# Real-Time Deformations of Function-Based Surfaces using Perturbation Functions

IOP Conf. Series: Journal of Physics: Conf. Series **1015** (2018) 032115 doi:10.1088/1742-6596/1015/3/032115

# Real-Time Deformations of Function-Based Surfaces using Perturbation Functions

**S I Vyatkin[1], A N Romanyuk[2], L A Savytska[2], T I Troianovska[2], N V Dobrovolska[3]**

[1]Institute of Automation and Electrometry, Siberian Branch of the Russian Academy of Sciences, Academician Koptug Ave. 1, Novosibirsk, Russia, 630090, sivser@mail.ru
[2]Vinnitsa National Technical University, Khmelnytsky Highway 95, Ukraine, 21000, rom8591@gmail.com
[3]Vinnytsia Institute of Trade and Economics of Kyiv National University of Trade and Economics Soborna St., 87, Vinnytsia, Ukraine, 21050, natali0212@ukr.net

**Abstract.** This paper presents an adaptive method for animating dynamic deformable objects using geometric operations. Transformations of geometric objects are described for offsetting and morphing. Free forms constructed on the basis of perturbation functions have the advantage of spline-representation of surfaces, thus achieving a high degree of smoothness and is an advantage of an arbitrary form for a small number of such perturbation functions. The user interacts in real-time with the dynamic object through the control of a tool, attached to a mouse device driven with forces derived from the method.

## 1. Introduction

Animating deformable objects in real-time are essential for many interactive virtual reality applications, such as surgery simulators, etc. An important point for a successful immersion is the liveliness of objects: deformations should be dynamic, and not just a succession of static postures. Another essential point is to make a strict guarantee for real-time use of GPU. The first thing that is necessary to state is that if one wants to propagate the deformation, it should be somehow added to every object/all objects that it affects. Actually the current scene-tree is organized so that there is no possibility to add any object only by referencing, i.e. without copying. This is done for avoiding situations when an object, changed somewhere, changes unintentionally the other part of the scene that is referenced to it too. Thus the additional perturbations should have parameters to assure the part-per-part connectivity for each pair of the object with the perturbation affects. A method of solving the problem using an adaptive physically-based model is proposed in the paper.

## 2. Geometric objects

The functionally defined object is completely determined by means of a real-valued description function of three variables (x1, x2, x3) in the form of $F(X) \geq 0$, then these objects are considered as separate closed subsets of the Euclidean space En, defined by this describing function $F(X) \geq 0$, where F is a continuous real-valued function and the X= (x1, x2, x3) is the point in Euclidean space En, and defined by the coordinate variables. Here function $F(X) > 0$ defines proper points inside the object,

function F(X) = 0 defines the boundary points, and function F(X) < 0 defines points which lie outside and do not belong to this object.

It is proposed to describe the complex geometric objects by specifying the function of deviation (an implicit second-order function) from the base surfaces [1]. So, the freeform is a composition of the base surface (surfaces) and the perturbation function:

$$F'(x,y,z) = F(x,y,z) + \sum_{i=1}^{N} R_i(x,y,z)$$

where the given perturbation function R(x, y, z) is defined as follows:

$$R_i(x,y,z) = \begin{cases} Q_i^3(x,y,z), if\ Q_i(x,y,z) \geq 0 \\ 0, if\ Q_i(x,y,z) < 0 \end{cases}$$

Here, Q(x, y, z) is the perturbing quadric.

It is assumed that max[Q + R] ≤max[Q] + max[R]. In order to estimate the maximum value of Q in some interval, it is necesarry to calculate the maximum value of the perturbation function of one definite interval. As a result, the resulting surfaces are smooth and require to create complex surface forms requiring few perturbation functions.

Thus, the problem of object construction is reduced to the problem of quadric surface deformation in a desired manner rather than to approximation by primitives (polygons or patches represented by B-spline surfaces).

## 3. Deformations of the surfaces

The set of geometric operations Φ is expressed mathematically in the following form [2]:

$$\Phi_i: M^1+M^2+\ldots+Mn \rightarrow M,$$

where n is the number of operation operand.

Let the object G1 be defined as $f1(X) \geq 0$. Unary operation (n = 1) of G1 means operation $G2=\Phi_i(G1)$ with definition $f2=\psi(f1(X)) \geq 0$, where ψ is a continuous real function of one variable.

The following advanced geometric operations should be considered in more detail.

Offsetting operations in geometric modeling generate expanded or contracted versions of an original object. These operations have been applied: to plan collision-free trajectories of a computer-controlled manipulator; to describe cutter path generation for numerically controlled machining; in the algorithms for computing volume, moments of inertia and other integral properties of solid; in design of mold profiles; to generate fillets and rounds in modeled parts, such as constant-radius offsetting [3].

There are several mathematical definitions for offsetting. Normal offsetting defines an offset curve or surface as a set of points of an initial curve displaced along the normal direction by a given distance. Constant-radius offset is a set of points with the fixed minimal distance to an initial point set. This operation is a Minkowski or vector sum of an initial object and a disk or a ball. It also can be regarded as sweeping by a disk with its center moving throughout an initial object. For an initial parametric curve a distance-like implicit function can be defined as an offset curve with a level set of this function.

Iso-valued offsetting is:

$$F = f(X) + C,$$

Negative constant C defines negative (internal) offset; positive C defines positive (external) offset.

Offsetting along the normal is:

F = f(X+DN), for positive (external) and

F = f(X-DN), for negative (internal) offsetting,

where D is the given distance value, N is the gradient vector of function f in the point with X coordinates.

This definition differs from standard one. The gradient is calculated in a given point.

Positive constant-radius offsetting with distance D: define an initial object by $f(x,y,z) \geq 0$; for any given point $(x,y,z)$ defined the sphere of radius D with the center in the point; find maximal value $f_0$ of $f(x',y',z')$ for points $(x',y',z')$ belonging to the sphere's surface; assign $f(x,y,z) = f_0$.

The offsetting operation with perturbation functions was implemented by transformation of perturbation function coefficients. Thus, one can create an enlarged or diminished copy of the initial object, i.e., make positive or negative offsetting, respectively. For example, solid beats can be simulated. Let the initial object be defined by the function $f(X)>0$, then in the case of this operation, the obtained solid will be described by the function $F=f(X)+C$, where $C<0$ determines the negative offsetting (compression) and $C>0$ determines the positive offsetting (extension). Otherwise, adding together the positive or negative constant and the free term of the perturbation function yields to extension or compression of the whole object.

3D morphing operation transforms the first defined object into the second one with obtaining multiple intermediate forms. The term originates from the word metamorphosis and refers to the animation technique in which one pattern is gradually transformed to another one. During the metamorphosis (morphing), the initial pattern is gradually transformed to the final one (Fig 1).

A sequence of frames of transformation of one object to another is generated by means of the initial, final, and key intermediate models.

Let F1 and F2 be values of the perturbation functions of the first and second objects, respectively. Then the resulting perturbation function is calculated as follows:

$$F=\beta F1 + (1- \beta)F2,$$

where β is the positive continuous function.

For function-based objects with the use of perturbation functions, one can perform 3D morphing of nongomeomorphic objects.

To create a complex scene, it is necessary to describe a certain number of primitives necessary for a particular problem. The entire three-dimensional scene is presented by an object, which is contacted by the rasterization algorithm by means of queries. Therefore, the geometric model should allow one to design objects and their compositions of arbitrary complexity. This is primarily reached by using Boolean operations of union and intersection. The entire scene has the form of a tree, each node of this tree is a design object, which performs logical operations with its descendants, and the top of the tree consists of primitives used by the system. When the rasterization algorithm addresses the design object with a certain query, this object asks his descendants, transforms the result, and provides an appropriate answer to the query. The descendant can be a primitive or another design object. If geometric operations, rotations, displacements, or scaling is applied to the design object, it repeats all these operations with his descendants.

Thus, the scene is described by a binary tree whose root is the object representing the entire three-dimensional scene and possessing a property of answering a query about its intersection with a volume. The nodes of the tree contain unite or intersection operators. The operator is also an object, while the leaves are objects expressed by quadrics.

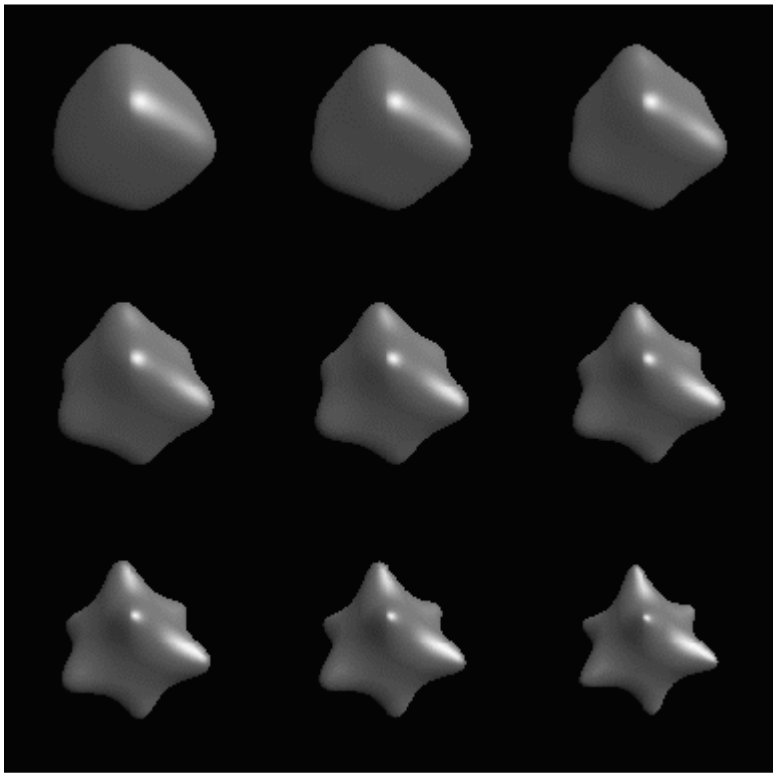

**Figure 1.** Animated 3D surface

The library of classes of functionally defined objects has a static chart for names of objects registered in the library and for creating functions. The names of the registered objects are automatically defined as lexical units in the syntax analysis of the scene. The goal of the preparatory phase is to construct a geometric model. The input data for the code are several text files, and each file describes a logically self-consistent element of the scene and all the necessary parameters of this element. A pre-processor translates these files into one file containing a complete description of the scene. The resultant file is transferred to the input of the syntax analyzer (parser), which analyzes the text structures and constructs the model. When the model is successfully constructed, it is subjected to a projective transformation, which finalizes the pre-processing phase.

The problem of object design is reduced to the problem of deformation of the basic surface in a required manner rather than to its approximation by primitives. This process resembles play dough sculpturing with the use of geometric operations described in it [4]. The coordinates of the points of the deformed surface are determined by the method of detection of collisions of functionally defined objects. This method of collision detection is based on the relation of intersection of the objects and recursive division of the object space for searching for the first point of the contact between the objects.

Deformation implies a possibility of adding a perturbation with parameters defined by the tool to an arbitrary point on the surface. The tool is chosen by the user (e.g., a thin cylinder) and defines the perturbation type and the domain of its action. For this purpose, a library of objects (tools) is created, i.e., a perturbation with prescribed parameters can be added to an arbitrary point of the surface by using a chosen tool. There is also a function of object extraction from the scene, aimed at performing certain operations with the object: rotation, compression–extension, motion, etc. Thus, the perturbation is added onto the object surface at a place indicated by the mouse. As it was already mentioned, the data are stored in the memory in the form of a tree, where each object of the scene is presented as a description of the function with its properties. The tree is shown on the screen; it is possible to copy, cut, and paste individual branches or the tree as a whole. It is also possible to open a window of

properties for each element of the tree and change the element properties in this window. The actions in the main window of the code are performed over the beloning to the object and for adding the depth levels to the object and references to the parent, the code has a flag, which signalizes sub tree participation in rendering. After that, the reference to the object is stored if this object has no descendants. Thus, the tree roots are found. A field in the class VxObject is created, which indicates the tree depth. The sub tree depth is calculated directly before finding the minimum common parent for all objects in the list. If the depth is calculated during scene construction, it may become necessary to recalculate the depths of all nodes after the unite or intersection operations, which requires a lot of time; for this reason, only the depths of the roots are calculated. An important issue is the possibility of deformation of the initial shape, not only of adding new perturbations.

One part of the interacting system is the VxDemo code designed for visualization of functionally defined objects. The input data for VxDemo are contained in a text file (with the SCN extension) created in some text editor or interactively. The file contains the scene description. The text of the file contains a set of lexical units, which can be conventionally divided into the following categories: graphical primitives, properties of primitives, operators, texture, and illumination parameters.

The output data are the resultant images of the scenes in the graphical format. The parameters of scene visualization can be changed in the Options window, which is opened by using the corresponding item in the View window.

The tool defines the perturbation type and the domain of its action. For this purpose, the user is offered certain graphical information (including extraction of the object, e.g., by a certain color or by a bounding box, drawing of axes, etc.). All drawings supporting user's work with operations are performed by OpenGL for obtaining a higher speed and functionality.

Thus, the main specific feature of the rendering class is extraction into a separate thread, OpenGL support (compatibility of buffers, depth testing. etc.), and also the formation of bounding boxes for the objects.

An individual thread is necessary to ensure a possibility of interrupting and restarting of rendering after any change in the scene performed by the user.

The following problems were solved: response of the code to events indicated by the mouse (choosing and modifying the objects), extraction of the object (or several objects) in the scene and possibility for the user to perform some operations with the object(s); choosing the operation — affine transformations MRS (move, rotate, scale), geometric operations, and deformation, possibility of working with the list of tools, recording into a file and loading of the tree of the scene, and possibility of creating new tags and their recognition.

The code implementing the method of interactive modeling of functionally defined objects is written in the high-level language C++. This language is well suitable for the object model of the described system because it is an object-oriented language.

The necessary methods and classes of C++ were developed and implemented: the method of binary search for image elements, classes of functionally defined objects, classes of rendering of functionally defined objects, and classes of the system interface of the interactive volume-oriented geometric modeling with a possibility of implementing a simple mechanism of extension of new algorithms and characteristics.

These hierarchical classes compose the core of the computational system, which can be extended to add other functions or to change the characteristics.

The visualization time is reduced by using the computational resources of a graphics processing unit with compute unified device architecture (CUDA) (NVIDIA). The CUDA system is a parallel programming model that allows implementing programs in C on a standard graphics processing unit. The result of running the programs on different processing units is the same even if they may have a different number of streaming multiprocessors. Among the functions of the graphics processing unit there was the calculation of the coordinates of points of the surfaces, normals, and illumination. Geometric transformations were performed by the central processing unit (CPU). The DirectX application programming interface was used for visualization. Testing was performed on Intel Core2

CPU E8400 3.0 GHz, GPU and 9800 GT 470 GTX processors [5].

## 4. Conclusion

In this present paper, the authors have considered the deformations on function-based objects. New methods for program implementation of complex geometric offsetting and morphing operations on the basis of the perturbation functions have been proposed. Thus, the proposed method for describing objects of 3D scenes by the base surfaces and the perturbation functions has a more compact description than the known methods for defining the function-based objects. Many systems for interactive modeling and editing of functionally defined shapes involve intermediate triangulation of the edited surface for the purpose of decreasing the computation time, which complicates the modeling method itself and the entire system as a whole. Moreover, the computation accuracy is deteriorated by these intermediate manipulations. It should be also noted that successful triangulation is not possible for all geometric shapes; hence, there are constraints on the complexity of modeled shapes. In the proposed approach, the authors managed to avoid the additional triangulation operation, while the interactive regime was retained. This is ensured by using appropriate methods of definition of three-dimensional objects, visualization, and detection of object collisions, geometric operations, and possibility of local and global deformation.

## References

[1] Vyatkin S I 2007 Complex Surface Modeling Using Perturbation Functions *Optoelectronics, Instrumentation and Data Processing. Growth* **43(3)** 40-47

[2] Ricci A 1973 A constructive geometry for computer graphics *The Computer Journal. Growth* **16(2)** 157-160

[3] Rossignac J R and Requicha A A G. 1986 Offsetting operations in solid modeling *Computer Aided Geometric Design*. *Growth* **3** 129-148

[4] Vyatkin S I 2014 An Interactive System for Modeling, Animating and Rendering of Functionally Defined Objects *American Journal of Computer Science and Engineering Survey*. *Growth* **2(3)** 102−108

[5] Vyatkin S I 2014 Method of binary search for image elements of functionally defined objects using graphics processing units *Optoelectronics, Instrumentation and Data Processing*. *Growth* **50(6)** 606–612