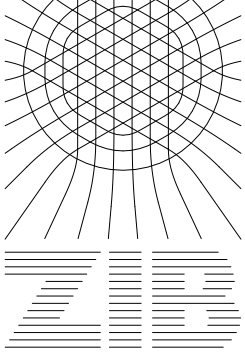
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Heilbronner Str. 10, D-10711 Berlin - Wilmersdorf



Gerhard W. Zumbusch

Visualizing Functions of the h-p-Version of Finite Elements

Technical Report TR 94-5 (July 1994)

Visualizing Functions of the h-p-Version of Finite Elements

Gerhard W. Zumbusch*

Abstract

Results from finite-element-calculations are usually visualized by colored surface- and contour-line-plots or polygonal patches or simply displaced lines and grids. In computer graphics however more advanced techniques like texture-mapping and NURBS are well established and there exist efficient algorithms and implementations. We show that these techniques are not only easy to use, but form a very natural and far more efficient approach for visualization of higher order finite-element's solutions like in p- and h-p-version. Texture-mapping is useful for displaying vector-valued data, too.

Key Words
computer graphics, FEM, p-version, multivariate interpolation
AMSMOS, MSC 1991 subject classification
Primary 65N30, 41A10.

*Konrad–Zuse–Zentrum Berlin, Heilbronner Str. 10, D-10711 Berlin-Wilmersdorf, Germany. Zumbusch@ZIB-Berlin.De

Contents

1 Introduction

The Finite Element Method is a generally accepted and widely used method for the solution of many problems in engineering and science. There are many different approaches and a vast number of computer codes implementing them. Some of them can be categorized under the terms of h-, p- and h-p-version. The first part of this text deals with higher degree polynomial shape functions, which can occur in all finite element versions, but are essential for the p- and h-p-version.

Let us assume that we have calculated a solution of our problem with a finite element code. This solution typically resides in a real space $\Phi : \Omega \subset \mathbb{R}^d \rightarrow \mathbb{R}^s$. In the case that we are able to reduce the data of interest (the solution) to a single set of numbers, we can analyze the result easily. If we perform eigenvalue computations or some postprocessing like calculation of the maximum or an integral of specific components of the solution, we get only a couple of numbers. In all other cases, e.g. if we just want to analyze eigensolutions, we have to visualize a solution over the domain Ω. This is easy in one space dimension $d = 1$, drawing lines and graphs. In this text we consider the two-dimensional case ($d = 2$). The corresponding graphical representations can be subsumed under the subject of surface plots. If we want to visualize more than one component of the solution ($s > 1$) simultaneously, we can use common techniques like arrow plots or a more sophisticated approach with texture maps.

We want to show that traditional visualization of finite elements can easily be extended by state of the art techniques of computer graphics. This permits visualization without loss of information and with a lower amount of data in the context of higher degree polynomial shape functions by e.g. NURBS. In addition the visualization can be extended to transmit additional information via texture mapping, too. We stress algorithmic and implementational details that are important for performance and for correctness of the results. For a background in computer graphics, see e.g. [FDFH90].

In the first part of the paper we review common methods of visualization of surfaces $\Phi : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$. We do this with the fact in mind, that Φ is not a piecewise linear function, which is often assumed in current visualization packages, especially in the context of finite elements. The second part is concerned with NURBS for the exact representation of Φ and with the visualization of NURBS. In the third part we combine the visualization with texture mapping and show some applications, where additional information is transmitted via texture mapping.

1.1 Contour Lines

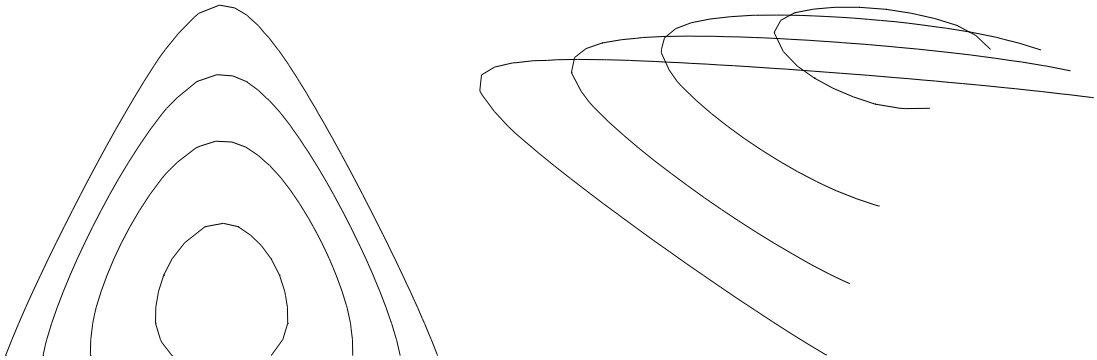


Fig. 1. Contour line plot on a triangle, black&white, vector/plotter style, in the plane and in 3D orthographic projection.

A very early graphical representation of surfaces are contour lines. They require only line drawing devices like pen plotters and do not need any hidden line or hidden surface computations. The calculation of the contour lines itself for arbitrary functions Φ is a non-trivial task. For piecewise linear Φ the algorithm is rather simple and numerically stable. This was done in figure 1. In the case of piecewise quadratic and cubic Φ a numerically stable algorithm is possible, if data is represented in the right way. For implementations in the case of regular grids see [RAW90] and in the case of irregular, triangular tesselations see [PS77]). For higher polynomial degrees all algorithms are only approximative and heuristic for algebraic reasons and may be inefficient. For pen plotter devices there is the additional task of connecting contour line segments and optimizing plotter paths. This is not necessary for raster

devices used nowadays. Nevertheless drawing contour lines is the cheapest visualization method, measured in the number of graphics operations.

1.2 Mesh Plots

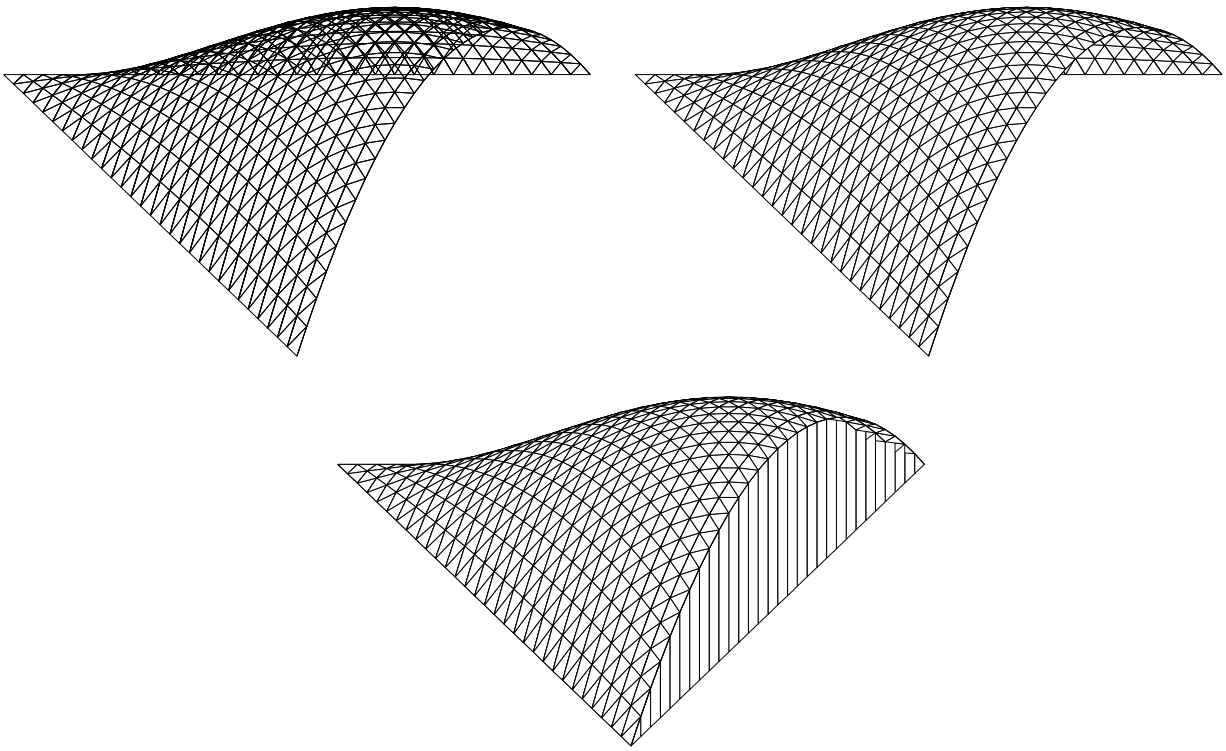


Fig. 2. Mesh plot on a triangle; black&white, vector/plotter style, drawn transparent and with hidden lines by painter's algorithm and in a co-ordinate box.

Another method of plotting surfaces is the drawing of mesh plots. A regular mesh in the domain Ω is projected onto the solution-manifold in $\mathbb{R}^3$. In most cases the pictures are very comprehensive, but sometimes it is difficult to grasp at the first glance whether some visible parts of the surface represent the upper or lower side. If there are different colors or textures available, the lower side can be presented in a visually distinguishable way.

Algorithmically there are different approaches. The simplest is: Draw the mesh transparently. This results in rather complicated pictures. The more

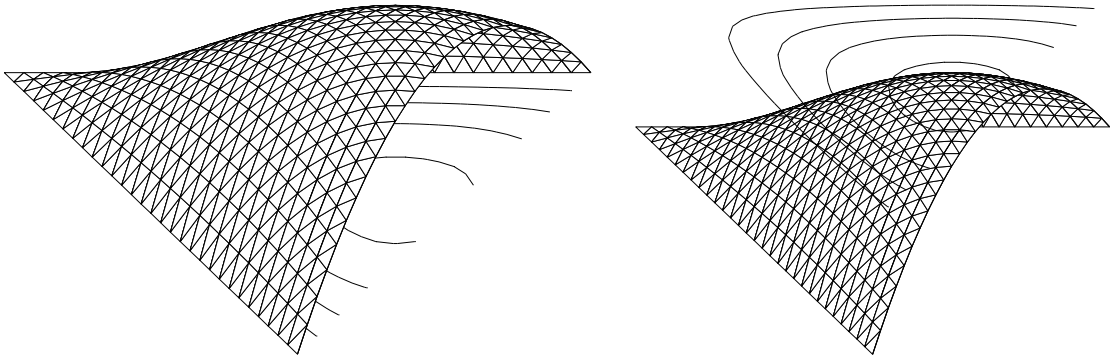


FIG. 3. Mesh plot on a triangle with contour lines beneath and above.

sophisticated methods use hidden line algorithms. Algorithms for plotter devices have to implement a skyline buffer which accumulates maximum and minimum skyline for the drawing from front to back. An emerging difficulty is a correct labeling of the drawing interacting with the surface in $\mathbb{R}^3$. Labels of axis may be hidden by the surface or hide it themselfes. They may also be placed into cuts of the surface. Being unable to erase parts of the picture is a general problem of plotter devices.

A cheaper algorithm which is only applicable for raster devices is a simplified version of painter's algorithm. Just draw the lines and redraw the patches between them from back to front. This utilizes no buffer and bookkeeping but the possibility of erasing areas in the picture. Labels can be drawn under (before) and over (after) the mesh. For this algorithm much more pixels (resolution dependend) have to be manipulated. The pictures shown in figure 2–3 have been produced by a more complicated version of painter's algorithm which actually sorts all patches and lines, both for display and for print. For this we used the matlab package [Mat92].

1.3 Surface Plots

Using raster devices the step from mesh plots to surface plots is not a big one. Pen plotters do not appreciate filled patches anyway. We apply the painter's algorithm as for the mesh plots. Now we utilize different colors or gray-scales filling the patches which were erased (filled with background color) for the

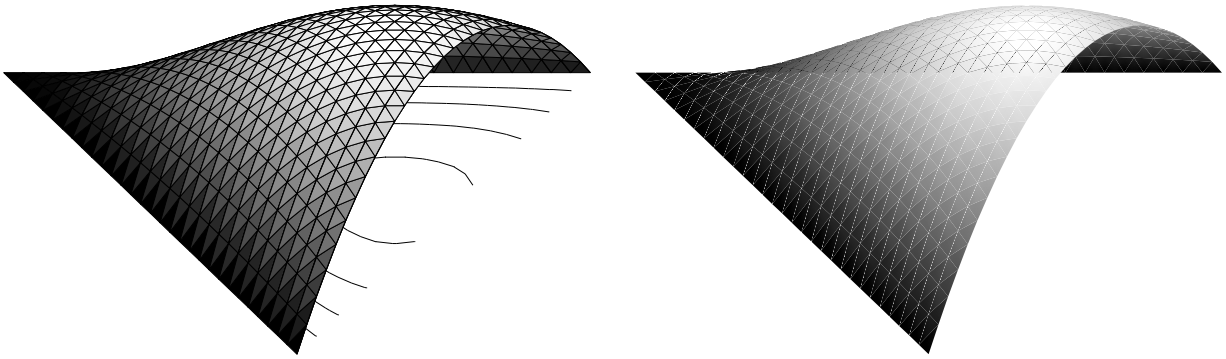


FIG. 4. Surface plot on a triangle. Produced with hidden surface removal by painter's algorithm (interpolated intensities).

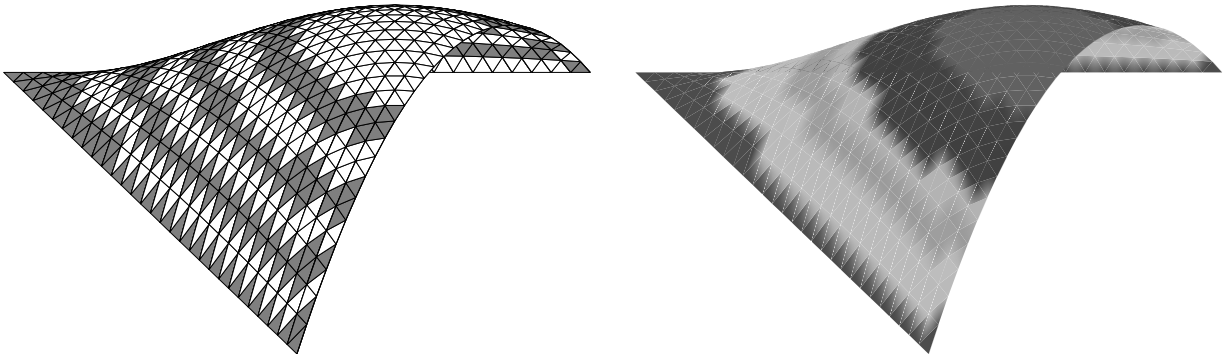


FIG. 5. Surface plot on a triangle. The elevation is mapped to the intensity via a color lookup table.

mesh plots. We have a new degree of freedom if we choose the color of the patches not uniform, but varying over the surface. We could have colored the lines for the mesh plot, too, but this would not be very impressive in print, since most printers have difficulties separating different colored lines.

A standard way of assigning colors is via a color lookup table. This is commonly used for color displays with a restricted frame buffer depth and therefore a restricted number of colors that can be displayed simultanously. There arise two questions: How do I assign the indices of the lookup table to

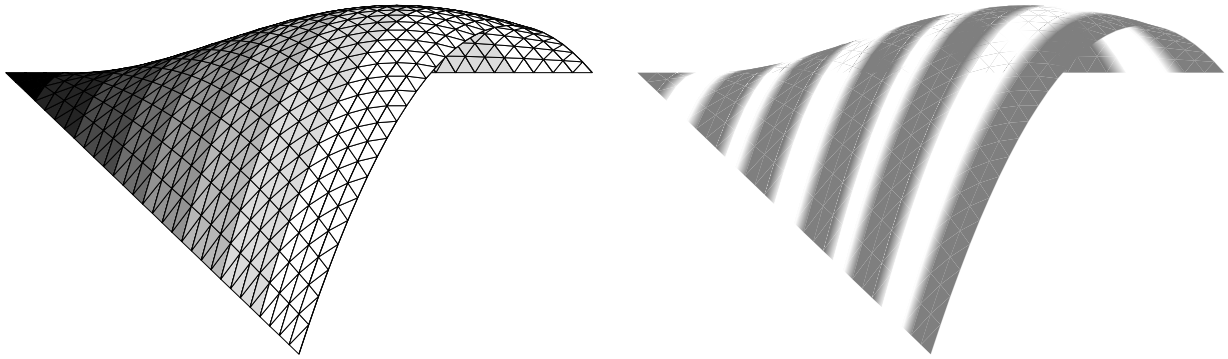


FIG. 6. Surface plot on a triangle. One co-ordinate is mapped to the intensity via a color lookup table.

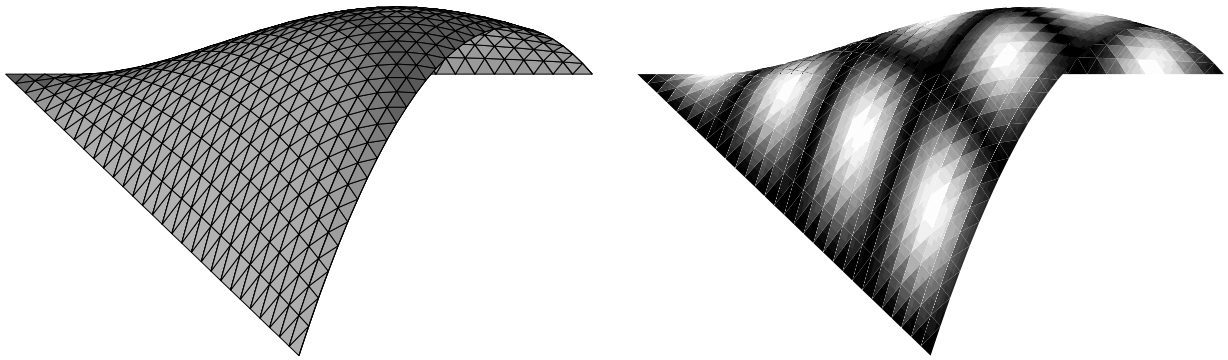


FIG. 7. Surface plot on a triangle. The surface is considered as a real object with lightening model and with texture mapping (texture $|\sin \pi x_1 \sin \pi x_2|$).

the surface and what colors do I put into the lookup table. Some examples are shown here.

The examples can be classified into groups by the geometric properties parametrizing the lookup table. We choose elevation which is the value of the solution itself, one axis of the picture (depth) and the geometric properties of the surface in $\mathbb{R}^3$. Colors are calulated via geometric normals to the surface in combination with an ambient, diffuse and speculiar light reflection model using Gouraud-shading. Another classification is by the color lookup tables,

which can be quasi-continuous over the whole spectrum or some parts of it or discontinuous, yielding geometrical effects like contour bands.

In figure 7) texture mapping is used. This technique will be used in chapter 3 for visualizing additional information, but is also useful for piecewise linear functions Φ.

2 NURBS

If the graphical results of the piecewise linear representations are not satisfying and we want to increase precision, we have to supply additional data. This can be done either by increasing the number of linear patches or by using patches with a higher level of precision. The decision is analogous to the decision in finite element methods whether to do an h- or an p-refinement (possibly both, h-p). If we use higher order polynomials in the numerical calculation, linear patches with only continuous surfaces (not differentiable) do not suffice anyway. In order to improve the patches, we can choose polynomials and rational functions, which are both heavily used in computer graphics. Continuously differentiable surfaces emerge from the calculus of (graphical-) G^1-continuous functions, which equal standard C^1-continuous modulo a proper parametrization [Boe88]). Surfaces emerging from non-conforming global finite-element-solutions are not continous and surfaces from conforming finite-elements with second order operator need not be continously differentiable. The surfaces are be in all cases piecewise differentiable on every local finite-element.

We now concentrate on the exact graphical representation of higher order polynomial shape funtions in finite element context. The solution may be represented by any set of complete polynomials well suited for graphical computations [Zum93]. If we use central projection, starting with polynomials, we have to do the remaining parts of computation with rational functions. Hence we can operate on rational functions anyway. They transform to rational functions of the same degree under central perspective transformations. Orthographic perspective is considered a special case of central perspective. Another argument for rational functions is that they are already efficiently implemented in many graphic subsystems.

central projection in projective co-ordinates

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} \mapsto A\vec{x} = \begin{pmatrix} \vec{a}_1\vec{x} \\ \vec{a}_2\vec{x} \\ \vec{a}_3\vec{x} \\ \vec{a}_4\vec{x} \end{pmatrix} \equiv \begin{pmatrix} \vec{a}_1\vec{x}/\vec{a}_4\vec{x} \\ \vec{a}_2\vec{x}/\vec{a}_4\vec{x} \\ \vec{a}_3\vec{x}/\vec{a}_4\vec{x} \\ 1 \end{pmatrix}$$

with transformation matrix

$$A = \begin{pmatrix} \vec{a}_1 \\ \vec{a}_2 \\ \vec{a}_3 \\ \vec{a}_4 \end{pmatrix} \in \mathbb{R}^{4,4}$$

2.1 Trimming Bivariate NURBS

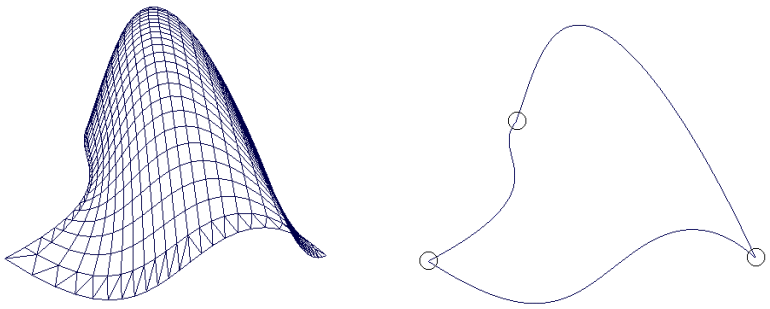


Fig. 8. A bivariate NURBS on a square, automatically resolved into patches and a triangular trim curve for it, hardcopy from screen.

Unfortunately in most cases only the bivariate rational functions (= bivariate NURBS, see [Far90]) for the rectangle are implemented in graphical subsystems [McL91, Hew92], but we need them on the triangle. NURBS on the triangle are in some sense simpler, because they are determined by a lower number of coefficients (control points). But NURBS on the triangle are completely contained in the space of NURBS on the rectangle of the same degree and the double area. Hence we can compute the coefficients of the NURBS on the rectangle instead of the ones on the triangle. For graphical representation we have to remove one half of the rectangle which can be done by trimming. This is shown in the sequence of figures 8 and 9. The NURBS is defined in parameter space on $[0,1]^2 \to \mathbb{R}^3$. We can restrict this space to a domain $T \subset [0,1]^2 \to \mathbb{R}^3$. The domain T can be defined as the interior of an oriented closed curve called "trimming curve". In our case T will be the triangle $T = \{(x_1, x_2) | x_1, x_2, 1 - x_1 - x_2 \geq 0\}$.

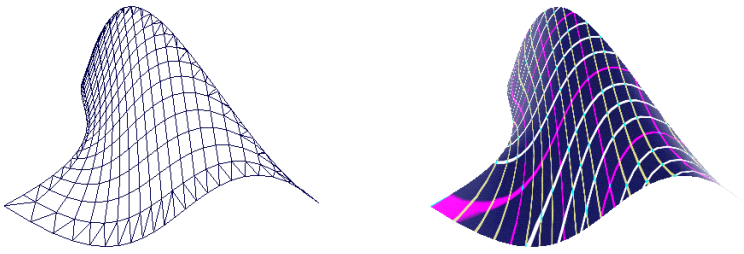


FIG. 9. The bivariate NURBS trimmed to a NURBS on a triangle, additionally with texture mapping.

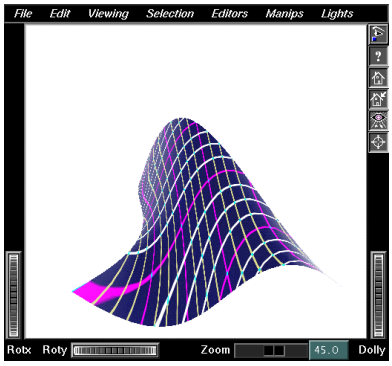


FIG. 10. A NURBS on a triangle with texture mapping in the interactive scene viewer tool (IRIS Inventor).

We use the term NURBS in the sense of one spline per finite element patch. Usually the term "spline" is connected with the coupling of these splines on different patches (e.g. G^1 continouity). But in our context properties like continuity are delivered (or maybe not) by the solution itself. In this section we use hardcopys form screen rather than patch-wise prints.

2.2 Display and Print

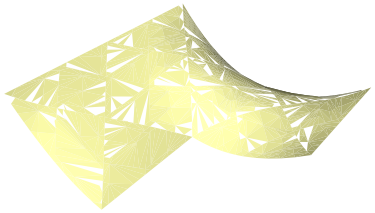


FIG. 11. A solution on an L-shaped domain with quadratic triangle shaped elements.

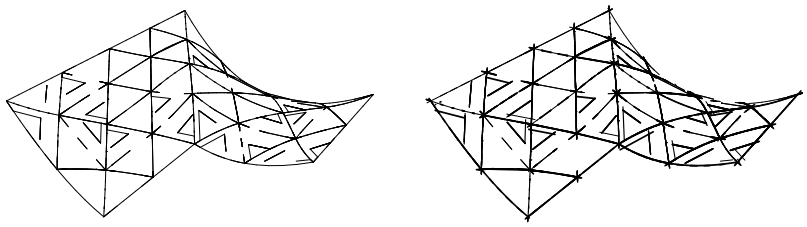


FIG. 12. A solution on an L-shaped domain with quadratic triangle shaped elements, printed with plain lines and with extended lines.

Now using the graphical subsystem for displaying NURBS, we also rely on its printing capabilities. We show different settings of the postscript driver of the package IRIS Inventor [Wer92], which uses the package IRIS GL [McL91, Sil] for display (now available as Open GL and Open Inventor). The printer driver of IRIS Inventor is also available in the visualization package Explorer [Hal93a, Hal93b]. A very similar approach to NURBS is permited by the graphical system Starbase [Hew92].

The display of the figures on screen exploiting special graphics hardware is fast, but the output of the printer drivers in comparison to is very slow.

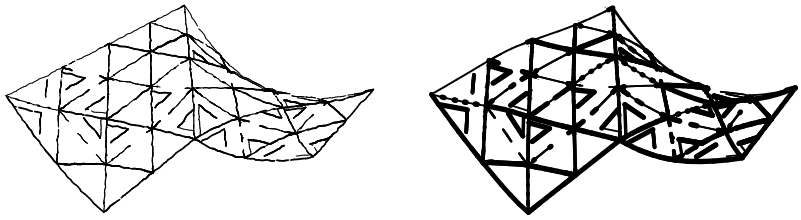


FIG. 13. A solution on an L-shaped domain with quadratic triangle shaped elements, printed with wobbly lines and with warped lines. Some lines are intentionally left out for an impression of a sketch.

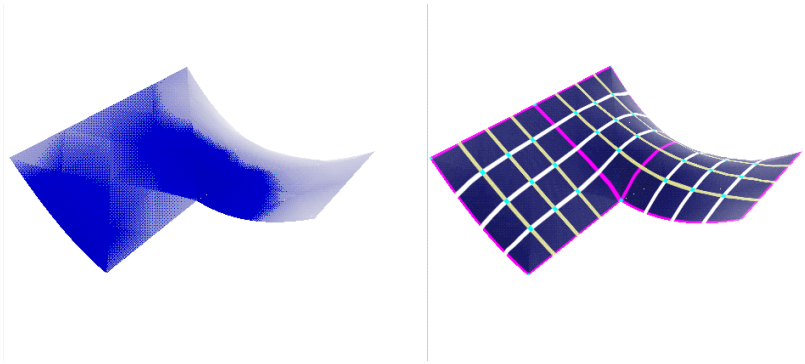


FIG. 14. A solution on an L-shaped domain with quadratic triangle shaped elements.

The time expense depend on the hidden-line algorithm, and more on the bandwith of the graphics channel which is usually narrow compared with the huge amount of data. The pictures in figure 11–13 are drawn by painter's algorithm. The grahical objects are sorted and drawn from back to front. Objects on the front may cover objects behind them drawn earlier. In the case of two objects overlapping each other, the sorting fails and at least one object has to be splitted into parts.

Painter's algorithm results in relatively small data files. Nevertheless the

processing of this data consumes an equal amount of printer/processor time as the processing of a much larger but better structured file of a screen dump, which only needs transfer bandwith. One argument for the patch-wise printing is the resolution. Resolution is only bounded by the precision of number representation, which grows logarithmically in file size. Resolution much greater than printer's resolution is only usefull for filtering. Hardcopy resolution is restricted by the number of pixels saved. Filtering and other techniques only can enhance quality of pictures with low resolution a little bit. On the other hand patch-wise printing may be implemented in a numericaly unstable way. The postscript printer drivers used for printing figure 11 does make errors (sorting errors due to implementation), which are clearly visible. The used drivers for the screen show no visible errors, though they use related algorithms. Hence printer drivers sometimes have low quality.

Another point to be mentioned is sorting. Painter's algorithm for patch-wise drawing requires sorting of patches, the (hardware) Z-buffer for pixel drawing does not. Hence in general the complexity of painter's algorithm grows with $n \log n$ Operations whereas Z-buffer is optimal with n operations for n equal-sized patches. The graphical packages mentioned use a general sorting for painter's algorithm which makes them potentially slow. Taking advantage of the structure of finite-elements (no intersection, back to front with known neighbourhoods) would reduce sorting down to an n operation process, too. In finite-elements one could look for a point on the boundary of the domain which is the point farthest away. Proceeding from one element only to its neighbours reduces the problem to local sorting. Additional knowledge of hierarchies in finite elements faciliates a kind of hierarchal sorting. Often a previous hierarchy has been sorted already so there is only little additional work left.

2.3 Adaptive Tesselation

A general problem concerning higher degree polynomials we mentioned in chapter 1.1. We have to render an higher order surface. We can do this directly. But if the degree p of the polynomials exceeds three we cannot find the roots and do some of the computations symbolically for algebraic reasons. We also can render the surface by linear patches. We apply adaptive subdivision of patches until the area of display is resolved fine enough (dis-

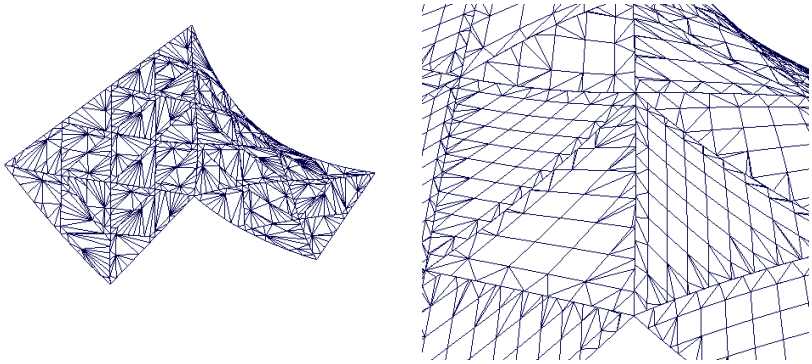


FIG. 15. A solution on an L-shaped domain with quadratic triangle shaped elements. It is automatically resolved into patches. We show a global view and a detail of the adaptive tesselation. The picture enlargement triggers a finer tesselation.

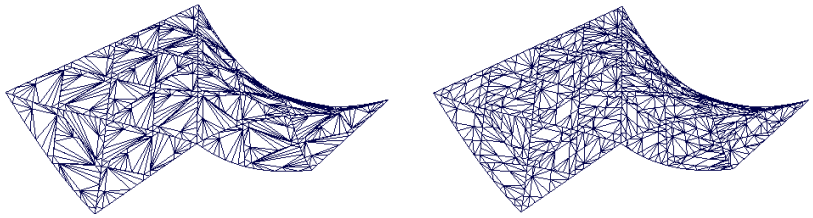


FIG. 16. Quadratic triangle shaped elements, geometric complexity for adaptive control of the tesselation is set to 0.01 and to 0.7.

play space), or the object itself is resolved fine enough (object space). We postponed this problem to the graphical software package (here Iris GL). We only supply the coefficients of the polynomial function Φ in a suitable polynomial basis. We choose a parameter $c \in [0, 1]$ for adaptivity in display space, called "geometric complexity". In this implementation it is unfortunately independent from real geometric complexity of the surface, like curvature or other more heuristic measures. A more general approach to adaptivity on

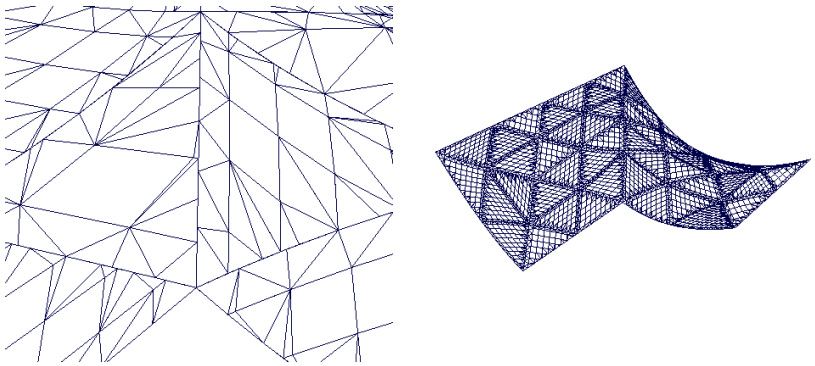


FIG. 17. Quadratic triangle shaped elements, geometric complexity for adaptive control of the tesselation is set to 0.01 (detail) and to 0.9.

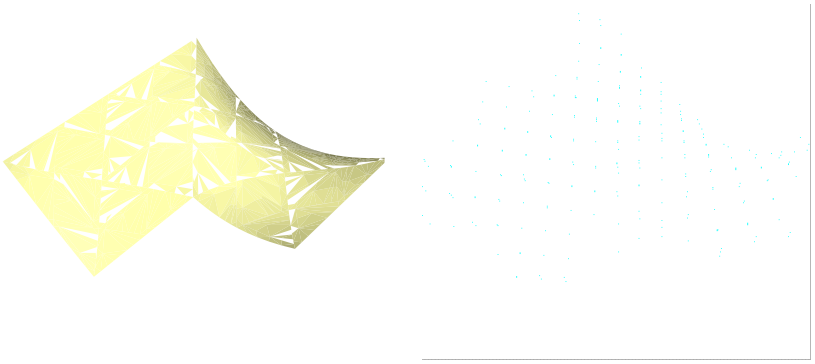


FIG. 18. L-shaped domain with quadratic triangle shaped elements, only vertices of the elements, the cheapest representation, shaded print/ hardcopy.

this kind of surfaces is contained e.g. in [Hoh91].

2.4 POLYNOMIAL DEGREES

Here we try to show the influence of the polynomial degrees onto the picture and onto the visual impression (figures 19 and 20). The influence of polynomial degrees on the approximation properties are clear analytically. In the case of regular functions we get an approximation error of $\mathcal{O}(\sqrt[\alpha]{\langle})$. This means that we need a lot of linear patches to approximate an higher

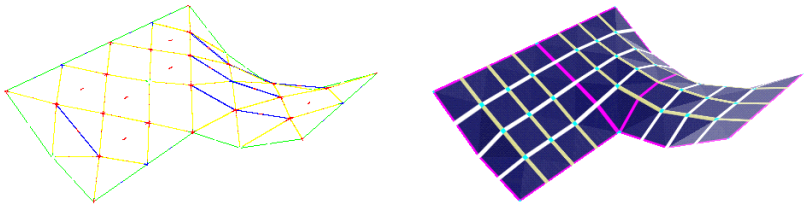


FIG. 19. A solution on an L-shaped domain with linear triangle shaped elements, additionally texture mapping with projection into co-ordinate plane.

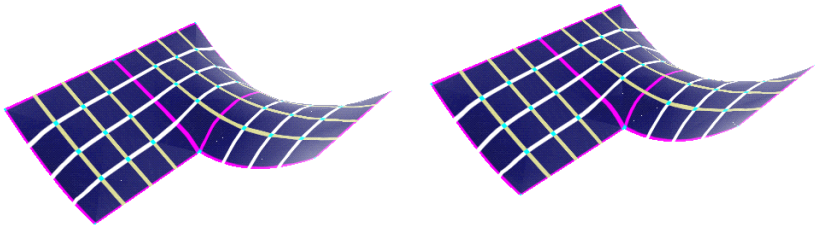


FIG. 20. The solution calculated with quadratic and cubic ploynomials, represented by quadratic and cubic triangle shaped elements.

order surface (figure 23). Using higher order patches reduces data drastically. One effect which is visible (and obvious) is, that linear elements can be represented by a linear patch, whereas higher polynomials are subdivided into (many) linear patches.

We now show the influence of the polynomial degrees onto the tesselation (figures 21 and 22). The graphical object of the figures itself comes from h-p-version with non-uniform degree p and is a rather rare one in graphical representation. Up to rounding errors, the functions on the boundary of the

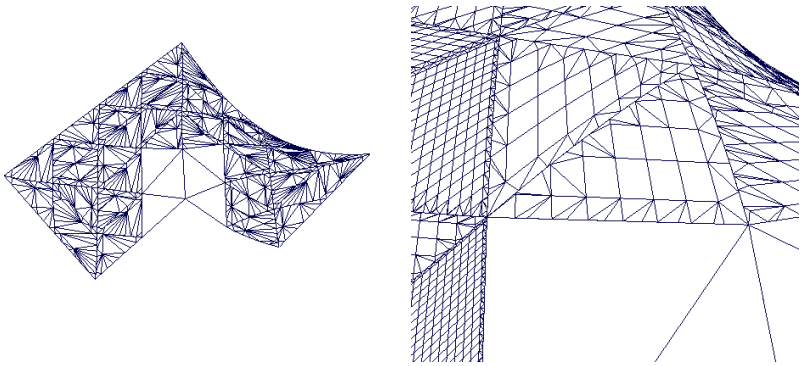


FIG. 21. L-shaped domain with linear, quadratic and cubic triangle shaped elements. Linear shape functions are not resolved into smaller patches because they are drawn exactly. Adaptive tesselation, global view and detail.

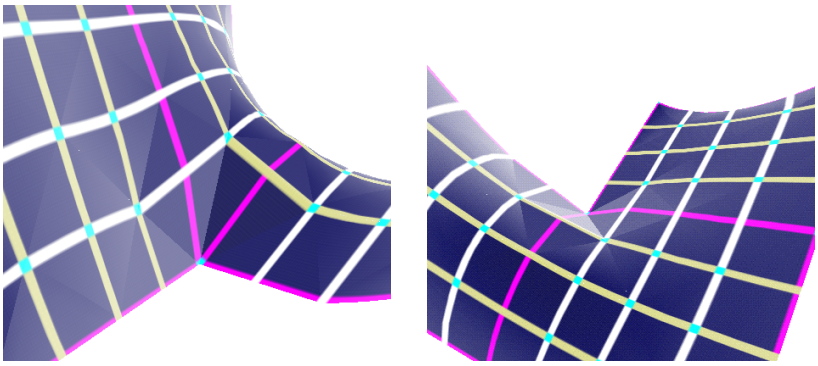


FIG. 22. Two different views of an L-shaped domain with linear, quadratic and cubic triangle shaped elements. Linear shape functions have got a constant brightness due to reflection calculation.

patches are equal, because the global function is approximated continuously. Hence the dense subdivision of quadratic elemements which have an edge in common with a linear element do not produce errors in continuity. A key point is either a stable (unambigous) representation of the polynomials or a high precision of the coefficients. Relying on precision, excessive subdivision or excessive enlargement of the picture will result in such display errors.

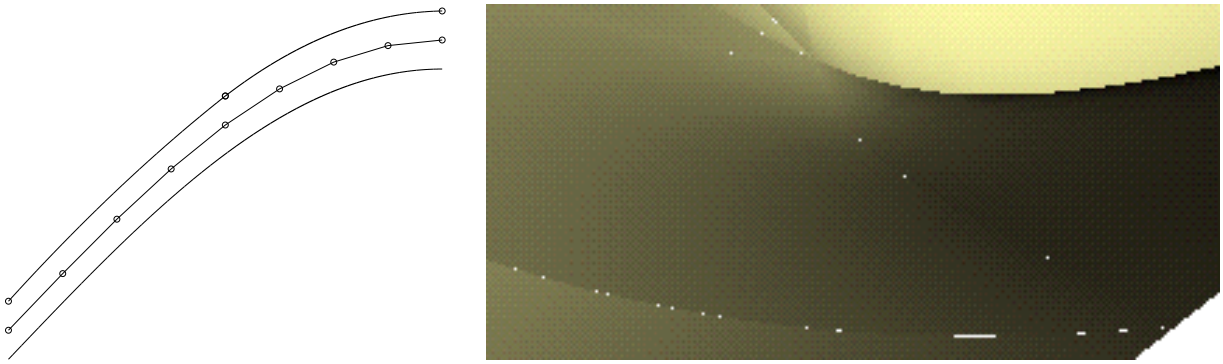


FIG. 23. Approximation of $\sin x$ (bottom) by 8 equidistant (linear) lines (middle) and by 2 quadratic curve segments (top). Rendering errors due to lack of coefficients' precision.

Such errors will always occur on edges of two triangles with different represented polynomials or different subdivided polynomials, which is the case for different polynomial degrees.

2.5 NON-CONFORMING TESSELATIONS

If we consider non-conforming tesselations, we come into the same kind of trouble concerning continuity. Non-conformity means that two neighboured elements may have in common only some part of an edge, not the whole one. This does not make more trouble than the conforming case, a sufficient precision of coefficients and graphical computations supposed. But there is a general problem: If we stress the term numerically stable, which e.g. means that the behaviour of the algorithm on one edge only depends on the data of that edge, we come into trouble. One has the problem that the different elements obtain very different sets of coefficients on the same edge, even for a stable basis like the NURBS. This leads to acceptable low, but certain errors. This errors become unacceptable, if there are questions like: Is this specific pixel filled or is it not? Hence the global surface may become non-continuous under some view transformations, which can be visible.

A general solution for this problem is not a change of the NURBS basis. There are two alternatives: One can use the original representation of data of the finite element method and do the adaptive tesselation in a global

conforming way. This can be done in a approach like [WR92]. Another way is used in [RR93a, RR93b], where the NURBS basis is retained, but additional information concerning topology of the tesselation is supplied. The renderer "knows", that the surface is continuous and that specific functions have to be equal on an edge. In the case of critical decisions for drawing, an algorithm can use this information by some kinds of modified skyline algorithms, which cannot miss a pixel.

3 Texture Mapping

We want to show results using a second technique of computer graphics, which is not directly related to NURBS, texture mapping. Some figures have already shown texture mapping, but only in connection with a linear mapping in $\mathbb{R}^2$ and using simple maps. Texture mapping is algorithmically simple, but computationally intensive. At a first glance one would omit texture mapping therefore for finite element visualization. We have arguments against: For a hardware supported texture mapping [Sil] it is no problem to cope with the complexity of finite elements, although a simple line drawing (without hidden lines) performs faster of course. But using hardware Z-buffer, additional texture mapping (sometimes also in hardware) is not a big deal and it has the same measure of complexity $\mathcal{O}(n)$. Another point, which we will make more clear in the following is that we are able to transmit additional information via texture mapping. It is a natural generalization of the linear color lookup tables, which nearly everybody uses already. The pictures in this section were made with the use of Inventor [Wer92]. The postscript printer drivers we did use are not capable of texture mapping, but other drivers are. A better quality printer driver (but pixel oriented) is contained in the package AVS [UFK+90].

3.1 Linear Mapping

Like in the previous sections we use a linear mapping of coordinate space $\mathbb{R}^2$ into the periodic parameter space of the texture $[0,1]^2$. We always use the same geometry and the same mapping but change the texture. There are libraries full of textures. We choose some characteristic color textures. They are simply pixmaps generated by scanning photographs or purely synthetic. Sometimes they are modified and processed for periodicity of the picture which excludes gaps when mapping them.

Some pictures like the rock example (figure 25) give a realistic impression of an artificial object in $\mathbb{R}^3$ which is the solution. The feature texture mapping extends the imagination of the surface plots. Another alternative is the application of geometric textures like the brick-pattern (figure 25) or grids (figure 27). Here we can easily perceive curvature and relative height. Compared to classic line plots with parallel lines we can recognize here curvature

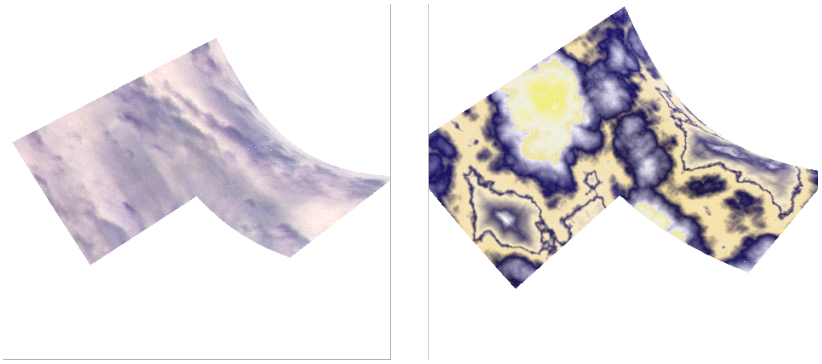


FIG. 24. L-shaped domain with quadratic triangle shaped elements, textured with clouds and with marble.

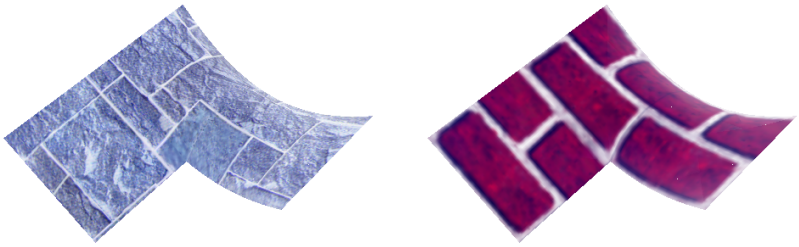


FIG. 25. L-shaped domain with quadratic triangle shaped elements, textured with faces of rock and with bricks.

in both directions because of the lines running in two orthogonal directions. We have a widening and a thickness of the lines when they are near to the observer. This is a very intutive effect similar to a textured membrane (a balloon) put onto the surface. Choosing a proper texture, one add text and annotations onto the texture, inscriptions like name of the axis and numerical values, and graphics like contour lines.

A general problem is the printing of the pictures. One problem is the impact of the reversal of black and white to realistic lightening models. One problem is the proper transfer of colors from screen to print. It is possible

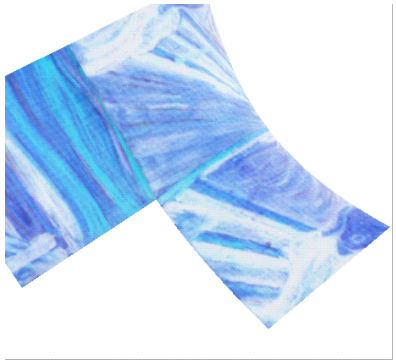


FIG. 26. L-shaped domain with quadratic triangle shaped elements, textured with Munch's painting "Scream".

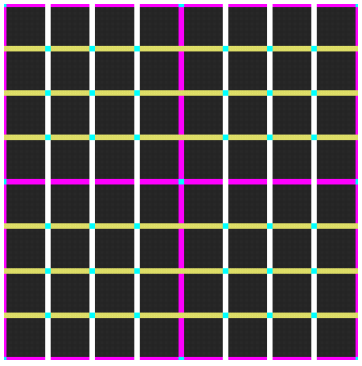


FIG. 27. Texture with grid lines, often used for the pictures here.

to reverse all colors, which would have been disastrous for real textures like the Munch painting (figure 26). Another method is to only change black and white which is bad in cases of near black or near white pictures. One also can substitute the black color of the border by white, which changes the appearance of the picture, too. For optimal results one only can tune the colors either for print or for display on the screen.

3.2 SOLUTION DEPENDENT MAPPING

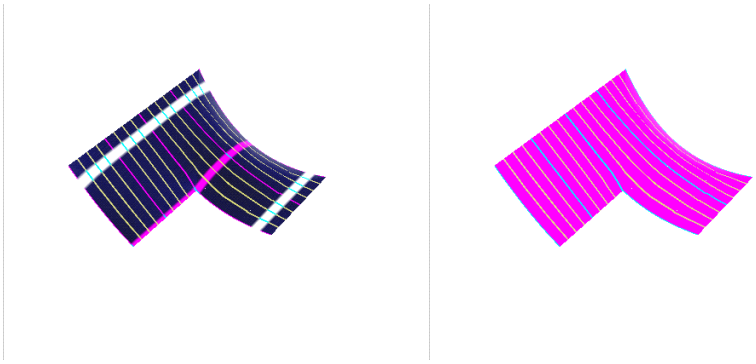


FIG. 28. L-shaped domain with quadratic triangle shaped elements, different scaling of the texture, streched in x_2- and shrunken in x_1-direction, resulting in parallel lines.

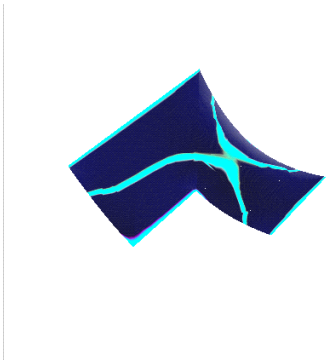


FIG. 29. L-shaped domain with quadratic triangle shaped elements, different scaling of the texture, scaling in x_1- and x_2-direction by the solution itself which equals an one dimensional color lookup table.

We continue with linear texture mapped views of the geometric object. Now we modify the parameters of the linear mapping $\mathbb{R}^3 \to \mathbb{R}^2$. We choose the geometric map of figure 27. We can distort the map in the x_1, x_2-plane which leads to line plots instead of grid plots (figure 28). The lines can have different thickness and focal points in the picture. The focal point depends on the resolution of the texture in relation to its magnification and the related texture filter. Here we use the standard bilinear filter which is

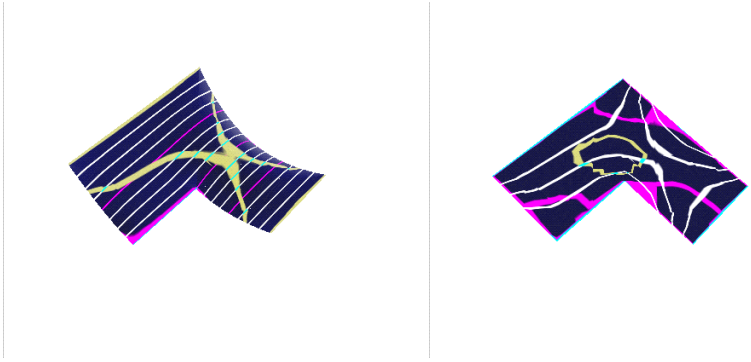


FIG. 30. L-shaped domain with quadratic triangle shaped elements, different scaling of the texture, x_1 in x_1-direction and the solution in x_2-direction/ one component of the solution in x_1 and the other in x_2-direction, no elevation (in x_3-direction).

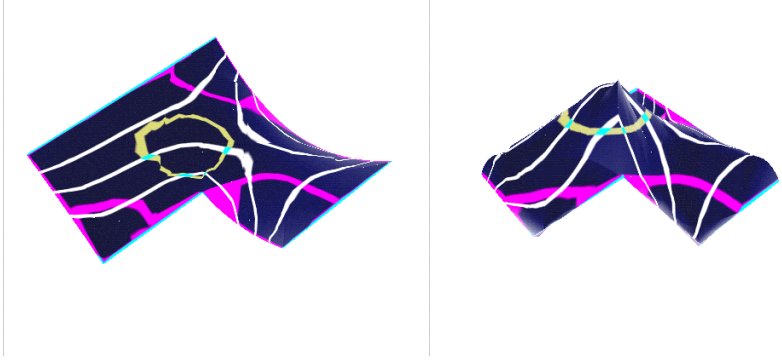


FIG. 31. L-shaped domain with quadratic triangle shaped elements, different scaling of the texture, one component of the solution in x_1 and the other in x_2-direction, the first one in x_3-direction and the second one in x_3-direction.

bilinear interpolation in texture co-ordinate space. More interesting is the usage of the third dimension which is the solution itself for the mapping. The simplest case (figure 29) is the scaling (x_3, x_3). Both texture co-ordinates are choosen as x_3. This means a color that is only solution dependent. Geometrically this is a one dimensional color lookup table indexed by the height of the surface. We could play the same game as in chapter 1.3, creating

colored contour bands and so on.

We choose more complicated mappings like mixtures of coordinates x_1, x_2 and a two-component solution which exploits full two dimensions of the texture. This only delivers different views of the same geometric object of the surface. Depending on the parameters and the actual mapping they may lead to new insights, but are at first difficult to handle and rather confusing.

A real new opportunity which is only possible by texture mapping is the visualization of vector valued solutions. In our case we take a solution (f_1, f_2) defined on a domain of $\mathbb{R}^2$, which is a two dimensional manifold in $\mathbb{R}^4$. This is rather complicated to imagine. Nevertheless some effects of it can be visualized. Using texture mapping, we now have a chance to show both components, which traditionally is done by smale arrows. Both techniques have to visualize two components in every point of the domain in $\mathbb{R}^2$. The arrows show a magnitude and an angle. Here we use the grid texture. This shows the interaction of both components in cartesian coordinates. We could have used a pattern of concentric circles or a system of meridians, also showing angles and magnitude in polar coordinates. Another promising technique is the convolution of textures along streamslines. This also visualizes vector fields, but it requires a different renderer.

4 Beyond Graphics

We have used only a smal set of techniques of computer graphics. Much more techniques are available and some of them are or become implemented these days fast enough for interactive applications in data visualization. We can only point out some features which may be useful. There are techniques developed for the use in flight simulators and the design in automobile industry today, which deliver even more advanced visualization techniques. One is volume rendering instead of surface modelling. This enables to use clusters of particles or fog of different intensities for visualization. Another one is a improved texture mapping facility with adaptive resolution textures, which resolve the pattern only in a size small enough, not in full detail. An additional improvement is 3-D texture mapping. 3-D textures enhance the number of parameters for the texture and use three dimensional arrays of pixels (voxels).

A real problem is the quality of the pictures. It is amazing that the algorithms for display on the screen are usually better or as good as the ones for printing. Printing now is some orders of magnitude slower than real-time graphics on screen, which would allow a higher quality. Techniques like anti-aliasing and filtering, not to mention a correct hidden line algorithm, are standard. State of the art algorithms perform these tasks in a very efficient way, but in our opinion quality goes first. Connected with this is the question of the optimal resolution and the optimal set of colors for a print. Both questions depend on many parameters like the question of reproduction of the prints, but in general the printer drivers should be improved.

Acknowledgements

The author wants to express his appreciation for helpful discussions with and corrections by H. C. Hege and D. Stalling. He also wants to thank the remaining members of the Devision "Visualization and Parallel Computing" for technical support and collaboration.

References

[Boe88] W. Boehm. On the definition of geometric continuity. *Comput. Aided Design*, 20(7):370–372, 1988.

[DLY89] P. Deuflhard, P. Leinen, and H. Yserentant. Concepts of an adaptive hierarchical finite element code. *IMPACT Comput. Sci. Engrg.*, 1:3–35, 1989.

[Far90] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design – A Practical Guide, 2nd ed.* Academic Press, San Diego, 1990.

[FDFH90] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics - Principles and Practice.* Addison-Wesley, 2nd edition, 1990.

[Hal93a] M.-A. Halse. *IRIS Explorer Module Writer's Guide.* Silicon Graphics, Inc., Moutain View, CA 94039-7311, 1993.

[Hal93b] M.-A. Halse. *IRIS Explorer User's Guide.* Silicon Graphics, Inc., Moutain View, CA 94039-7311, 1993.

[Hew92] Hewlett-Packard Company, 1000 N.E. Circle Blvd., Corvallis, OR 97330. *Starbase Graphics Techniques*, 1992.

[Hoh91] A. Hohmann. An adaptive continuation method for implicitly defined surfaces. Preprint SC 91–20, Konrad-Zuse-Zentrum, Berlin, 1991.

[Mat92] The Math Works, Inc., Natick, Mass. 01760. *Matlab, High-Performance Numeric Computation and Visualization Software*, 1992.

[McL91] P. McLendon. *Graphics Library Programming Guide.* Silicon Graphics, Inc., Moutain View, CA 94039-7311, 1991.

[PS77] M. J. D. Powell and M. A. Sabin. Piecewise quadratic approximations on triangles. *ACM Trans. Math. Software*, 3(4):316–325, 1977.

[RAW90] C. Reinsch, V. Apostulecu, and K. Weidner. *Das LRZ-Graphiksystem, Benutzer-Manual, Teil II.* Leibniz-Rechenzentrum, Barer Str., D-80333 München, Germany, 9th edition, 1990.

[RR93a] L. Rose and D. Ramey. *Advanced Visualizer User's Guide, Version 4.0.* Wavefront Technologies, Inc., Santa Barbara, CA 93103, 3nd edition, 1993.

[RR93b] L. Rose and D. Ramey. *Wavefront File Formats, Version 4.0.* Wavefront Technologies, Inc., Santa Barbara, CA 93103, 1st edition, 1993.

[Sil] Power series, technical report. Technical report, Silicon Graphics, Inc., Moutain View, CA 94039-7311.

[UFK+90] C. Upson, T. Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gruwitz, and A. van Dam. The application visualization system: A computational environment for scientific visualisation. *IEEE Computer Graphics and Applications ?*, 9, 1990.

[Vis93] Abt. Visualisierung und Paralleles Rechnen. Handbuch zur Visualisierung am ZIB. Technical Report TR 92-7, Konrad-Zuse-Zentrum, Berlin, 1993.

[Wer92] J. Wernecke. *IRIS Inventor Programming Guide.* Silicon Graphics, Inc., Moutain View, CA 94039-7311, 1992.

[WR92] A. Wierse and M. Rumpf. GRAPE - Eine objektorientierte Visualisierungs- und Numerikplattform. *Informatik Forsch. Entw.*, 7, 1992.

[Zum93] G. W. Zumbusch. Symmetric hierarchical polynomials for the *h*-*p*-version of finite elements. Preprint SC 93-32 ZIB, 1993.