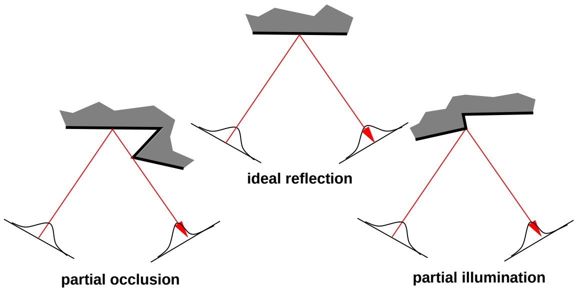
Because of their increased robustness, active systems are by far dominant in industrial applications. Problems may still arise, though, e.g. if textures interfere with dense stripe patterns. In general, scanning works best with "well–behaved" surfaces, which are smooth and have low reflectance.

Triangulation scanners have the common problem of shadowing, due to the separation of light source and detector; parts of a non–convex object may not be reached by the light from the projector or may not be seen by the detector, as shown here:



Obviously, the longer the base line, the more shadowing occurs. On the other hand, a longer base line increases numerical precision of the distance measurements. So there is a tradeoff between getting enough data and achieving high accuracy. Some approaches use more than one camera to get more data and to simultaneously increase the reliability of the distance estimate[29, 31].

Related to shadowing is the *beam location error*. To locate the precise position of a detected light spot or stripe projected onto the object surface, it is generally assumed that the light beam and its reflection have Gaussian intensity distribution, with the highest intensity at the center of the "bump". This results in problems in locating the stripe, if the basic assumption of a planar surface and a fully visible stripe is violated; finding the center of the stripe (or spot) fails in this case.



A possible solution to this problem is so–called *spacetime analysis*[13]: Instead of guessing *where* the center of the beam on the surface is at a given time, the change in intensity per pixel over time as the beam sweeps across the surface is considered. It is assumed that the beam is wide compared to the step width of the sweep; over time, the intensity of a specific pixel increases, reaches its maximum, then decreases again. This time–varying profile is always Gaussian, so it can be reliably estimated, *when* the beam was centered on that pixel.

2.1.5. Registration

When scanning complex objects, multiple scans are taken – which usually means, that either the object or the scanner have to be repositioned. After scanning, the range images are given in their own local coordinate system; these datasets need to be put into one common frame for reconstructing the surface.

The problem can be seen as equivalent to finding the rigid transformations of the scan sensor between the individual scans, which for some systems is already known, e.g. cylindrical scanners, where the scan head is moved under software control – the relative motion is thus likely to be directly available. Often, especially for surfaces with little inherent structure, special markers are applied to the physical object; the desired transformation is then the one that matches a marker in one image onto the other.

If no such external information can be used, or if refinement of the solution is necessary due to lack of precision, registration is done by an iterative optimization process that tries to match the point clouds of individual scans by minimizing distances of point pairs. It is generally assumed that the scans have enough overlap and are already roughly aligned (e.g. by interactively pairing prominent surface features).

For matching multiple range images the standard approach is to register them *pair–wise* using a variant of the *iterated closest point method* (ICP)[9, 12, 40]. For two point sets A and B this roughly works as follows:

1. Determine starting transformation T from A to B.
2. Estimate correspondences between sample points in A to points in B (using the current T), where the corresponding point in B either is one of the original samples or lies on the surface reconstructed from B.

3. Update T with the rigid transformation that minimizes the mean quadratic distance between the points of these pairs.
4. Repeat steps 2 and 3 (potentially re–pairing points) until convergence.

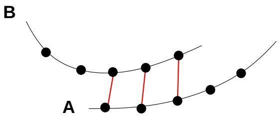
A and B are then locked in place and another scan is added. Ideally, the transformations are computed precisely and after "working your way around", all scans fit together perfectly. In practice though, the data is already contaminated: Surface points measured from different angles or distances result in slightly different sampled positions. So even with a good optimization procedure, it is likely that the n–th scan won't fit to the first. Multiple iterations are necessary to find a global optimum and to distribute the error evenly.

We have to solve two problems at once here: a) Find corresponding point pairs; b) find the transformation that minimizes the distance. Since the second step depends on the first, it is essential to find "good" point pairs. As we are dealing with a non–convex optimization problem, the algorithm can get stuck in a local minimum, if the initial starting point is not good (which lowers the chances of correctly pairing points).

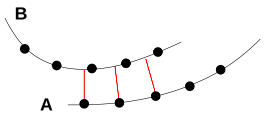
Finding good pairs is a non–trivial task. Common heuristics are to pair each point A_i:

- with its nearest neighbor B_j in space;
- with the nearest B_j in normal direction (point normals are usually available with the range data);
- with the intersection point of the normal and the surface reconstructed from B.

When pairs have been fixed, the transformation that aligns them can be computed as a closed–form solution[37]. Finding better point pairs becomes easier and more reliable from one iteration to the next, so the process converges to some minimum.



match closest neighbor in space



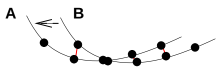
project point onto plane

These methods for pairing points often fail, if the surfaces are not smooth enough or the current registration is too bad. A number of heuristics have been proposed to choose "good" pairs and discard the rest; among these are[36, 19]:

- Only consider overlapping regions.
- Don't match points if their distance is too large.
- Only match points with similar point normals.
- Don't pair boundary vertices.
- Only consider the 90% best matches.

Searching closest points in sets of hundreds of thousands of points in space can take considerable time. The procedure can be sped up by projecting points directly from one range map into another[38] and performing the pairing there, reducing the problem to 2D. It is also possible to use image intensity information for the pairing decisions[31].

The distance minimization procedure can lead to problems, if most of the paired points are very close to each other and only a few are far apart. Then the surface is prevented from moving closer to the correct solution, sticking to an unacceptable local optimum:



registration of A and B stuck in local optimum

For this reason, other approaches have been proposed that do not directly minimize the distance between paired points. A detailed discussion of these methods is beyond the scope of this tutorial, so we only give some pointers to the literature here: Chen and Medioni[12] e.g. minimize the distance of a point to the tangent plane through it's peer, Masuda and Yokoya[26] minimize the sum of squared distances between *all* given pairs.

2.2. Triangulation of point clouds

2.2.1. Introduction

After the data acquisition (including the registration) phase the next step in the surface reconstruction process is the generation of a single surface representation, e.g. one overall triangle mesh, from the acquired data.

Depending on the application and the structure of the input data different reconstruction problems arise:

Unorganized data: We have no additional information other than the sampled points. This is the most general approach and therefore also the computationally most expensive one, because we do not exploit any additional information we might have. There was a state-of-the-art report at Eurographics '98 by Mencl and Müller[28].

Contour data: In medical applications the input model is often sliced into thin layers each of which is then digitized to a contour line. Here one can exploit the fact that these contours are closed polygons arranged in parallel stacks.

Volumetric data: Also in the medicine sector we have volumetric data measured by e.g. MRT or CT imaging. Here we get a 3D-grid as input data and basically have to do iso-surface extraction, e.g. using the well-known *Marching Cubes* algorithm[25]. But the Marching Cubes algorithm does not produce optimal results. If the edge length of the voxel grid is too big aliasing artifacts are clearly visible. Additionally arbitrarily bad shaped triangles can occur in the resulting mesh (cf. Fig. 2). The grid size should be chosen carefully since the mesh size grows quadratic.

Range data: The input data is a collection of range images that are assumed to be registered into a common coordinate system (cf. Section 2.1.5). A range scanner typically produces a rectangular grid of depth values or 3D-points, so we have adjacency information for the points of each

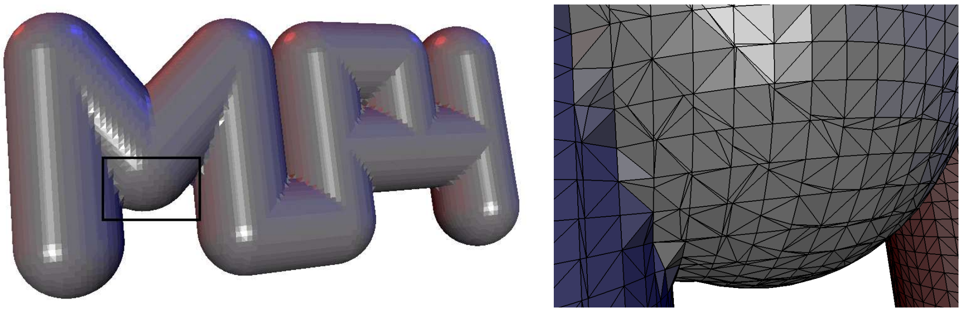


Figure 2: *A model generated by applying the Marching Cubes algorithm to a volumetric distance function (left). Notice the aliasing artifacts at sharp feature lines and the badly shaped triangles (right).*

single scan. The difficulty is the integration of the different scans into one single mesh. Another problem is the huge amount of data, that is typically generated by the uniform (over-) sampling of the scanner.

Regardless of the underlying structure of the data we can divide the possible approaches into two groups, depending on whether they produce an:

interpolation of the input data: the vertices in the resulting mesh are the original sampled points.

approximation to the sample points. Especially for range data we want an approximating rather than an interpolating mesh to get a result of moderate complexity.

The approaches presented in this section can be classified as follows:

Sculpting based: This class of algorithms is used to reconstruct a surface from an unorganized point cloud and produces an interpolating mesh. These algorithms have in common that they first build a tetrahedrization (usually the 3D Delaunay triangulation[30, 6]) of the point set to get some kind of global shape of the object. Afterwards heuristics are used to select a subset of the 2-simplices (i.e. triangles) as the resulting mesh. These approaches are capable of reconstructing surfaces from very sparsely sampled data. The drawback is the computational complexity and memory consumption for building the initial tetrahedrization.

Volume based: This technique can be used to reconstruct a surface from structured or unstructured sample data. Here an estimated distance for each sampled point is inserted in a voxel or octree structure and the result is extracted from this structure, e.g. using the Marching Cubes algorithm. Therefore these approaches produce approximations to the sampled points, and the edge length of the volumetric grid controls the complexity of the output.

Incremental/region-growing: This class of algorithms starts with a seed and incrementally grow this seed until the whole input data is covered. The seed may be a triangle, an edge, a first range image or a wireframe approximation.

2.2.2. Sculpting based approaches

2.2.2.1. Alpha-Shapes Edelsbrunner and Mücke[17] generalized the notion of convex hull to the parameterized α-*shapes*. The α-shape of a set of points in 3-space is a polytope that does not need to be convex, connected or a 2-manifold. A triangle, edge or vertex belongs to the α-shape iff it has a circumsphere of radius at most α that is empty of other sample points. For $\alpha = \infty$, the α-shape is the convex hull of the point set, for $\alpha = 0$ it is the point set itself. As α decreases the α-shape shrinks by developing cavities (cf. Fig. 3).

The α-shape is related to the Delaunay triangulation (DT, cf. [30, 6]) in the following way:

$$\forall \text{ simplex } s \in \text{DT} \;\; \exists\, \alpha_s > 0 : \quad s \in \alpha_s\text{-shape}.$$

Conversely for $0 \le k \le 2$ every k-simplex of the α-shape is a simplex of the DT and therefore

$$\{k\text{-simplices of DT}\} \;=\; \bigcup_{0 \le \alpha \le \infty} \{k\text{-simplices of } \alpha\text{-shape}\}\,.$$

The α-shape of a point set can be calculated by first building the DT and eliminating all k-simplices, $0 \le k \le 3$, whose minimum enclosing sphere has radius greater than α. Think of the DT filled with styrofoam and the vertices being more solid. Then the geometric intuition behind α-shapes is to use a spherical eraser tool with radius α that carves out the styrofoam wherever it can pass between the vertices.

In a last step the triangles that belong to the resulting surface are extracted out of the α-shape based on the following heuristic: A triangle belongs to the surface if at least one of the two α-spheres that interpolate the triangle's vertices is empty of other points.

The difficulty with this approach is to find a suitable value for the global parameter α. If it is too small, holes and gaps will occur; if it is too big, cavities may not be preserved. In the presence of varying sampling density this gets even more complicated.

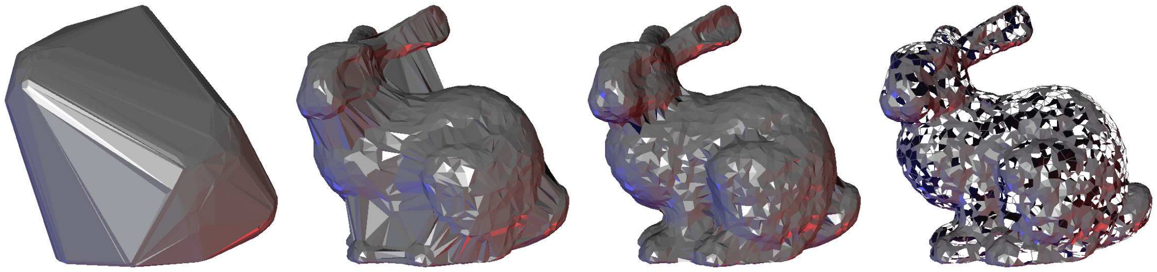


Figure 3: *α-shapes for decreasing values of α. The models have been created using the "Alpha Shapes" software by Edelsbrunner et al.* (`http://www.alphashapes.org/alpha/index.html`).

Pros/Cons:

+ Elegant theoretical formulation
– Global variable α has to be found experimentally.
– Problems with varying sampling density.
– Computationally expensive.

2.2.2.2. Voronoi-Filtering Amenta et al.[3, 2] present an algorithm for surface reconstruction from a given point cloud. The remarkable feature is the provability of their algorithm: given a "good" sample of a smooth surface (i.e. twice-differentiable manifold) without boundary the algorithm will reconstruct a surface that is topologically equivalent to the original surface and will converge to it both pointwise and in surface normal as the sampling density increases.

The intuition behind the definition of a "good" sample is that featureless areas can be reconstructed from fewer sample points, while in detailed regions the sampling has to be more dense. Actually a set of sample points S of a surface F is good, if the sampling density is (up to a factor r) inversely proportional to the distance to the *medial axis* (ME) of F:

$$\begin{aligned} &S \text{ is a "good" sampling } :\Leftrightarrow \\ &\quad \forall p \in F : \; dist(p,S) \;\leq\; r \cdot dist(p, ME(F)) \end{aligned}$$

The *medial axis* of F is the closure of all points having more than one closest point on F and can be thought of as the continuous extension of the *Voronoi diagram*[30]. Therefore this definition of a good sample respects the curvature of F as well as the proximity of two sheets of the surface. Amenta et al. prove their results for $r \leq 0.06$, but also state that in practice $r = 0.5$ is generally sufficient.

For every sample point $s \in S$ two *poles* p_s^+ and p_s^- are defined as the two vertices of the *Voronoi cell* of s that are farthest from s and on opposite sides of the surface F. The observation is that the Voronoi cell of a vertex is long and thin and roughly perpendicular to the surface F. Therefore $n_s^+ := p_s^+ - s$ and $n_s^- := p_s^- - s$ are good approximations to the surface normal at s (it can be proven that the angular error is linear in r).

The actual algorithm is roughly sketched as follows:

1. Compute the Voronoi diagram of S.
2. $\forall s \in S$: compute the poles p_s^+ and p_s^-.
3. Compute the Delaunay triangulation of the union of S and all poles.
4. Keep the triangles whose vertices are all sample points (*Voronoi filtering*).
5. Normal filtering and manifold extraction.

After performing steps 1-4 one gets what Amenta et al. call the *crust* of the sample points. This is not necessarily a manifold and therefore a normal filtering step discards triangles whose normals differ too much from n^+ or n^-. Afterwards a manifold extraction keeps only the outside surface of the remaining triangles.

Pros/Cons:

+ Reconstruction from point clouds.
+ Provable reconstruction.
+ Adaptive resolution (by adaptive sampling).
– Complexity of $O(n^2)$ and high memory needs because of the Delaunay triangulation.
– Problems with noise, sharp edges, boundaries.

2.2.2.3. Others An approach similar to the one by Amenta et al. is the "Delaunay sculpting" of Boissonnat[10]. He also builds the Delaunay triangulation in a first step and removes tetrahedra using a heuristic based on their circumspheres. To overcome the problem of the global parameter α in α-shape based approaches, Edelsbrunner[16] introduced *weighted α-shapes*. Teichmann and Capps[34] use varying values of α to adapt to the local sampling density. Bernardini et al.[7] automatically determine the optimal value for α and use the α-shape of the point set for the construction of a signed distance function.

2.2.3. Volumetric approaches

2.2.3.1. Reconstruction from unorganized points In this paper Hoppe et al.[21] address the issue of reconstructing a surface from unorganized points. The algorithm consists of two stages. In the first stage a *signed distance function* is constructed. This function maps a point in 3-space to an estimated signed distance to the unknown surface. Therefore the unknown surface is approximated by the zero-set of this function, points outside the object have positive, points inside have negative distance. In the second stage the zero-set is extracted using the Marching Cubes algorithm.

The critical step is the first stage, the estimation of the signed distance. To construct it, an *oriented tangent plane* is associated to each sample point. These tangent planes are a good local approximation to the unknown surface and the distance of a point $p \in \mathbb{R}^3$ to it is defined to be the distance to the plane associated to the sample point nearest to p. While the estimation of an *unoriented* tangent plane to a sample point $s \in \mathbb{R}^3$ only requires a least square fit to s and its k nearest neighbors, the difficulty is the consistent orientation of these planes. Consider two samples points s_i, s_j that are geometrically close. If the sampling is dense and the surface is smooth, then the corresponding normal vectors at these points are assumed to be almost parallel, i.e. $|n_i n_j| \approx 1$. This would give us an unsigned distance function. To get a signed one we have to ensure that the two normals are not only almost parallel, but also pointing in approximately the same direction, i.e. $n_i n_j \approx +1$. This condition should hold for all sample points that are sufficiently close to each other.

This can be formulated as a graph optimization problem. The nodes represent the tangent planes and are connected by an edge if the corresponding sample points are sufficiently close. To decide which nodes to connect, an *enriched Euclidean Minimum Spanning Tree* is constructed. The orientation is propagated along this tree by traversing the minimum spanning tree with the cost of edge (s_i, s_j) being $1 - |n_i n_j|$, therefore favoring the propagation between samples with almost parallel normals. After this orientation step we are now able to define the signed distance function as described above.

Using a variation of the Marching Cubes algorithm the zero-set of the distance function is extracted. To alleviate the problem of badly shaped triangles Hoppe et al. collapse small edges in a postprocessing step using an aspect ratio criterion.

Pros/Cons:

- + It can handle large, unstructured point clouds.
- + It is robust in the presence of noisy input data.
- + One can control the output size by adjusting the grid size of the iso-surface extraction step.
- – There is no adaptive resolution. The vertices in the output are uniformly distributed, regardless of highly curved or featureless areas of the surface. If one does not want to lose small features this will lead to complex output meshes.
- – As already mentioned, the Marching Cubes algorithm can produce bad shaped triangles. This makes a postprocessing step necessary.

2.2.3.2. Reconstruction from range images The approach of Curless and Levoy[14] is similar to the method of Hoppe et al., but tuned for handling complex range data. They also build a signed distance function and get their result by an iso-surface extraction step. Additionally they take into account the special problems and structures that come with the integration of separate range images:

- *Range uncertainty*: For every sampled point of a range image one can derive a reliability value.
- *Utilization of all range data*, including redundant samplings.
- *Incremental updating*: After each scan a reconstruction can be done and be improved by adding another scan. This should be independent of the order in which the separate scans are processed.
- *Robustness*: The algorithm should produce stable results even in the presence of noise and outliers in the data.
- *Ability to fill holes*: When scanning non-trivial objects, there are always regions that cannot be captured because of self-shadowing (cf. Section 2.1.4).

The global distance function is generated by a weighted average of the distance functions of the individual range surfaces. The weighting should be chosen specific to the range scanning technology in order to take the range uncertainty into account. In their case the weights depend on the dot product between the vertex normal and the scanning direction, leading to greater uncertainty in regions measured under a flat angle. The averaging of redundant measurements can reduce sensor noise.

The distance function of a single range image is constructed by triangulating the sampled points using the pixel-grid neighborhood information and assigning a weight to each vertex. To evaluate this function at a point $v \in \mathbb{R}^3$, this point gets projected onto the range mesh along the sensor's line of sight. If an intersection x occurs at a triangle t the weight w is computed by barycentric interpolation of the weights of t's vertices and the result is the distance from v to x weighted by w.

The global distance function is evaluated on the voxel grid that is used for the iso-surface extraction afterwards. Whenever an additional range image is generated, the global distance and weighting function and their values at the grid vertices are updated.

The hole filling operates not on the reconstructed mesh, but directly on the volume. All points in the volume are classified to one of three states:

- *Unseen*: The initial value for all voxels.
- *Near the surface*: These voxels take on the signed distance and weighting values as described above.
- *Empty*: By performing a *space carving* step, i.e. by tracing all lines of sight from a sensors position to the observed points in the corresponding range image the tra-

versed voxels are marked as empty. This technique is very likely to remove outliers.

Holes in the extracted surface are frontiers between unseen and empty regions. To fill these holes the iso-surface extraction is not only performed for the zero-set of the distance function, but also between unseen and empty regions. By assigning specific distance and weighting values to empty and unseen grid vertices this can be done by one single extraction step.

To achieve space efficiency the volume gets run-length encoded. To achieve time efficiency by faster volume traversal each range image is resampled so that its scanlines are aligned to the scanlines of the volume grid. Therefore both the range image and the volume can simultaneously be processed in scanline order. This approach is well suited for handling huge amounts of data, as it was proven in the Digital Michelangelo project[18].

Pros/Cons:

+ Can handle huge amounts of data.
+ Robust (noise, outliers).
+ Output size controllable.
– No adaptive resolution.
– Marching Cubes problems.

2.2.3.3. Others Pulli et al.[32] present a method for the reconstruction from a sequence of range maps. They first build a hierarchical octree representation of the object. Every cube that is neither completely inside nor completely outside the object is recursively subdivided up to a maximum level. Afterwards a triangle mesh is extracted from the volume data. Their approach is able to handle noise and outliers and fill holes at missing data. Bajaj, Bernardini and Xu[4] build a signed distance function by first computing the α-shape of the object to which they fit implicit Bernstein-Bézier patches.

2.2.4. Incremental approaches

2.2.4.1. Ball pivoting The Ball-Pivoting Algorithm (BPA) of Bernardini et al.[8] generates an interpolating triangle mesh from a given unstructured point cloud. They assume that an oriented normal vector is available for each point and that the sampling density has a global minimum. The normal vectors are used to determine the surface orientation and for consistency checks when generating triangles (all three normals should point in roughly the same direction). Range data automatically fulfills both requirements. Otherwise techniques like the ones described in [21, 3] can be used to estimate the normal vectors, but this can be quite expensive (orientation problems, Voronoi diagram).

The basic principle of the BPA is very simple and intuitive: we start with a seed triangle and a ball of user defined radius ρ sitting on this triangle (i.e. interpolating its three vertices). This ρ-ball is pivoted around an arbitrary edge of the current boundary (initially the edges of the seed triangle) until it touches another sample point. This pivoting is basically a circular movement of the ball's center in the plane perpendicularly bi–secting the edge whilst always staying in contact with the two edge vertices. If the ρ-ball hits another sample point on its movement around the edge a new triangle is created from the edge and this point. The boundary is adjusted accordingly and the pivoting is continued. The update of the boundary front may change its topology, e.g. a loop of edges may be split, or two loops may be merged into one. As the ball walks on the sample points the mesh grows until the whole connected component is reconstructed. If there are multiple connected components, a new seed triangle is chosen and the process is repeated.

A nice feature of the BPA is that it is strongly related to α-shapes: by construction every BPA-generated triangle has an empty circumsphere of radius $\leq \rho$ (cf. Section 2.2.2.1). Therefore all triangles resulting from pivoting a ρ-ball are a subset of the ρ-shape of the point set. Because of this relationship it can be proven that for a sufficient sampling of a smooth manifold the BPA produces a homeomorphic approximation with an error bound of ρ. Additionally the BPA is guaranteed to generate 2-manifolds, so no cleaning up step is necessary. By using *out-of-core* processing techniques this algorithm is able to handle very large datasets (cf. [1]).

Pros/Cons:

+ Can handle large real-world scans.
+ Out-of-core triangulation possible.
+ Moderate memory consumption (no Delaunay triangulation).
– No adaptive resolution (requires uniform sampling).

2.2.4.2. Interactive approach While the existing techniques are off-line algorithms, the approach of Kobbelt and Botsch[23] incorporates user interaction *during* the surface reconstruction. Their approach generates a mesh approximating hybrid input data (e.g. points, triangles, or even NURBS patches). Resolution and alignment of the triangles can be adapted manually to varying detail level and quality requirements in different regions of the object (cf. Fig. 4).

The concept behind the user interface is to simulate a *virtual 3D scanning device*. The input data is displayed in an OpenGL window, the user can view it from arbitrary positions and angles. In an interactive scanning session the user adds one scan after the other to the previously generated model. One iteration consists of placing, scaling ($\rightarrow$ resolution) and orienting ($\rightarrow$ alignment) the object on-screen, determining the valid region of interest, extracting the patch and automatically stitching it to the already existing mesh.

When rendering geometry with enabled depth-buffering the depth value for every pixel is stored in the z-buffer. While the graphics system uses this information to determine mutual occlusion, the virtual scanner reads this data and unprojects it back into 3-space. The result is a range image con-

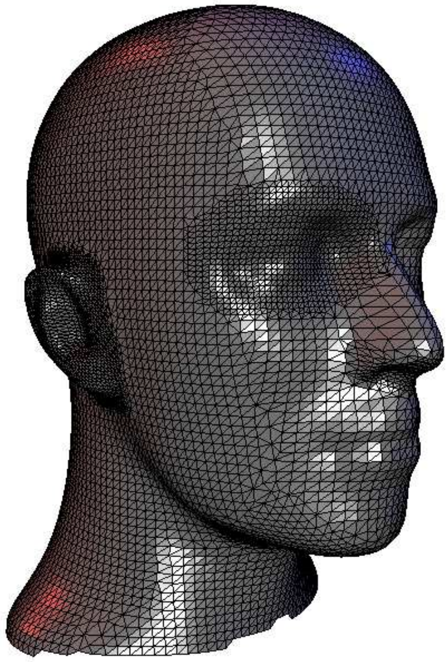


Figure 4: *The interactive approach provides an intuitive interface to locally adapt the mesh resolution to specific requirements for the resulting model by simply zooming the point cloud. Here the ear and the eye have been z-buffer scanned with a higher density.*

taining a 3D sample point for every pixel. This range image is filtered, a user defined mask selects the region of interest and the remaining vertices can be trivially triangulated because of the underlying pixel-coordinate parameterization. Zooming in on the object will result in a finer triangulation. By rotating the object so that geometric features are horizontally or vertically aligned, the edges of the triangulation will be aligned to these features by construction.

Since the scanning process is mainly the outcome of one rendering step all objects that can be rendered can also be z-scanned. So this approach can also be used for the remeshing of objects with poor triangle quality (e.g. meshes from CAD systems). When scanning point clouds the sampling is assumed to be sufficiently dense, so that the rendered points form a connected area without (large) holes on the screen. Smaller holes can be interpolated by the graphics hardware by choosing a bigger point size ($\rightarrow$ nearest neighbor interpolation).

If the acquired scan overlaps with the previously generated model, an automatic quality mask ensures that the better part is kept. This requires a projection of the new scan onto the old mesh. This expensive quadratic search can be reduced to one *ID-rendering* step that is done by the graphics hardware. The remaining patch gets stitched into the existing mesh using a modification of the *mesh zippering* algorithm of Turk and Levoy[36].

By out-sourcing the computationally expensive tasks to the graphics hardware (subsampling, range image, mesh projection) the program stays interactive even for large input data. The amount of input data only effects the rendering speed, the scanning and stitching only depends on the size of the object in screen space. The memory consumption is much lower than for most other approaches, since no additional space partitioning structures have to be generated (like e.g. Delaunay triangulation, Voronoi diagram or search structures).

An extension of this approach that uses feature-sensitive sampling instead of the z-buffer in order to avoid aliasing artifacts at curved feature lines is be presented in [11] (cf. Fig. 5).

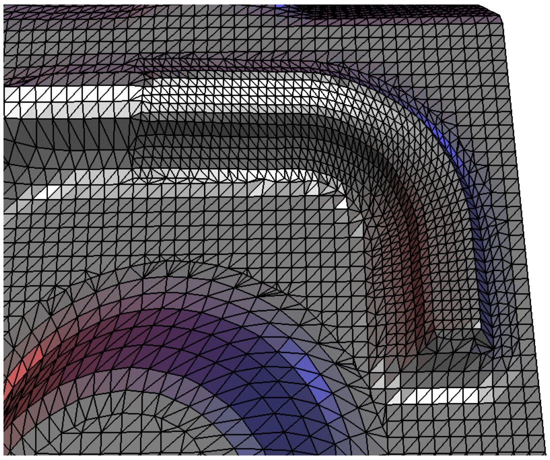


Figure 5: *Using feature sensitive sampling for curved feature lines enables optimal alignment of the mesh to the underlying geometry.*

Pros/Cons:

+ Interactivity: the user can directly influence the result.
+ Adaptive resolution and adjustable alignment.
+ Fast by using graphics hardware.
+ Low memory consumption.
– Interactivity: no automatic algorithm.

2.2.4.3. Others Boissonnat[10] starts with the edge connecting the two closest sample points and generates an interpolating mesh by iteratively adding triangles to the boundary front. Mencl and Müller[27] build a *surface description graph*, i.e. an enhanced Euclidean Minimum Spanning Tree, perform a feature detection step and finally fill this wireframe model with triangles, also leading to an interpolating mesh. The following methods all do reconstruction from range images. Soucy and Laurendeau[19] use *Venn diagrams* to partition the input data into non-overlapping regions which are re-parameterized and merged together. In a similar approach Turk and Levoy[36] incrementally take scans, erode overlapping parts and zipper them together, followed by a consensus geometry step in order to minimize registration errors. Rutishauser et al.[33] merge depth images by taking into account the error along the sensor's line of sight and a retriangulation step for the redundant data.

References

1. J. Abouaf. The florentine pietá: Can visualization solve the 450–year old mistery? *IEEE Computer Graphics and Applications*, 19:6–10, 1999.
2. N. Amenta and M. Bern. Surface reconstruction by voronoi filtering. In *Annual ACM Symposium on Computational Geometry*, 1998.
3. N. Amenta, M. Bern, and M. Kamvysselis. A new voronoi–based surface reconstruction algorithm. In *SIGGRAPH 98 Conference Proceedings*, pages 415–422, 1998.
4. C. L. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *SIGGRAPH 95 Conference Proceedings*, pages 109–118, 1995.
5. S. T. Barnard and M. A. Fischler. Computational stereo. *ACM Computing Surveys*, 14(4):553–572, 1982.
6. M. Bern and D. Eppstein. *Mesh generation and optimal triangulation*. World Scientific, 1992.
7. F. Bernardini, C. L. Bajaj, J. Chen, and D. R. Schikore. Automatic reconstruction of 3D CAD models from digital scans. *International Journal of Computational Geometry and Applications*, 9(4&5):327–370, 1999.
8. F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball–pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.
9. P. J. Besl and N. D. McKay. A method for registration of 3–D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–258, 1992.
10. J.-D. Boissonnat. Geometric structures for three–dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266–286, 1984.
11. M. Botsch, Ch. Rössl, and L. Kobbelt. Feature sensitive sampling for interactive remeshing. *Preprint*, 2000.
12. Y. Chen and G. Medioni. Object modelling by registration of multiple range images. *International Journal of Image and Vision Computing*, 10(3):145–155, 1992.
13. B. Curless and M. Levoy. Better optical triangulation through spacetime analysis. Technical Report CSL–TR–95–667, Stanford University, Computer Systems Laboratory, 1995.
14. B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH 96 Conference Proceedings*, pages 303–312, 1996.
15. P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry– and image–based approach. In *SIGGRAPH 96 Conference Proceedings*, pages 11–20, 1996.
16. H. Edelsbrunner. Weighted alpha shapes. Technical Report UIUCDCS–R–92–1760, Department of Computer Science, University of Illinois, Urbana–Champagne, IL, 1992.
17. H. Edelsbrunner and E. P. Mücke. Three–dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.
18. M. Levoy et al. The Digital Michelangelo Project: 3D scanning of large statues. In *SIGGRAPH 00 Conference Proceedings*, to appear.
19. H. Gagnon, M. Soucy, R. Bergevin, and D. Laurendeau. Registration of multiple range views for automatic 3–D model building. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 581–586, 1994.
20. J. Heikkilä and O. Silvén. A four–step camera calibration procedure with implicit image correction. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1106–1112, 1997.
21. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Computer Graphics (SIGGRAPH 92 Conference Proceedings)*, volume 26, pages 71–78, 1992.
22. S. B. Kang, J. A. Webb, C. L. Zitnick, and T. Kanade. An active multibaseline stereo system with active illumination and real–time image acquisition. In *Proc. International Conference on Computer Vision*, 1995.
23. L. Kobbelt and M. Botsch. An interactive approach to point cloud triangulation. In *Computer Graphics Forum (Proc. Eurographics 2000)*, to appear.
24. E. Kruppa. Zur Ermittlung eines Objecktes aus zwei Perspektiven mit innerer Orientierung. *Sitz.–Ber. Akad. Wiss., Wien, Math. Naturw., Kl. Abt. IIa*, 122:1939–1948, 1913.
25. W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. In *Computer Graphics (SIGGRAPH 87 Conference Proceedings)*, volume 21, pages 163–170, 1987.
26. T. Masuda and N. Yokoya. A robust method for registration and segmentation of multiple range images. *Computer Vision and Image Understanding*, 61(3):295–307, 1995.

27. R. Mencl and H. Müller. Graph–based surface reconstruction using structures in scattered point sets. In *Proceedings of the Conference on Computer Graphics International*, pages 298–311. IEEE Computer Society, 1998.

28. R. Mencl and H. Müller. Interpolation and approximation of surfaces from three–dimensional scattered data points. In *Proceedings of Eurographics '98, State of the Art Reports*, 1998.

29. M. Okutomi and T. Kanade. A multiple–baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4):353–63, 1993.

30. F. P. Preparata and M. I. Shamos. *Computational geometry : an introduction*. Springer, 1985.

31. K. Pulli. *Surface Reconstruction and Display from Range and Color Data*. PhD thesis, University of Washington, 1997.

32. K. Pulli, T. Duchamp, H. Hoppe, J. McDonald, L. Shapiro, and W. Stuetzle. Robust meshes from multiple range maps. In *Proc. IEEE Int. Conf. on Recent Advances in 3–D Digital Imaging and Modeling*, 1997.

33. M. Rutishauser, M. Stricker, and M. Trobina. Merging range images of arbitrarily shaped objects. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 573–580, 1994.

34. M. Teichmann and M. Capps. Surface reconstruction with anisotropic density–scaled alpha shapes. In *IEEE Visualization '98 Conference Proceedings*, pages 67–72, 1998.

35. R. Tsai. A versatile camera calibration technique for high–accuracy 3–D machine vision metrology using off–the–shelf TV cameras and lenses. In L. Wolff, S. Shafer, and G. Healey, editors, *Radiometry – (Physics-Based Vision)*. Jones and Bartlett, 1992.

36. G. Turk and M. Levoy. Zippered polygon meshes from range images. In *SIGGRAPH 94 Conference Proceedings*, pages 311–318, 1994.

37. Z. Wang and A. Jepson. A new closed–form solution for absolute orientation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 129–134, 1994.

38. S. Weik. Registration of 3–d partial surface models using luminance and depth information. In *Proc. IEEE Int. Conf. on Recent Advances in 3–D Digital Imaging and Modeling*, pages 93–100, 1997.

39. M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide. Second edition. The Official Guide to Learning OpenGL, Version 1.1*. Addison–Wesley, 1996.

40. Z. Zhang. Iterative point matching for registration of free–form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–152, 1994.

3. Discrete differential geometry

3.1. Discrete curvature

While differential geometry analyses surfaces that are sufficiently often differentiable – which does not apply to meshes directly – discrete differential geometry is based on the fact that meshes can be interpreted as approximations of such smooth surfaces. In the following we will start with a short survey on differential geometry where we shortly explain the most important geometric intrinsics, i.e. surface properties that purely depend on the geometry and not on its specific parameterization. After that survey we will present some popular discretization methods for various geometric invariants.

3.1.1. Geometric intrinsics

Let S be a sufficiently smooth surface and $q \in S$ be a point on that surface.

Surface normal vector
The *normal vector* $\vec{n}$ at q is a unit vector that is perpendicular to the surface.

Normal curvature
The *normal curvature* $\kappa(T)$ is the curvature of the planar curve at q that results if one intersects the surface S with a plane that contains the surface normal direction $\vec{n}$. Here T is a unit vector inside the tangent plane of q, i.e. $< \vec{n}, T >= 0$, that specifies the direction of the normal cut.

The tensor of curvature
The *tensor of curvature* assigns to each q the function that measures the normal curvature $\kappa(T)$ in the direction determined by T. As function of the tangent vector T the normal curvature can be formulated as

$$\kappa(T) = \begin{pmatrix} t_x \\ t_y \end{pmatrix}^T \cdot \begin{bmatrix} \kappa_{11} & \kappa_{12} \\ \kappa_{21} & \kappa_{22} \end{bmatrix} \cdot \begin{pmatrix} t_x \\ t_y \end{pmatrix}, \qquad (1)$$

where t_x and t_y are the coordinates of T in an arbitrary orthonormal basis of the tangent space and $\kappa_{12} = \kappa_{21}$. The maximal normal curvature κ_1 and the minimal normal curvature κ_2 are called *principal curvatures*, the associated tangent vectors T_1 and T_2 are called *principal directions*.

It is always possible to choose an orthonormal basis of the tangent space built by two principal directions T_1 and T_2. Using such a basis, equation (1) simplifies to Eulers's theorem:

$$\kappa(\theta) = \kappa(T) = \kappa_1 \, cos^2(\theta) + \kappa_2 \, sin^2(\theta), \qquad (2)$$

where θ is the angle between T and T_1.

Gaussian curvature
The *Gaussian curvature* K is defined to be the determinant of the matrix that defines the quadratic form (1). If we change the orientation of the surface normal vector the Gaussian curvature does not change. If $K > 0$ both principal curvatures have the same sign, for $K < 0$ they have different signs. K can be expressed in terms of the principal curvatures:

$$K = \kappa_1 \kappa_2. \qquad (3)$$

Mean curvature
The *mean curvature* H is defined to be half the trace of the matrix in (1). It can be expressed in terms of the principal curvatures:

$$H = \frac{\kappa_1 + \kappa_2}{2}. \qquad (4)$$

If we change the orientation of the surface normal vector the mean curvature changes its sign.

The mean curvature can be interpreted as the average of the normal curvatures since it satisfies

$$H = \frac{1}{\pi} \int_0^{\pi} \kappa(\theta) \, d\theta.$$

3.1.2. Discretization techniques

Surface normal vector
A popular way to discretize the normal vector $\vec{n}$ is to use an average of the unit normals of the triangular faces that surround the vertex. Various averages occur in the literature, e.g. arithmetic, area weighted or angle weighted average[8].

Normal curvature
Given a normal vector $\vec{n}$ at a vertex q, we can discretize the normal curvature in the direction given by a unit tangent vector T_j that results if we project a vertex q_j that is adjacent to q into the tangent plane defined by $\vec{n}$. A frequently used discretization for such a normal curvature is given by the formula

$$\kappa(T_j) = 2 \frac{< q_j - q, \vec{n} >}{< q_j - q, q_j - q >}. \qquad (5)$$

This equation results if one discretizes the mathematical formula for the continuous case[11], but there is also a geometric explanation. This equation can be interpreted as interpolating the vertices q and q_j with a circle whose center lies on the line defined by q and $\vec{n}$ and to use the inverse of the resulting radius as normal curvature[9].

Local polynomials
A popular method to discretize geometric intrinsics is based on the idea to interpolate or approximate a local submesh by using polynomials. In practice the most common type are quadratic polynomials

$$f(x) = a_1 x^2 + a_2 y^2 + a_3 xy + a_4 x + a_5 y + a_6.$$

To be able to use that approach, one requires a planar parameter domain and has to assign planar coordinates to every vertex in a local neighborhood of q. Perhaps the most well–known approach is to first estimate a surface normal vector at a vertex and then project the vertices into that plane. However, this requires that the estimated normal is chosen accurately. The surface triangulation induces an ordering on the adjacent neighbors and this order should be preserved in the parameterization. If the normal plane is badly chosen, this is not guaranteed. A solution is to choose the exponential map[12] of the 1-neighborhood around q. This operator maps the neighborhood onto the plane while preserving the distances of the vertex to its neighbors and the ratio of the adjacent angles. The drawback here is that this approach is more expensive to calculate because it requires the usage of trigonometric functions.

Quadratic interpolation requires that the vertex has valence 5. Therefore, to be able to use all vertex information for arbitrary vertex valences one uses least square approximation. Here the most efficient method to solve the problem is to use the normal equations approach[5], since this mainly involves to calculate the inverse of a symmetric matrix of low dimension.

Once the polynomial is determined, the geometric invariants of the polynomial can be used as discretization values. For the discrete curvatures this only requires to determine the first and second fundamental forms of the polynomial.

However, problems occur if the parameter values that are assigned to the vertices lie on a curve of degree 2. In that case the least squares approximation fails and a special case treatment - e.g. reduction of the number of basis functions - is necessary.

Taubin's approach
Taubin[11] proposed an algorithm to derive the tensor of curvature. The principal curvature and principal directions are obtained by computing in closed form the eigenvalues and eigenvectors of certain 3×3 symmetric matrices defined by integral formulas, and closely related to the matrix representation of the tensor of curvature. As input to his algorithm he requires discretized normal vectors and discretized normal curvatures (equation (5)).

Moreton and Séquin's approach
Another algorithm to estimate the tensor of curvature at q is derived by Moreton and Séquin[9]. The idea behind this approach is to use the fact that the normal curvature distribution can't be arbitrary, but is determined by Euler's theorem (2). The 2×2 matrix occurring in equation (1) can be expressed as

$$K = \begin{bmatrix} e_x & e_y \\ -e_y & e_x \end{bmatrix} \cdot \begin{bmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{bmatrix} \cdot \begin{bmatrix} e_x & e_y \\ -e_y & e_x \end{bmatrix}^{-1},$$

where e_x and e_y are the coordinates of the principal direction T_1 in the chosen orthonormal basis. Moreton and Séquin's idea now is to estimate a discrete normal $\vec{n}$ and use the normal curvatures given by equation (5) to create a linear system. Solving this system using a least squares algorithm one can find estimates for the unknown principal curvatures and principal directions. A special case treatment here is only necessary, if the projection of the adjacent vertices to the tangent plane are intersection points of two lines passing the vertex q, so only vertices of valence 3 or 4 may need a special case treatment[10].

Estimating the Gaussian curvature
The advantages of the last three methods is that once the principal curvatures are discretized, one can also derive other important invariants from that information, as the Gaussian curvature or the mean curvature using eq. (3) or (4). If only the Gaussian curvature is needed, one can use the spherical image method or the angle deficit method. The idea of these approaches is to discretize a theorem for defining the Gaussian curvature on a smooth surface derived from a theorem by Rodrigues (see e.g.[3]).

Using the angle deficit method one can discretize the Gaussian curvature at q with the well known formula

$$K = \frac{2\pi - \sum_j \theta_j}{\frac{1}{3} \sum_j A_j},$$

where θ_j are the inner angles adjacent to q and A_j the corresponding triangle areas.

3.2. Quality control for meshed surfaces

This section gives a brief overview over techniques for rating the quality of triangular meshes. Quality control may be applied on the raw data immediately after mesh generation or on preprocessed data (cf. Section 2).

The techniques presented here are used to visualize surface quality in order to detect surface defects. They can be implemented in a straightforward way while still being effective and can be used interactively. The methods are adapted from smooth free-form surfaces (e.g. NURBS), and we may take advantage of the previously introduced concepts of discrete differential geometry (cf. Section 3).

3.2.1. What is the *quality of a triangle mesh*?

The answer to this question depends on what the mesh is used for. Different applications may require different quality criteria. We concentrate on the following:

Smoothness: Take a brief look on the situation when using NURBS patches: They have inherent C^n (e.g. C^2 for cubic splines) smoothness by construction and one is usually interested in smooth connections across patch boundaries (e.g. C^1).
The situation for triangle meshes is similar. Naturally, one would not expect something like "patch boundaries" inside a mesh. But such boundaries may emerge e.g. when

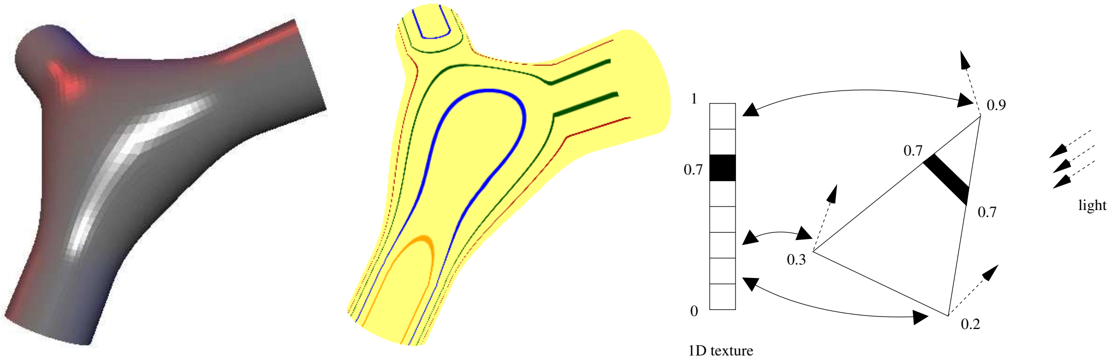


Figure 6: *Isophotes. The middle part of the surface was blended between the three tubes, the boundary conditions are C^1. The discontinuities of the curvature at the joints are hard to detect from the flat shaded image (left), but clearly visualized by isophotes (middle) since C^1 blends cause C^0 isophotes. The right image sketches the rendering of isophotes with 1D-textures: The illumination values are calculated for the vertices of a triangle from vertex normals and the light direction. These values are used as texture coordinates. The texel denoting the iso-value is colored black, so iso-lines are interpolated within the triangle.*

filling holes by trivially triangulating the hole and subsequent fairing.

Fairness: A smoothness criterion is not sufficient for generating high quality, highly aesthetic surfaces. A so called *fairness criterion* must be added. Usually this is defined as low variation of curvature in contrast to continuity of curvature (smoothness). So, we will use the results of the previous section for rating fairness. (see also Section 5.2)

Shape of triangles: Some applications require "well shaped" triangles (e.g. simulations using *Finite Element Methods* (FEM) *Computational Fluid Dynamics* (CFD)) while other applications (e.g. rendering) may neglect this property. So we need to check constraints on shape parameters such as angles and area. (cf. Section 5.1.7)

3.2.2. Visualizing smoothness

Our aim is interactive surface visualization; hence we try to get maximum advantage of graphics hardware. Therefore a given surface is tessellated to a set of triangles for rendering (in contrast to non-interactive rendering techniques like ray-tracing). As we can think of the mesh as an accurate tessellation of e.g. a set of NURBS patches we will use the same techniques for quality control that are used for smooth surfaces (cf. eg. [6]).

3.2.2.1. Specular shading The simplest visualization technique is to use standard lighting and shading (Phong illumination model, flat- or Gouraud shading) as provided by the graphics subsystem [4, 13].

The local illumination of a vertex depends on the position of the light sources, on surface normals and on the view point/direction if a specular lighting term is used.

This approach to surface interrogation is the most straightforward one, but it is difficult to find minor perturbations of a surface (cf. Fig. 6, left).

3.2.2.2. Isophotes Isophotes are lines of constant illumination on a surface. Here, one assumes that there is only *diffuse* or *Lambertian* reflection. As a consequence, isophotes are independent of the view point. For this application one single, infinitely distant point light source is assumed. So the illumination I_P of a surface point P is given by

$$I_P = \max\{\langle N_P \,|\, L \rangle, 0\},$$

where N_P is the surface normal at P and L is the direction of light (cf. [4]). Both vectors are normalized, so the value of I_P is in the interval $[0,1]$. Now some values $I_{c,j} \in [0,1] = \text{const}$ (e.g. $I_{c,j} = \frac{j}{n}$, $j = 0,\ldots,n$) are chosen and the isophotes/iso-curves $I = I_{c,j}$ are rendered.

The resulting image makes it easier to detect irregularities on the surface compared to standard shading. The user can visually trace the lines, rate their smoothness and transfer these observations to the surface: If the surface is C^k continuous then the isophotes are C^{k-1} continuous (cf. Fig. 6).

There are two approaches to rendering iso-curves such as isophotes: The first approach is to explicitly extract the curves or curve segments and then display them as lines. Here, in principle the same algorithms as for extracting iso-surfaces can be applied (*Marching Cubes*[7], cf. Section 2.2). Fortunately the situation for extracting a curve on a surface is easier (*"marching triangles"*).

The second approach takes advantage of the graphics hardware and allows direct rendering of isophotes from illumination values in the vertices of a triangle mesh:

A *1-dimensional texture* is initialized with a default color C. Illumination values I_P are now treated as texture coordinates, and for the isophote values $I_{c,j}$ the corresponding texels are set to color $C_j \neq C$. The illumination values $I_{c,j}$ are evaluated at every vertex and used as texture coordinates. With this setup the graphics subsystem will linearly interpolate the 1D texture within the triangles resulting in a rendered image of the isophotes (colors C_j) that are drawn onto the surface (color C) [13] (cf. Fig. 6).

The 1D textures approach benefits more from the graphics hardware in contrast to explicitly calculating line segments. A drawback is that the width of the curves varies due to texture interpolation.

3.2.2.3. Reflection lines In contrast to isophotes a specular surface is assumed for reflection lines. As a consequence reflection lines change when the point of view is modified resp. when the object is rotated or translated. The light source consists of a set of "light-lines" that are placed in 3D space. Normally, the light-lines are parallel lines (cf. Fig. 7).

Traditionally, reflection lines have been used in the process of designing cars. An arrangement of parallel fluorescent tubes is put in front of the car model to survey the surface and its reflection properties.

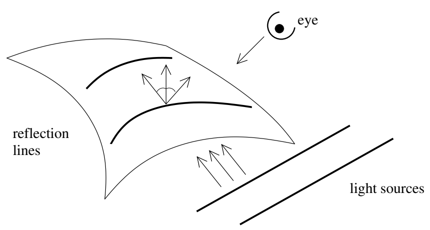


Figure 7: *Reflection lines. The light source consists of two parallel lines that are reflected by the surface. The reflection property requires that angles of incidence (light,normal) are equal to angles of emission (viewing direction,normal).*

Under the assumption that the light source is infinitely far away from the object, *environment mapping*[13] can be used to display reflection lines in real-time. A texture for environment mapping is generated once by ray-tracing the light sources over a sphere. The graphics subsystem will then automatically generate appropriate texture coordinates for every vertex depending on its relative position and normal.

Reflection lines are an effective and intuitive tool for surface interrogation. If the surface is C^k continuous then the reflection lines are C^{k-1} continuous. Just like isophotes, they can be efficiently rendered by taking advantage of graphics hardware and they are also sensitive to small surface perturbations. In addition, the concept that a real-world process is simulated makes their application very intuitive even for unexperienced users. Fig. 8 shows reflection lines for a C^0, C^1 and a C^2 surface. Reflection lines are also used to show surface quality in Section 5.2.

3.2.3. Smoothness vs. fairness

The first two quality criteria listed above are *smoothness* and *fairness*. *Smoothness* denotes the mathematical definition of continuous differentiability (C^n). While smoothness is necessary in order to guarantee high quality surfaces it is not sufficient.

A surface may be smooth in a mathematical sense but still looking awkward from an aesthetical point of view. This is where *fairness* comes in: fairness is an aesthetic measure of "well-shapedness", it is therefore more difficult to define in technical terms than smoothness (distribution vs. variation of curvature) [1]:

An important rule is the so called *principle of simplest shape* that is derived from fine arts. A surface is said to be well-shaped if it is simple in design and free of unessential features. So a *fair* surface meets the mathematically defined design goals (e.g. interpolation, continuity) while being nice looking in this sense. There are several approaches to formulate the latter term in a precise, mathematical way.

The most common measures for fairness are motivated by physical models like the strain energy of a thin plate

$$\int \kappa_1^2 + \kappa_2^2 dA$$

or differential geometry like the variation of curvature

$$\int \left(\frac{\partial \kappa_1}{\partial \vec{e}_1}\right)^2 + \left(\frac{\partial \kappa_2}{\partial \vec{e}_2}\right)^2 dA$$

with principal curvatures κ_i and principal directions $\vec{e}_i$, $i = \{1,2\}$.

In general, some surface energy is defined that punishes "bad-shapedness", and curvature is used to express these terms as it is independent from the special parameterization of a surface. A fair surface is then designed by minimizing these energies (cf. Section 5.2). Our current goal is not to improve but to check surface quality, so we need to visualize these energies.

Note that there are also different characterizations of fairness like "aesthetical run of isophotes/reflection lines".

3.2.4. Visualizing curvature and fairness

If fairness is expressed in terms of curvature we have to visualize curvature to evaluate fairness. We obtain curvature values in every vertex of a triangle mesh by applying the techniques presented in the previous section (cf. Section 3.1).

3.2.4.1. Curvature values The technique suggested in the following sections will visualize arbitrary "curvature values". Any useful scalar value d that is a measure for discrete curvature can be used (cf. textbooks on differential geometry like [3] for detailed explanations). Here are some examples:

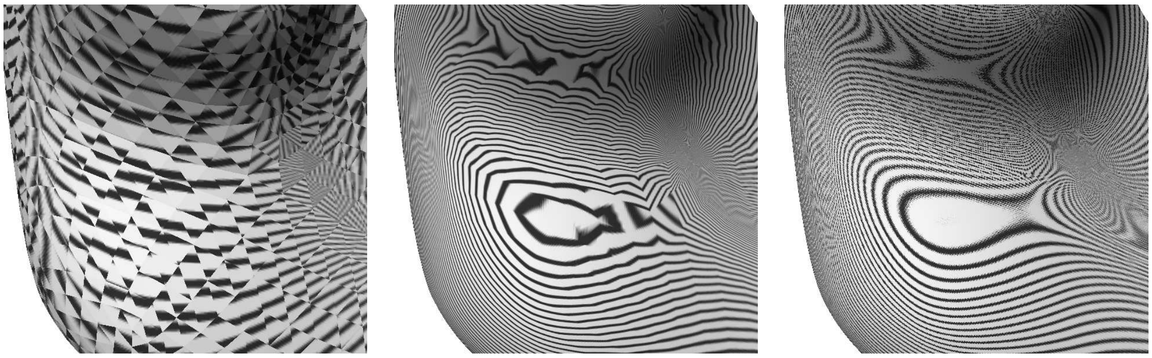


Figure 8: *Reflection lines on C^0, C^1 and C^2 surfaces. One clearly sees that the differentiability of the reflection lines is one order lower, i.e. C^{-1}, C^0 and C^1 respectively.*

- the Gaussian curvature $K = \kappa_1\kappa_2$ that indicates the local shape of the surface (elliptic for $K > 0$, hyperbolic for $K < 0$ and parabolic for $K = 0 \wedge H \neq 0$ resp. flat for $K = 0 \wedge H = 0$). A local change of the sign of K may denote a (even very small) perturbation of the surface.
- the mean curvature $H = \kappa_1 + \kappa_2$
- the maximum curvature $\kappa_{\max} = \max\{|\kappa_1|, |\kappa_2|\}$
- the total curvature $\kappa_1^2 + \kappa_2^2$ that is used in the previously mentioned thin plate energy.

3.2.4.2. Color coding scalar values We visualize scalar values directly on the surface, i.e. every vertex is assigned a color value. The graphics subsystem will then linearly interpolate colors within the triangles. All lighting calculations must be turned off for this purpose. There are lots of ways for color coding, one of them is the following:

Assume a scalar value d_i is given for every vertex V_i, e.g. d_i may denote any type of curvature. Now let $d_{\max} := \max\{d_i\}$ and $d_{\min} := \min\{d_i\}$. Data values are scaled by the following function **scale** : $[d_{\min}, d_{\max}] \to [-1, 1]$ with

$$\mathbf{scale} : d \mapsto \begin{cases} -d/d_{\min} & : d < 0 \\ d/d_{\max} & : d \geq 0 \end{cases}$$

Positive and negative values are scaled separately such that the zero level is preserved. Notice that the value 0 is usually of special interest. So $d_{\min} \leq 0 \leq d_{\max}$ is assumed. If not so, the origin ("green line", see below) should be shifted appropriately.

The red and blue color components are used to indicate positive resp. negative data values. All vertices and all displayed pixels are to have equal intensity (r+g+b=1). So the green component is used to "fill up" intensity. Assume color components range from 0 to $c_{\max}$, e.g. $c_{\max} = 255$. The function **rgb** : $[-1, 1] \to [0, c_{\max}]^3$ assigns to each value a RGB triple with intensity $c_{\max}$.

$$\mathbf{rgb} : d \mapsto \begin{cases} (0, (1+d)c_{\max}, -dc_{\max}) & : d < 0 \\ (dc_{\max}, (1-d)c_{\max}, 0) & : d \geq 0 \end{cases}$$

Fig. 9 shows the RGB mapping on the right side. Zero values are displayed bright green, $d_{\min}$ and $d_{\max}$ result in blue and red respectively.

Data values $d \in [d_{\min}, d_{\max}]$ can now be mapped to RGB values by $\mathbf{rgb}(\mathbf{scale}(d))$. Many applications need enhanced contrast in the vicinity of zero and less near $d_{\min}$ and $d_{\max}$. Therefore a new parameter $\gamma \in (0, 1]$ is introduced that adjusts the "contrast" of the visualized data. Then value d is mapped to a RGB triple by

$$\mathbf{rgb}(\mathbf{scale}(d)^\gamma)$$

For $\gamma = 1$ we obtain the original mapping. With decreasing γ, the resolution increases for values near 0, i.e. a greater range in the color table is used for those values. Fig. 9 (left) illustrates the color coding for $\gamma = 1$ and $\gamma < 1$.

3.2.4.3. Isocurvature lines Isocurvature lines are lines of constant curvature on a surface. They can be displayed similarly to isophotes. Instead of illumination values curvature values are used. If the surface is C^k continuous then the isocurvature lines are C^{k-2} continuous, so isocurvature lines are also very sensitive to discontinuities.

A problem when rendering isocurvature lines with 1D-textures may be the wide range of curvature values that may not map appropriately to the $[0, 1]$ interval of texture coordinates resp. the actual texels. One solution is to clamp the curvature values to a suited interval, the other solution is to explicitly extract the curves and draw them as lines.

3.2.4.4. Lines of curvature Besides the scalar principal curvatures the principal directions also carry information on the local surface properties. They define a discrete direction field on the surface with one tangential direction per vertex. By linearly interpolating directions over triangles using barycentric coordinates a continuous field can be defined.

Lines of curvature can then be traced on this direction field. The tracing algorithm does Euler integration steps in-

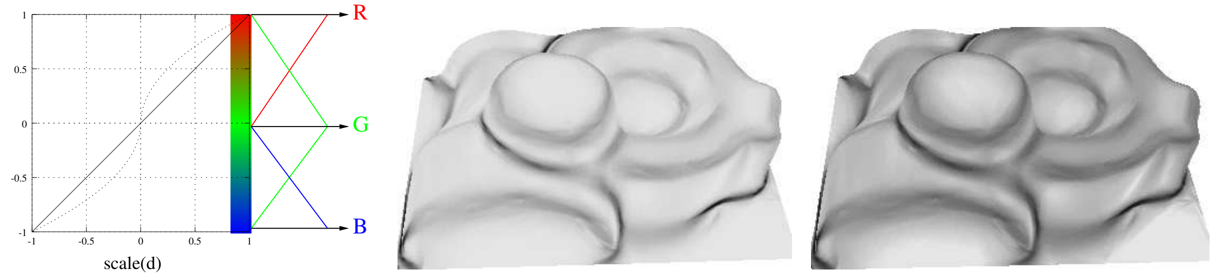


Figure 9: *Color coding. Left: d is mapped to* $[-1,1]$ *by* ***scale****, resolution near 0 may be enhanced by using* $\gamma < 1$*, the transformed value is then coded as* (r, g, b)*. Middle: maximum curvature* $\kappa_{\max} = \max\{|\kappa_1|, |\kappa_2|\}$ *with* $\gamma = 1$*. Right with enhanced contrast* $\gamma = \frac{1}{2}$.

Figure 10: *Lines of curvature. The (signed) maximum curvature is color coded and lines of curvature are superimposed.*

side a triangle until an edge is crossed. Then a neighboring triangle is entered. Fig. 10 shows lines of curvature.

The visualized curvature flow may give a very good and intuitive impression of the surface. Alternatively texture based techniques like line integral convolution (LIC) [2] can also be used on triangle meshes. Tracing and constructing a huge number of lines of curvature is rather expensive compared to the other techniques.

3.2.5. The shape of triangles

Some applications need "well-shaped", round triangles in order to prevent them from running into numerical problems. This includes numerical simulations based on FEM or CFD. For this purpose, "round" triangles are needed, i.e. the ratio of the radius of the circumcircle to the shortest edge should be as small as possible (cf. Fig. 3.2.5).

The most common way to inspect the quality of triangles is to view a wireframe or hidden-line rendered image. This may not be an option for very complex meshes. A straightforward solution is a color coding criterion based on triangle shapes. This helps to identify even single "badly shaped" triangles.

3.2.6. Summary

The most important issue about quality control is probably the fact that techniques that are well known from the interrogation of smooth surfaces can be adapted and used for triangle meshes in a straightforward way. Discrete curvature analysis is the key to achieve this result. In addition to smoothness and fairness there are criteria on triangle shape that may be important for specific applications.

References

1. H. G. Burchard, J. A. Ayers, W. H. Frey, and N. S. Sapidis. Approximation with aesthetic constraints. In *Designing Fair Curves and Surfaces*, pages 3–28, 1994.

2. B. Carbal and L. C. Leedom. Imaging vector fields using line integral convolution. In *SIGGRAPH 93 Conference Proceedings*, pages 263–274, 1993.

3. M. P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice–Hall, Inc Englewood Cliffs, New Jersey, 1993.

4. J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice, second edition in C*. Addison–Wesley, 1996.

5. G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1989.

6. H. Hagen, S. Hahmann, Th. Schreiber, Y. Nakayima, B. Wördenweber, and P. Hollemann-Grundstedt. Surface interrogation algorithms. *IEEE Computer Graphics and Applications*, 12(5):53–60, 1992.

7. W. E. Lorensen and H. E. Cline. Marching cubes: a

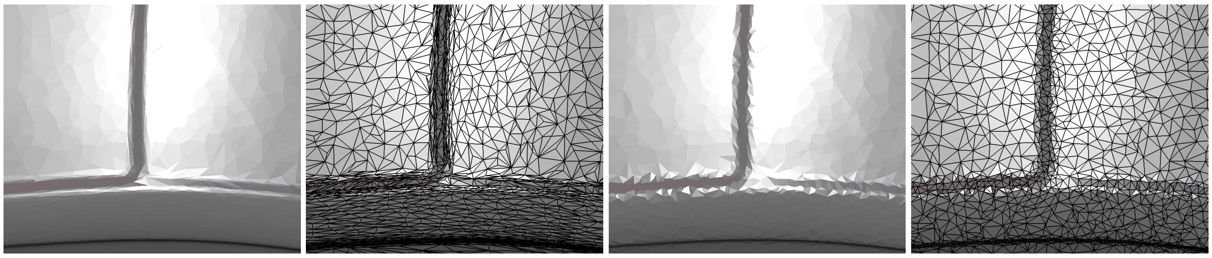


Figure 11: *A triangle mesh optimized for smooth appearance, leading to flat triangles (left), and for triangle shape, leading to surface artifacts (right).*

high resolution 3D surface construction algorithm. In *Computer Graphics (SIGGRAPH 87 Conference Proceedings)*, volume 21, pages 163–170, 1987.

8. D. S. Meek and D. J. Walton. On surface normal and gaussian curvature approximations given data sampled from a smooth surface. *Computer Aided Geometric Design*, 17:521–543, 2000.

9. H. P. Moreton and C. H. Séquin. Functional optimization for fair surface design. In *Computer Graphics (SIGGRAPH 92 Conference Proceedings)*, volume 26, pages 167–176, 1992.

10. R. Schneider and L. Kobbelt. Geometric fairing of irregular meshes for free–form surface design. submitted.

11. G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral. In *Proc. International Conference on Computer Vision*, pages 902–907, 1995.

12. W. Welch and A. Witkin. Free–form shape design using triangulated surfaces. In *SIGGRAPH 94 Conference Proceedings)*, pages 247–256, 1994.

13. M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide. Second edition. The Official Guide to Learning OpenGL, Version 1.1*. Addison–Wesley, 1996.

4. Coarse-to-fine hierarchies

Coarse–to–fine hierarchies are build up by successively refining a coarse base mesh (i. e. by inserting new vertices and triangles). The resulting degrees of freedom can be used in two ways: Subdivision schemes position the new vertices such that the resulting meshes become smooth—the geometric information inherent to the base mesh is therefore not changed. Remeshing methods on the other hand position the vertices such that more and more geometric detail becomes visible by sampling points from the original surface—the resulting meshes therefore need not be smooth. The following two sections describe these approaches in more detail.

4.1. Stationary subdivision

4.1.1. Introduction

Subdivision schemes have become increasingly popular in recent years because they provide a uniform and efficient way to describe smooth curves and surfaces. Their beauty lies in their elegant mathematical formulation and their simple implementation: Given an arbitrary control polygon perform the following subdivision step ad infinitum (see Figure 12):

1. *Splitting step:* Insert a new vertex at the midpoint of each edge.
2. *Averaging step:* Relocate each vertex according to given refinement rules.

If the refinement rules are chosen carefully the resulting control polygons will converge to a smooth limit curve. In practice the algorithm is stopped after a sufficient number of subdivision steps and the resulting control polygon is rendered as an approximation of the curve.

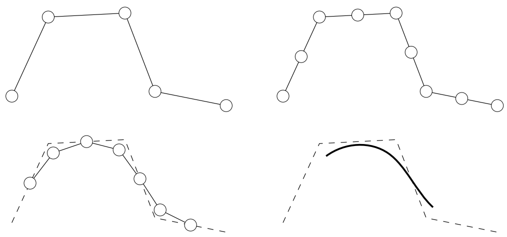


Figure 12: *Subdivision step: original polygon (upper left), after splitting step (upper right), after averaging step (lower left) and limit curve (lower right).*

Even though subdivision schemes have many applications in the univariate case their real strengths are only revealed when looking at the bivariate case: they are easily able to handle surfaces of *arbitrary* topology while automatically maintaining continuity properties.

In the following we give a brief historical overview and cite the necessary references. Subdivision methods (for curves) were introduced and mathematically analyzed for the first time by G. de Rham in 1947. However, only their re–invention in 1974 by G. M. Chaikin[5] made them available for the computer graphics community. Chaikin used them to derive a simple algorithm for the high–speed generation of curves. In 1978 the concept of subdivision was carried over from curves to surfaces: Catmull and Clark described a generalization of bicubic tensor product B–splines[4] and Doo and Sabin introduced a generalization of biquadratic tensor product B–splines[9]. In the following decades many new schemes were proposed: the Butterfly scheme[12], the Loop scheme[28] and variational[20] schemes (see also Section 5.2) to name only a few. However, the subdivision rules of the early schemes were only “ad hoc” generalizations of known rules for regular cases (knot insertion) and lacked a precise mathematical analysis with respect to convergence behavior and differentiability properties. A first and important approach was already given by Doo and Sabin: they performed a spectral analysis of the so–called *subdivision matrix* to prove the convergence of their scheme in extraordinary vertices. This approach was further enhanced by Storry and Ball[2]. A second way to analyze subdivision schemes are the so called *generating functions*[10]: this formalism allows the subdivision step to be expressed as a simple multiplication of two formal power series. However, it was not until 1995 when U. Reif introduced the concept of the *characteristic map* of a subdivision scheme[30] that one was finally able to rigorously prove the continuity properties of subdivision schemes[33].

Nowadays the research is focusing on applications of subdivision schemes: Methods to interpolate points and curves were developed[16, 27], physical simulations based on subdivision methods were examined[6]. Subdivision techniques are used for animations in computer generated movies[7] as well as in raytracing applications[22]. New techniques to efficiently handle and render subdivision surfaces were developed and even implemented in hardware[29]. The modeling power of subdivision schemes was greatly enhanced by introducing schemes that are able to model corners and creases[17, 3]. At present almost everything one can do with traditional NURBS–based systems can also be achieved by subdivision techniques[32].

Before starting we need some preliminaries: In the following we are dealing mostly with triangular and quadrangular meshes, i. e. the faces are triangles and quadrangles, respectively. The number of edges emanating from a vertex is called the *valence* of the vertex. A vertex of a quadrangular mesh is said to be *regular*, if it is an inner vertex of valence 4 or a boundary vertex of valence 3, otherwise it is called *extraordinary*. Likewise a vertex of a triangular mesh is called *regular*, if it is an inner vertex of valence 6 or a boundary vertex of valence 4, and *extraordinary* otherwise.

4.1.2. Catmull–Clark subdivision

In order to get a feeling for subdivision schemes we will describe one of the first of them in more detail: the Catmull–

Clark scheme, which was introduced in 1978. It is a generalization of ordinary tensor product B–spline subdivision and widely used because its quadrangular structure fits well into existing CAD systems.

Let $s(u,v) = \sum_i \sum_j c_{ij} N_i^3(u) N_j^3(v)$ be a bicubic tensor product B–spline: the c_{ij} are the control points arranged in a regular quadrilateral grid (see Figure 13) and the $N_i^3(\cdot)$ are the uniform cubic B–splines over the knot–vector $\mathbb{Z}$. The surface s is build of quadrilateral polynomial patches each of them being determined by a regular 4×4 submesh of the original mesh (compact support of the B–splines!).

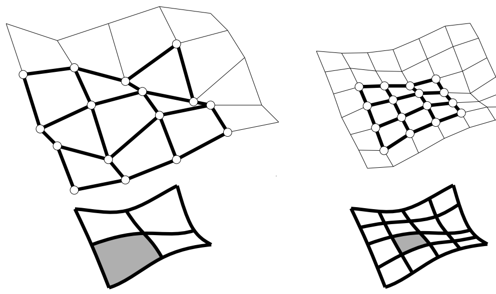


Figure 13: *B–Spline subdivision: Each patch of a tensor product B–Spline surface is fully determined by a 4×4 submesh of the original mesh (left). Knot insertion yields a refined mesh (right).*

By inserting knots we see that s can be rewritten as a tensor product B–spline over the refined knot vector $\frac{1}{2}\mathbb{Z}$, i. e. $s(u,v) = \sum_i \sum_j d_{ij} N_i^3(2u) N_j^3(2v)$. The control points d_{ij} constitute a finer quadrilateral mesh (see Figure 13) and are easily computed by the following masks:

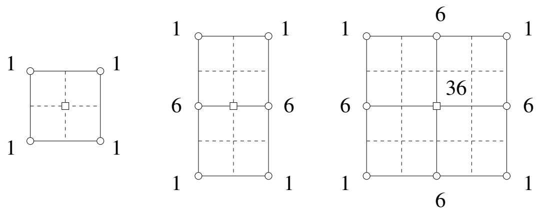


Figure 14: *Masks for knot–insertion of bicubic tensor product B–splines.*

In this and the following figures already existing edges are shown as solid lines while new edges are dashed. To compute the new position of the vertex marked by a square one has to take the weighted average of the depicted vertices. By repeatedly inserting knots like this the resulting control meshes converge to the limit surface s. Now suppose we have a quadrilateral mesh with extraordinary vertices; still we can interpret every 4×4 submesh as control net of a polynomial patch—but this will leave a hole in the surface (see Figure 15).

In order to carry over the concept of subdivision to extraordinary vertices Catmull and Clark extended the masks for the regular case by a set of new masks—one mask for each valence (see Figure 16).

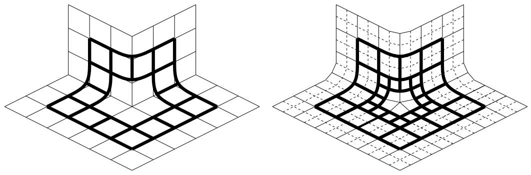


Figure 15: *Catmull–Clark subdivision: Interpreting each regular 4×4 submesh as the control net of a polynomial patch still leaves holes (left), subdividing the control mesh results in a larger regular region and thus gradually fills these holes with rings of patches (right).*

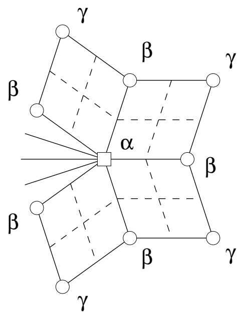


Figure 16: *Catmull–Clark subdivision: mask for extraordinary vertices*[14]*. The coefficients depend on the valence n of the center vertex:* $\beta = \frac{3}{2n^2}$, $\gamma = \frac{1}{4n^2}$, $\alpha = 1 - n\beta - n\gamma$.

Now they could do subdivision as above and add a further ring of patches to the surface thus making the hole smaller (see Figure 15). The new masks were designed such that the resulting control meshes converge to a limit surface which is C^2 except for the extraordinary vertex where it is C^1. Note that the choice of the masks is not unique—there exist other variants which also generalize bicubic tensor product B–splines.

4.1.3. Analysis of subdivision schemes

There are two major approaches to mathematically analyze the properties of subdivision schemes (convergence behavior, continuity/differentiability of the limit function, reproduction properties): generating functions and spectral analysis of the subdivision matrix.

Generating functions

Generating functions are used to analyze univariate subdivision schemes and the regular parts of bivariate subdivision schemes. To avoid messy multi–indices we will restrict ourselves to the univariate case. We start with a definition:

Let $P = (p_j)_{j=-\infty,\ldots,\infty}$ be an arbitrary sequence, then we call the formal series $P(z) = \sum_j p_j z^j$ the *generating function* (or the *symbol*) of P.

As an example consider the 4–point scheme (see Figure 17):

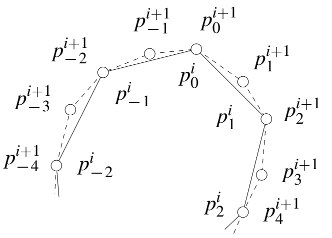


Figure 17: *The 4–point scheme*[11]*: A polygon $P^i = (p^i_j)$ (solid lines) is mapped to a refined polygon $P^{i+1} = (p^{i+1}_j)$ (dashed lines). Note that this is an interpolatory scheme: $p^{i+1}_{2j} = p^i_j$.*

A polygon $P^i = (p^i_j)$ is mapped to a refined polygon $P^{i+1} = (p^{i+1}_j)$ by applying the following two subdivision rules:

$$\begin{aligned} p^{i+1}_{2j} &= p^i_j, \\ p^{i+1}_{2j+1} &= \frac{1}{16}(-p^i_{j-1} + 9p^i_j + 9p^i_{j+1} - p^i_{j+2}). \end{aligned}$$

In general such a subdivision step can be compactly written in a single equation

$$p^{i+1}_j = \sum_{k=-\infty}^{\infty} \alpha_{2k-j} p^i_k,$$

where the α_j are coefficients depending on the subdivision rules. Note that the index $2k-j$ alternatingly selects the even indexed αs or the odd indexed αs. In our case we have

$$\alpha = (\alpha_j) = \frac{1}{16}[\ldots, 0, 0, -1, 0, 9, 16, 9, 0, -1, 0, 0, \ldots]$$

After some computation we see that the subdivision step can be expressed in the generating function formalism as a simple multiplication of the corresponding symbols:

$$P^{i+1}(z) = \alpha(z) P^i(z^2).$$

Note that $\alpha(z)$ is the symbol of the sequence α! The basic idea of proving convergence to a continuous limit function is to show that the polygons P^i form a *Cauchy sequence*, which follows if the distance $||P^{i+1} - P^i||_\infty$ of two successive polygons decreases exponentially in i. Some computation shows that this is the case, if the so–called *difference scheme* which is given by the symbol

$$\alpha'(z) = \frac{z}{1+z}\alpha(z)$$

exists (i. e. $\alpha(-1) = 0$) and is contractive. This is the case if

$$\max\left\{ \sum_j |\alpha'_{2j}|, \sum_j |\alpha'_{2j+1}| \right\} = q < 1.$$

So we have an easy criterion to check the convergence of a subdivision scheme to a continuous limit function. Likewise one can prove convergence to higher order continuous functions by examining higher order difference schemes.

Subdivision matrix formalism

The subdivision matrix formalism is used to describe the behavior of bivariate subdivision schemes near extraordinary vertices. The basic idea is to keep track of a vertex p_0 through different subdivision levels. To do this, it turns out that it is necessary not only to keep track of p_0 but also of the vertices $p_1, \ldots, p_n$ in a finite neighborhood of p_0—in the case of Loop subdivision this is the 1–ring of neighbors, i. e. all vertices adjacent to p_0 (see Figure 18). In general, the size of the neighborhood is determined by the support of the subdivision masks.

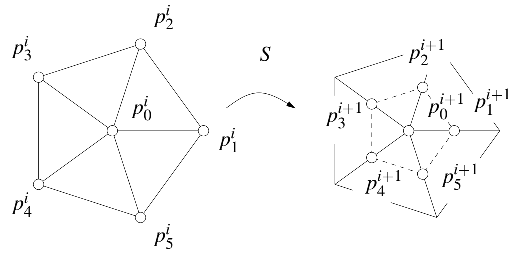


Figure 18: *Subdivision matrix formalism: The subdivision matrix S maps a neighborhood of a vertex on level i to the neighborhood of the same vertex on level $i+1$.*

Now let $p^i = [p^i_0, p^i_1, \ldots, p^i_n]$ be a column vector comprising the positions of $p_0, p_1, \ldots, p_n$ at subdivision level i. Then there is a $(n+1) \times (n+1)$ matrix S satisfying

$$p^{i+1} = S p^i.$$

This matrix is called the *subdivision matrix* of the scheme. Let $\lambda_0 \geq \lambda_1 \geq \ldots \geq \lambda_n$ be the eigenvalues of S and $x_0, x_1, \ldots, x_n$ the corresponding eigenvectors (i. e. $Sx_j = \lambda_j x_j$). Note that due to the affine invariance of subdivision schemes we have $\lambda_0 = 1$ and $x_0 = [1, \ldots, 1]^T$. Since the x_j form a basis we can expand p^0 to

$$p^0 = \sum x_j \omega_j$$

for some vector–valued coefficients ω_j. Subdividing the mesh m times means applying S^m to p^0:

$$p^m = S^m p^0 = \sum (\lambda_j)^m x_j \omega_j = \lambda_0^m x_0 \omega_0 + \lambda_1^m x_1 \omega_1 + \cdots.$$

Now suppose that $1 = \lambda_0 > \lambda_1$. Then it is easy to see that the limit position $\lim_{i\to\infty} p^i_0$ of p_0 is given by ω_0. Similar formulas can be derived for the limit tangents in ω_0. Thus the analysis of the subdivision matrix provides formulas (masks) for limit positions and tangents (if they exist).

However, it turns out that the above analysis is not enough to show that the limit surface in the neighborhood of an extraordinary vertex is smooth. This means that the derived formulas are only valid, if the convergence to a differentiable function has already been shown by some other method. For this one has to consider a larger neighborhood of p_0 and analyze the spectral properties of the corresponding subdivision matrix. Furthermore one needs to analyze the so called *characteristic map* of the subdivision scheme. It can be shown that if this map is regular and injective the subdivision scheme itself produces C^1–continuous surfaces in the limit[30, 33].

4.1.4. Technical terms

In this section we explain some of the common technical terms used in the subdivision literature.

According to the way they relate the topology of the original mesh to the topology of the subdivided mesh, subdivision schemes are classified (see Figure 19) as

- *primal* or *face–split* schemes
- *dual* or *vertex–split* schemes
- *other*, e. g. $\sqrt{3}$–scheme, honeycomb refinement and bisection[31]

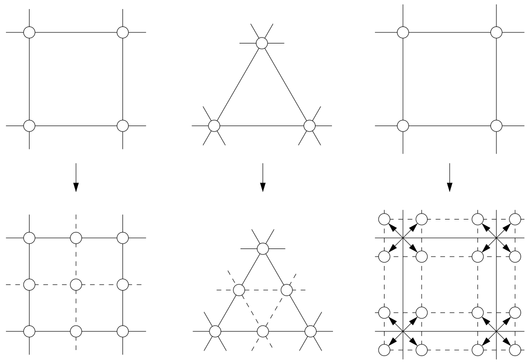


Figure 19: *Classification of subdivision schemes: face split schemes—quadrangular (left) and triangular (middle)—and vertex split schemes (right).*

Furthermore one distinguishes between *interpolatory* (old vertices are not relocated) and *approximating* schemes. A subdivision scheme is said to be *stationary*, if the subdivision rules do not depend on the overall structure of the mesh nor on the subdivision level (all subdivision schemes presented here are stationary), otherwise it is called *non–stationary*. Variational schemes [20] are an example of non–stationary schemes which are based on optimization techniques to generate smooth meshes (see also Section 5.2). A subdivision scheme has *compact support*, if only a finite number of coefficients in the subdivision masks is non–zero.

4.1.5. Common subdivision schemes

In this section we present some common subdivision schemes. The following table gives a brief overview of the basic properties (note that C^k really means C^k *almost everywhere*, i. e. except for the extraordinary vertices, where all schemes are C^1).

Doo–Sabin[9]	approx.	C^1	quadrilateral	dual
Catmull–Clark[4]	approx.	C^2	quadrilateral	primal
Kobbelt[19]	interp.	C^1	quadrilateral	primal
Butterfly[12]	interp.	C^1	triangular	primal
Loop[28]	approx.	C^2	triangular	primal
$\sqrt{3}$[21]	approx.	C^2	triangular	other

We will only give the basic refinement rules and discuss some of the properties. For further information (refinement rules for boundary edges, masks for limit positions, etc.) the reader is referred to the literature. Generally interpolatory schemes allow for a more intuitive control of the limit surface (vertices are interpolated, no shrinking effect) and for a more simple implementation (in–place) of many algorithms. However, the surface quality is usually not as good as that of approximating schemes.

Doo–Sabin scheme The Doo–Sabin scheme[9] (see Figure 20) generalizes quadratic tensor product B–splines. Its refinement rules are given in Figure 21. It is interpolatory in the sense that the barycenters of the faces of the original mesh are interpolated.

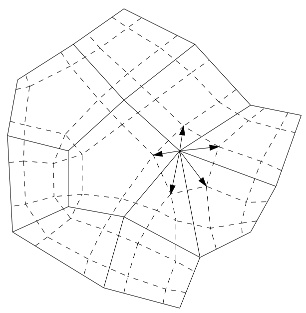


Figure 20: *The Doo–Sabin scheme is the most prominent example of a* dual *or* vertex–split *subdivision scheme. Note that the number of extraordinary faces (i. e. faces which are not quadrilateral) remains constant.*

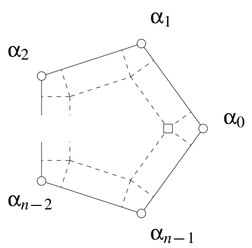


Figure 21: *Doo–Sabin scheme: There is only one mask parameterized by the number n of vertices adjacent to the face*[14]*:* $\alpha_0 = \frac{1}{4} + \frac{5}{4n}$, $\alpha_i = \frac{3+2\cos(2i\pi/n)}{4n}$ *for* $i = 1, \ldots, n-1$.

Kobbelt scheme The Kobbelt scheme[19] is an interpolatory scheme for quadrilateral meshes which emerges from generating the tensor product of the 4–point scheme.

(Modified) Butterfly scheme This scheme was originally proposed by Dyn, Gregory and Levin[12] and modified by Zorin, Schröder and Sweldens[34] to yield an everywhere C^1–continuous surface. The refinement rule for the unmodified scheme is depicted in Figure 22.

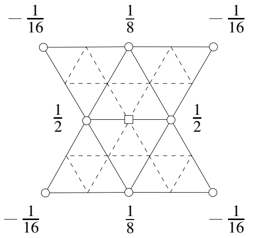


Figure 22: *Butterfly scheme: As this is an interpolatory scheme we only need a mask for the newly inserted vertices*[14].

Loop scheme The Loop scheme[28] generalizes quartic box-splines. Its refinement rules are given in Figure 23.

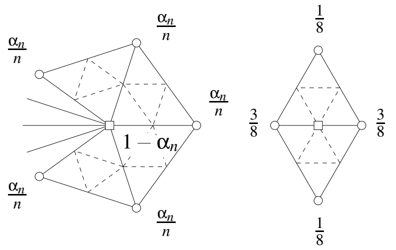


Figure 23: *Loop scheme: In case of a regular mesh this scheme produces quartic boxsplines. We have* $\alpha_n = \frac{1}{64}(40 - (3+2\cos(2\pi/n))^2)$ *where n is the valence of the center vertex*[28].

$\sqrt{3}$–scheme This scheme was only recently proposed by Kobbelt[21]. It produces a C^2 surface almost everywhere but is not based on polynomials. It is especially well suited for adaptive subdivision since one doesn't need to insert auxiliary triangles. In a first step every original triangle is split in three by inserting a new vertex at its barycenter. In the second step the original edges are flipped, yielding a triangle mesh rotated by 30 degrees (see Figure 24). The subdivision masks are shown in Figure 25.

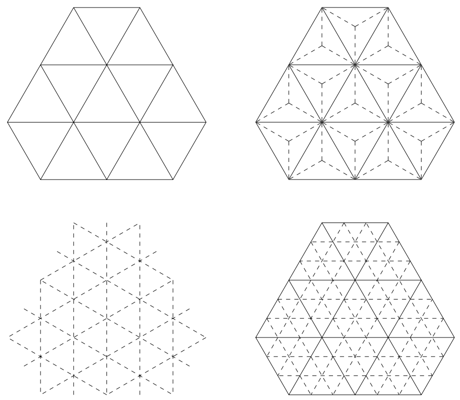


Figure 24: $\sqrt{3}$*–scheme : original mesh (upper left), after inserting barycenters (upper right) and after edge flipping (lower left). Note that two* $\sqrt{3}$*–subdivision steps result in a tri–section of the original triangle edges (lower right), hence the name of the scheme.*

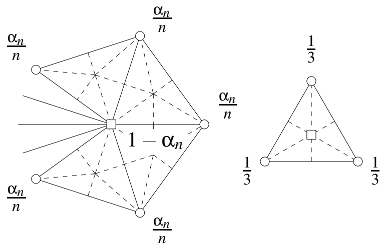


Figure 25: $\sqrt{3}$*–scheme : This scheme has the masks with the smallest possible support*[21]*. We have* $\alpha_n = \frac{4-2\cos(2\pi/n)}{9}$ *where n is the valence of the center vertex.*

4.2. Remeshing

Using the powerful means of *subdivision*, the preceding section illustrates how one can define a surface as the limit of a sequence of successively refined polyhedral meshes. In this section we do not deal with the geometric part of the subdivision that leads to mathematically smooth and visually appealing surfaces, but we focus on the special connectivity, the so called *subdivision–connectivity* that emerges, when iteratively applying a regular refinement operator to a coarse triangle mesh. A well–known refinement operator is the 1–to–4 split that recursively splits each triangular face in 4 subtriangles by introducing 3 new vertices on the edges. Since every submesh that corresponds to one base triangle has the structure of a regular grid and the whole hierarchy is based on an arbitrary coarse base mesh (cf. Fig. 26), the resulting meshes are said to be *semi-regular*.

The implicitly defined connectivity established on a coarse base mesh and the direct availability of multiresolution semantics gives rise to many techniques exploiting this convenient representation as the following enumeration shows.

Compression/Progressive Transmission Lounsberry et al. [8] perform a multiresolution analysis, i.e. they introduce a wavelet decomposition for meshes with subdivision–connectivity. By suppressing small wavelet coefficients, a compressed approximation within a given error–tolerance can be achieved. Moreover such a wavelet representation can easily be transmitted in a progressive fashion. (Send the base mesh first and refine it with successively arriving wavelet coefficients.)

Multiresolution editing For instance Zorin and co–workers[35] combine subdivision and smoothing techniques and present an interactive multiresolution mesh editing system, which is based on semi–regular meshes and

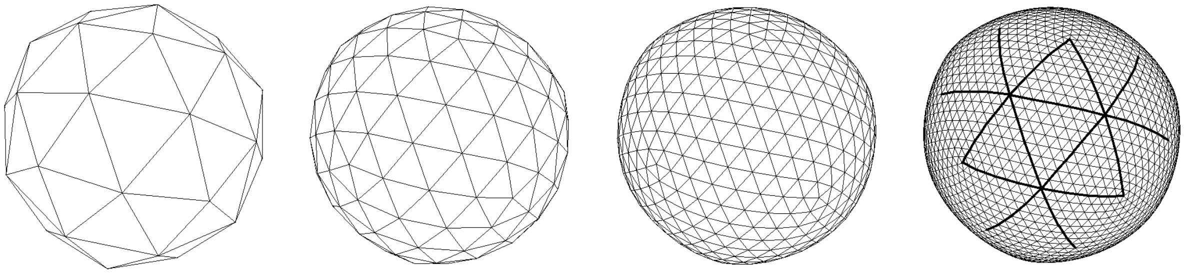


Figure 26: *Meshes with subdivision–connectivity are generated by uniformly subdividing a coarse base mesh $\mathcal{S}_0$ (left). On the refined meshes we can easily identify regular submeshes (right) which topologically correspond to a single triangle of the base mesh $\mathcal{S}_0$ (left).*

enables efficient modifications of the gross shape while preserving little detailed features.

Parameterization Each submesh *(subdivision–patch)* can be parameterized naturally by assigning barycentric coordinates to the vertices. Combining the local parameterizations of the subdivision–patches yields a global parameterization. Texturing is just one application of such a parameterization.

Level–of–detail control Standard rendering libraries are able to display objects at various levels–of–detail, that is they display a coarse approximation, if the object is far away and switch to a finer one, if the viewer approaches. The different subdivision levels naturally support this feature. In combination with multiresolution analysis, switching to finer resolutions can be done smoothly by fading in the wavelet coefficients.

Recent investigations show, that compact and convenient representations for multiple of the applications above can be derived when using semi-regular meshes[15, 25, 18].

However, even if semi–regular meshes are particularly convenient for various types of applications, in practice it is rather unlikely that a given triangle mesh has this special type of connectivity (except those meshes originating from subdivision). During the last years, a couple of methods have been presented to convert a manifold triangle mesh into one having subdivision–connectivity and thus having access to the powerful methods developed for semi–regular meshes when an arbitrary input mesh is given.

Before we give an overview over three popular conversion schemes, we start by establishing the notation and describe some quality criteria. Let an arbitrary (manifold) triangle mesh $\mathcal{M}$ be given. The task of *remeshing* the input data $\mathcal{M}$ means to find a sequence of meshes $\mathcal{S}_0,\ldots,\mathcal{S}_m$ such that each $\mathcal{S}_{i+1}$ emerges from $\mathcal{S}_i$ by the application of a uniform subdivision operator which performs a 1–to–4 split on every triangular face of $\mathcal{S}_i$ (cf. Fig. 26). Since the $\mathcal{S}_i$ should be differently detailed approximations of $\mathcal{M}$, the vertices $\mathbf{p} \in \mathcal{S}_i$ have to lie on the continuous geometry of $\mathcal{M}$ but they not necessarily have to coincide with $\mathcal{M}$'s vertices. Furthermore it is allowed but not required, that the vertices of $\mathcal{S}_i$ are a subset of $\mathcal{S}_{i+1}$'s vertices.

In general it would be enough to generate the mesh $\mathcal{S}_m$ since the coarser levels of detail $\mathcal{S}_i$ can be extracted by subsampling. Nevertheless, building the whole sequence $\mathcal{S}_0,\ldots,\mathcal{S}_m$ from coarse to fine often leads to more efficient multi–level algorithms.

The quality of a subdivision–connectivity mesh is measured in two different aspects. First, we want the resulting parameterization which maps points from the base mesh $\mathcal{S}_0$ to the corresponding points on $\mathcal{S}_m$ to be close to *isometric*, i.e. the local distortion of the triangles should be small and evenly distributed over the patch. To achieve this, it is necessary to adapt the triangles in the base mesh $\mathcal{S}_0$ carefully to the shape of the corresponding surface patches in the given mesh $\mathcal{M}$.

The second quality requirement rates the base mesh $\mathcal{S}_0$ itself according to the usual quality criteria for triangle meshes, i.e. uniform size and aspect ratio of the base triangles.

As previously stated, a global parameterization of $\mathcal{M}$ can easily be derived if $\mathcal{S}_m$ is given. This is also possible the other way around, i.e. a semi–regular mesh can easily be constructed, if a global parameterization is available. Therefore, schemes that build a global parameterization on $\mathcal{M}$ can also be adapted to our task[24, 1, 17].

However, this tutorial focuses on the "classical" schemes that convert an input mesh into one with subdivision–connectivity.

4.2.1. Eck's scheme

In '95 Eck and co–workers[13] were the first who came up with a three step remeshing–scheme. This can roughly be described as follows.

Partitioning partitions $\mathcal{M}$ into regions $\mathcal{T}_0, \ldots, \mathcal{T}_r$ using discrete *Voronoi tiles*. The union of these tiles is dual to a triangulation which is used as the base triangulation.

Parameterization constructs a local parameterization for each submesh of $\mathcal{M}$ that corresponds to a base triangle and joins them which results in a global parameterization.

Resampling recursively performs 1–to–4 splits on the base domain triangles in $\mathcal{S}_0$ and maps the new vertices into $\mathbf{R}^3$.

In order to be able to compare this scheme with the two other schemes, that were recently published, we focus on some aspects of the algorithm. For a complete coverage we refer to the original paper.

The partitioning starts with a single seed–triangle. By sucessively adding adjacent triangles, the seed grows to a tile. New seed–triangles are added, if one of the restrictions, that ensure that the dual of the tiles builds a proper triangulation, is violated. The growing process terminates, when the tiles cover the complete surface $\mathcal{M}$ and the tiles' dual serves as the base triangulation. The vertices are located at the centroids of the seed–triangles. Thus the base–vertices are relatively equally spread over $\mathcal{M}$. However, the positions of the vertices and because of that the whole remesh heavily depends on the growing process, i.e. the algorithm cannot opt for the second quality criterion.

Using harmonic maps, a parameterization is constructed as follows. At first, one computes a harmonic map that carries each Voronoi tile $\mathcal{T}_i$ into a planar polygon $\mathcal{P}_i$. The inverse mapping is a parameterization of $\mathcal{T}_i$ over $\mathcal{P}_i$. This parameterization is evaluated to get the triangular submeshes which are dual to the Voronoi tiles $\mathcal{T}_i$. Finally, harmonic maps are used to map these submeshes to the corresponding triangle in $\mathcal{S}_0$. The combination of these mappings leads to an optimal parameterization for the base–triangles but it provides C^0–continuity over the edges of adjacent triangles only.

Finally, here is a summary of the algorithm's main features.

+ arbitrary manifold input–meshes
+ parameterization on base–triangles is close to isometric
– C^0–continuity over the edges of base–triangles only
– base–domain $\mathcal{S}_0$ not optimal according to quality criteria
– costly computations (harmonic mappings)

4.2.2. MAPS

An alternative approach addressing the remeshing problem is the MAPS algorithm presented by Lee et al. [26] Within the scope of this tutorial we roughly sketch the key features of the scheme. For a more elaborate discussion, please refer to the original paper. The base domain $\mathcal{S}_0$ is found by applying an incremental mesh decimation algorithm to the original mesh. (Cf. Sec. 5.1 for a closer look at mesh decimation). Compared to Eck's scheme, this provides more control on the generation of the base mesh since feature lines can be taken into consideration and local curvature estimates can be used to determine regions where more or less vertices are removed. The atomic simplification step is the *vertex removal*. Figures 27 and 28 illustrate how a parameterization over $\mathcal{S}_0$ can be derived. This parameterization is, again, not globally smooth but only locally within each patch corresponding to a base triangle. The actual remeshing is not done by sampling the initial parameterization at the dyadic barycentric parameter values as usual but an additional smoothing step based on a variant of Loop's subdivision scheme[28] is used to shift the sample sites within the parameter domain.

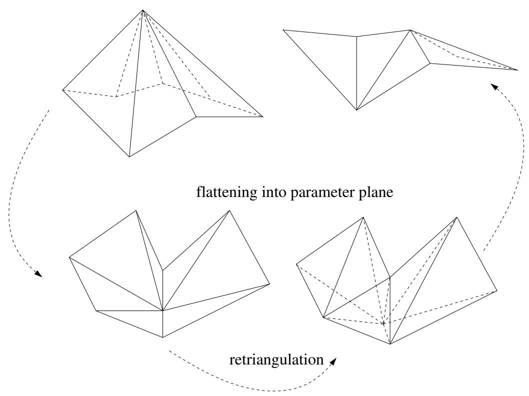


Figure 27: *To remove a vertex, its one–ring is mapped into the plane (exponential–map), the vertex removed, the resulting hole retriangulated and finally mapped back into 3 space (figure inspired by the original paper[26]).*

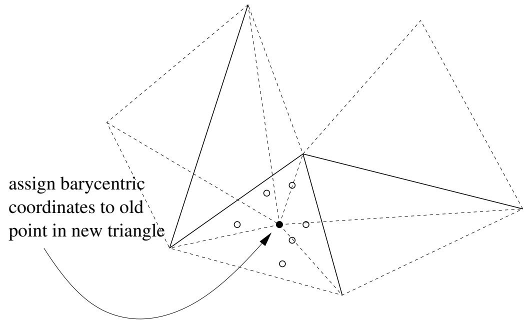


Figure 28: *After retriangulation (cf. Fig. 27), the removed vertex (black dot) gets assigned barycentric coordinates with respect to the containing triangle on the coarser level. Barycentric coordinates of previously removed vertices (hollow dots) are updated accordingly (figure inspired by the original paper[26]).*

The pros and cons of the scheme are enlisted below.

+ arbitrary manifold input–meshes
+ user defined feature lines
+ parameterization on base–triangles is close to isometric

+ $\mathcal{S}_m$ smooth over the edges of $\mathcal{S}_0$ ("sampling at Loop–positions")
– Without user input: base–domain $\mathcal{S}_0$ not optimal according to quality criteria, i.e. it depends on the mesh decimation only.

4.2.3. Shrink wrapping

A completely different approach to the remeshing problem for genus–zero objects is presented by Kobbelt and co–workers[23]. The physical model behind the algorithm is the process of *shrink wrapping* where a plastic membrane is wrapped around an object and shrunk either by heating the material or by evacuating the air from the space inbetween the membrane and the object's surface. At the end of the process, the plastic skin provides an exact imprint of the given geometry. To simulate the shrink wrapping process, they approximate the plastic membrane by a semi–regular mesh $\mathcal{S}_m$. Two forces are applied to its vertices. An attracting force pulls them towards the surface. A relaxing force is applied in order to optimize the local distortion energy and to avoid folding. This ensures an even distribution of the vertices. The attracting part is realized by a projecting operator **P** that simply projects $\mathcal{S}_m$'s vertices on $\mathcal{M}$. The relaxing is done by applying an operator **U** to all vertices in $\mathcal{S}_m$, that iteratively balances the vertex distribution. Thus, the shrink–wrapping is an iterative process, where one alternates the **P** and the **U** operator.

Nevertheless, the proposed scheme works slightly different in order to accelerate the underlying optimization process. Instead of immediately shriveling up $\mathcal{S}_m$, the remeshing process starts with an initial convex mesh $\mathcal{S}_0$ (e. g. an icosahedron). Once applying **P/U** converges on level $\mathcal{S}_i$, the scheme switches to the next refinement level $\mathcal{S}_{i+1}$. Hence, this multi–level approach generates intermediate levels, which are close to the final solution, with relatively low computational costs.

Unfortunately, the algorithm described so far works for simple input meshes $\mathcal{M}$ only. One of the problems that arise is that especially for the coarser approximations, the projection operator **P** might produce counter–intuitive results.

For this reason, they extend the basic shrink–wrapping algorithm with the aid of a parameterization F of $\mathcal{M}$ over the unit sphere. Both, $\mathcal{M}$ (using F's inverse) and $\mathcal{S}_0$ (projection) are mapped onto a sphere. Thus, **P** becomes trivial. The relaxation operator **U** is adapted to this in such a way, that it still measures the geometry on the original surface. This is done by associating triangles of $\mathcal{M}$ to corresponding surface areas of $\mathcal{S}_0$ (which is trivial, if both meshes are mapped to a sphere). This guarantees an equal distribution of $\mathcal{S}_0$'s vertices on $\mathcal{M}$ when evaluating $F(\mathcal{S}_0)$. In areas where the surface metric of $\mathcal{S}_0$ and $\mathcal{M}$ differ considerably, which would lead to severe stretching in the resulting remesh, new vertices are inserted into $\mathcal{S}_0$ by performing simple edge–splits. Once $\mathcal{S}_0$ is found, successive levels can be computed by either using the stabilizing parameterization over the sphere or directly, if $\mathcal{S}_i$ and $\mathcal{M}$ do not differ too much.

Summing up, here are the main features/limitations of the shrink–wrapping approach to the remeshing problem.

+ custom tailored base–domain $\mathcal{S}_0$ which is optimized with respect to both quality criteria
– in its basic form the algorithm works for genus–zero objects only

References

1. C. L. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *SIGGRAPH 95 Conference Proceedings*, pages 109–118, 1995.
2. A. A. Ball and D. J. T. Storry. Conditions for tangent plane continuity over recursively generated B–spline surfaces. *ACM Transactions on Graphics*, 7(2):83–102, 1988.
3. H. Biermann, A. Levin, and D. Zorin. Piecewise smooth subdivision surfaces with normal control. In *SIGGRAPH 00 Conference Proceedings*, to appear.
4. E. Catmull and J. Clark. Recursively generated B–spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10:350–355, 1978.
5. G. Chaikin. An algorithm for high speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974.
6. F. Cirak, M. Ortiz, and P. Schröder. Subdivision surfaces: A new paradigm for thin–shell finite–element analysis. *Internat. J. Numer. Methods Engrg.*, 47, 2000.
7. T. DeRose, M. Kass, and T. Truong. Subdivision surfaces in character animation. In *SIGGRAPH 98 Conference Proceedings*, pages 85–94, 1998.
8. T. DeRose, M. Lounsbery, and J. Warren. Multiresolution analysis for sufaces of arbitrary topological type. Technical Report 93–10–05, Department of Computer Science and Engineering, University of Washington, 1993.
9. D. Doo and M. Sabin. Behaviour of recursive subdivision surfaces near extraordinary points. *Computer Aided Design*, 10:356–360, 1978.
10. N. Dyn. Subdivision schemes in computer aided geometric design. *Advances in Numerical Analysis II, Wavelets, Subdivisions and Radial Functions*, pages 36–104, 1991.
11. N. Dyn, D. Levin, and J. A. Gregory. A 4–point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design*, 4:257–268, 1987.

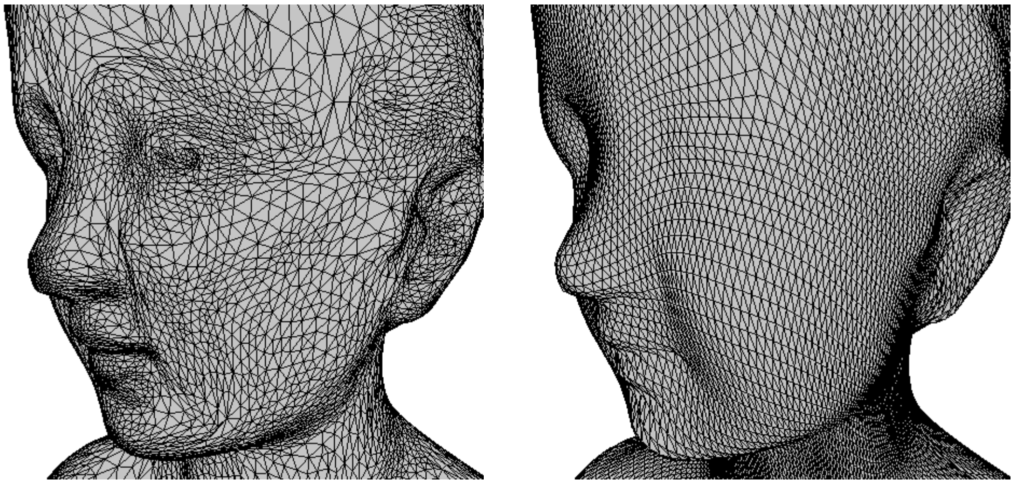


Figure 29: *A remeshed version (right) of a bust model (left) obtained with Kobbelt's shrink–wrapping approach.*

12. N. Dyn, D. Levin, and J. A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169, 1990.

13. M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH 95 Conference Proceedings*, pages 173–182, 1995.

14. D. Zorin et. al. Subdivision for modeling and animation. In *SIGGRAPH 00 Course Notes*, to appear.

15. I. Guskov, K. Vidimce, W. Sweldens, and P. Schröder. Normal meshes. In *SIGGRAPH 00 Conference Proceedings*, to appear.

16. M. Halstead, M. Kass, and T. DeRose. Efficient, fair interpolation using catmull–clark surfaces. In *SIGGRAPH 93 Conference Proceedings*, pages 35–44, 1993.

17. H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. In *SIGGRAPH 94 Conference Proceedings*, pages 295–302, 1994.

18. A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. In *SIGGRAPH 00 Conference Proceedings*, to appear.

19. L. Kobbelt. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. *Computer Graphics Forum*, 15(3):409–420, 1996.

20. L. Kobbelt. A variational approach to subdivision. *Computer Aided Geometric Design*, 13(8):743–761, 1996.

21. L. Kobbelt. $\sqrt{3}$–subdivision. In *SIGGRAPH 00 Conference Proceedings*, to appear.

22. L. Kobbelt, K. Daubert, and H.-P. Seidel. Ray tracing of subdivision surfaces. *Eurographics Rendering Workshop*, pages 69–80, 1998.

23. L. Kobbelt, J. Vorsatz, U. Labsik, and H.-P. Seidel. A shrink wrapping approach to remeshing polygonal surfaces. *Computer Graphics Forum*, 18(3):119–130, 1999.

24. V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH 96 Conference Proceedings*, pages 313–324, 1996.

25. A. Lee, H. Moreton, and H. Hoppe. Displaced subdivision surfaces. In *SIGGRAPH 00 Conference Proceedings*, to appear.

26. A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. MAPS: Multiresolution adaptive parameterization of surfaces. In *SIGGRAPH 98 Conference Proceedings*, pages 95–104, 1998.

27. A. Levin. Interpolating nets of curves by smooth subdivision surfaces. In *SIGGRAPH 99 Conference Proceedings*, pages 57–64, August 1999.

28. C. T. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, Department of Mathematics, 1987.

29. K. Pulli and M. Segal. Fast rendering of subdivision surfaces. *Eurographics Rendering Workshop*, pages 61–70, 1996.

30. U. Reif. A unified approach to subdivision algorithms near extraordinary vertices. *Computer Aided Geometric Design*, 12(2):153–174, 1995.

31. M. C. Rivara. Mesh refinement processes based on the generalized bisection of simplices. *SIAM J. Numer. Anal.*, 21(3):604–613, 1984.

32. J. Stam. Exact evaluation of catmull–clark subdivision surfaces at arbitrary parameter values. In *SIGGRAPH 98 Conference Proceedings*, pages 395–404, 1998.

33. D. Zorin. *C^k Continuity of Subdivision Surfaces*. PhD thesis, California Institute of Technology, Department of Computer Sciences, 1996.

34. D. Zorin, P. Schröder, and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *SIGGRAPH 96 Conference Proceedings*, pages 189–192, 1996.

35. D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *SIGGRAPH 97 Conference Proceedings*, pages 259–268, 1997.

5. Fine-to-coarse hierarchies

In the previous section methods were presented that generate mesh hierarchies from a coarse base mesh by refining "bottom–up" to a specified detail level. In this section we are going to discuss ways to build a hierarchy "top–down": Given a detailed mesh, a coarse base mesh is created by successively removing detail information.

Again, two types of hierarchies can be discerned: Levels of *complexity* are created through mesh decimation (Section 5.1) while levels of *smoothness* are achieved by fairing techniques (Section 5.2).

5.1. Mesh decimation

5.1.1. Introduction

The aims of this section are

- to provide an overview of the current body of literature that can be used as a starting point for further investigations;
- to aid in identifying application–specific needs and point you to common problems with simplification algorithms;
- to sum up what's involved in "rolling your own" mesh simplification tool.

First of all, to give you an idea of what geometry simplification can be used for, here are some typical applications:

Oversampled scan data: Meshes generated from scan data are usually huge and mostly uniformly sampled; the density of the mesh doesn't correspond to the detail needed locally.

Overtesselated surfaces: Dense meshes are also often generated from other surface representations; a typical example is the extraction of iso–surfaces using e.g. the marching cubes algorithm (see Section 2.2), which samples the surface at a fixed detail level. Also parametric surfaces, like NURBS patches (as used in CAD programs), are often just uniformly sampled in parameter space and thus also not adapted to the detail actually needed to represent local geometry.

Level–of–detail rendering: To speed up rendering, many systems support switching to a lower resolution model when the object is far away from the viewer. It is desirable to generate these simplified models automatically.

Progressive transmission: On low–bandwidth channels, transferring a complete object description might not be feasible. With progressive meshes, the basic shape of an object is transmitted first and can already be viewed; detail is subsequently transmitted until the target resolution is reached.

Multiresolution editing: If an object can be transformed into a representation of varying coarseness, large–scale modifications can be accomplished by working on a coarse level and automatically re–applying the detail information (see Section 6).

So, how do you go about geometry simplification? As a first approach, we could define the task at hand in loose terms as "creation of a low–polygon approximation of a complex model that is good enough for the intended application".

A close examination of this preliminary definition already raises a lot of questions:

- What are the properties of the model we have to consider? Geometry, topology, attributes may or may not be important.
- What exactly is an "approximation"? We have to define the actual *error* in some way.
- Finally, what is "good enough"? This is also entirely dependent on the application. We probably have to specify quality criteria on the model.

As you can see, the above problem statement is very vague and application–dependent, which might help to explain why there are so many mesh simplification algorithms, with their own particular strengths and weaknesses. In the following section we briefly review the major developments and approaches that have been taken until today.

5.1.2. A brief history

For domain–specific geometric models automated methods to generate lower levels of detail exist since the 1980s[37, 38]. The first more general simplification algorithms that were not tied to a particular application and data structure appeared in 1992:

Re–tiling[34] places new vertices on the existing geometry, inserting more points in areas of high surface curvature, and distributes them by point repulsion forces. Most of the existing vertices are then removed and a triangulation is created from the remaining set of vertices. While the optimization process leads to well–shaped triangles, there is no real way to guide the simplification process and to bound the surface error with this method, as Turk points out himself.

The same year, an incremental method was proposed in the pioneering work of Schroeder et al.[32]; simplification of the mesh is accomplished by removing a vertex and retriangulating the resulting hole. Approximation error is measured as the distance of the removed vertex to the (average) resulting plane of the retriangulation. There is no bound for the deviation from the *original* mesh.

Rossignac and Borrel introduce *vertex clustering*[27]: A spatial grid is superimposed onto the geometry, and all vertices in a cell are merged into one, collapsing the associated parts of the geometry. The maximum vertex error in this case is the size of a cell. The algorithm is simple and efficient, but since it is purely vertex–based, it ignores properties of the surface such as curvature and does not preserve the original topology.

In mesh optimization[17], minimization of an energy function is used to find an approximation of a triangular mesh that has a low number of vertices, which is achieved by removing and repositioning vertices and flipping edges. The desired approximation constraints are directly modeled into this function. The work is motivated by the surface reconstruction problem, but applies to mesh simplification as well.

Hoppe[15] later proposes a simpler variant of mesh optimization (using now only one operation, the edge collapse) to generate levels of detail and presents the *progressive mesh* representation for transmitting and storing a coarse approximation plus the detail information from which the original surface can then be reconstructed. The method extends to scalar surface attributes, such as vertex colors and normals. Feature edges (here called “discontinuity curves”) are supported as well. To generate a progressive mesh representation, the history of the applied simplification operations is stored together with the information needed to invert the process (basically the positions of the removed vertices). Beginning with the coarse base mesh, the original geometry can now be successively reconstructed. With efficient data structures[16], this representation usually consumes even less memory than the original, non–hierarchical mesh. Nearly all current mesh simplification algorithms support progressive mesh representations.

A number of incremental methods have been proposed in the years following, mainly differing in the way the approximation error is estimated[12, 21, 11, 7], with different consequences for memory consumption and running time, respectively. The incremental approach dominates the field of mesh decimation today in various refinements; because of their modular structure flexible choices in mesh data structure, evaluated criteria and local operations performed on the mesh are possible, thus enabling a wide range of applications; for this reason, the following text will concentrate on incremental methods.

Most authors concentrate on simplification within some *geometric* error bound, though many algorithms extend to surface attributes as well. A few authors have concentrated on visual appearance, most notably Cohen et al.[8] who introduce a *texture deviation metric* to guarantee a maximum screen–space error. Other than in previous algorithms, surface attributes are decoupled from the geometry by means of texture and normal maps, which leads to fewer restrictions on the simplification of geometry in order to retain a level of visual faithfulness.

Probably the biggest current challenge for mesh simplification methods lies in handling the ever–increasing amount of data in an efficient way: Polygonal models with millions or even billions of vertices are becoming more and more commonplace, and efficient as well as effective reduction of the complexity is often a prerequisite for any further processing. Due to limited main memory capacities the model has somehow to be processed without ever loading it completely into memory[23].

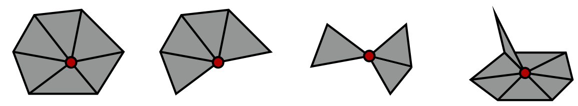


Figure 30: *Manifold and non–manifold triangle configurations*

Nearly all decimation algorithms restrict to triangle meshes, for a number of reasons:

- Triangles are always planar, wherever you move a vertex.
- Following from this: The surface is piecewise linear throughout, thus simplifying evaluations.
- Every polygonal surface can be triangulated.
- Constant–size data structures can be implemented more efficiently.

In the following, we first discuss suitable mesh data structures and then explain the general incremental decimation approach; existing methods will be discussed and classified by noting how particular subproblems are handled.

5.1.3. Prerequisites: mesh data structures

A mesh decimation algorithm usually needs more information than just vertex positions: the connectivity of the mesh needs to be accessible (and mutable), in order to easily examine and update mesh regions. Common queries to such a mesh data structure are:

- Which are the vertices forming a given triangle?
- Which triangles are adjacent to a given triangle?
- Which triangles are adjacent to a given vertex?
- Is a vertex/edge on a boundary?

There are a number of data structures that contain neighborhood information and can be used in mesh decimation, for example winged–edge[1] or half–edge data structures[4, 18].

These data structures cannot represent arbitrary triangulations, but are basically restricted to *bounded two–manifolds*, i.e. the surface has to be topologically disk–like at every point, or halfdisk–like on boundaries. This rules out edges with more than two adjacent triangles or points where two triangles meet only at their tips. Figure 30 shows triangle configurations, where the left two are legal for a manifold surface, the others are not.

This is a common restriction for topology–preserving mesh decimation algorithms. A topology–changing algorithms though is allowed to merge disconnected components of a triangle mesh. To deal with “real world” meshes, which often have some non–manifold parts, the offending vertices are usually marked as such and ignored during simplification.

5.1.4. Incremental methods outline

This is the outline of a simple incremental decimation algorithm[32]:

```
Repeat:
  find region with estimated error < epsilon
  remove triangles in region
  retriangulate hole
Until no further reduction possible
```

The problem with this naive approach is, that we are trying to solve a *global* optimization task by applying operations only based on *local* decisions. A greedy algorithm like this is prone to get stuck in a local optimum that may be very far away from the globally optimal solution.

A way to alleviate this problem is to introduce a global *ranking* for all candidate operations: Each currently possible operation is assigned a computed cost value and then – if the cost is acceptable – inserted into a priority queue. On each step of the algorithm the current least–cost operation is applied and the priorities for the affected neighborhood is reevaluated and updated. It should be noted, that this still doesn't guarantee that a global optimum will be found: No operation is ever performed that violates the prescribed error bound, even if at a later stage the surface would lie well within these bounds.

The improved queue driven algorithm now has an initial preprocessing phase to initialize the queue:

```
For all possible regions:
  op := operation on region
  c := cost of local decimation
  if c < epsilon
    (op,c) -> queue

Repeat:
  Apply least-cost operation
  Update queue
Until queue empty
```

Having a closer look at the local simplification step, there are still open questions:

- What exactly is a "region"?
- How should it be retriangulated?

Before we can answer these, we need to decide on the required behavior of the mesh decimation algorithm:

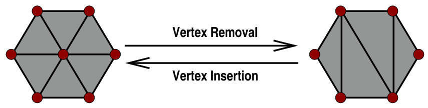
- Are we allowed to change the mesh topology or not?
- Does the simplified mesh have to consist of a subset of the original vertices or do we want to optimize the shape by shifting vertices around?
- Should the process be invertible for reconstruction (progressive mesh)?
- What region of the mesh is affected by the change? (We need to know this to compute an error estimate.)

The answers to these questions are all connected to the choice of the atomic changes we apply to the mesh; since they affect the connectivity of the mesh, they are called *topological operators*. The idea is, to have a small set of operators, or even only one operator, that changes the mesh in a clearly defined region in a computationally cheap, useful and well–defined way; this concept has been first introduced by Hoppe[17].

The following section gives an overview of common elementary operators. All the following operations have a corresponding inverse operation, varying in the information that has to be stored to perform it.

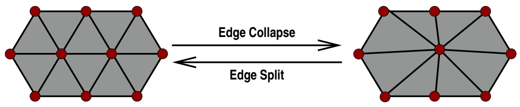
5.1.5. Topological operators

The first operator that may come to mind is close to our definition above: Remove a vertex and retriangulate the hole.

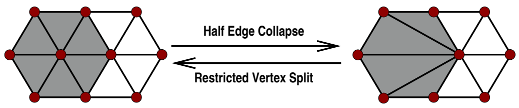


Simple as it is, we still have some degrees of freedom here, namely how we retriangulate the hole after removing the vertex. Optimizing for the best retriangulation can be algorithmically cumbersome for special cases have to be considered[32], and computationally expensive, depending on the *valence* of the vertex, i.e. the number of vertices connected to the removed vertex by triangle edges. As an additional drawback, a lot of information needs to be kept to invert the operation.

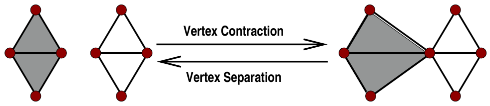
To reduce the number of degrees of freedom, a very common operation is the *edge collapse*[17], where two end vertices of an edge are merged, effectively removing one vertex and two triangles. Here the only degree of freedom left is the position of the merged vertex.



This operator can be restricted even further, if we choose the target position to be at one of the original two vertex positions. No new vertex positions are "invented", the only choice left is into which vertex to collapse the edge. This operation is called *restricted edge collapse* or *half–edge collapse*, because giving the half–edge completely specifies the operation[21].

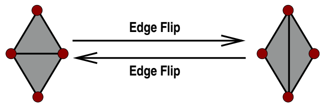


The previous operators do not modify the topology of the mesh, i.e. no unconnected regions of the mesh will be connected and no tears will disappear or be introduced. In topology–modifying algorithms, two vertices are allowed to merge even if they are *not* connected by an edge:



In this operation we may again choose to position the target vertex freely or restrict to one of the two previous vertex positions (this variant is shown in the picture).

Another common operator, the *edge flip*, does not change the vertices at all, but only the connectivity. This is often used as an optimization step, because flipping an edge may result in a better local approximation of the surface[17].



In the above pictures only the case of inner vertices are shown. To apply them to boundary vertices/edges/triangles, special care has to be taken to ensure a topologically consistent mesh.

Even in the non–boundary case, situations arise where an operator cannot be applied, e.g. it may generate coplanar triangles, as shown in figure 31. So, when choosing an operation, there has to be a check of the local topology, whether this operation can actually be applied[21].

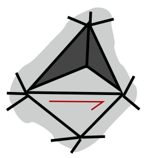


Figure 31: *A collapse along the arrow would cause the dark triangles to share all three vertices and thus cause a non-manifold configuration.*

5.1.6. Error metrics

Now we have clearly specified local operations to perform on a given mesh. Given that the operation of choice is topologically legal, we still have to check whether the approximation error it introduces lies within the given bounds. The definition of this error metric is where incremental decimation algorithms differ most. There are three main points one has to consider for classifying these metrics:

Measured properties: Topology, geometry, attributes

Global vs. local: Is the error evaluated only from one step to the next, or is there a comparison to the original geometry?

Interpretation: To be useful for specifying the error bound, the metric must be intuitive; also, there shouldn't be too many parameters, that the user has to adjust manually.

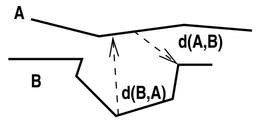


Figure 32: *Hausdorff distance $d(A,B)$ compared to $d(B,A)$*

An example for a local measure is the distance to the previous mesh[32]. The problem here is of course, that no useful statement can be made as for how faithful the simplified geometry is compared to the original. Even if one accumulates the error values obtained at each step, this results only in an overly conservative estimate of the global error, prohibiting high simplification rates. Having a tight, reliable upper bound for the surface error is crucial in e.g. CAD design or medical applications.

Many algorithms try to guarantee an upper bound on the *distance error* between the original and the simplified surface, as in the conceptually straightforward approach by Cohen et al.[9]: inner and outer offset surfaces are computed and used by an incremental procedure to simplify the mesh within these global bounds.

A common way to define distances between meshes is the *Hausdorff distance* between to shapes A and B, which is defined as $max(d(A,B),d(B,A))$, where $d(X,Y)$ (the so–called *one–sided Hausdorff distance*) is the maximum of the distances from any point on X to the closest point on Y. It should be noted, that the distance from shape A to shape B is in general not the same as the distance from B to A (see figure 32).

The Hausdorff distance is an intuitive global error measure, but difficult to compute[19]. In the case of scanned data, one can argue that only the *vertices* of the original mesh hold actual information, while the triangulation merely interpolates these points. From this point of view, it is sufficient to estimate the *one–sided Hausdorff distance* by measuring the distance from the original surface *vertices* to the *triangles* of the simplified surface[21].

Measuring the global error usually requires to carry along some sort of global history during simplification, e.g. the removed vertices[21] or information about the current error bound in the form of scalar values, error volumes or other data[11,31].

By preserving global volume of the shape with each local simplification step[24,25], no global history needs to be maintained, but although the method yields in general very good approximations, no upper bound on the surface error can be given. Recently, error quadrics where introduced[11], which can be computed very fast and in general give pleasing results, but unfortunately the interpretation of the surface error is unintuitive ("squared distance of vertex to a set of planes"). It is not obvious how this measure relates to the Hausdorff distance.

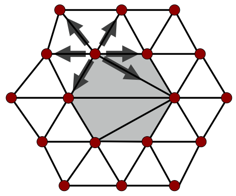


Figure 33: *Situation after an edge collapse; the dark region of the mesh has changed, thus for each vertex adjacent to that region (here shown exemplary for one vertex only), the potential collapses (marked by arrows) need to be reevaluated.*

Only a few authors elaborate explicitly on how not only the geometry of a surface can be approximated, but also surface attributes like colors, vertex normals and texture, though most of the methods extend to these properties[8, 15].

Meshes with boundaries (as opposed to solids) involve treatment of special cases. E.g. if a vertex on a boundary is removed, the distance error now needs to be measured from that vertex to the boundary polygon, not a triangle. Some methods apply completely different criteria than for simplification of inner vertices (such as preservation of the length of the boundary polygon[13]).

When rejecting removal of a vertex due to violation of the error bound, it is important to re–evaluate that vertex in a later stage, since it may again become a candidate due to changes in the nearby regions of the mesh. This is usually achieved by updating priority queue entries for the neighborhood of the removed vertex (commonly involving all triangles in the *2–ring* of that vertex, i.e. the triangles adjacent to the changed regions, as shown in figure 33.

For validation of the error bounds the Metro tool[6] is a useful, publically available software to compare meshes in terms of surface distances and volume difference.

5.1.7. Quality metrics

The error metric explained in the previous section serves to select the valid candidate operations for the next simplification step. But we still need to know, in what *order* these operations should be performed, to guide the process towards "better" surfaces. This is achieved by evaluating the operations using a *quality metric*, which is conceptually separate, but in practice often intertwined with the error metric.

As a simple example, an algorithm that relies on bounding the distance error between two surfaces can't detect *fold–overs*: A point moving in the plane can cause the orientation of adjacent triangles to flip. To generate "reasonable" approximations, more control needs to be exerted (e.g. by disallowing a more than 90 degree change of the face normal). Also, *self intersections* of a shape can appear, if the opposite sides of a model are closer together than the bound on the distance error.

We can group quality criteria by their domain:

Geometry: Smooth surfaces are usually desirable; apart from the "triangle flipping" problem, there may be user–imposed limits on surface curvature. Also, good preservation of features can be understood as controlling quality. Decimation algorithms often optimize for well–shaped triangles employing some measure for "roundness" (generally maximizing triangle area with respect to edge lengths).

Topology: A regular mesh structure is often preferable. In a topology–changing decimation algorithm, one might want to minimize the number of separate components.

Attributes: Common surface attributes are colors, texture coordinates and vertex normals. Faithful reproduction of the original appearance requires minimizing texture distortion, preserving discontinuities in surface colors and minimizing normal deviation (cf. Fig. 34).

Often, the evaluated criteria are used both for error and quality estimation: E.g. we may optimize for low distortion of a triangle, given some scalar measure, but have a threshold where we simply reject the operation, if the achievable quality is too low.

5.1.8. Practicalities

Besides the core decimation routines, authoring a general simplification tool often involves more to make it valuable in practice. The following list states in a nutshell a few key points to consider; this does of course not claim to be complete, as domain–specific demands always have to be added.

Feature preservation: The user should be able to specify features on the surface that are retained; this can be done completely manually by selecting individual vertices or edges, or semi–automatically (the program detects and marks features based on user–specified criteria).

Configuration: The default parameters for error bounds and quality criteria should already be useful; named presets are a way to hide complex parameter sets ("optimize for triangle shape"; "optimize for smoothness").

Progressive mesh generation: The decimation algorithm needs to store the record of applied operations along with information to invert them.

Robustness: If the decimation algorithm cannot handle non–manifold or degenerate mesh regions, these should be detected and dealt with in a graceful way – e.g. by simply ignoring them. This also influences the mesh data structure of choice: If a mesh cannot even be represented internally, there is obviously no way to deal with degeneracies.

Speed: The choice of decimation criteria has a great effect on speed and scalability of the algorithm. It should be carefully investigated, whether exact error bounds are a

Figure 34: *Example of mesh decimation using error metrics for preservation of surface attributes; left: original model (400K triangles); center: decimated without respecting color attributes (5K triangles); right: decimated preserving color discontinuities (5K triangles). The algorithm trades geometry preservation for color preservation in this case.*

necessity – this can make the difference between a running time of a few seconds versus several minutes, or even hours.

Memory Requirements: When meshes in the range of a few million triangles have to be simplified in–core, easily hundreds of megabytes of RAM are consumed. Efficient data structures are very important here, and again the choice of error bounds plays a role, because memory usage is higher, if e.g. a global history has to be maintained.[25]

5.2. Discrete fairing

Meshes usually have to satisfy aesthetic requirements. While the aesthetic appearance of a surface is always subjective, it is nevertheless possible to formulate a principle that makes it possible to quantify the surface fairness, the principle of the simplest shape[3]. Generally speaking, a shape should be free of unnecessary details as noise or oscillations. Discrete fairing creates a mesh that is free of such unnecessary detail. To speed up the fairing process, multigrid algorithms based on mesh decimation are often integrated in the construction algorithms.

There are two major cases where mesh fairing algorithms are applied. In one case one is interested in smoothing out the high frequency details of an existing mesh with the constraint to keep the low frequency components. Typical fields of application are smoothing of polyhedral surfaces as those extracted from volumetric medical data by iso-surface construction algorithms or those resulting from laser-range scanners. Because of the huge mesh size in such applications, it is especially important to have fast algorithms here. The other application is free form surface modeling. Here the problem is to create a well defined smooth surface that is fair. Tightly connected to this application is multiresolution editing of arbitrary meshes. Here fairing algorithms are needed that are able to locally preserve the shape of the triangulation to enable a simple detail encoding. In this context mesh hierarchies resulting from mesh decimation in combination with discrete fairing are used to generate a geometric hierarchy.

When quantifying the quality of a mesh, in contrast to smooth surfaces, we have to be aware that there are two different fairness types, *outer* and *inner* fairness. The outer fairness measures the fairness of the mesh geometry. This controls the quality of the shape that is approximated by the mesh. The inner fairness addresses the parameterization of the mesh vertices. It determines how the mesh vertices are distributed over the resulting surface (Fig. 35).

Most mesh fairing schemes are based on linear methods. These algorithms do not strictly separate the outer and inner fairness concept, so the inner fairness strategy greatly influence the outer fairness and therefore the shape of the resulting mesh. Other important influences are mesh size and connectivity (Fig. 35). Therefore, linear approaches produce results that are in most cases clearly not as fair as nonlinear approaches that depend on intrinsic surface properties only and there is no guarantee that the surfaces are free of parameterization artifacts. Moreover, it is also harder for a designer to develop an intuitive understanding for the manipulation behavior.

To avoid such influences, one has to use fairing algorithms that depend on intrinsic surface properties only, properties that purely depend on the geometry. Altering the mesh size, the connectivity or the inner fairness condition only produces another discretization of the same smooth surface and hence leads to geometries that will be close to each other (Fig. 35). In practice this puts the designer in the position to choose the inner fairness condition freely.

However, linear approaches have some important advantages against their nonlinear counterparts. The resulting algorithms are usually simple and enable a fast implementation. Another important property is that such linear approaches are in general mathematically well understood. Here it is usually possible to give practicable statements under which conditions a solution exists respectively when the construction process converges. Intrinsic fairing is a nonlinear problem and often considerable more costly. Here, for many popular fairing functionals, it is also still unknown if a solution always exists within specific smoothness classes.

5.2.1. Linear methods

Energy minimization

The standard approach for fair surface construction is based on the idea to minimize a fairness metric, punishing intrinsic features that are inconsistent with the fairness principle of the simplest shape[3]. While this leads to nonlinear functionals, a popular technique to simplify this approach is to give up the parameter independence by approximating the geometric invariants with higher order derivatives. For some important fairness functionals this results in algorithms that enable the construction of a solution by solving a linear system. This is the discrete fairing approach that was presented by Kobbelt in [20].

Diffusion flow

A very effective method for smoothing polyhedral surfaces is the discrete diffusion flow[33]. Here the idea is to iteratively update each vertex q_i

$$q_i^{new} = q_i + t\,\Delta q_i \tag{6}$$

by adding a displacement vector that is a scaled discrete Laplacian. Assigning a scalar weight λ_{ij} to every vertex q_j that is adjacent to q_i with the constraint $\sum_j \lambda_{ij} = 1$, the discrete Laplacian Δq_i is defined as

$$\Delta(q_i) = \sum_{q_j \in N(q_i)} \lambda_{ij} q_j \; - \; q_i.$$

Here $N(q_i)$ denotes the set of all vertices adjacent to q_i. This algorithm is linear in time and space complexity and therefore well suited to fair even very large polyhedral surfaces. In this formulation, for stability reasons the scale factor t has to be a positive number satisfying $0 < t < 1$. This approach can be interpreted as forward Euler discretization of the diffusion equation. Using backward Euler discretization, Desbrun et al.[10] showed that it is possible to develop a diffusion flow that is unconditionally stable and hence enables larger scaling factors t.

The main purpose of the diffusion flow is to smooth the high frequencies in noisy meshes. Since the equilibrium surface of the flow only enables C^0 boundary conditions, in this form it is of only limited use in free-form fair surface design. To enable smooth boundary conditions one has to consider diffusion equations of higher order. In [33] Taubin proposed to combine two such smoothing steps with positive and negative scale factors and developed an algorithm that enables various interpolation constraints. Another idea that enables smooth boundary conditions would be to use higher powers of the Laplacian in the diffusion flow. A good trade-off between efficiency and quality is the bilaplacian flow, enabling C^1 boundary conditions. This flow also results if we chose scale factors of equal absolute value in Taubins algorithm.

Laplacian smoothing

A special case of the diffusion flow is known as Laplacian smoothing. Here the idea is quite simple: Each vertex q_i is replaced such that $\Delta(q_i) = 0$ is satisfied locally, that means we have

$$q_i = \sum_{q_j \in N(q_i)} \lambda_{ij} q_j.$$

As we can see this equation results from (6) by setting $t = 1$. Although this value is outside the specified range allowed for the diffusion flow, convergence is guaranteed if boundary conditions are specified. There are two principal methods to update the vertices q_i, *simultaneous* and *sequential* Laplacian smoothing[35]. The first updates the vertices in a Jacobi like manner, the second depends on partially previously calculated new vertices in a Gauss-Seidel like manner. Although the simultaneous version needs more storage space for holding the old positions of q_i, it produces better results, since the outcome does not depend on the update order of the local smoothing steps.

PDE discretization

Another mesh fairing approach is based on the idea to discretize the PDE approach of Bloor and Wilson[2]. In [22] Kobbelt et al. proposed to discretize

$$\Delta^2 f = 0, \tag{7}$$

to create surfaces satisfying prescribed C^1 boundary conditions. This equation results if one uses variational calculus to describe the surface that minimizes the thin plate energy

$$\iint f_{xx}^2 + 2 f_{xy}^2 + f_{yy}^2 \, dxdy.$$

In practice equation (7) is discretized and the resulting linear system is solved using an iterative multigrid solver. The multigrid approach considerably speeds up the construction algorithm. In this formulation the algorithm is fast enough to be used in interactive free form mesh modeling. The hierarchies for the multigrid approach are determined using a mesh decimating algorithm based on the progressive mesh approach presented by Hoppe[15] (see 5.1.2).

The last fairing schemes mentioned above can be partitioned into three categories, depending on whether fairing is based on some kind of diffusion flow, Laplacian smoothing or whether they discretize a PDE that characterizes the smooth solution. All three categories are tightly connected. For example the two Laplacian smoothing algorithms can

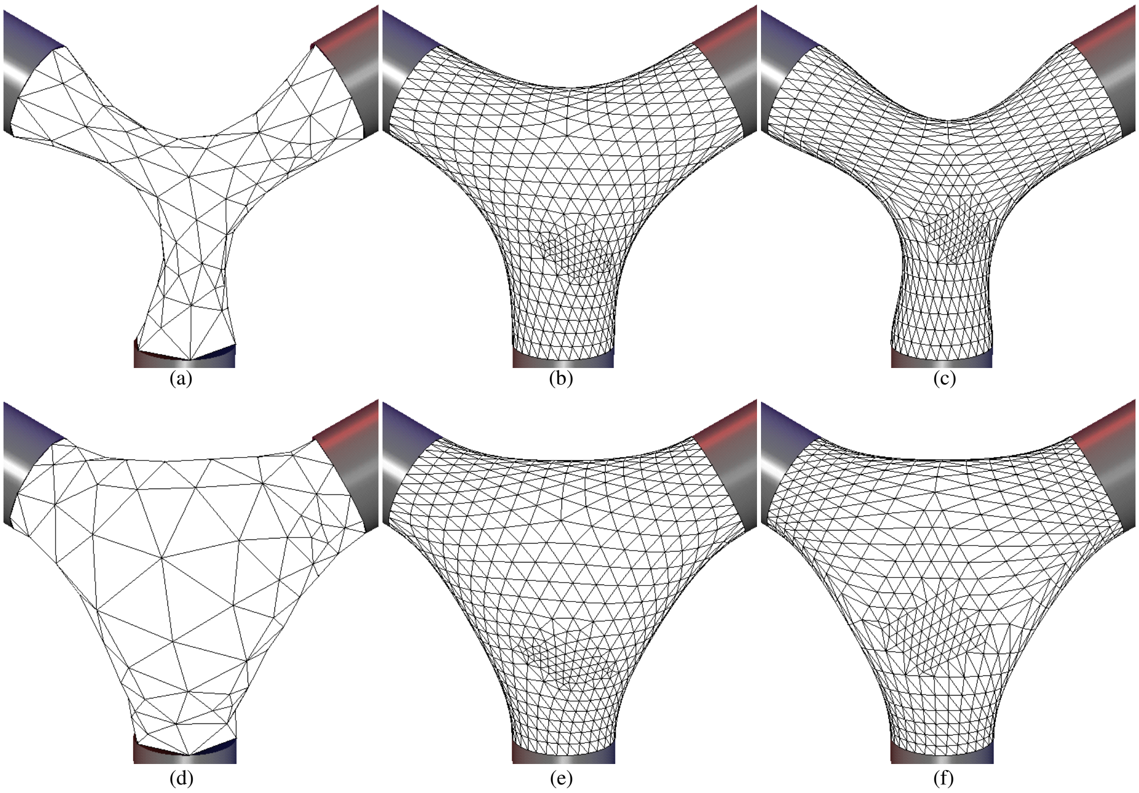


Figure 35: *a) - c) show examples of mesh fairing based on discretizing the equation $\Delta^2 f = 0$. The boundary condition is determined by 3 cylinders that are arranged symmetrically. In a) and b) we see the results for two different mesh sizes, if a local uniform parameterization is assumed. In c) we used a discretization of a Laplacian that locally preserves the shape of the triangles. We can see that the mesh size and the local parameterization strategy strongly affect the results. The rectangular patch introduced in the original mesh leads to local as well as global shape distortions and prevents a symmetric solution. d) - f) show examples of an intrinsic fairing operator, whose outer fairness is based on the non-linear PDE $\Delta_B H = 0$. We can see that the chosen inner fairness operator and the mesh structure have only marginal influence on the shape. The influence of the mesh size is also much weaker than in the linear setting.*

be seen as performing some steps of an Jacobi or Gauss-Seidel iterative solver on the linear system that is derived from the global equation $\Delta f = 0$. And it is also possible to create the surface that is given by the PDE (7) using the fact that the bilaplacian diffusion flow has this equation as its equilibrium. Moreover, in all three categories the discretization of the Laplacian plays a central role. During the last years various linear discretizations of the Laplacian have been developed[33, 22, 10, 14], differing in how the geometry of the mesh influences the discretization. The chosen discretization strategy determines the inner mesh fairness, but it also greatly influences the geometry of the resulting mesh (Fig. 35).

5.2.2. Nonlinear methods

Energy minimization

A common method to create fair surfaces is based on the idea to minimize a fairness metric that punishes unaesthetic surface properties. In [36] Welch and Witkin proposed a mesh fairing algorithm that enables G^1 boundary conditions based on the idea to minimize the total curvature

$$\int_A \kappa_1^2 + \kappa_2^2 \, dA,$$

thus punishing large curvature values. The necessary intrinsic curvature values were estimated using local quadratic approximations over local planar parameterizations. The local parameter domains are constructed using an exponential map of the 1-neighborhood around each vertex. This operator maps the neighborhood onto the plane while preserving the distances of the vertex to its neighbors and the ratio of the adjacent angles. The quadratic approximations are then constructed using local quadratic least square approximation. If the least square approximation is underdetermined, the number of the basis functions is reduced, making the discretization process discontinuous and thus leading to a potential instability risk in the mesh fairing algorithm. To overcome that risk, Welch and Witkin propose to optimize some auxiliary norm in the underdetermined cases, which requires an orthogonal matrix decomposition which is computationally expensive.

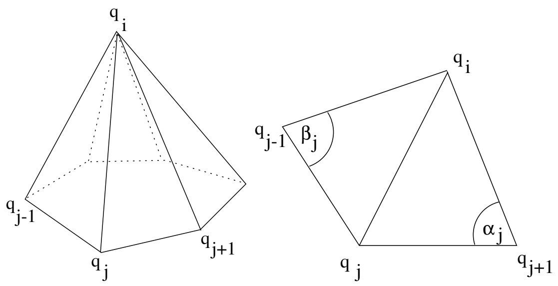


Figure 36: *The mean curvature vector at the vertex q_i can be discretized using the* 1*-disk.*

Diffusion flow

An intrinsic diffusion operator that is the nonlinear analogon to (6) was presented by Desbrun et al. in [10] using the discrete mean curvature flow

$$q_i' = q_i + \lambda H\vec{n},$$

where H is the mean curvature and $\vec{n}$ the outer unit surface normal, leading to an excellent noise reduction algorithm. They proved that the mean curvature vector can be discretized as

$$H\vec{n} = \frac{1}{4A} \sum_{q_j \in N(q_i)} (cot\alpha_j + cot\beta_j)(q_i - q_j),$$

where A is the sum of the triangle areas of the 1-disk at q_i and α_j and β_j are the triangle angles as shown in figure 36. Assuming that changes induced in a time step will be small, it can be handled quite analogously to the linear diffusion equation (6) and it is also possible to formulate an unconditional stable flow.

While this fairing algorithm is mainly designed to optimize the outer fairness in its original formulation, Ohtake et al.[26] combined this algorithm with an inner fairness criterion that leads to more regular meshes. To achieve this result, the authors propose to move the vertex in the direction defined by the linear diffusion flow using local uniform parameterization but with the speed equal to a properly chosen function of the mean curvature.

These intrinsic diffusion operators are excellent for smoothing noisy meshes, but since these algorithms converge to a discrete minimal surface satisfying $H = 0$, they only enable C^0 boundary conditions and are therefore, as their linear counterparts, of only limited use in free-form fair surface design. As for the linear diffusion flow algorithms, the solution would be to use curvature flows of higher order as for example the Laplacian of curvature flow[5].

PDE discretization

A fairing algorithm for arbitrary connected triangular meshes, that enables G^1 boundary conditions (prescribed vertices and unit normals) and allows the designer to completely separate outer and inner fairness conditions was presented by Schneider et al. in [30]. To achieve this, the outer fairness concept is based on the idea to discretize the intrinsic PDE

$$\Delta_B H = 0, \tag{8}$$

which can be interpreted as one possible nonlinear analogon to thin plate splines (7). Here Δ_B is the Laplace Beltrami operator and H the mean curvature. This equation characterizes the equilibrium state of the Laplacian of curvature flow[5]. The PDE only depends on geometric intrinsics and is comparatively simple for a forth order equation. The resulting surfaces extend an important fairing concept, that makes spiral curves so interesting in the planar case[28]. Because of the mean value property of the Laplacian, it is guaranteed that the extremal mean curvature values of a solution of (8) will be reached at the border and that there are no local extrema in the interior[29]. Thus it satisfies the important fairing principle of the simplest shape[3]. Since constant mean curvature surfaces satisfy this equation, important basic shapes as spheres and cylinders can be reconstructed. A coarse mesh already approximates the shape of the smooth surface that is implicitly defined by the PDE, so increasing the mesh size mainly improves the smoothness of the approximation. This property is exploited to improve the efficiency of the construction algorithm by using multigrid methods for arbitrary meshes[22]. The necessary mesh hierarchies are again created using the progressive mesh representation as introduced by Hoppe[15]. To further speed up the construction scheme, the fourth order PDE is factorized into two second order problems. Instead of trying to simulate the continuous case using local quadratic approximations, the authors use the discrete data of the construction algorithm directly.

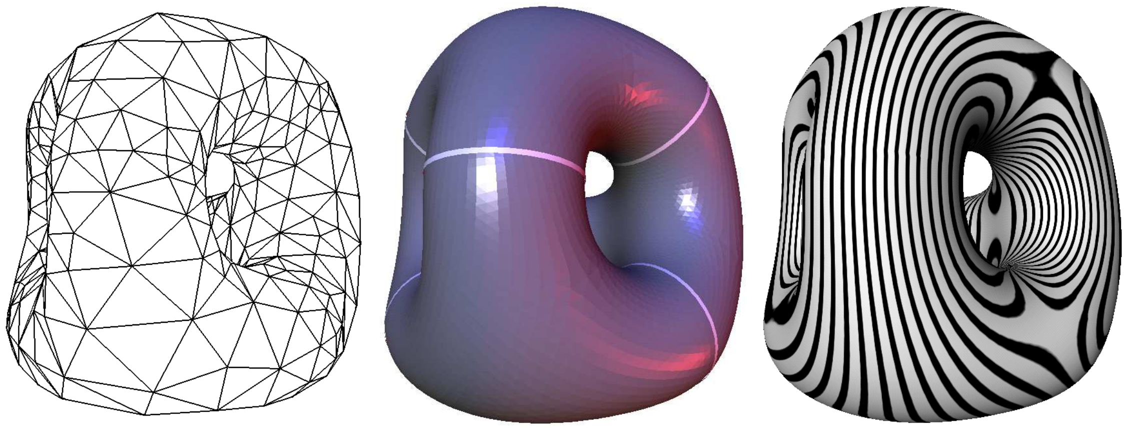
References

1. B. G. Baumgart. Winged–edge polyhedron representation. Technical Report STAN–CS–320, Stanford University, Computer Science Department, 1972.

2. M. I. G. Bloor and M. J. Wilson. Using partial differential equations to generate free–form surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 22:202–212, 1990.

3. H. G. Burchard, J. A. Ayers, W. H. Frey, and N. S. Sapidis. Approximation with aesthetic constraints. In *Designing Fair Curves and Surfaces*, pages 3–28. SIAM, Philadelphia, 1994.

4. S. Campagna, L. Kobbelt, and H.-P. Seidel. Directed



(a) (b) (c)

Figure 37: 6 *circles with* C^1 *boundary–conditions are used to define a "tetra thing". Due to the symmetry the final solution here is actually* G^2 *continuous, which is indicated in c) by the smooth reflection lines. The pictures were constructed using an intrinsic fairing operator, whose outer fairness is based on* $\Delta_B H = 0$.

edges: A scalable representation for triangle meshes. *Journal of Graphics Tools*, 3(4):1–11, 1998.

5. D. L. Chopp and J. A. Sethian. Motion by intrinsic laplacian of curvature. In *Interfaces and Free Boundaries 1*, pages 1–18, 1999.

6. P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

7. J. Cohen, Dinesh M., and M. Olano. Simplifying polygonal models using successive mappings. In *IEEE Visualization '97 Conference Proceedings*, pages 395–402, 1997.

8. J. Cohen, M. Olano, and D. Manocha. Appearance–preserving simplification. In *SIGGRAPH 98 Conference Proceedings*, pages 115–122, 1998.

9. J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. P. Brooks, Jr., and W. Wright. Simplification envelopes. In *SIGGRAPH 96 Conference Proceedings*, pages 119–128, 1996.

10. M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH 99 Conference Proceedings*, pages 317–324, 1999.

11. M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Conference Proceedings*, pages 209–216, 1997.

12. A. Guéziec. Surface simplification with variable tolerance. In *Second Annual Intl. Symp. on Medical Robotics and Computer Assisted Surgery*, pages 132–139, 1995.

13. A. Guéziec. Locally toleranced surface simplification. *IEEE Transactions on Visualization and Computer Graphics*, 5(2), 1999.

14. I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *SIGGRAPH 99 Conference Proceedings*, pages 325–334, 1999.

15. H. Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings*, pages 99–108, 1996.

16. H. Hoppe. Efficient implementation of progressive meshes. *Computers and Graphics*, 22(1):27–36, 1998.

17. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *SIGGRAPH 93 Conference Proceedings)*, pages 19–26, 1993.

18. L. Kettner. Designing a data structure for polyhedral surfaces. Technical Report TR 278, ETH Zürich, Institute of Theoretical Computer Science, 1997.

19. R. Klein, G. Liebich, and W. Straßer. Mesh reduction with error control. In *IEEE Visualization '96 Conference Proceedings*, pages 311–318, 1996.

20. L. Kobbelt. Discrete fairing. In *Proceedings of the Seventh IMA Conference on the Mathematics of Surfaces*, pages 101–131, 1996.

21. L. Kobbelt, S. Campagna, and H.-P. Seidel. A general framework for mesh decimation. In *Graphics Interface*, pages 43–50, 1998.

22. L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi–resolution modeling on arbitrary meshes. In *SIGGRAPH 98 Conference Proceedings*, pages 105–114, 1998.

23. P. Lindstroem. Out–of–core simplification of large polygonal models. In *SIGGRAPH 00 Conference Proceedings*, to appear.

24. P. Lindstrom and G. Turk. Fast and memory efficient polygonal simplification. In *IEEE Visualization '98 Conference Proceedings*, pages 279–286, 1998.

25. P. Lindstrom and G. Turk. Evaluation of memoryless simplification. *IEEE Transactions on Visualization and Computer Graphics*, 5(2), 1999.

26. Y. Ohtake, A. G. Belyaev, and I. A. Bogaevski. Polyhedral surface smoothing with simultaneous mesh regularization. In *Proceedings Geometric Modeling and Processing*, pages 229–237, 2000.

27. J. Rossignac and P. Borrel. Multi–resolution 3D approximation for rendering complex scenes. In *Second Conference on Geometric Modelling in Computer Graphics*, pages 453–465, 1993. Genova, Italy.

28. R. Schneider and L. Kobbelt. Discrete fairing of curves and surfaces based on linear curvature distribution. In *Curve and Surface Design*, pages 371–380, 1999. Saint Malo, France.

29. R. Schneider and L. Kobbelt. Generating fair meshes with g^1 boundary conditions. In *Proceedings Geometric Modeling and Processing*, pages 251–261, 2000.

30. R. Schneider and L. Kobbelt. Geometric fairing of irregular meshes for free–form surface design. submitted.

31. W. J. Schroeder. A topology modifying progressive decimation algorithm. In *IEEE Visualization '97 Conference Proceedings*, pages 205–212, 1997.

32. W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. In *Computer Graphics (SIGGRAPH 92 Conference Proceedings)*, volume 26, pages 65–70, 1992.

33. G. Taubin. A signal processing approach to fair surface design. In *SIGGRAPH 95 Conference Proceedings*, pages 351–358, 1995.

34. G. Turk. Re–tiling polygonal surfaces. In *Computer Graphics (SIGGRAPH 92 Conference Proceedings)*, volume 26, pages 55–64, 1992.

35. J. Vollmer, R. Mencl, and H. Muller. Improved laplacian smoothing of noisy surface meshes. In *Computer Graphics Forum (Proc. Eurographics)*, pages 131–138, 1999.

36. W. Welch and A. Witkin. Free–form shape design using triangulated surfaces. In *SIGGRAPH 94 Conference Proceedings)*, pages 247–256, 1994.

37. L. Williams. Pyramidal parametrics. In *Computer Graphics (SIGGRAPH 83 Conference Proceedings)*, volume 17, pages 1–11, 1983.

38. S. A. Zimmerman. Applying frequency domain constructs to a broad spectrum of visual simulation problems. Technical report, Evans & Sutherland, 1987. Presented at the IMAGE IV Conference, Phoenix, Arizona.

6. Geometric modeling based on polygonal meshes

Finally we come to the last stage in our virtual mesh processing pipeline. Until now we presented many powerful methods to efficiently process polygonal meshes. Having all the means at hand, we are able to focus on (interactive) multiresolution modeling, where a designer modifies a given surface by sophisticated editing operations. We distinguish between two different approaches: *Freeform modeling* is the process of modifying subregions of the surface in a *smooth* manner whereas the notion of *multiresolution modeling* describes edits where we can additionally preserve little geometric features.

Traditionally, geometric modeling is based on polynomial surface representations[1, 8, 9]. However, while special polynomial basis functions are well suited for describing and modifying smooth triangular or quadrilateral *patches*, it turns out to be rather difficult to smoothly join several pieces of a composite surface along common (possibly trimmed) boundary curves. As flexible patch layout is crucial for the construction of non–trivial geometric shapes, spline–based modeling tools spend much effort to maintain the global smoothness of a surface. For this reason, considerable effort has been spend to extend these traditional methods to polygonal meshes.

Just like in the two previous sections (where we carried many of the advantageous features of parametric surfaces over to polygonal surfaces, while getting rid of the severe restrictions inherent to them) this section discusses some approaches that make freeform and multiresolution modeling available for triangle meshes. Opposed to splines, where the control vertices provide a convenient way to smoothly edit the surface, this is a challenging task, since plain triangle meshes do not have any reasonable control mechanism. Before we describe in detail, how intuitive modeling metaphors for triangle meshes can be accomplished, we describe the general requirements a modeling tool should satisfy.

Intuitive I.e. editing the overall shape with an easy to use control mechanism (cf. control vertices of splines) in a broad, smooth manner while preserving little features residing on the surface should be possible.

Independent The editing interface should abstract from the underlying mesh representation, since in general a designer is not interested in how the surface is actually represented.

Interactive This is crucial, since a designer heavily depends on immediate visual feedback when performing further edits.

This part of the tutorial is organized as follows. At first we will deal with freeform modeling. This can be done with the help of *discrete fairing* or *subdivision* respectively. Since we explained these methods in previous sections (cf. 5.2,4.1), we restrict ourselves to describe, how an efficient control mechanism (similar to the control–points of spline–based methods) can be derived. In the second part we will show, how to build a hierarchical structure for semi–regular– and for unstructured meshes. Combined with freeform modifications, this enables us to perform multiresolution modeling. Finally we will briefly discuss some specific modeling schemes, that were proposed during the last years.

6.1. Freeform modeling

Subdivision schemes can be considered as the algorithmic generalization of classical spline techniques enabling control meshes with arbitrary topology. They provide easy access to globally smooth surfaces of arbitrary shape by iteratively applying simple refinement rules to the given control mesh. A coarse–to–fine hierarchy of meshes generated by this process quickly converges to a smooth limit surface. For most practical applications, the refined meshes are already sufficiently close to the smooth limit after only a few refinement steps. Lets assume we are given a **semi–regular mesh** $\mathcal{M}_n$, which was generated by applying some subdivision operator $\mathbf{S}$ to a base mesh $\mathcal{M}_0$, and we want to modify $\mathcal{M}_n$ with specific support. The usual way to implement this operation is to run a decomposition scheme several steps until the desired resolution level/mesh $\mathcal{M}_i$ is reached. In our setting, this can simply be done by subsampling, i.e. we just switch to $\mathcal{M}_i$. On this level the mesh $\mathcal{M}_i$ is edited. Applying $\mathbf{S}$ to the modified mesh $\mathcal{M}_i'$ $(n-i)$–times yields the final result. This operation can be performed quite efficiently due to the simplicity and numerical robustness of $\mathbf{S}$. (Fig. 38 illustrates the varying support of modifications at different levels). The major drawback of this procedure is the fact, that edits are restricted to vertices residing on a specific level. However, one can fake low–frequency modifications by moving a group of vertices from a finer level simultaneously. But besides being cumbersome, this annihilates the mathematical elegance of the multiresolution representation.

In order to apply global and smooth modifications to **arbitrary (manifold) triangle meshes** we make use of a completely different approach. In Sec. 5.2 we introduced the notion of discrete fairing that provides elegant means for our purposes. The key idea is relatively simple and can roughly be stated as follows.

> Define the edit by imposing boundary conditions to the mesh, choose an appropriate fairing scheme and solve the corresponding optimization problem.

Fig. 39 shows a convenient way how boundary conditions can be defined by the user. However, more sophisticated methods can easily be derived. The following sections show, how to apply discrete fairing in the context of interactive modeling.

Since interactivity is crucial, an efficient solver for the chosen fairing scheme has to be available. Linear methods (cf. Sec. 5.2) lead to large sparse linear problems and numerical analysis shows, that straight forward iterative solvers

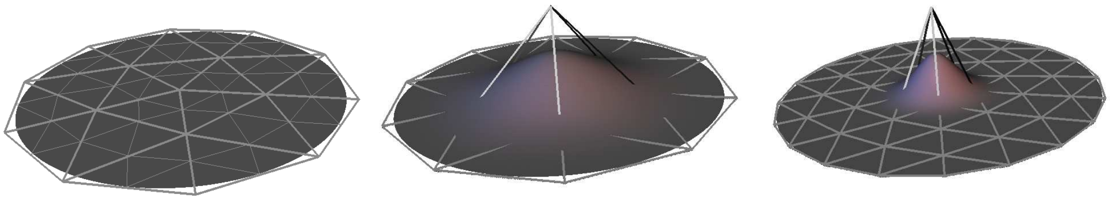


Figure 38: *A simple subdivision–surface (left) is modified by moving the vertices of corresponding control meshes. Editing the coarse control mesh leads to a wide "bump" (middle) whereas altering a vertex on a finer level affects a smaller area (right).*

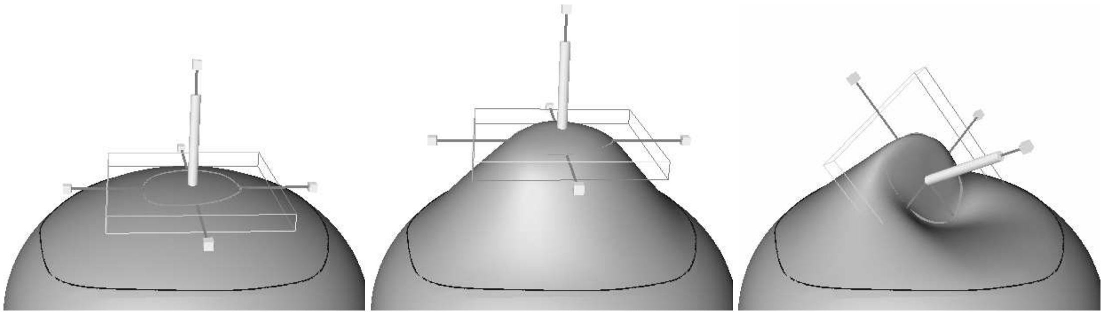


Figure 39: *Freeform edits for unstructured meshes: The dark line indicates the area which is subject to the modification. The bright line defines the handle geometry which can be moved by the designer (middle,right). Both boundaries can have an arbitrary shape and hence they can, e.g. be aligned to geometric features in the mesh. The dark and the bright line impose $\mathcal{C}^1$ and $\mathcal{C}^0$ boundary conditions to the mesh respectively and the modified smooth version is found by discrete fairing while preserving these conditions. Notice, that the designer can apply arbitrary modifications to the handle polygon and she does not have to take the mesh connectivity into account.*

(Jacobi/Gauß–Seidel) are not appropriate in this case. Nevertheless, more sophisticated solvers exploit knowledge about the structure of the problem, e.g. the large class of multi–grid schemes solve the problem hierarchically and thus achieve linear running times in the number of degrees of freedom. These multi–level schemes solve the problem on a coarse level first and use this solution to predict initial values for a solution on the next refinement level. In our case, we can use incremental mesh decimation (cf. Sec. 5.1) to build a fine–to–coarse hierarchy of the mesh in such a way that intermediate meshes can be used to solve the optimization problem (`OP`). This is done in the following way.

```
  go to coarsest level
  solve OP directly
Repeat:
  reinsert some vertices
  solve OP in vicinity of new vertices
Until mesh is reconstructed
```

Discrete fairing adjusts for vertex positions only, which might be insufficient for certain boundary conditions, i.e. extremely large/distorted triangles can occur. Fortunately another degree of freedom inherent to triangle meshes can be exploited. In other words the connectivity can be changed to get rid of badly shaped triangles. This can e.g. be done by prescribing a maximal/minimal edge–length (cf.[4, 5]). Long edges are removed by inserting new vertices in their center (the two triangles adjacent to this edge are split into four subtriangles). Subsequently a simple local optimization procedure balances the vertices' valences thus strongly distorted triangles can be avoided.

6.2. Multiresolution modeling

The previous section shows how to perform freeform modeling on triangle meshes. Let us now assume we want to modify the face of the bust model (see Fig. 41) and we would e.g. like to shift its nose. This could be accomplished with the above methods but the face would lose its features like eyes and mouth since this detail information would be removed by the optimization process. In order to enable such types of edits, we have to extend freeform modeling to *Multiresolution modeling* which means that we have to be able to distinguish between high–frequency detail information that

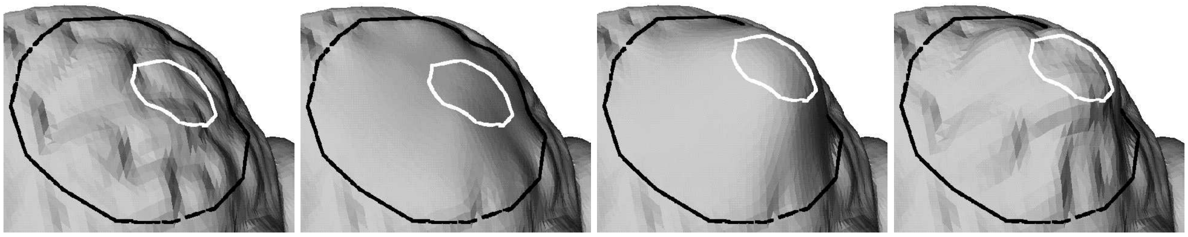


Figure 40: *A flexible metaphor for multiresolution edits. On the left, the original mesh is shown. The smooth version of the original mesh is found by applying discrete fairing while observing the boundary constraints (dark and bright line,cf. Fig. 39). The center left shows the result of the curvature minimization. The geometric difference between the two left meshes is stored as detail information with respect to local frames. Now the designer can move the handle polygon and this changes the boundary constraints for the curvature minimization. Hence the discrete fairing generates a modified smooth mesh (center right). Adding the previously stored detail information yields the final result on the right. Since we can apply fast multi-level smoothing when solving the optimization problem, the modified mesh can be updated with several frames per second during the modeling operation. Notice that all four meshes have the same connectivity.*

has to be preserved and the low–frequency shape we want to edit. This is where multiresolution representations for triangle meshes come in. In the course of this tutorial we already got to know two different ways to build hierarchies (coarse–to–fine and fine–to–coarse). In the context of multiresolution modeling however, we do not want hierarchies of different *coarseness*, i.e. with varying triangle count but of different *smoothness* Nevertheless, it will turn out, that both types of hierarchies are closely related.

Given an arbitrary surface $\mathcal{S}_m$, a multiresolution *decomposition* consists of a sequence of topologically equivalent surfaces $\mathcal{S}_{m-1},\ldots,\mathcal{S}_0$ with decreasing level of geometric detail. The difference $\mathcal{D}_i = \mathcal{S}_{i+1} - \mathcal{S}_i$ between two successive surfaces is the *detail* on level i which is added or removed when switching between the two approximations. The reconstruction $\mathcal{S}_m = \mathcal{S}_i + \mathcal{D}_i + \ldots + \mathcal{D}_{m-1}$ of the original surface $\mathcal{S}_m$ can start on any level of detail $\mathcal{S}_i$. Multiresolution modeling means that on some level of detail, the surface $\mathcal{S}_i$ is replaced by $\mathcal{S}'_i$. This operation does not have any effect on $\mathcal{S}_0,\ldots,\mathcal{S}_{i-1}$ but $\mathcal{D}_{i-1}$ and hence $\mathcal{S}_{i+1},\ldots,\mathcal{S}_m$ change since the (unchanged) detail information $\mathcal{D}_i,\ldots,\mathcal{D}_{m-1}$ is now added to the modified base surface $\mathcal{S}'_i$ for the reconstruction of $\mathcal{S}'_m$. To guarantee the intuitive preservation of the shape characteristics after a modification on some lower level of detail, this basic setting has to be extended in the sense that the detail information $\mathcal{D}_i$ is encoded with respect to *local frames*. These frames are aligned to the surface geometry of $\mathcal{S}_i$ [2, 1, 6, 10]. Sec.6.2.1 will elaborate on this.

For semi–regular meshes based on subdivision the *reconstruction operator* is given by the underlying subdivision scheme. We transform the mesh $\mathcal{S}_i$ to the next refinement level $\mathcal{S}'_{i+1} = \mathbf{S}\mathcal{S}_i$ by applying the stationary subdivision operator $\mathbf{S}$ and move the obtained control vertices by adding the associated detail vectors: $\mathcal{S}_{i+1} = \mathcal{S}'_{i+1} + \mathcal{D}_i$. To generate a smooth low–frequency approximation of $\mathcal{S}_m$, we simply suppress the detail reconstruction starting from some intermediate level j ($\mathcal{D}_i = 0$, i > j). The *decomposition operator* has to be the inverse of the subdivision operator, i.e. given a fine mesh $\mathcal{S}_{i+1}$ we have to find a mesh $\mathcal{S}_i$ such that $\mathcal{S}_{i+1} \approx \mathbf{S}\mathcal{S}_i$. In this case the detail vectors become as small as possible [6].

If we build the hierarchy by using an incremental mesh decimation scheme, the *decomposition operator* **D** applies to arbitrary meshes. Given a fine mesh $\mathcal{S}_{i+1}$ we find $\mathcal{S}_i = \mathbf{D}\mathcal{S}_{i+1}$, e.g. by applying a number of edge collapse operations (cf. Sec. 5.1). However, it is not clear how to define the detail coefficients, since inverse mesh decimation (*progressive meshes*) always reconstructs the original mesh and there is no canonical way to generate smooth low frequency geometry by suppressing the detail information during reconstruction. To solve this problem we split each step of the progressive mesh refinement into a topological operation (vertex insertion) and a geometric operation which places the re–inserted vertex at the original position. In analogy to the plain subdivision operator without detail reconstruction we have to figure out a heuristic which places the new vertices such that they lie on a smooth surface (instead of their original position). This can be done by discrete fairing (see. Sec. 5.2). The difference between this "predicted" position and the original location can then be used as the associated detail vector.

6.2.1. Detail encoding

In order to guarantee intuitive detail preservation under modification of the global shape, we cannot simply store the detail vectors with respect to a global coordinate system but we have to define them with respect to local frames which are aligned to the low–frequency geometry. Usually, the associated local frame for each vertex has its origin at the location predicted by the reconstruction operator with suppressed de-

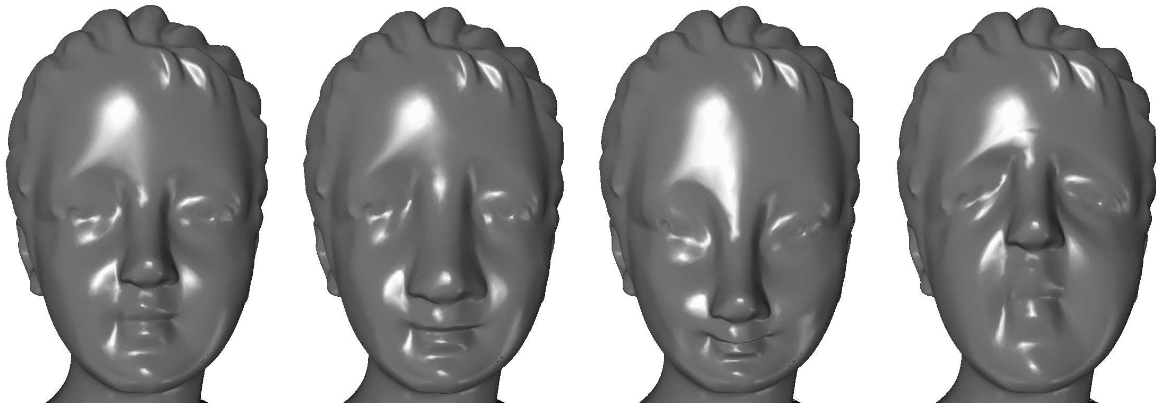


Figure 41: *Multiresolution editing of a bust (62k triangles,left). The handle lies around the nose, the whole face is marked as the area of influence. The edits where achieved by scaling and translating the nose (from left to right).*

tail. However, in many cases this can lead to rather long detail vectors with a significant component within the local tangent plane. Since we prefer short detail vectors to guarantee intuitive reconstructions, we use a different origin for the local frame. In fact, the optimal choice is to find that point on the low–frequency surface, whose normal points directly to the original vertex. In this case the detail is not given in (x,y,z)–coordinates but rather by the base point $\mathbf{p} = \mathbf{p}(u, v)$ on the low–frequency geometry plus a scalar value h for the offset in normal direction.

The general setting for detail computation is that we have given two meshes $\mathcal{S}_{i+1}$ and $\mathcal{S}'_{i+1}$ where $\mathcal{S}_{i+1}$ is the original data while $\mathcal{S}'_{i+1}$ is reconstructed from the low–frequency approximation $\mathcal{S}_i$ with suppressed detail, i.e. for coarse–to–fine hierarchies, the mesh $\mathcal{S}'_{i+1}$ is generated by applying a stationary subdivision scheme and for fine–to–coarse hierarchies $\mathcal{S}'_{i+1}$ is optimal with respect to some global bending energy functional. Encoding the difference between both meshes requires us to associate each vertex $\mathbf{p}$ of $\mathcal{S}_{i+1}$ with a corresponding base point $\mathbf{q}$ on $\mathcal{S}'_{i+1}$ such that the difference vector $\mathbf{p} - \mathbf{q}$ is parallel to the normal vector at $\mathbf{q}$. Any point $\mathbf{q}$ on $\mathcal{S}'_{i+1}$ can be specified by a triangle index i and barycentric coordinates within the referred triangle.

To actually compute the detail coefficients, we have to define a normal field on the mesh $\mathcal{S}'_{i+1}$. The most simple way to do this is to use the normal vectors of the triangular faces for the definition of a piecewise constant normal field. However, since the orthogonal prisms spanned by a triangle mesh do not completely cover the vicinity of the mesh, we have to accept negative barycentric coordinates, if $\mathcal{S}'_{i+1}$ is not sufficiently smooth. This may lead to "unnatural" detail reconstruction if the low–frequency geometry is modified.

We therefore propose a different approach[7] where the normal vectors are estimated at the vertices and a continuous normal field for the interior of the triangles is computed by linearly blending the normal vectors at the corner.

6.3. Modeling tools based on triangle meshes

6.3.1. Semi–regular meshes

In '97 Zorin et al. came up with a modeling tool for semi–regular meshes[10]. It implements the control mechanism we described in the context of freeform modeling for semi–regular meshes and it uses a decomposition operator similar to the one we sketched in Section 6.2.

+ intuitive modifications (detail preserving large scale edits)
+ fast and stable due to simple analysis/synthesis operators
– restricted to semi–regular input meshes
– topological hierarchy pinpoints degrees of freedom for edits on different scales.

6.3.2. Unstructured meshes

Kobbelt and co–workers[6] generalized multiresolution techniques to arbitrary triangle meshes. They introduced the fine–to–coarse geometric hierarchy by using the simple *Umbrella* algorithm (cf. Equation 7 in Sec. 5.2) to generate the low–frequency version of the input mesh. In[7] they extend the two stage hierarchy to multiple geometric hierarchies. A similar approach has been investigated by Guskov[3].

+ intuitive modifications (detail preserving large scale edits)
+ unstructured input meshes
+ flexible modeling metaphor (arbitrary area/handle)
– fixed mesh–connectivity

6.3.3. Dynamic vertex connectivity

In [5] Kobbelt et al. introduced the notion of multiresolution modeling for meshes with dynamic connectivity. They no longer require a global hierarchical structure that links the different detail levels, but they represent the detail information implicitly by the difference between independent surfaces. Since describing the scheme in detail would go beyond the scope of this tutorial, we refer to the original paper in the proceedings.

+ unstructured input meshes
+ adaptive connectivity
– no flexible/intuitive modeling metaphor (modeling with ellipsoids)

References

1. D. Forsey and R. H. Bartels. Surface fitting with hierarchical splines. *ACM Transactions on Graphics*, 14(2):134–161, 1995.

2. D. R. Forsey and R. H. Bartels. Hierarchical B–spline refinement. In *Computer Graphics (SIGGRAPH 88 Proceedings)*, volume 22, pages 205–212, 1988.

3. I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *SIGGRAPH 99 Conference Proceedings*, pages 325–334, 1999.

4. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *SIGGRAPH 93 Conference Proceedings)*, pages 19–26, 1993.

5. L. Kobbelt, T. Bareuther, and H.-P. Seidel. Multiresolution shape deformations for meshes with dynamic connectivity. In *Computer Graphics Forum (Proc. Eurographics 2000)*, to appear.

6. L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi–resolution modeling on arbitrary meshes. In *SIGGRAPH 98 Conference Proceedings*, pages 105–114, 1998.

7. L. Kobbelt, J. Vorsatz, and H.-P. Seidel. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry: Theory and Applications*, 14, 1999.

8. V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH 96 Conference Proceedings*, pages 313–324, 1996.

9. S. Lee. Interactive multiresolution editing of arbitrary meshes. *Computer Graphics Forum*, 18(3):73–82, 1999.

10. D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *SIGGRAPH 97 Conference Proceedings*, pages 259–268, 1997.

I N F O R M A T I K

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Anja Becker
Stuhlsatzenhausweg 85
66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

MPI-I-2000-4-002	L.P. Kobbelt, S. Bischoff, K. Kähler, R. Schneider, M. Botsch, C. Rössl, J. Vorsatz	Geometric Modeling Based on Polygonal Meshes
MPI-I-2000-4-001	J. Kautz, W. Heidrich, K. Daubert	Bump Map Shadows for OpenGL Rendering
MPI-I-2000-2-001	F. Eisenbrand	Short Vectors of Planar Lattices Via Continued Fractions
MPI-I-2000-1-003	P. Fatourou	Low-Contention Depth-First Scheduling of Parallel Computations with Synchronization Variables
MPI-I-2000-1-002	R. Beier, J. Sibeyn	A Powerful Heuristic for Telephone Gossiping
MPI-I-2000-1-001	E. Althaus, O. Kohlbacher, H. Lenhof, P. Müller	A branch and cut algorithm for the optimal solution of the side-chain placement problem
MPI-I-1999-4-001	J. Haber, H. Seidel	A Framework for Evaluating the Quality of Lossy Image Compression
MPI-I-1999-3-005	T.A. Henzinger, J. Raskin, P. Schobbens	Axioms for Real-Time Logics
MPI-I-1999-3-004	J. Raskin, P. Schobbens	Proving a conjecture of Andreka on temporal logic
MPI-I-1999-3-003	T.A. Henzinger, J. Raskin, P. Schobbens	Fully Decidable Logics, Automata and Classical Theories for Defining Regular Real-Time Languages
MPI-I-1999-3-002	J. Raskin, P. Schobbens	The Logic of Event Clocks
MPI-I-1999-3-001	S. Vorobyov	New Lower Bounds for the Expressiveness and the Higher-Order Matching Problem in the Simply Typed Lambda Calculus
MPI-I-1999-2-008	A. Bockmayr, F. Eisenbrand	Cutting Planes and the Elementary Closure in Fixed Dimension
MPI-I-1999-2-007	G. Delzanno, J. Raskin	Symbolic Representation of Upward-closed Sets
MPI-I-1999-2-006	A. Nonnengart	A Deductive Model Checking Approach for Hybrid Systems
MPI-I-1999-2-005	J. Wu	Symmetries in Logic Programs
MPI-I-1999-2-004	V. Cortier, H. Ganzinger, F. Jacquemard, M. Veanes	Decidable fragments of simultaneous rigid reachability
MPI-I-1999-2-003	U. Waldmann	Cancellative Superposition Decides the Theory of Divisible Torsion-Free Abelian Groups
MPI-I-1999-2-001	W. Charatonik	Automata on DAG Representations of Finite Trees

MPI-I-1999-1-007	C. Burnikel, K. Mehlhorn, M. Seel	A simple way to recognize a correct Voronoi diagram of line segments
MPI-I-1999-1-006	M. Nissen	Integration of Graph Iterators into LEDA
MPI-I-1999-1-005	J.F. Sibeyn	Ultimate Parallel List Ranking ?
MPI-I-1999-1-004	M. Nissen, K. Weihe	How generic language extensions enable “open-world” desing in Java
MPI-I-1999-1-003	P. Sanders, S. Egner, J. Korst	Fast Concurrent Access to Parallel Disks
MPI-I-1999-1-002	N.P. Boghossian, O. Kohlbacher, H.-. Lenhof	BALL: Biochemical Algorithms Library
MPI-I-1999-1-001	A. Crauser, P. Ferragina	A Theoretical and Experimental Study on the Construction of Suffix Arrays in External Memory
MPI-I-98-2-018	F. Eisenbrand	A Note on the Membership Problem for the First Elementary Closure of a Polyhedron
MPI-I-98-2-017	M. Tzakova, P. Blackburn	Hybridizing Concept Languages
MPI-I-98-2-014	Y. Gurevich, M. Veanes	Partisan Corroboration, and Shifted Pairing
MPI-I-98-2-013	H. Ganzinger, F. Jacquemard, M. Veanes	Rigid Reachability
MPI-I-98-2-012	G. Delzanno, A. Podelski	Model Checking Infinite-state Systems in CLP
MPI-I-98-2-011	A. Degtyarev, A. Voronkov	Equality Reasoning in Sequent-Based Calculi
MPI-I-98-2-010	S. Ramangalahy	Strategies for Conformance Testing
MPI-I-98-2-009	S. Vorobyov	The Undecidability of the First-Order Theories of One Step Rewriting in Linear Canonical Systems
MPI-I-98-2-008	S. Vorobyov	AE-Equational theory of context unification is Co-RE-Hard
MPI-I-98-2-007	S. Vorobyov	The Most Nonelementary Theory (A Direct Lower Bound Proof)
MPI-I-98-2-006	P. Blackburn, M. Tzakova	Hybrid Languages and Temporal Logic
MPI-I-98-2-005	M. Veanes	The Relation Between Second-Order Unification and Simultaneous Rigid *E*-Unification
MPI-I-98-2-004	S. Vorobyov	Satisfiability of Functional+Record Subtype Constraints is NP-Hard
MPI-I-98-2-003	R.A. Schmidt	E-Unification for Subsystems of S4
MPI-I-98-2-002	F. Jacquemard, C. Meyer, C. Weidenbach	Unification in Extensions of Shallow Equational Theories
MPI-I-98-1-031	G.W. Klau, P. Mutzel	Optimal Compaction of Orthogonal Grid Drawings
MPI-I-98-1-030	H. Brönniman, L. Kettner, S. Schirra, R. Veltkamp	Applications of the Generic Programming Paradigm in the Design of CGAL
MPI-I-98-1-029	P. Mutzel, R. Weiskircher	Optimizing Over All Combinatorial Embeddings of a Planar Graph
MPI-I-98-1-028	A. Crauser, K. Mehlhorn, E. Althaus, K. Brengel, T. Buchheit, J. Keller, H. Krone, O. Lambert, R. Schulte, S. Thiel, M. Westphal, R. Wirth	On the performance of LEDA-SM
MPI-I-98-1-027	C. Burnikel	Delaunay Graphs by Divide and Conquer
MPI-I-98-1-026	K. Jansen, L. Porkolab	Improved Approximation Schemes for Scheduling Unrelated Parallel Machines
MPI-I-98-1-025	K. Jansen, L. Porkolab	Linear-time Approximation Schemes for Scheduling Malleable Parallel Tasks
MPI-I-98-1-024	S. Burkhardt, A. Crauser, P. Ferragina, H. Lenhof, E. Rivals, M. Vingron	q-gram Based Database Searching Using a Suffix Array (QUASAR)

MPI-I-98-1-023	C. Burnikel	Rational Points on Circles
MPI-I-98-1-022	C. Burnikel, J. Ziegler	Fast Recursive Division
MPI-I-98-1-021	S. Albers, G. Schmidt	Scheduling with Unexpected Machine Breakdowns
MPI-I-98-1-020	C. Rüb	On Wallace's Method for the Generation of Normal Variates
MPI-I-98-1-019		2nd Workshop on Algorithm Engineering WAE '98 - Proceedings
MPI-I-98-1-018	D. Dubhashi, D. Ranjan	On Positive Influence and Negative Dependence