-

Painting gradients: free-form surface design using shading patterns

C.W.A.M. van Overveld

Department of Mathematics and Computing Science
Eindhoven University of Technology
PO Box 513, 5600 MB Eindhoven, The Netherlands
Email: wsinkvo@win.tue.nl

Abstract

An interactive system for designing curved surfaces is proposed which is based on direct manipulation of a single-view shaded image of this surface. Apart from functions like carving in the surface or adding to it, and local smoothing, sharpening or shifting of selected surface regions, the system supports painting local gradient distributions.

An iterative algorithm enforces conservativity of the gradient map at all times, so that the shaded image corresponds everywhere to a possible 3-dimensional surface. This surface can be output either as a gradient map, to be used e.g. in bump mapping, or as an adaptively tiled triangular mesh.

Keywords: surface design, shade-to-shape transform

1. Introduction

Designing curved surfaces in 3-d space has long been a major application of Computer Graphics and direct manipulation interaction techniques. In order to input such surfaces, techniques can be distinguished to be either 0, 1, or 2-dimensional.

In CAD practice, 0-dimensional input seems to be the most often used: indeed, a multitude of surface representation schemes ([Boehm 84]), based on control points has been developed since the introduction of Bezier curves in the early 60-s (see for instance [Catmull 78]).

Input techniques based on 1-dimensional data became available with the Coons and Gordon patches ([Boehm 84]), where boundary curves provide a rich visual cue and design aid as to the local shape of the surface. Although in present day CAD-practice, modern versions of Coons patches do occur [Kuriyama 94] they seem to be less widely used than control points-based techniques.

The possibility of genuine two-dimensional input techniques has been explored occasionally in the Computer Graphics community ([L.Williams 90]). Interestingly, in non-computer practice, the popularity of 0, 1, and 2 dimensional input techniques appears to be exactly in opposite order. Since the dramatic improvement of the technique of realistic painting in the 12th and 13th century, artists succeeded to create convincing and plausible visual impressions of a variety of curved surfaces. They essentially reproduce the light and shade distributions that reflect from these surfaces, thus communicating their geometric shape (of course, correct rendering of cast shadows, as well as more subtle artistic techniques, adds to the realism of paintings containing curved surfaces).

A painting, when regarded as a specification of the geometry of the painted object, classifies in our category of 2-dimensional input techniques. Drawing and sketching outlines, contours and structural details such as hatching is the non-computer based equivalent to 1-dimensional input. The equivalent of 0-dimensional (i.e. control point-based) surface specification seems to be rare in non-computer graphics contexts, although some artists use distributions of dots to convey the impression of curved surfaces.

The observed mismatch between the popularity of 0, 1, and 2-dimensional techniques in CAD-practice and non-CAD practice, respectively, might arouse some mistrust as to the appropriateness of the user interfaces of current free-form surface design tools. This paper aims at provoking a discussion, (a) as to what alternatives might provide for a truly intuitive input method for 3-dimensional curved surfaces, and (b) as to what extend paradigms from image processing and pattern recognition might be integrated into 3-dimensional CAGD practice. To this objective, a system is described in section 3, based on representing a surface as a depth-map.

The system, called GRADED (from GRADient EDiting), described in this paper is both inspired by traditional pixel-paint systems and image processing systems (to be discussed in section 4) and by a technique in pattern recognition (the so called shape-from-shading problem; cf. [Horn 70], [Horn 90]). Technical aspects of the latter are discussed in section 5.

Converting the curved surface from its depth-map representation to an adaptively tessellated wire frame model and vice versa is the topic of section 6. Section 7 shows some results by means of a sample test case, *viz.* free-hand modelling the details of a facial mask starting from a coarse and rather featureless polygonal model. Also we discuss the differences between our approach and earlier related work, but we can only do so after having explained the technical aspects of GRADED; hence the overview of earlier work is postponed until section 8.

But first, section 2 lists the requirements that the technique should meet.

2. Requirements

Under normal circumstances, humans have no difficulty in mentally reconstructing the geometric shape from merely looking at, say, three white eggs in a white bowl, even though the perceived image only contains shades of gray. Moreover, with little training, even amateurs are capable of producing a very similar visual impression with charcoal on paper in few minutes. Apparently, shaded images are natural sources for visual communication of geometric features, both in passive tasks (understanding an image) and active tasks (drawing, i.e., defining a shape). This means that an intuitive user interface for surface design should:

- 1. provide visual feedback of the surface under construction in the form of shaded images;
- 2. allow modification of the surface under construction via modification of its shaded image.

Traditionally, sculptors and ceramists have developed techniques for shaping surfaces by carving or (in the case of ceramists) adding material. Their progress is controlled by visual inspection, both of the pattern of shading when viewing the surface under construction from one direction, or of the shape of silhouette contours by looking from several directions. To match with these intuitive notions, the user interface should also:

- 3. allow addition and removal of 'material', i.e. locally decreasing or increasing the proximity of the surface to the viewer, simulating a variety of 'tool shapes' (i.e. a variety of height-distributions to be used for these editing operations);
- 4. allow for 3-dimensional rotation of the surface under construction for inspection purposes.

Finally, with the advent of intuitive image processing techniques in the realm of 2-d images (scaling, smoothing, sharpening, warping, et cetera) a variety of (local) transformations can be applied to images. If these transformations are applied to 2-dimensional images of 3-dimensional objects, they can be interpreted as modifications of the displayed 3-dimensional objects instead of merely their the 2-dimensional projections. For instance, an apparent local reduction of the contrast in a shading pattern may result from smoothing the associated area in the surface. Therefore, applying such a contrast reduction operation to the 2-dimensional image should indicate a smoothing of the underlying surface. Since these local image transformations are highly intuitive, an intuitive user interface for 3-dimensional surface design therefore should also:

- 5. support the most common image processing operations on the shaded images, with a corresponding modification of the underlying surface.

Although a full implementation of the above requirements for truly arbitrary surfaces forms our ultimate goal, we start in this paper by studying the feasibility of this approach by restricting the class of modelled surfaces to partial functions $z = Z(x, y)$ where x and y denote the coordinates parallel to the screen, say normalized from 0...1, and z is the local proximity to the viewer. We have to allow Z to be a partial function on $(0...1) \times (0...1)$ since the modelled surface may have an arbitrarily shaped boundary and it may contain holes. So part of the domain is not mapped to the surface.

This class is sufficiently large to contain complicated surfaces such as facial masks and bas-reliefs, as well as all separable casting-molds, and it matches with the class of surfaces usually studied in the context of the shade-to-shape problem. This restriction implies that requirement 4 only involves rotation of the surface for inspection, not for modification. The extension to arbitrary surfaces (e.g. surfaces with overhangs) is left for future work.

Further, requirement 5 asks for a connection with image processing techniques, and requirement 2 will rely on some notions from pattern recognition.

3. Free hand modification of depth-maps

A surface of the form $z = Z(x, y)$ is conveniently represented in an $M \times M$-sized depth map $D(X, Y)$ where $0 \leq X < M$ and $0 \leq Y < M$. The type of D contains a boolean b and a real value z. The relation between z and D is defined as follows:

$$D(X,Y).z = \begin{cases} Z(\frac{X}{M}, \frac{Y}{M}) & \equiv D(X,Y).b = \textbf{TRUE} \\ & (\frac{X}{M}, \frac{Y}{M}) \in \text{domain of } z \\ & \\ \text{undefined} & \equiv D(X,Y).b = \textbf{FALSE} \\ & \text{otherwise} \end{cases}$$

In the presentation of algorithmic aspects in the rest of this paper (with the exception of section 6) we ignore the fact that $Z(x, y)$ is a partial function, as well as the limited domain of $D(X, Y)$ in order to arrive at a simpler notation (in the implementation of GRADED, these aspects are of course fully taken care of). Therefore, in the remainder of the paper we abbreviate $D(X,Y).z$ by $D(X,Y)$.

Given $D(X,Y)$, a shaded image is obtained by orthogonal projection, where pixel (X,Y) receives a color $c = C(n)$ depending on the normal vector $n = N(X,Y)$ in the point $(X, Y, D(X,Y))$. This normal vector in turn depends on the local gradient $\rho(X,Y)$ of the surface. The n and ρ are defined as:

$$\rho_x(X,Y) = D(X+1,Y) - D(X,Y), \tag{1}$$

$$\rho_y(X,Y) = D(X,Y+1) - D(X,Y), \tag{2}$$

$$N_x(X,Y) = \rho_x(X,Y)/m, \tag{3}$$

$$N_y(X,Y) = \rho_y(X,Y)/m, \tag{4}$$

$$N_z(X,Y) = 1/m, \tag{5}$$

where

$$m = \sqrt{(}1 + \rho_x(X,Y)^2 + \rho_y(X,Y)^2). \tag{6}$$

As with the z values, the ρ values are kept in a buffer indexed by X, Y; this will be called the *gradient buffer*.

For the color function $c = C(n)$, any local illumination light model may be used. Here, 'local' means that the function c is a function of the local normal vector n of the surface only; that is: cast shadows, color bleeding or distance attenuation effects cannot be accommodated. We note in passing that in many approaches to the shade-to-shape problem, a strictly Lambertian surface is needed, as well as a single point shaped light source with an exactly known direction. We won't need any of these assumptions in GRADED.

In section 5 we will discuss how painting in GRADED may result in modifying the gradient values in the gradient buffer.

4. Operations from image processing

In the GRADED system, two buffers are kept: the depth buffer contains the $D(X,Y)$ values and the gradient buffer contains the $\rho(X,Y)$ values that are derived from the depth buffer. The contents of the depth buffer may change arbitrarily, since with every depth distribution there corresponds a gradient distribution. The converse is not true: as we will see in section 5, the gradient buffer is only allowed to represent gradient distributions that meet the so called *conservativity* constraint. Section 5 discusses editing operations that operate on the gradient buffer.

sec	depth buffer	gradient buffer	rendered view (=shading pattern)
4	manipulated by the user	derived from depth buffer	computed from gradient buffer
5	derived from the gradient buffer	derived from shading pattern	manipulated directly by the user

In the current section we discuss the manipulations that operate directly on the contents of the depth buffer. The effect of these operations are visible due to the changes in gradient buffer they cause, since the shading pattern is derived from the normal vectors and hence from the gradient buffer via a local illumination model.

Most current pixel-paint systems support operations for direct manipulation of the color of a selected region of pixels, such as tinting and opaque painting. Tinting is an operation which gradually increases or decreases the red, green and/or blue components of the color of a set of pixels in a region. Tinting and opaque painting are local in the sense that they affect the color of a pixel only in dependence of the color of the current brush (opaque painting) or in dependence of the color of the current brush in combination with the previous color of that pixel (tinting).

The *shape* of the brush defines the shape of the affected region, and brushes may have a *profile* which specifies a variation of the effect of the brush (such as the density of the tint) over the affected region. The shape can be viewed as a connected area of pixels; the profile is a real-valued function defined on these pixels with values from 0 to 1.

'Shape' and 'profile' are attributes that apply to brushes, as well as to all image processing tools having a limited geometric extent.

Tinting (in the case of pixel painting) generalizes straightforwardly in GRADED to an operator that increases or decreases the local height of the surface, $D(X,Y)$ for (X,Y) in an affected region. These operators are the *add* and *carve* operators, respectively. For their shape and profile we implemented a variety of shapes (rectangular block, (truncated) pyramid, Gaussian, ellipsoid, cone, rhomboid). They are parameterized by the size of the shape, the gradient and the eccentricity.

To set the size, orientation and eccentricity parameters interactively, the control widget in the user interface to specify the tool shape has a circular shape; the polar coordinates r and ϕ of a selected point within this circular area define both an orientation (ϕ) and an eccentricity (derived from r). See figure 1. A separate conventional linear slider is used to scale the size of the brush shape. Examples of the effect of the add-operator with several brush shapes are shown in the top row of figure 5. This figure shows the user interface of the GRADED system. Apart from the painting canvas (on the right) and the circular tool selection widget, a low-resolution wire frame is displayed in the lower left hand corner of the window. This wire frame rendering can be rotated in 3 directions interactively and it displays in real-time (albeit at a low resolution) the shape of the edited surface.

Apart from the local operators tinting and opaque painting in pixel paint systems, there are operators which are non-local. The new color at pixel (X, Y) then not only depends on the previous color in (X, Y) but also on the previous colors in other pixels, e.g. the neighbors of (X, Y). Similar as with colors, such operators can be defined for $z-$values in the depth buffer. We implemented the following operators:

- smooth: $D(X,Y)_{new} = (1-\alpha)D(X,Y) + \alpha z_{average}$ where $z_{average}$ is a weighted average of $D(X,Y)$ and its neighbors. The coefficient α is the tool profile; it varies between 0 and 1 over the affected region of the tool depending on the current tool shape and the definition of the tool profile function. Smoothing has the effect of averaging shape details of the surface under the tool.
- sharpen: $D(X,Y)_{new} = (1-\alpha)D(X,Y) - \alpha z_{average}$. Sharpen is the inverse operator of smooth.
- contract: $D(X,Y)_{new} = (1-\alpha)D(X,Y) - \alpha D(X+\chi, Y+\eta)$; the vector (χ, η) is pointing in the direction $(X - X_0, Y - Y_0)$ where (X_0, Y_0) is the centre of the affected region of the brush. The magnitude of (χ, η) is taken proportional to the local value of the tool's profile function. The contract operator locally shrinks shape features in the tool's affected region. Also the $D(X,Y).b$-attributes are affected; this means that applying contraction near the boundary of the surface also contracts part of the boundary.
- expand: $D(X,Y)_{new} = D(X_0, Y_0)$ if (X, Y) is closer than a given threshold to (X_0, Y_0), the centre of the tool's affected region. Otherwise, it is the contract-operator with a vector $(-\chi, -\eta)$ instead of (χ, η). The expand operator locally widens shape features in the tool's affected region. Also the $D(X,Y).b$-attributes are affected.
- shift: $D(X,Y)_{new} = D(X + \alpha\xi, Y + \alpha\epsilon)$. Again, α emulates the tool profile; the vector (ξ, ϵ) is taken along the main orientation direction of the tool shape. Also the $D(X,Y).b$-attributes are affected; this means that applying a shift operator near the boundary of a surface also shifts part of the boundary.
- cut: $D(X,Y).b =$ **FALSE** for (X, Y) in the tool' s affected region where the profile value exceeds 0.5.
- create: $D(X,Y).b =$ **TRUE** for (X, Y) in the tool's affected region where the profile value exceeds 0.5.

For all non-local operators, the $D(X,Y)_{new}$ has to be computed first for all pixels (X, Y) in the tool's affected region before the old $z-$values may be overwritten by $D(X,Y)$, z_{new} in order to avoid unwanted order-dependencies. The effect of some of the non-local operators is shown in the top right image in figure 5.

5. Painting gradients: a pattern recognition-based technique

Although the image processing-based operators from section 4 add to the intuition of a direct manipulation-based interface for curved surface design, they are not sufficient to comply with requirement 2 of section 3: "allow modification of the surface via modification of its shaded image". A possible work-around could be to implement a full shade-to-shape algorithm (see for instance [K.M.Lee 94]), which takes a user defined shaded image as input for conversion to a depth map. Three observations plead against this option:

- most shade-to-shape algorithm require rather restricted illumination conditions. This means, among other things, that the light source's shining direction be known exactly. The user (=designer) has to comply with these conditions up to an impractically high degree of accuracy. Or stated differently: the user has to paint a very accurate shading pattern in order to get the surface shape that she/he had in mind.
- a full-fledged shade-to-shape algorithm requires a large numerical effort. This may be unacceptable in interactive contexts, where a (nearly) real time response is essential.
- the problem to be solved in surface design is less involved than a full shade-to-shape problem, since an artist who paints a shaded image of a curved surface can do so *because he knows what the local surface gradient has to be*. So the local surface gradient (which is available and can be edited directly in the gradient buffer) is in principle available as input to the shape recovery algorithm.

We can phrase this last observation differently (see also figure 2): given the known direction of the illumination, a painter does an (approximate) mental calculation to convert the known surface gradient into a shade. To this aim, we provide the designer with a method to *directly* input gradients $\rho(X,Y)$ into GRADED. This facilitates the task for the system since it does not have to invert the illumination equation ([K.M.Lee 94]). Still we can give the user the impression that he/she is painting directly with shades, thanks to the one-to-one relation between the gradient ρ, the normal vector n and the associated color $C(n)$. Again making use of the circular input widget we introduced in section 4, this works as follows.

The circular area mentioned before can be interpreted as a 2-dimensional projection of a hemisphere which is placed in the same illumination field $C(n)$ as the surface under design. Now observe that all possible orientations ρ occur somewhere on this hemisphere. This means that the circular area looks like an illuminated hemisphere, displaying all possible colors in the co-domain of $C(N(\rho))$ distributed over its surface. Selecting a point in this area therefore selects a point on the surface of the hemisphere, *and* at the same time a unique orientation (hence also a normal vector), *and* the color to use to paint in the gradient buffer those parts of the surface having the same orientation as the selected part of the hemisphere. Therefore the user may either conceive his selection on the basis of a shade-color or of a surface orientation (normal): in both cases he chooses a consistent (color, orientation)-pair.

Points X, Y where $D(X,Y)$ is undefined are rendered in a color which does not occur in the co-domain of $C(n)$. In the examples in section 7, the illumination function is chosen to consist of the contribution of two differently colored light sources in two different directions; the material is predominately diffuse although a small specular reflection component adds somewhat to the perceived plasticity of the surface.

If the user paints or edits in this manner, directly into the gradient buffer $\rho(X,Y)$, the remaining problem is to convert the resulting gradient distribution to $D(X,Y)$-values such that requirement holds for all (X,Y). Such a conversion, however, is not possible for arbitrary $\rho(X,Y)$ (see also [Brooks 92]).

Indeed, conservativity must be maintained. Conservativity holds that the accumulated displacement in $z-$direction over any closed loop has to sum to zero. See figure 3. A closed loop consists of pixels that are either horizontally or vertically adjacent (in a 4-connected metric). Let a closed loop of N subsequently adjacent pixels be given by $(X_i, Y_i), i = 0 \cdots N-1$ (the points A, B, C, D in 3).

The subsequent $z-$values from the depth buffer are $z_i = D(X_i, Y_i)$, and the subsequent differences as specified from editing the gradient buffer $\rho(X,Y)$ are d_i. This means that, for 4-connected points,

$$d_i = \begin{cases} \rho_x(X_i, Y_i) & \text{if} \quad (X_{i+1}, Y_{i+1}) = (X_i + 1, Y_i) \\ \rho_y(X_i, Y_i) & \text{if} \quad (X_{i+1}, Y_{i+1}) = (X_i, Y_i + 1) \\ -\rho_x(X_i, Y_i) & \text{if} \quad (X_{i+1}, Y_{i+1}) = (X_i - 1, Y_i) \\ -\rho_y(X_i, Y_i) & \text{if} \quad (X_{i+1}, Y_{i+1}) = (X_i, Y_i - 1) \end{cases}$$

Conservativity requires that

$$\sum_{i=0}^{N-1} d_i = 0.$$

Now, suppose the user desires to alter the orientation of the surface. He edits a set of ρ values, as indicated by the arrows in the upper left diagram of figure 3. A non-conservative loop may result, in which case the user has painted a gradient distribution that does not correspond to any possible $z-$function. So the gradient components that correspond with the d_i for these loops must be adjusted. Since we assume no a priori knowledge about which gradients should be adjusted most (in other words, all gradients are assumed to be equally likely), an equal amount of adjustment should be applied to all d_i, say

$$d'_i = d_i - \frac{1}{N} \sum_{i=0}^{N-1} d_i.$$

In the upper right diagram in figure 3 these corrections are indicated by $\Delta = -\frac{1}{N} \sum_{i=0}^{N-1} d_i$. Now conservativity holds for the corrected gradient components:

$$\sum_{i=0}^{N-1} d'_i = 0.$$

This is indicated in the lower left diagram in figure 3: the loop A', B', C', D' is closed. It is not clear yet, however, how this adjustment should apply to the $z-$values in the loop, in other words: all $z-$ values in the loop can be offset by an arbitrary constant without affecting the loop's conservativity. Let z_i be adjusted with an amount $d_{z;i}$. Then we can demand

$$\frac{1}{2} \sum_{i=0}^{N-1} {d_{z;i}}^2$$

to be minimal under the boundary condition that

$$\begin{aligned} d'_i &= z'_{i+1} - z'_i \\ &= z_{i+1} + d_{z;i+1} - z_i - d_{z;i} \end{aligned} \quad (7)$$

for $i = 0 \cdots N-1$. The latter set of boundary conditions is dependent, however, owing to the loop being closed. An independent set of boundary conditions is obtained by having i run from 0 to $N-2$. We can solve for the $d_{z;i}$ using the Lagrange multiplier technique. This amounts to adding terms $\lambda_i(z_{i+1} + d_{z;i+1} - z_i - d_{z;i} - d'_i)$ to

$\frac{1}{2}\sum d_{z;i}{}^2$ for $i = 0...N-2$. The λ's introduced here are the Lagrange multipliers that can be solved for later (although it turns out that we don't actually need their values). In vector notation, this amounts to finding a stationary value of

$$\frac{1}{2}(\mathbf{d}_z, \mathbf{d}_z) + (\lambda, \mathbf{S}(\mathbf{z} + \mathbf{d}_z) - \mathbf{d'}).$$

Here $\mathbf{d}_z$ is the vector of N components $d_{z;i}$; λ is the vector of $N-1$ Lagrange multipliers, λ_i; $\mathbf{z}$ is the vector of N components z_i; **d'** is the vector of N components d'_i. The $N-1 \times N$ matrix **S** is defined by $\mathbf{S}_{i,i} = -1$ and $\mathbf{S}_{i,i+1} = 1$ for $0 \le i < N-1$; all other elements are 0.

The minimization problem gives rise to the $2N-1$ sets of linear equations in the unknowns $\mathbf{d}_z$ and λ:

$$\begin{pmatrix} \mathbf{I} & \mathbf{S}^T \\ \mathbf{S} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{d}_z \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{d'} - \mathbf{Sz} \end{pmatrix}$$

The coefficient matrix in the left hand part is symmetric (**I** is the $N \times N$ identity matrix) and has a 2×2 block structure. It can be inverted in closed form for arbitrary N. For instance, the inverse for $N = 6$ reads:

$$\frac{1}{8}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & -5 & -4 & -3 & -2 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & -4 & -3 & -2 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & -3 & -2 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 3 & -2 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 3 & 4 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 3 & 4 & 5 \\ -5 & 1 & 1 & 1 & 1 & 1 & -5 & -4 & -3 & -2 & -1 \\ -4 & -4 & 2 & 2 & 2 & 2 & -4 & -8 & -6 & -4 & -2 \\ -3 & -3 & -3 & 3 & 3 & 3 & -3 & -6 & -9 & -6 & -3 \\ -2 & -2 & -2 & -2 & 4 & 4 & -2 & -4 & -6 & -8 & -4 \\ -1 & -1 & -1 & -1 & -1 & 5 & -1 & -2 & -3 & -4 & -5 \end{pmatrix},$$

which nicely shows the generic structure of the inverse matrices (note that the entries in the lower right block are minus the products of the row and column indices in that block). With this matrix, the $z-$adjustments for the considered loop, $d_{z;i}$, represented by the vector $\mathbf{d}_z$, are obtained by merely multiplying the lower right block with the vector $\mathbf{d'} - \mathbf{Sz}$. The lower right diagram in figure 3 shows the final adjustment: the z-values in the loop have been shifted by an amount δ which results from the minimisation procedure outlined above.

It does not follow from the conservativity constraint, however, what adjustments should apply to the gradients $\rho(X,Y)$ and the $z-$values $D(X,Y)$ for pixels (X,Y) that are *within* the loop; moreover, since conservativity has to hold for the *entire* domain (X,Y), the above adjustment should occur for a set of loops which is in some sense 'complete'. Two strategies have been implemented:

- Cover the entire domain with loops of size $N = 4$, consisting of (X,Y), $(X+1,Y)$, $(X+1,Y+1)$, and $(X,Y+1)$ for all (X,Y). These minimal loops have no interior pixels, so adjusting merely the gradients and the $z-$values on the loops proper enforces conservativity.
- Cover the entire domain with loops of size $N = 8$, consisting of (X,Y), $(X+1,Y)$, $(X+2,Y)$, $(X+2,Y+1)$, $(X+2,Y+2)$, $(X+1,Y+2)$, $(X,Y+2)$, and $(X,Y+1)$ for X and Y even. These loops have one interior pixel, namely $(X+1,Y+1)$ for X and Y even.

  The adjustment for the $z-$value in this interior pixel and those components of ρ that do not affect conservativity of the loop as a whole is computed using a similar minimization principle, demanding conservativity for the four square sub-loops of size $N = 4$ (note that enforcing conservativity for any three of them automatically yields a conservative fourth loop).

For both strategies we have to cope with the fact that most pixels are shared by several (up to four) loops. Therefore an iterative procedure has been implemented where first the adjustments are calculated and accumulated for all loops separately but not yet applied to the contents of the depth-buffer and the $\rho-$buffer. Only after all loops have been processed, the accumulated values of all adjustments are applied. This approach avoids any unwanted order dependencies. In pseudo-code, this reads:

```
do {
  for all pixels (x,y)  set d_z(x,y):=0;
  for all loops {
     compute corrections c_i for the z-values
     of the pixels in that loop;
     set d_z(x,y):=d_z(x,y)+c_i;
     } /* this does not affect the pixel's z-values yet */
  for all pixels (x,y)  set D(x,y):=D(x,y)+d_z(x,y);
  for all pixels (x,y)  derive rho(x,y) from D(x,y);
} until convergence /* the corrections are
                     * sufficiently small */
```

One iteration of the $N = 4$ algorithm tends to result in corrections in the direct neighborhood of the painted regions whereas the $N = 8$ algorithm affects a somewhat more extended region. On the other hand, the $N = 4$ algorithm requires somewhat more iterations until convergence, which spreads out the affected region. When converged, the result of both algorithms is very similar; also, since computing the correction of a $N = 8$ loop is more expensive than a $N = 4$-loop, the total computational effort of both methods is about the same.

6. Converting depth-maps to wire frames and back

Although useful applications can be thought of for a surface $z = Z(x,y)$ in the form of a discrete depth map

$D(X, Y)$, such as offset mapping or merging of depth buffers in the context of a Z-buffer hidden surface algorithm, most applications rely on an object space representation instead.

A naive way to convert the $D(X, Y)$ map into an object space representation is to produce a wire frame mesh with a regular triangular topology with one vertex for every pixel (X, Y) within the domain of D, and next reducing the number of triangles, e.g. using the algorithm of ([Turk 92]).

As an alternative, we describe an algorithm based on an adaptive tessellation of the domain (X, Y) within the domain of D. It is a three pass algorithm which initially triangulates the $M \times M$ domain with two triangles (one with vertex coordinates $(0, 0, f_z(0, 0))$, $(M - 1, 0, f_z(M - 1, 0))$, $(M - 1, M - 1, f_z(M - 1, M - 1))$ and the other one with $(0, 0, f_z(0, 0))$, $(0, M - 1, f_z(0, M - 1))$, $(M - 1, M - 1, f_z(M - 1, M - 1))$; the function f_z returns the values $D(X, Y)$) even for X and Y on non-grid locations, using bi-linear interpolation on the depth-buffer.

The algorithm is parameterized with L_{max}, the maximal allowed edge length; L_{min}, the minimal allowed edge length, and E which is the maximal relative $z-$difference between the midpoint of any edge and the underlying surface.

Schematically, the three passes work as follows (see also figure 4):

Pass 1 serves to get a triangular mesh with a sufficiently high minimal density such that loops 2 and 3 won't miss any shape details:

```
reset all edge labels;
do  {
  loop over all edges e {
    if( |e| > L_max ) label e;
    introduce a split point halfway e;
  }
  loop over all triangles t {
    if t possesses one or more labelled edges,
    split t using the split points;
    update the edge administration;
    reset all edge labels;
  }
} until all edges are shorter than L_max;
```

Pass 2 serves to separate regions in the domain where $Z(x, y)$ is defined from regions that do not belong to the surface. To this aim, a three-valued function $f_b(X, Y)$ returns a value **IN**, **ON** or **OUT** for arbitrary, non-integer X and Y by bi-linearly interpolating the values of $D(X, Y).b$, treating them as real values (**FALSE** $= 0.0$ and **TRUE** $= 1.0$). We apply threshold values of < 0.45 to classify **OUT**, > 0.55 to classify **IN** and between 0.45 and 0.55 to classify **ON**. This function serves to detect if an edge crosses a domain boundary: if f_b is evaluated on the two extremes of an edge and classifies **IN** and **OUT**, respectively, the edge crosses a domain boundary.

```
reset all edge labels;
do {
  loop over all edges e with |e|>L_min {
    if e crosses a domain boundary {
      using binary sectioning on f_b, find an inter-
section point;
      label e;
    }
  }
  loop over all triangles t {
    if t possesses one or more labelled edges,
    split t using the found intersection points;
    update the edge administration;
    reset all edge labels;
  }
} until no edges longer than L_min cross a do-
main boundary;
```

Next, in pass 3 edges are split in regions where a high curvature exists.

```
reset all edge labels;
do  {
  loop over all edges e with |e|>L_min {
    if the z-difference of the midpoint of e and the
    surface underneath is more than |e|/E,
    introduce a split point halfway e;
  }
  loop over all triangles t {
    if t possesses one or more labelled edges,
    split t using the split points;
    update the edge administration;
    reset all edge labels;
  }
} until no edges longer than L_min deviate too
  much from the surface;
```

Finally, the output wire frame mesh consists of all triangles of which no points classify **OUT**.

The inverse conversion, from a wire frame representation to a depth map takes place using a conventional scan conversion algorithm ([Foley 90]) using the depth buffer D to account for hidden surface elimination. The latter conversion is needed when reading an object represented as a wire frame into GRADED.

7. Results

In figure 6, the effect of enforcing conservativity is depicted. In the left part, the topmost shading pattern has been painted directly by the user; the lowermost pattern results from enforcing conservativity according to the algorithm in this paper. Figure 5, lower row, demonstrate a sample session with the GRADED system. We start from a simple facial mask that was modelled using conventional wire frame modelling techniques. It is input in GRADED, and with both local and non-local image processing tools and direct manipulation of the shaded image, much detail is added (especially note the eyes and the lips, the eyebrows, the cheeks and the labio-nasal folds). Finally, the mask is output as an adaptively tessellated wire frame mesh and rendered (see rightmost part in figure 6) from a different view point using a more advanced illumination model. It turns out that sketching complex surfaces that have curved features on relatively large scale, such as a facial mask, from scratch is still a quite elaborate task. On the other hand, adding detail to a surface which has the right global shape is very intuitive: the transformation between the first and last versions of the mask required under 20 minutes. A second example is shown in figure 6. The image in the middel is the result after 2 hours work in a gradient painting session where the author painted, from scratch, a picture of his own right hand. With a 180 X 180 resolution, painting and conservativity reconstruction operate near realtime on a SUN SPARC 10 workstation.

8. Related work

The development of GRADED has been inspired by the earlier work of several authors.

First, manipulating a surface which is represented in terms of a grid of samples can be seen as a special case of editing control-points in tensor product surfaces. In this regard, GRADED can be viewed as an editor for piecewise 1st order B-spline surfaces. An earlier example of editing spline surfaces is ([R.Parent 77]).

In the Thingworld system of Pentland, Essa, Friedman, Horowitz, and Sclaroff ([Pentland 90]) the user has access to a simulated lump of material where the shape can be modified by using 'forces'. Using modal analysis, the user may distinguish between global and local shape operations.

Galyean and Hughes ([Galyean 91]) describe a direct manipulation tool for surface design where the user has direct access to the surface representation which is an implicit surface, represented as voxels. The surface is modified by adding or removing voxels to the inside region of the (closed) surface.

In both the Thingworld system and the Galyean-Hughes system, no restrictions are imposed on the edited surface whereas in GRADED only surfaces of the form $z = Z(x, y)$ may occur. On the other hand, this means that a complete repertoire of image procesing operators is immediately available to express arbitrary shape details in GRADED.

In the 3-D paint system of Williams ([L.Williams 90]), also the connection with traditional image processing operations is made. But unlike GRADED, it uses non-standard video hardware. Also, unlike GRADED, it associates depth-values with a grey-scale distribution. From our experience with GRADED, it turns out that associating surface gradients to color values works quite intuitive, because the displayed colour values approximate what one would see in the 'real world' if the surface were illuminated by coloured light sources. On the other hand, natural illumination conditions don't associate shading values to depth value distribution, and therefore in Williams' system, the user is confronted with a non-standard image of the surface, which may cause a considerable learning time.

Haeberli ([Haeberli 90]) also uses the notion of user input surface gradients, although he uses this information to obtain brush parameters for the shape and orientation of a simulated brush. The resulting image is 2-D.

In the sculpting system of Coquillart ([Coquillart 90]) a truly 3-D relief is modelled, but the user controls the shape via a network of control vertices rather than via direct manipulation of the surface proper.

A system where truly direct manipulation operations to define surface relief are supported has been described by Hanrahan and Haeberli ([Hanrahan 90]). In their system, however, the relief is implemented as bump maps, so only small scale surface modifications can be achieved.

In the field of pattern recognition, much work has been done to solve the shade-to-shape problem ([K.M.Lee 94]). Since we provide orientation rather than shading information, however, in the GRADED system a particularly simple version of this problem occurs, and a relatively straightforward algorithm suffices to provide the near real-time performance needed in an interactive system.

References

[Boehm 84] W. Boehm, G. Farin, and J. Kahman. A survey of Curve and Surface methods in CAGD. *Computer Aided Geometric Design*, 1:1–60, 1984.

Figure 1: Clicking a point in the widget specifies orientation and the eccentricity of the brush shape.

[Brooks 92] M. Brooks, W. Chojnacki, and R. Kozera. Impossible and ambiguous shading patterns. *Int. J. Comput. Vision*, 7(2):119–126, 1992.

[Catmull 78] E. Catmull and J. Clark. Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes. *Computer Aided Design*, pages 350–355, November 1978.

[Coquillart 90] Sabine Coquillart. Extended free-form deformation: a sculpturing tool for 3d geometric modelling. *Computer Graphics (Proc. SIGGRAPH 90)*, 24(4):187–196, August 1990.

[Foley 90] James D. Foley, Andries van Dam, Steven Feiner, and John Hughes. *Computer Graphics Priniciples and Practice*. Addison-Wesley, 1990.

[Galyean 91] Tinsley A. Galyean and John F. Hughes. Sculpting: an interactive volumetric modeling technique. *Computer Graphics (Proc. SIGGRAPH 91)*, 25(4):267–274, August 1991.

[Haeberli 90] Paul Haeberli. Paint by numbers: abstract image representations. *Computer Graphics (Proc. SIGGRAPH 90)*, 24(4):207–214, August 1990.

[Hanrahan 90] Pat Hanrahan and Paul Haeberli. Direct WYSIWYG painting and texturing on 3d shapes. *Computer Graphics (Proc. SIGGRAPH 90)*, 24(4):215–223, August 1990.

[Horn 70] B. Horn. *Shape from shading: a method for obtaining the shape of a smooth opaque object from one view (PhD thesis)*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1970.

[Horn 90] B. Horn. Height and gradient from shading. *Int. J. Comput. Vision*, 5:584–595, 1990.

[K.M.Lee 94] K.M.Lee and C.C.Jay Kuo. Shape from shading with perspective projection. *CVGIP: image understanding*, 59(2):202–212, 1994.

[Kuriyama 94] Shigeru Kuriyama. Surface modelling with an irregular network of curves via sweeping and blending. *Computer Aided Design*, 26(8):597–606, August 1994.

[L.Williams 90] L.Williams. 3d paint. *Computer Graphics (Proc. SIGGRAPH 90)*, 24(2):225–233, March 1990.

[Pentland 90] A. Pentland, I. Essa, M. Friedmann, B. Horowitz, and S. Sclaroff. The Thingworld modeling system: virtual sculpting by modal forces. *Computer Graphics (special issue on 1990 symposium on interactive 3D graphics)*, 24(2):143–144, March 1990.

[R.Parent 77] R.Parent. A system for sculpting 3D data. *ACM Computer Graphics (Proceedings SIGGRAPH 1977)*, 11(2):138–147, July 1977.

[Turk 92] Greg Turk. Re-tiling polygonal surfaces. *Computer Graphics (Proc. SIGGRAPH 92)*, 26(2):55–64, July 1992.

Figure 2: Top: the classic painting technique requires the translation from imaginary surface orientation to reflected light intensity. Bottom: the proposed technique in GRADED allows the designer to bypass this translation. The actions on a shaded background require human mediation.

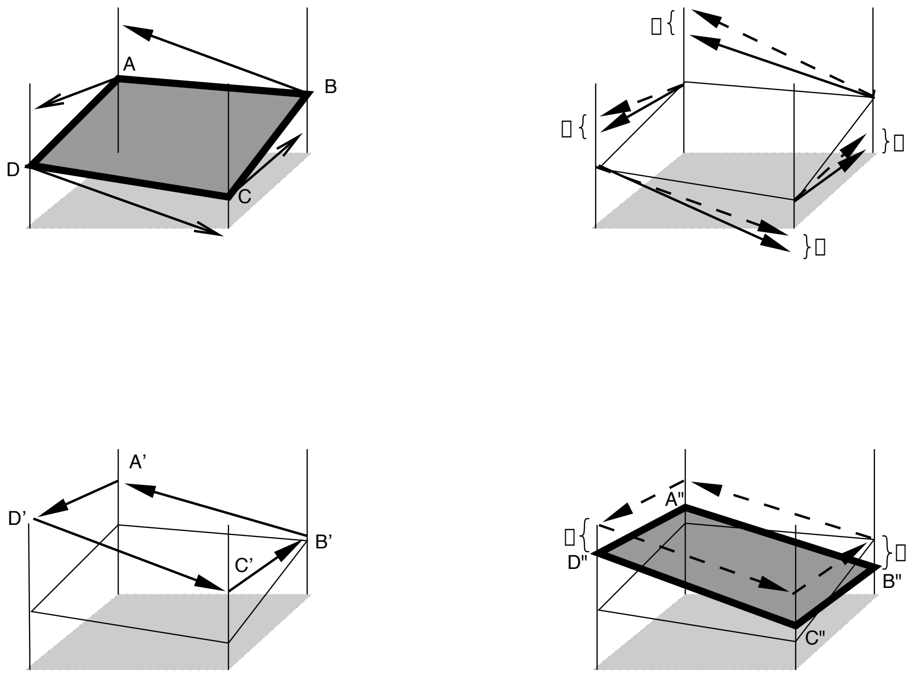

Figure 3: The phases in forcing conservativity for one loop. Upper left: the arrows result from the user's intention to edit the gradient. Upper right: these arrows are corrected. Lower left: now A', B', C', D' is a closed loop. Lower right: a translation in the z-direction causes the new loop to be as close as possible to the original one.

Figure 4: A schematic view of the surface triangulation. The shaded region represents the domain of z. a. Initially, the triangulation consists of two triangles. b. After pass 1, the triangles have been subdivided such that every edge has length at most L_{max}. c. After pass 2, the boundary of the domain of z has been found. The white dots indicate the intersection of previously existing edges with the boundary. These intersections are connected by thick lines. The dashed lines are needed to maintain the triangulation. d. After pass 3, further subdivisions have taken place to adapt to differences in curvature. In this case, two edges have been split; the split points are indicated by white circles and the dashed lines are needed to maintain the triangulation. e. The result consists of all triangles with no points classifying OUT.

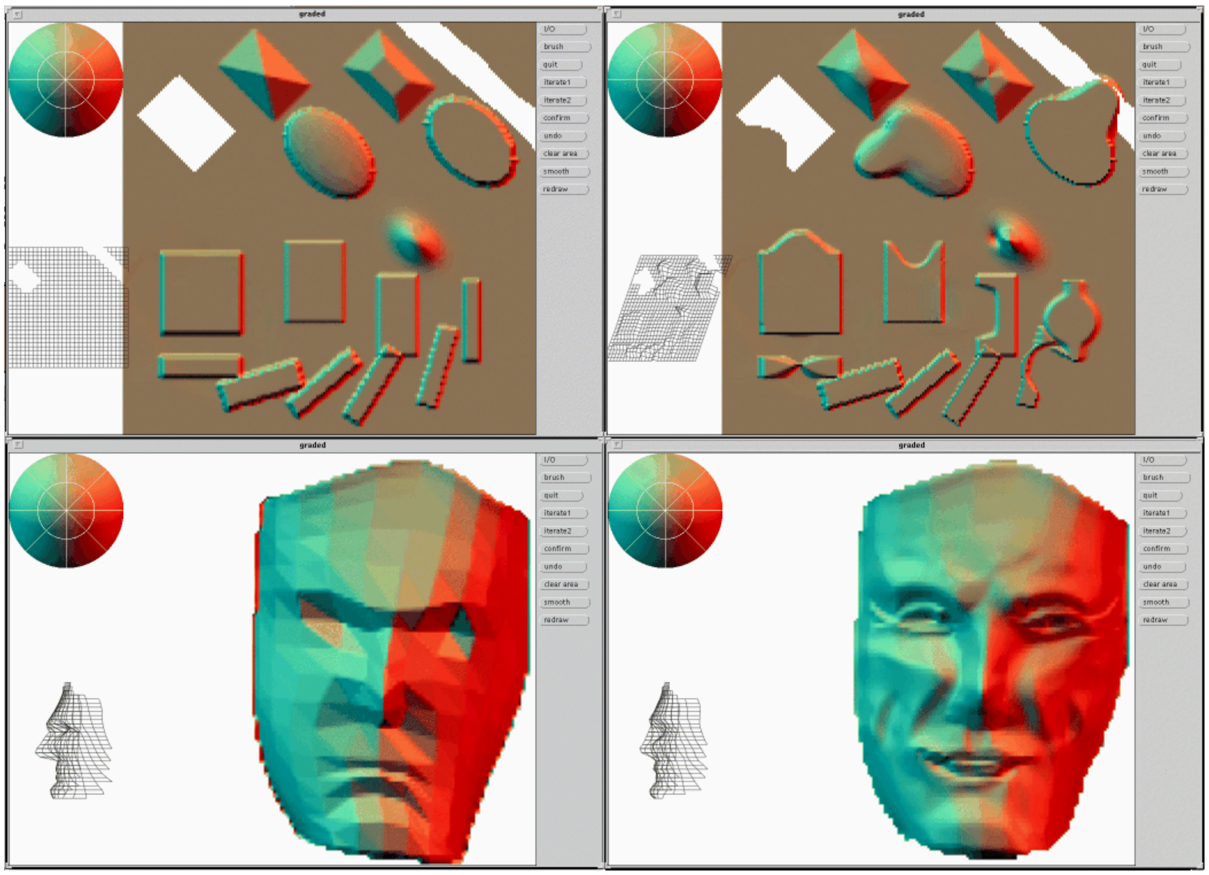

Figure 5: Upper left: a sample of brush shapes and brush profiles used to add, carve and cut in the depth buffer. Upper right: the application of some image processing operators to distort, shift, smoothen and sharpen the depth buffer. Lower left: a simple polygonal model of a face represented in GRADED. Lower right: the same face after editing in GRADED.

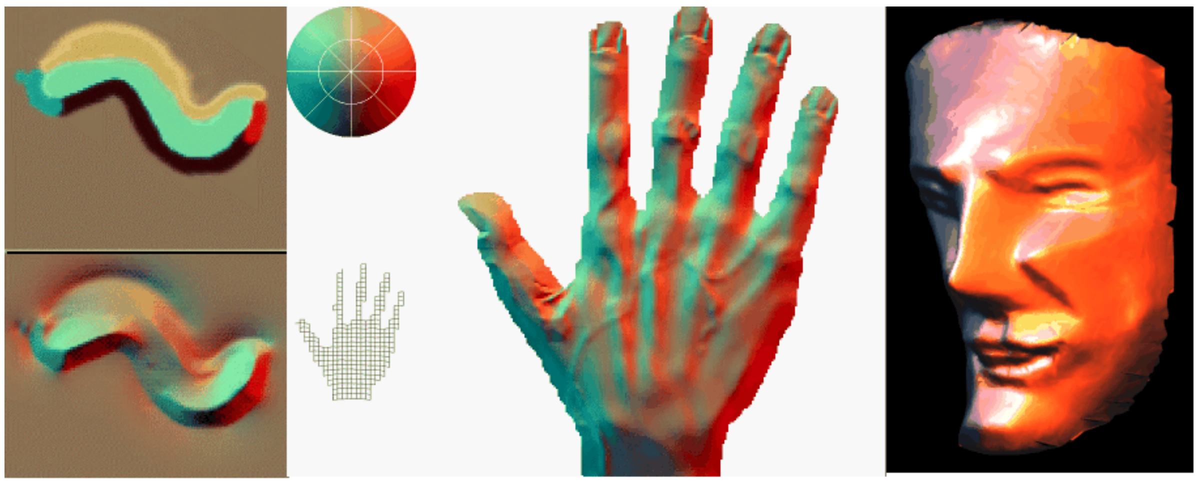

Figure 6: Upper left: a shading distribution painted into the gradient buffer. Lower left: the result of conservativity restoration. Middle: a relief-painting of a hand; Right: the face of¿ figure 5, rendered from a different view point.