Scientific Visualization, 2019, volume 11, number 1, pages 91 - 106, DOI: 10.26583/sv.11.1.08

Building and Visualization of Sleek 3D Surfaces without Misplaced Extremes

K.V. Ryabinin[1], K.A. Matkin[2]
Perm State University, Perm, Russia

[1] ORCID: 0000-0002-8353-7641, kostya.ryabinin@gmail.com
[2] ORCID: 0000-0002-0444-9476, matkin.k@yandex.ru

Abstract

The paper is devoted to the visualization of functional dependencies expressed as $y = f(x, z)$ by building sleek 3D surfaces based on discrete sets of points. The criteria of sleek surface quality are formulated taking into account the needs of scientific visualization and visual analytics. The most important criterion is the absence of misplaced extremes and oscillations on the result surface, because such artifacts can deliver false information about the process being represented by the visualized data. The methods of building smooth surfaces in the most popular scientific visualization software are examined against the formulated criteria and it is discovered that the misplaced extremes are an issue in modern visualization tools. To tackle this problem the new approach of building sleek surfaces is proposed. This approach is based on the previously developed algorithm of building smooth 2D curves that was generalized to the three-dimensional case.

The developed surface building algorithm consists of the following main steps. Assuming to have the input data as a regular grid of 3D points, which correspond to the table-defined function $y = f(x, z)$, we first propose to build the set of smooth 2D curves along X and Z axes. Afterwards, we propose to build bicubically blended Coons patches for all quads bounded by each 4 neighbor points from the original data set. Then we discretize each Coons patch by emitting new points to reach needed surface resolution. Next, we triangulate the created set of points and calculate vertex normals using smoothing groups algorithm. Last, we smooth the field of normals using Gaussian blur function.

The proposed algorithm meets the formulated criteria and ensures high visual quality of result surfaces. It was integrated into multiplatform charting library NChart3D and scientific visualization system SciVi, where it proved its correctness and stability by solving real-world scientific visualization tasks.

Keywords: smooth interpolation, Bezier curve, Coons patch, Gaussian filter, misplaced extremes, smooth surface, charts rendering, visual analytics.

1. Introduction

Almost every multi-purpose scientific visualization and visual analytics tool allows to build volumetric surfaces to graphically represent functional dependencies expressed as $y = f(x,z)$. When it is about scientific experiments, very often f is a table of discrete values obtained during measurements or mathematical modeling. Thereby the sample rate of f is limited by the experiment's circumstances and the result surface may become very rough. While the most popular rendering technique nowadays is polygonalization, the visual quality of corresponding images with low sample rates of f become unacceptable (see Fig. 1a). The one has to apply some smoothing interpolation to the set of values to achieve attractive and observable results.

One of the most popular ways in computer graphics to create smooth surface according to the discrete set of points is to build NURBS [1]. However, this way is often inappropriate for visual analytics needs, because NURBS surface does not contain the original point set using it as vertices of bounding lattice (as shown in Fig. 1b) and thereby does not really reflect the process under analysis.

The other way is to use smooth (continuously differentiable) interpolation functions like, for example, Hermite splines [2]. But this way in turn often gives so-called outliers: misplaced extremes on the surface, which do not belong to the original point set (as shown in Fig. 1c). Sometimes it is all right to have these outliers for sake of surface smoothness, but in some specific situations this can be a critical issue in terms of visual analytics. For example, if f is a table of temperature measurements across some area, the Fig. 1c will deliver potentially false information about the raise of temperature between two points in top-right corner. Especially if the samples of f are actually close to each other, the oscillation like in Fig. 1c is in fact very improbable and highly unwanted by showing continuously distributed values.

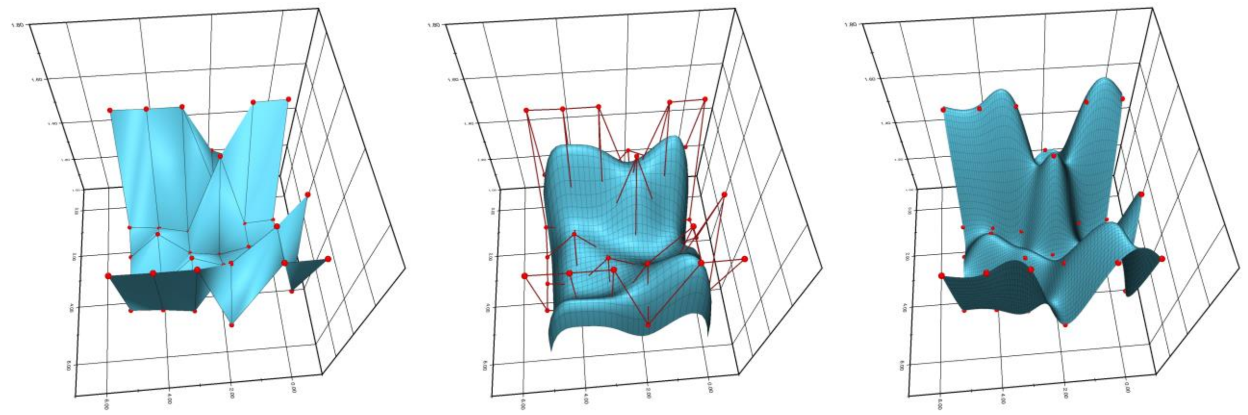

a b c

Fig. 1. Surface (painted blue) constructed by the set of control points (painted red) using different approaches: linear interpolation (a), NURBS (b), Hermite splines (c).

This paper addresses the problem of building the sleek-appearing 3D surface with no misplaced extremes according to the discrete set of points. While this problem is indeed crucial for solving several tasks of visual analytics, many visualization systems lack the efficient implementation of such kind smoothing. In the previous work [3] 2D case was considered and the solution for building sleek 2D curves according to the discrete point set was proposed. The present work is an improvement and generalization of that approach to the 3D case.

2. Criteria of Sleek Surface Quality

The surface *k*-smoothness is normally defined as the ability to be continuously differentiated *k* times, C^k, $k > 0$. Alternatively, according to E. Weisstein, a surface parameterized in variables u and v is called smooth if the tangent vectors $\vec{t}_u$ and $\vec{t}_v$ in the u and v directions satisfy $\vec{t}_u \times \vec{t}_v \neq 0$ [4].

However when it comes to the visualization and, in particular, visual analytics, the most important is how the surface is perceived and whether it can deliver proper information to the person examining the data. The perception in general is subjective, but it is based on the shape and *the shading*. While mathematically smooth surfaces are perceived sleek, the surfaces with derivative discontinuities (so-called C^0-surfaces) are not necessary perceived creasy. Moreover, if the shape is actually creasy, the proper shading can effectively mask it (which is widely used in computer graphics to perform visually attractive presentation of low-polygon approximations of 3D models).

Taking into account the visual analytics needs, the list of quality criteria for the surface built by the discrete set of points $\{P_{ij}\}$, $i = \overline{1,n}$, $j = \overline{1,m}$ is pretty the same as the list of quality criteria for curves in [3]:

1. The surface should be an interpolation of $\{P_{ij}\}$, this means, it should contain $\{P_{ij}\}$.
2. The surface should be perceived as sleek as possible: there should be no noticeable creases, or their number should be minimal.
3. There should be no misplaced extremes on the surface: minimum and maximum on the $[P_{ij}; P_{i+1j}] \times [P_{ij}; P_{ij+1}]$ should be in border points and the surface should not oscillate in the defined area.
4. There should be no self-intersections on the surface if the corresponding linearly interpolated surface has no ones.
5. The surface should not oscillate in the vertical direction and for each $[P_{ij}; P_{i+1j}] \times [P_{ij}; P_{ij+1}]$ should not intersect the bounding box with the sides parallel to vertical axis and containing $P_{ij}, P_{i+1j}, P_{ij+1}, P_{i+1j+1}$.
6. The building algorithm should be as efficient as possible.

The logic behind these criteria is described in detail in [3] related to the 2D curves and can be transferred as is to the 3D case.

3. Test Point Set

Smoothing algorithms may spawn misplaced extremes in different combinations of neighbor points, so it is matter of elaborate testing to prove that the particular algorithm gives stable results. Table 1 contains the data set used in this paper for demonstration purposes. This data set is quite random (the data are generated artificially), but it clearly shows the misplaced extremes problem in all the algorithms the developed one is compared to.

Table 1. Data set used for demonstration purposes

	$x = 0$	$x = 1$	$x = 2$	$x = 3$	$x = 4$	$x = 5$	$x = 6$
$z = 0$	1.321	1.657	1.165	1.215	1.623	1.236	1.657
$z = 1$	1.265	1.654	1.154	1.165	1.153	1.648	1.654
$z = 2$	1.324	1.264	1.547	1.125	1.246	1.465	1.264
$z = 3$	1.165	1.654	1.125	1.154	1.315	1.135	1.654
$z = 4$	1.157	1.654	1.165	1.300	1.136	1.168	1.654
$z = 5$	1.215	1.658	1.184	1.156	1.163	1.185	1.658

It should be stressed, that while the proposed algorithm has no misplaced extremes on this particular data set, it is not at all the proof of its stability, because this data set does not cover all possible combinations of points positions. The stability of the proposed method was proven by solving the real-world visual analytics tasks, see the Section 6.

4. Related Work

4.1. Subdivided Surfaces

The problem of creating smoothed versions of rough surfaces relates not only to the scientific visualization and visual analytics, but also to other branches of computer graphics and computational geometry (including photorealistic rendering, computer-aided design, etc.). In polygon-based 3D graphics the smoothing of surfaces is all about *subdivision* – representing surfaces with more polygons than the origin point cloud ensures. In terms of mathematics, there are generally two types of subdivision: *approximation* and *interpolation*.

Approximation does not necessary contain the original set of points. The basics of approximation are described in details by C.T. Loop [5]. The examples of this approach are NURBS surfaces with their different modifications (like T-Splines [6]). Modern 3D graphics editors implement similar algorithms, for example, Catmull-Clark subdivision [7] used in Blender [8]. But the problem of these methods is always the same: approximated surfaces are typically enclosed in the original ones as it is shown in [5] and can be seen in Fig. 2. This means, they do not suit the very first criteria mentioned in the Section 2.

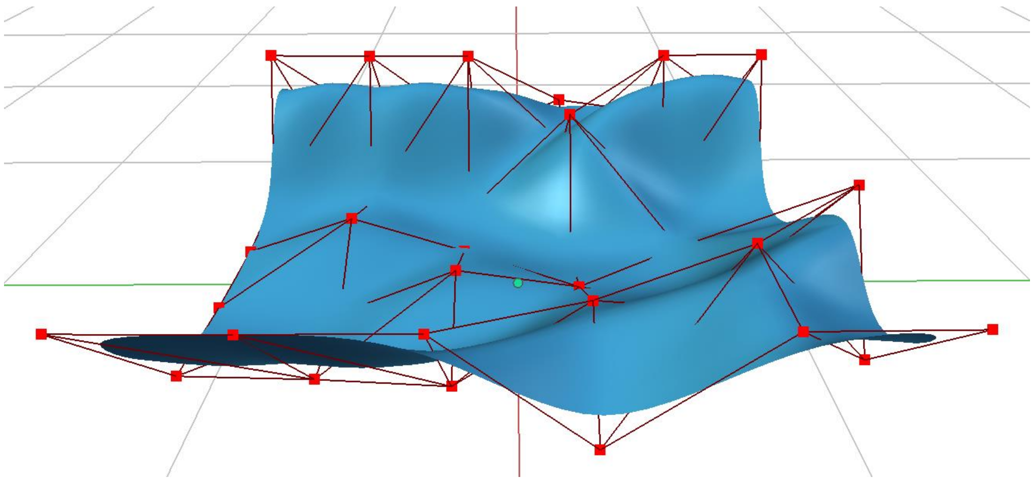


Fig. 2. Surface built by test point set and subdivided with Catmull-Clark algorithm in Blender.

Interpolated surfaces meet the first criterion containing their control points. The interpolation algorithms can widely vary. The most popular are based on the cubic polynomials, like cubic Hermite interpolation mentioned above. The variety and variability of interpolation approaches gives the ground to create a custom algorithm that could meet all the criteria formulated.

4.2. Monotone Interpolation

There are several methods of monotone interpolation in 3D space. For example, the works by M. Abbas et al. [9] and L. Allemand-Giorgis et al. [10] cover C^1-continuous smooth interpolation of gridded data, ensuring absence of misplaced extremes inside the input data domain. These methods normally meet all the criteria indicated in the Section 2, but being restricted by C^1-continuity they are unable to handle corner-cases like very steep slopes (for example, data sets of with non-functional dependency, where some neighbor points have the same abscissa). Also, as it is shown below, these methods are not yet integrated into the popular scientific visualization and visual analytics software.

4.3. Coons Patch

In case the interpolation between 4 control points P_1, P_2, P_3, P_4 is known and represented as curves $c_0(s)$, $c_1(s)$, $d_0(t)$, $d_1(t)$, $s \in [0; 1]$, $t \in [0; 1]$ $c_0(0) = d_0(0) = P_1$, $c_0(1) = d_1(0) = P_2$, $c_1(0) = d_0(1) = P_3$, $c_1(1) = d_1(1) = P_4$, the Coons patch [11] can be constructed to build the smooth surface between these curves using the following formula:

$$C(s,t) = I_c(s,t) + I_d(s,t) - B(s,t), \tag{1}$$

where I represents some interpolation and B represents bi-interpolation.
The interpolation method can vary. For example, in case of linear interpolation for I and correspondingly bilinear for B, the components of formula (1) yield in the following:
$I_c(s,t) = (1 - t)c_0(s) + tc_1(s)$,
$I_d(s,t) = (1 - s)d_0(t) + sd_1(t)$,
$B(s,t) = (1 - s)(1 - t)c_0(0) + s(1 - t)c_0(1) + (1 - s)tc_1(0) + stc_1(1)$.
The result surface is called *bilinearly blended Coons patch*. An example is shown in Fig. 3.

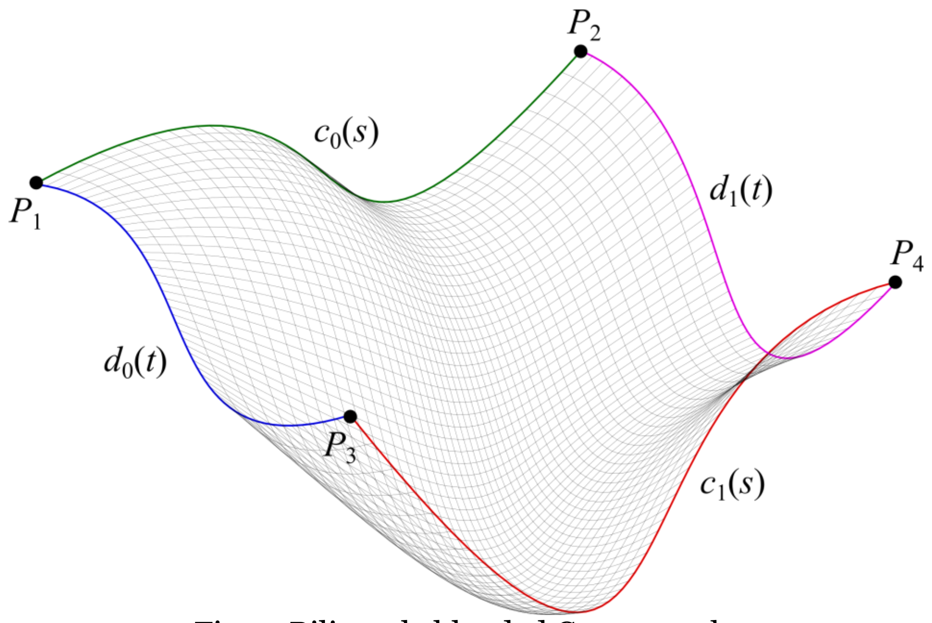


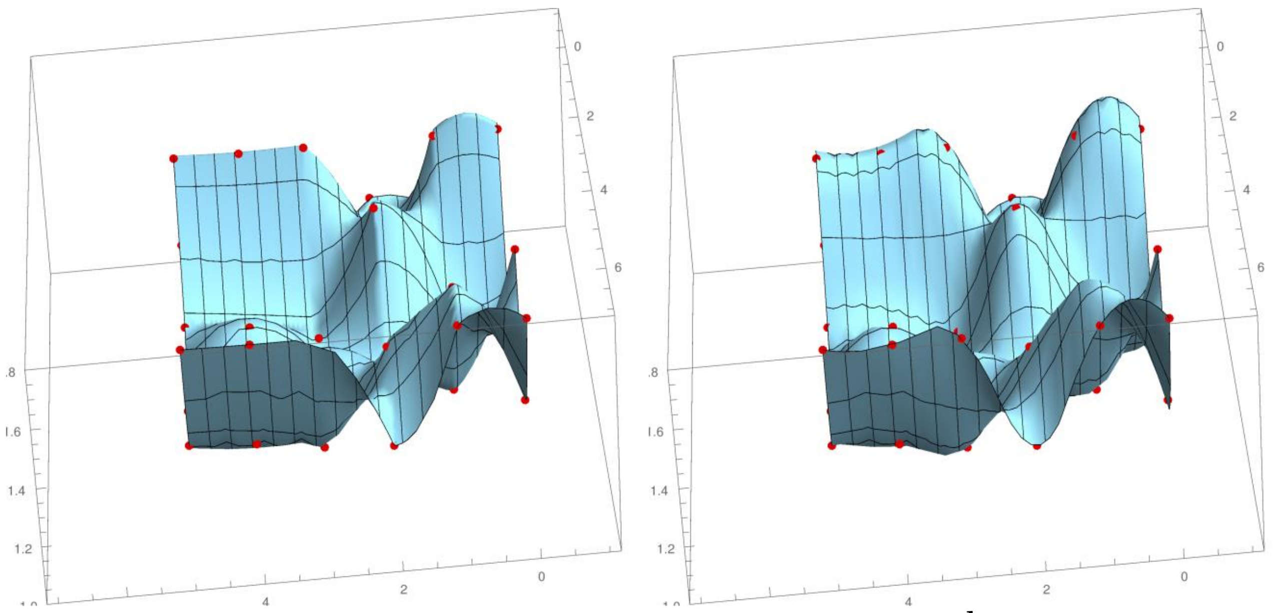
Fig. 3. Bilinearly blended Coons patch.

Discussed Coons patch exactly meets its boundary curves, but if multiple patches are joined, they do not necessarily have the same tangent planes at joint curves leading to the creases along those curves. To fix this problem, I can be cubic and B – bicubic interpolation. To find the exact representation of I and B, cubic Hermite splines can be used with the weights chosen to match the partial derivatives at the corners. The result is called *bicubically blended Coons patch*.
Coons patch is a handy tool to build the surface that potentially can meet the criteria mentioned in the Section 2, but the problem is to find the corresponding curves between control points.

4.4. Building Smooth Surfaces Using Popular Scientific Visualization Software

The modern scientific visualization software that is capable of 3D rendering normally provides functions to build surfaces by given set of points. The most popular tools provide automatic smoothing as well.

One of the most popular and powerful systems incorporating huge amount of mathematical solvers and providing a lot of visualization capabilities is Wolfram Mathematica [12]. Among other functions, it allows building surfaces by the discrete set of points. There are two interpolation modes available: Hermite (function Interpolation with parameter Method->"Hermite") and B-spline [13] (function Interpolation with parameter Method->"Spline"). The results of both applied to the testing data set from the Section 3 are shown in Fig. 4.



a b

Fig. 4. Surface built by Wolfram Mathematica using Hermite interpolation (a) and B-spline interpolation (b).

As it can be seen from the figure, Hermite interpolation (the exact weights are not mentioned in the documentation of Mathematica) is better than B-spline interpolation according to the criteria mentioned in the Section 2, but still has some false extremes (for example, in top-right corner).

A lot of scientific visualization software utilize VTK library [14] under the hood, for example ParaView [15] and its lightweight version for mobile devices KiwiViewer [16]. VTK supports wide range of rendering techniques and data visualization function. It provides abstract class vtkSubdivisionFilter to generalize approximation and interpolation algorithms for building surfaces. Currently, 3 methods are included in the VTK core: vtkLoopSubdivisionFilter that implements smoothing algorithm introduced by C.T. Loop [5], vtkButterflySubdivisionFilter that implements so-called butterfly scheme introduces by D. Zorin et al. [17] and vtkLinearSubdivisionFilter that implements regular linear interpolation (the surface is not smoothed). The rendering results of all these methods applied to the testing data set are shown in Fig. 5.

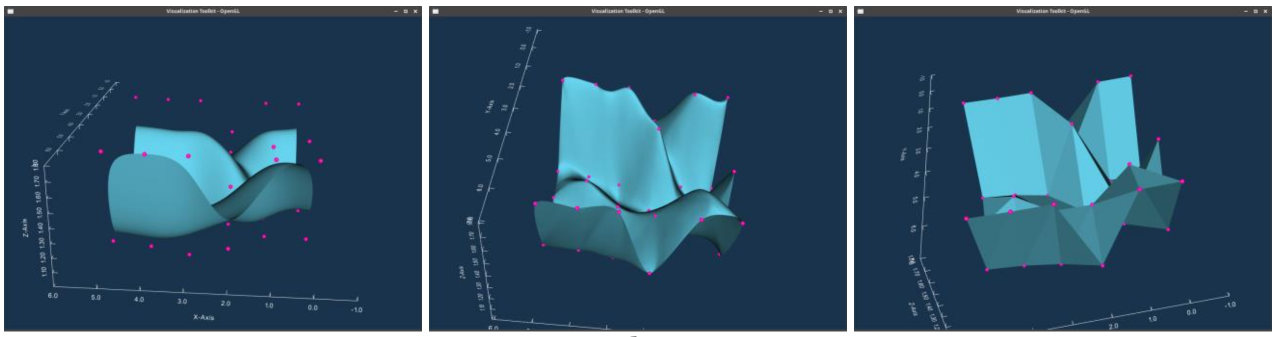

a b c

Fig. 5. Surfaces built by VTK using Loop’s algorithm (a), butterfly algorithm (b) and linear interpolation (c).

As seen from Fig. 5, Loop’s method behaves like NURBS: the surface does not contain the control points. Thereby, this method does not meet the very first criterion. The butterfly filter suites the first criterion, but spawns misplaced extremes and oscillations. The linear subdivision filter does not build sleek surface. Consequently, VTK does not provide the desired smoothing function.
The next popular library for scientific visualization is MathGL [18]. This library is not as versatile as VTK concentrating on the charts only, but it is also used in a wide range of applications requiring high-quality visualization. MathGL generally provides two ways to build the surface by the discrete set of points: spline-based interpolation (accessible with the refill function) and linear interpolation (accessible with the datagrid function). The rendering results are shown in Fig. 6.

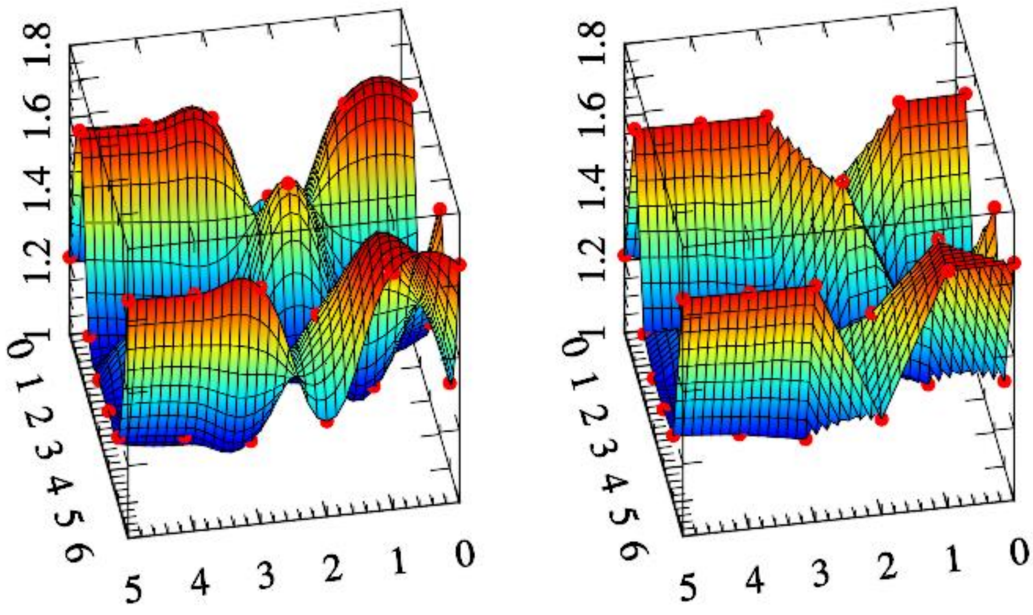

a b

Fig. 6. Surfaces built by MathGL using spline interpolation (a) and linear interpolation (b).

The shape of surface in Fig. 6a is very similar with the one in Fig. 4b. Probably, similar approaches are used. As in the previously considered software, MathGL either builds the surface that is not perceived sleek, or spawns misplaced extremes and oscillations.
Taking into account the above mentioned examples it can be stated that the misplaced extremes of smooth surfaces are still an issue, even in the world-leading visualization software solutions. Thereby the problem of creating the algorithm meeting all the criteria from the Section 2 is an important task in scientific visualization and visual analytics.

5. Proposed Solution

In the previous research we developed an algorithm of building sleek 2D curves without misplaced extremes [3]. The curves consist of cubic Bezier segments with the intermediate control points calculated using a set of heuristics. Taking this algorithm as a background, we propose its 3D generalization.

Assume having $\{P_{ij}\}, i = \overline{1,n}, j = \overline{1,m}$ – a set of input points distributed in as a regular grid in 3D space. Lets assume for disambiguation, that this grid is distributed in XOZ and Y is vertical axis. Building a smooth surface according to this point set consists of the following high-level steps:

1. For $i = \overline{1,n}$, build a smooth curve C_i by the points $P_{ij},\ j = \overline{1,m}$. This curve consists of $m - 1$ Bezier segments denoted as $C_{ik}, k = \overline{1,m-1}$.
2. For $j = \overline{1,m}$, build a smooth curve D_j by the points $P_{ij},\ i = \overline{1,n}$. This curve consists of $n - 1$ Bezier segments denoted as $D_{jl}, l = \overline{1,n-1}$.
3. For $i = \overline{1,n-1}, j = \overline{1,m-1}$, build a Coons patch based on curves C_{ij}, C_{i+1j}, D_{ji}, D_{j+1i} with the constant resolution $R \times R$. Currently no special heuristics for calculating R are developed and it is just an external algorithm parameter.
4. Triangulate the set of points built in step 3. As long as the input point set is assumed to be a regular grid, the triangulation is trivial.
5. Calculate vertex normals using smoothing groups algorithm.
6. Additionally smooth the field of normals with Gaussian blur function.

The asymptotic complexity of this algorithm is $O(mn)$, which suites the mentioned criteria of efficiency. The result of this algorithm applied to the testing data set is shown in Fig. 7.

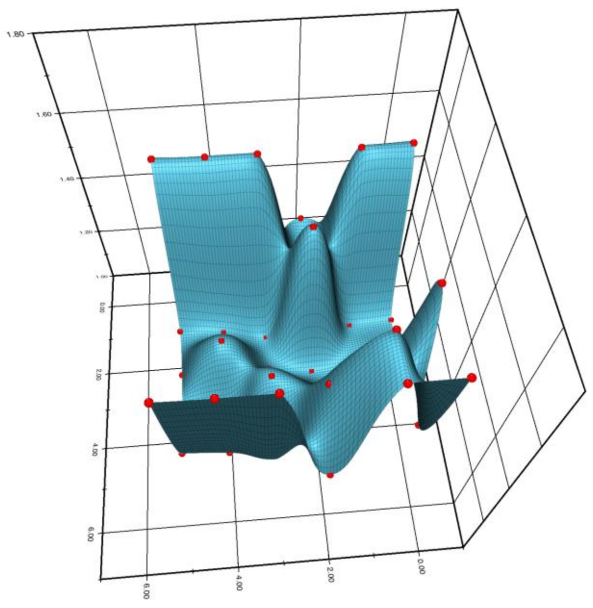


Fig. 7. Surface built by the proposed algorithm.

As it can be seen from the figure, neither misplaced extremes nor oscillations are presented. The above steps are described in details in the upcoming subsections. The results are discussed in the Section 6.

5.1. Building Sleek Curve with Smoothness Order Zero

As a first step of building result surface, its 2D slices are considered and each slice is treated as a piecewise-defined Bezier curve. The key contribution of [3] is the way to calculate intermediate control points $C_{i-1}^{(1)}$, $C_{i-1}^{(2)}$, $C_i^{(1)}$ and $C_i^{(2)}$ to join the neighbor Bezier segments

B_{i-1} and B_i without visible crease as shown in Fig. 8a (this figure is extracted from [3] for the sake of clarity). To ensure the absence of misplaced extremes and oscillations on the result curve, the following conditions should be fulfilled for each segment:

1. The points $C_{i-1}^{(2)}$ and $C_i^{(1)}$ should lay on the tangent to the result curve in the point P_i.
2. The lengths of tangent vectors should be equal: $\left|P_i C_{i-1}^{(2)}\right| = \left|P_i C_i^{(1)}\right|$.
3. The intermediate control points $C_i^{(1)}$ and $C_i^{(2)}$ should belong to the areas A_i and B_i respectively depicted in Fig. 8b.

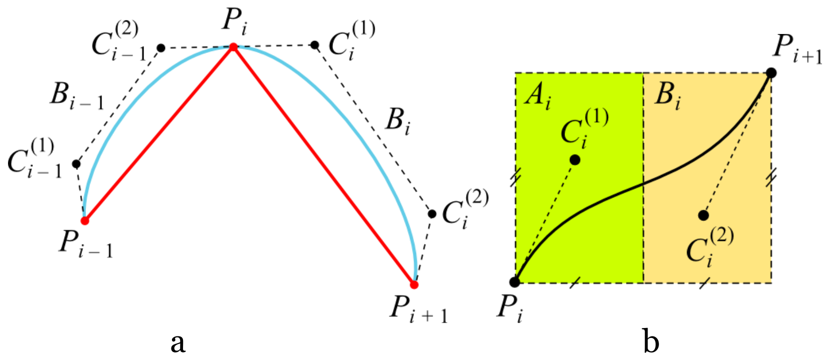


Fig. 8. Joining of two Bezier curves (a) and areas the intermediate control points belong to (b).

The algorithm of meeting the above conditions proposed in [3] is presented in pseudo code in Listing 1. It must be noted, that these conditions altogether generally lead to the curve of smoothness order 0, so formally speaking this curve is not smooth. The creases on the curve appear in the corner cases like equal abscissa of neighbor points or transition from one “plateau” (sequence of points with equal ordinate) to another. The algorithms ensuring smoothness order 1 and higher spawn misplaced extremes in these cases, but our algorithm does not. Instead, it “breaks” the formal smoothness locally in corner cases. Because normally the number of creases is low, the result curve is perceived sleek, which exactly matches the desired criteria of quality.

Listing 1. Pseudo code of the sleek curve building algorithm.

```
1. Input: array of 2D points $P_i$, $i = \overline{1, n}$.
2. Let $\vec{t}_L$ represent the tangent to the Bezier segment in its starting point. The initial value is zero vector.
3. Let $\vec{t}_R$ represent the tangent to the Bezier segment in its ending point. The initial value is zero vector.
4. Let $\vec{c}$ represent the vector from the previous point to the current one.
5. Let $\vec{s}$ represent the vector from the current point to the next one: $\vec{s} = \frac{P_2 - P_1}{|P_2 - P_1|}$.
6. For each $i = \overline{1, n-1}$:
    6.1. Reuse the previously calculated tangent: $\vec{t}_L = \vec{t}_R$.
    6.2. Reuse the previously calculated vector: $\vec{c} = \vec{s}$.
    6.3. Calculate the denormalized difference: $\vec{d} = P_{i+1} - P_i$.
    6.4. Calculate new $\vec{t}_R$:
        6.4.1. If $i < n - 2$, then:
            6.4.1.1. $\vec{s} = \frac{P_{i+2} - P_{i+1}}{|P_{i+2} - P_{i+1}|}$.
            6.4.1.2. If $x_{\vec{c}} = 0$ or $y_{\vec{c}} = 0$, then $\vec{t}_R = \vec{c}$.
            6.4.1.3. Else if $x_{\vec{s}} = 0$ or $y_{\vec{s}} = 0$, then $\vec{t}_R = \vec{s}$.
            6.4.1.4. Else $\vec{t}_R = \frac{\vec{c} + \vec{s}}{|\vec{c} + \vec{s}|}$.
    6.4.2. Else $\vec{t}_R = 0$.
  6.5. Clamp $\vec{t}_L$ and $\vec{t}_R$ to the areas $A_i$ and $B_i$ respectively:
    6.5.1. If $\mathrm{sign}(x_{\vec{t}_L}) \neq \mathrm{sign}(x_{\vec{d}})$, then $x_{\vec{t}_L} = 0$.
    6.5.2. If $\mathrm{sign}(y_{\vec{t}_L}) \neq \mathrm{sign}(y_{\vec{d}})$, then $y_{\vec{t}_L} = 0$.
    6.5.3. If $\mathrm{sign}(x_{\vec{t}_R}) \neq \mathrm{sign}(x_{\vec{d}})$, then $x_{\vec{t}_R} = 0$.
    6.5.4. If $\mathrm{sign}(y_{\vec{t}_R}) \neq \mathrm{sign}(y_{\vec{d}})$, then $y_{\vec{t}_R} = 0$.
  6.6. Let $z_L$ be the flag indicating whether $x_{\vec{t}_L} = 0$.
  6.7. Let $z_R$ be the flag indicating whether $x_{\vec{t}_R} = 0$.
  6.8. Calculate $l_L$ and $l_R$ – lengths of the corresponding tangents for the current Bezier
       segment (assuming $W \in [2; +\infty)$ – algorithm's parameter):
    6.8.1. If $z_L$ is true, then $l_L = 0$, else $l_L = \frac{x_{\vec{d}}}{W x_{\vec{t}_L}}$.
    6.8.2. If $z_R$ is true, then $l_R = 0$, else $l_R = \frac{x_{\vec{d}}}{W x_{\vec{t}_R}}$.
    6.8.3. If $|l_L y_{\vec{t}_L}| > |y_{\vec{c}}|$, then:
      6.8.3.1. If $y_{\vec{t}_L} = 0$, then $l_L = 0$, else $l_L = \frac{y_{\vec{c}}}{y_{\vec{t}_L}}$.
    6.8.4. If $|l_R y_{\vec{t}_R}| > |y_{\vec{c}}|$, then:
      6.8.4.1. If $y_{\vec{t}_R} = 0$, then $l_R = 0$, else $l_R = \frac{y_{\vec{c}}}{y_{\vec{t}_R}}$.
    6.8.5. If both $z_L$ and $z_R$ are false, then:
      6.8.5.1. Let $\Delta = \frac{y_{\vec{t}_L}}{x_{\vec{t}_L}} - \frac{y_{\vec{t}_R}}{x_{\vec{t}_R}}$.
      6.8.5.2. If $\Delta \neq 0$, then:
        6.8.5.2.1. Let $x = \frac{1}{\Delta}\left(y_{P_{i+1}} - \frac{y_{\vec{t}_R}}{x_{\vec{t}_R}} x_{P_{i+1}} - y_{P_i} + \frac{y_{\vec{t}_L}}{x_{\vec{t}_L}} x_{P_i}\right)$.
        6.8.5.2.2. If $x > x_{P_i}$ and $x < x_{P_{i+1}}$, then:
          6.8.5.2.2.1. If $|l_L| > |l_R|$, then $l_L = 0$, else $l_R = 0$.
    6.8.6. Create the Bezier segment $B_i$ with the following control points: $P_i$, $P_i + l_L t_L$,
           $P_{i+1} + l_R t_R$, $P_{i+1}$.
```

5.2. Building Coons Patches

According to the assumption, the input points are distributed in the regular grid. Each cell of this grid is an area between 4 neighbor points bounded with 4 corresponding Bezier segments obtained in the previous steps. To build the result surface, each cell is treated as a Coons patch and the intermediate points inside this cell are calculated according to the formula (1) with the resolution R, which means, $R^2 - 4$ new points are emitted.

To ensure better shading of the result surface, its wireframe should be as close to the regular grid as possible. However, each cell is bounded by parametric Bezier segments, and the result points depend nonlinearly on the parameter. This means, if the parameter changes linearly, X and Z coordinates of result points change nonlinearly. To ensure a regular grid, the parameter should be changed in a nonlinear way.

The cubic Bezier segment is calculated as follows:

$$B_i(t) = (1-t)^3 P_i + 3t(1-t)^2 C_i^{(1)} + 3t^2(1-t) C_i^{(2)} + t^3 P_{i+1}, (2)$$

where P_i, P_{i+1} are the points from the input data set,

$C_i^{(1)}$, $C_i^{(2)}$ are the intermediate control points calculated according to the algorithm shown in Listing 1,

$t \in [0; 1]$.

Each bounding curve is parallel to either X- or Z-axis being a part of regular grid. This means, only the pairs $\{X, Y\}$ or $\{Z, Y\}$ are calculated by (2).
Consider the curve parallel to X-axis. Z-coordinates of its points are all the same and X changes according to (2). Lets assume, the parameter t is changed linearly from 0 to 1 with the step $^1/_R$. To ensure linear changing of X-coordinate, B_i should be calculated using the new parameter t' that should be found by solving the following equation:

$$x_{P_i} + t\,(x_{P_{i+1}} - x_{P_i}) = (1-t')^3\, x_{P_i} + 3t'(1-t')^2\, x_{C_i^{(1)}} + 3t'^2(1-t')\, x_{C_i^{(2)}} + t'^3 x_{P_{i+1}}. \quad (3)$$

This equation can be solved by well-known Cardan formula. After this, the points B_i become equidistant in X-direction. The similar calculations are applied to the curves parallel to Z-axis. As a result, the regular wireframe for each Coons patch is ensured.
The next problem to be solved is the joining of neighbor patches. Using the bilinear blending in formula (1) results in the surface shown in Fig. 9. The field of normals is calculated using trivial smoothing groups algorithm [19]: each vertex normal is an average of normals of incident triangles.

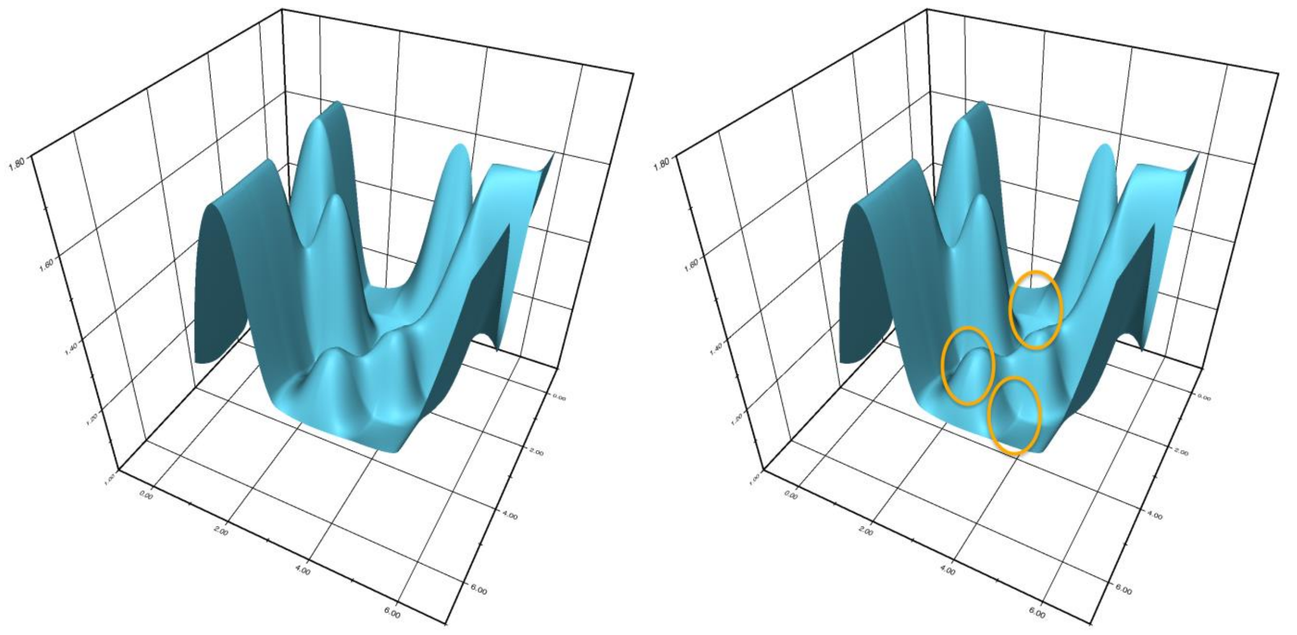


Fig. 9. Surface built using bilinearly blended Coons patches without marks (a) and with the visible creases marked by ovals (b).

As it can be seen, the creases are visible disturbing the sleek appearance (for the sake of clarity, the most problematic places are marked with ovals in the Fig. 9b). The common solution of this problem is using bicubic blending instead of bilinear one.
The cubic interpolation can be expressed by the following formula [20]:

$$f(P_1, P_2, P_3, P_4, t) = t^3\left(-\frac{1}{2}P_1 + \frac{3}{2}P_2 - \frac{3}{2}P_3 + \frac{1}{2}P_4\right) + t^2\left(P_1 - \frac{5}{2}P_2 + 2P_3 - \frac{1}{2}P_4\right) + \\ + t\left(-\frac{1}{2}P_1 + \frac{1}{2}P_3\right) + P_2 \quad (4)$$

Bicubic interpolation yields in the following [20]:

$$g(s,t) = \\ f(f(P_{11}, P_{12}, P_{13}, P_{14}, t), f(P_{21}, P_{22}, P_{23}, P_{24}, t), f(P_{31}, P_{32}, P_{33}, P_{34}, t), f(P_{41}, P_{42}, P_{43}, P_{44}, t), s) \quad (5)$$

The bicubic blending in formula (1) is the based on formulas (4) and (5). The non-trivial part is that this kind of blending requires neighbor curves to build the current patch as shown in the Fig. 10.

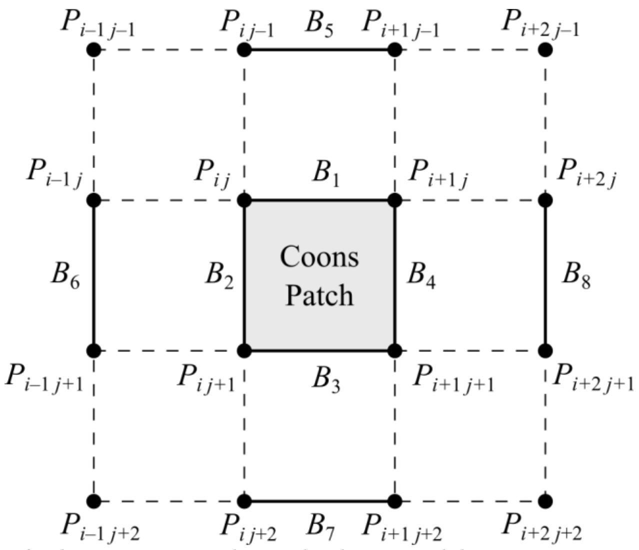


Fig. 10. Bicubically blended Coons patch with the neighbor curve segments that are used in bicubic interpolation.

In the corner cases, where some of B_5, B_6, B_7, B_8 do not exist (on the surface' boundary), the non-existing segments are assumed to be equal to B_1, B_2, B_3, B_4 respectively for the sake of unification.
The result of bicubically blended Coons patches is shown in the Fig. 11.

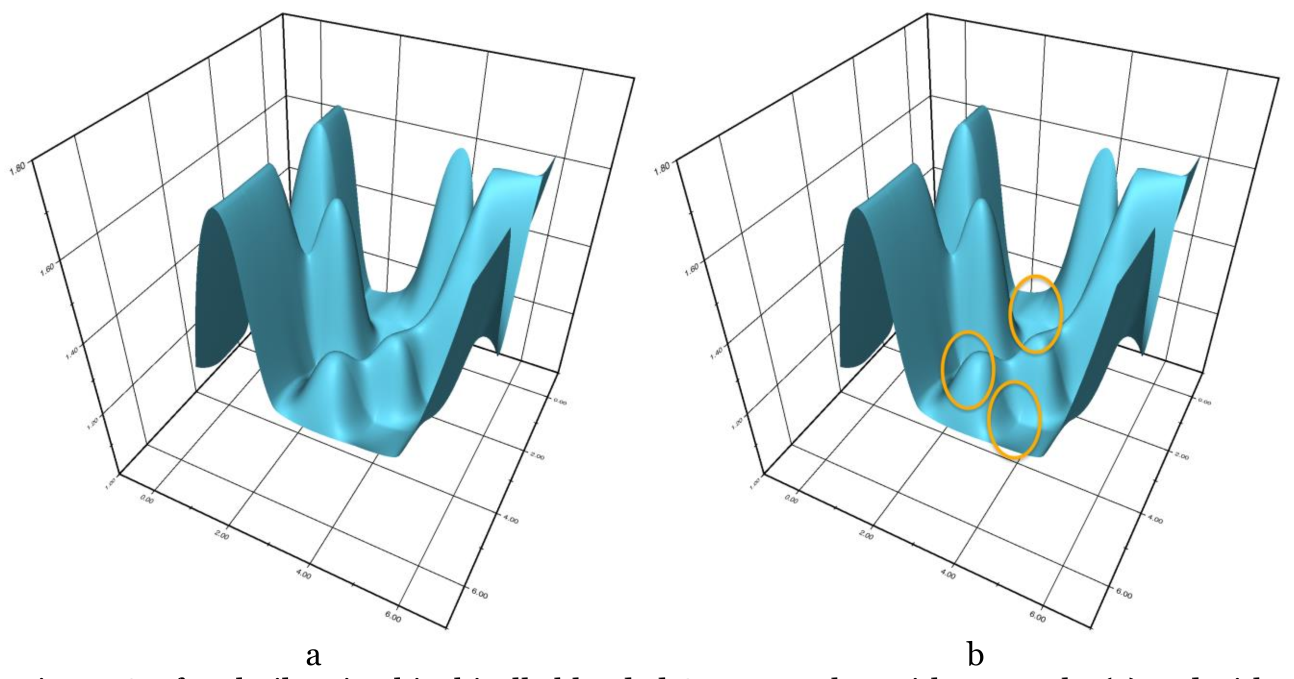


Fig. 11. Surface built using bicubically blended Coons patches without marks (a) and with the visible creases marked by ovals (b).

As it can be seen, the quality gets higher, but still is not high enough. The problem is, that the initial curves indeed have non-continuous derivative in these places. While they are perceived sleek when viewed in 2D, the shading of corresponding 3D surface makes the creases remarkable, because it is calculated by non-smooth normals' field.
The possible solutions are either to increase the radius of smoothing groups used for calculating vertex normals, or to perform artificial smoothing of normals' field.

5.3. Smoothing the Field of Normals

We decided to remove the creases by applying the smoothing to the field of normals. This approach appears more flexible because enables different smoothing functions. We have chosen Gaussian blur filter, because it effectively smooths out the values preserving the high influence of the median and thereby not spoiling the surface' curvature information represented by the field of normals.

We apply the Gaussian blur filter as a convolution like it is traditionally done in image processing. The vertex normals are processed componentwise. To build the convolution kernel, the following formula is used:

$$G(i,j) = e^{-\frac{2}{r^2}\left((i-r)^2+(j-r)^2\right)}, \tag{6}$$

where r is the blur radius,

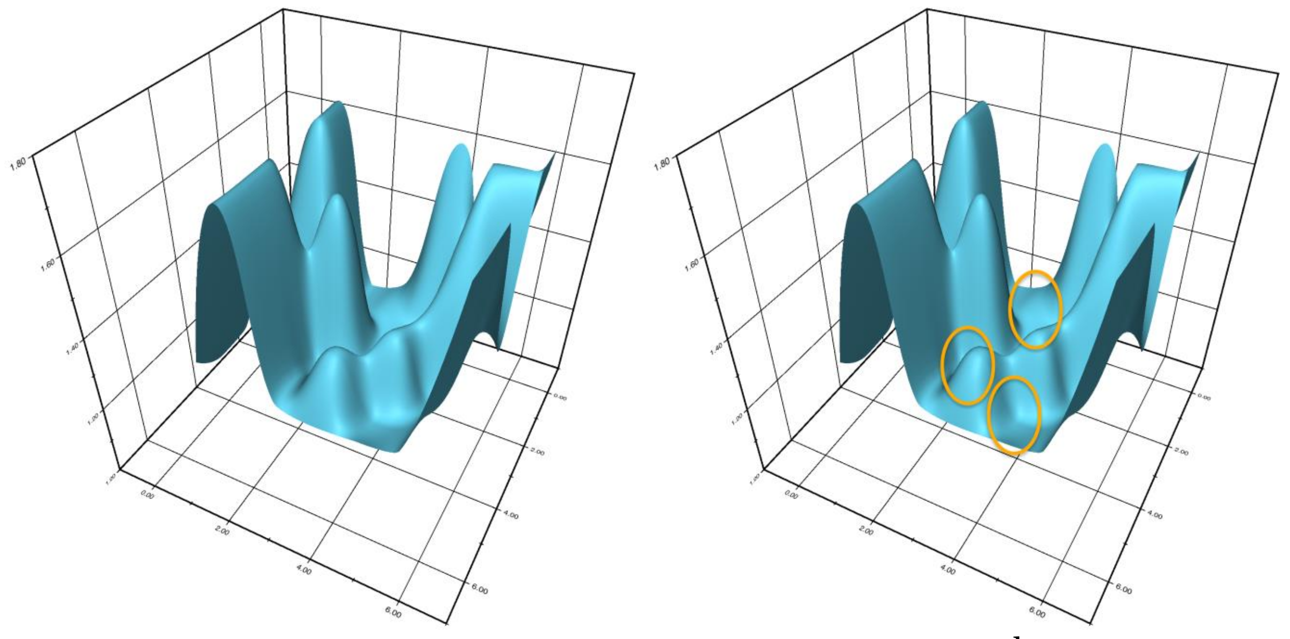
$i, j \in \overline{1, 2r+1}$ – indices of kernel items.

The kernel is not normalized; instead, the normals are renormalized after blurring.

The radius r is a parameter to be tuned. It affects the strength of blur; therefore it should be big enough to remove the creases, but not too big to preserve the curvature information of the surface (or the shading gets unnatural because the normal won't reflect the actual surface shape). Obviously this parameter depends on the resolution, because the bigger is the resolution, the more vertex normals are on the surface, the bigger should be the kernel to cover the areas with the creases. We conducted a lot of experiments and found out, that the acceptable balance between smoothness and correct shape shading is achieved when the radius is about one fifth of the resolution. So, we propose the following formula:

$$r = \left[\frac{R}{5}\right]. \tag{7}$$

For example, for the resolution 17 the kernel radius will be 3. The result rendering after smoothing the normals field with Gaussian filter using the mentioned parameters is presented in Fig. 12.



a b

Fig. 12. Surface built using bicubically blended Coons patches and Gaussian filtering of the normals field without marks (a) and with the ovals marking the places the creases used to be in the previous steps (b).

We consider the visual quality of the obtained result high enough.

6. Results and Discussion

As demonstrated on the test data set, the proposed method allows building the surfaces perceived sleek and having no misplaced extremes. The core of the developed algorithm is the method of building sleek 2D curves avoiding misplaced extremes that we proposed more than a year ago [3]. Since then, this algorithm was intensively tested in the production by visualizing different real-world data within the multiplatform charting library NChart3D [21] and scientific visualization system SciVi [22]. The proposed generalization of this algorithm to the 3D case is as well integrated into both NChart3D and SciVi. It already runs in production delivering high-quality results.

According to this testing we can state that the proposed solution completely suites the criteria mentioned in the Section 2 and is thereby applicable for solving the corresponding scientific visualization and visual analytics tasks.

The proposed algorithm has the following limitation: it assumes, the points of the input set are distributed in the regular grid, covering this grid completely without tears. To bypass this limitation, some kind of regularization should be applied to the initial data set as a preprocessing stage. The regularization can be based on interpolation to fill up the tears in the grid. However in the case of unstructured (irregular) grid or a sparse grid the proposed method in its current form is hardly usable.

7. Conclusion

In the present work we considered the problem of high-quality graphical representation of functional dependencies expressed as $y = f(x, z)$. By investigating the well-known software capable of 3D surface building and rendering we found out that the misplaced extremes and oscillations are an issue of almost all modern interpolation techniques in 3D case.

We proposed an approach to build sleek 3D surfaces interpolating the given discrete set of points and avoiding misplaced extremes and oscillations. This approach ensures rendering results of high visual quality and can be used by solving scientific visualization and visual analytics tasks. The developed algorithm was integrated into the software NChart3D and SciVi and used in solving real-world visualization tasks in various application domains. During its usage in the production it proved its correctness and stability.

The only limitation is the requirement that the input data should be presented in a form of a regular grid. As a future work we plan to investigate the possible ways to bypass this limitation without much computational effort.

The authors' implementation of the proposed algorithm written in C++ is available on GitHub under terms of MIT license [23]: https://github.com/icosaeder/sleek-surface.

References

1. Weisstein, E.W. NURBS Surface [Electronic Resource] // MathWorld – A Wolfram Web Resource. URL: http://mathworld.wolfram.com/NURBSSurface.html (last accessed 01.09.2018).
2. Spitzbart, A. A Generalization of Hermite's Interpolation Formula // The American Mathematical Monthly. – Mathematical Association of America, 1960. – Vol. 67, No. 1. – PP. 42–46.
3. Ryabinin, K.V. Visualization of Smooth Curves without Misplaced Extremes Based on Discrete Point Set // Scientific Visualization. – National Research Nuclear University “MEPhI”, 2017. – Q. 1, Vol. 9, No. 1. – PP. 50–72.
4. Weisstein, E.W. Smooth Surface [Electronic Resource] // MathWorld – A Wolfram Web Resource. URL: http://mathworld.wolfram.com/SmoothSurface.html (last accessed 01.09.2018).

5. Loop, C.T. Smooth Subdivision Surfaces Based on Triangles // Master Thesis. – Department of Mathimetics, University of Utah, 1987. – 74 P.
6. Sederberg, T.W., Zheng, J., Bakenov, A., Nasri, A. T-splines and T-NURCCs // ACM Transactions on Graphics. – 2003. – 22, 3 (July). – PP. 477–484.
7. Catmull, E., Clark, J. Recursively generated B-spline surfaces on arbitrary topological meshes // Computer-Aided Design. – Elsevier, 1978. – Vol. 10 I. 6. – PP. 350–355. DOI: 10.1016/0010-4485(78)90110-0.
8. Blender 3D Editor [Electronic Resource]. URL: https://www.blender.org/ (last accessed 01.09.2018).
9. Abbas, M., Majid, A.A., Awang, M.N.H., Ali J.M. Monotonicity-Preserving Rational Bi-Cubic Spline Surface Interpolation // ScienceAsia. – 2014. – Vol. 40S. – PP. 22–30. DOI: 10.2306/scienceasia1513-1874.2014.40S.022.
10. Allemand-Giorgis, L., Bonneau, G.-P., Hahmann, S., Vivodtzev, F. Piecewise Polynomial Monotonic Interpolation of 2D Gridded Data // Topological and Statistical Methods for Complex Data – Springer, 2014. – PP. 73–91.
11. Coons, S.A. Surfaces for Computer-Aided Design of Space Forms // Technical Report. – Project MAC, MIT, 1967. – 105 P.
12. Wolfram Mathematica [Electronic Resource]. URL: https://www.wolfram.com/mathematica/ (last accessed 01.09.2018).
13. Lee, E.T.Y. A Simplified B-Spline Computation Routine // Computing. – Springer, 1982. – Vol. 29, I. 4. – PP. 365–371.
14. VTK Library [Electronic Resource]. URL: https://www.vtk.org/ (last accessed 01.09.2018).
15. ParaView Scientific Visualization System [Electronic Resource]. URL: https://www.paraview.org/ (last accessed 01.09.2018).
16. KiwiViewer Scientific Visualization System [Electronic Resource]. URL: https://www.kitware.com/kiwiviewer/ (last accessed 01.09.2018).
17. Zorin, D., Schröder, P., Sweldens, W. Interpolating Subdivision for Meshes with Arbitrary Topology // SIGGRAPH'96 Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. – ACM, 1996. – PP. 189–192.
18. MathGL Library [Electronic Resource]. URL: http://mathgl.sourceforge.net/doc_en/Main.html (last accessed 01.09.2018).
19. Ryabinin, K.V. Virtual Reality and Multimedia. Building a Virtual World with OpenGL // Tutorial Book. – Perm State University, 2018. – 100 p.
20. Breeuwsma, P. Cubic interpolation [Electronic Resource]. URL: http://www.paulinternet.nl/?page=bicubic (last accessed 01.09.2018).
21. Ryabinin, K.V. Development of Multiplatform Charting Library // Scientific Visualization. – National Research Nuclear University "MEPhI", 2014. – Q. 1, Vol. 6. No. 1. – PP. 51–67.
22. Ryabinin, K.V., Chuprina, S.I. Development of Ontology-Based Multiplatform Adaptive Scientific Visualization System // Journal of Computational Science. – Elsevier, 2015. – Vol. 10. – P. 370–381.
23. The MIT License [Electronic Resource]. URL: https://opensource.org/licenses/MIT (last accessed 01.09.2018).