Original Article

An Interactive System for Modeling, Animating and Rendering of Functionally Defined Objects

Sergey I. Vyatkin*

Institute of Automation and Electrometry, SB RAS, Novosibirsk, Russian Federation

Address for Correspondence

Institute of Automation and Electrometry, SB RAS, Novosibirsk, Russian Federation
E-mail:
sivser@mail.ru

ABSTRACT

This paper describes interactive shape modeling of geometric objects defined by perturbation functions. 3D objects based on the perturbation functions have an advantage of spline representation of surfaces, that is, a high degree of smoothness, and an advantage of arbitrary form for a small number of perturbation functions. Interactive modification of the function-based model with fast visualization allows us to provide both the interactivity and any required level of detail leading to a photo-realistic appearance of the resulting shapes. An interactive system modeling is presented.

Keywords: Animating, Rendering, Modules, Modeling.

INTRODUCTION

Several representations of geometric objects are currently used in computer graphics. Each of the objects, according to its properties, is used in different fields, beginning from 3-D simulation and CAD systems up to real-time visualization systems. The functional representation describes most accurately the object geometry and has the smallest size of the required data. Procedures of functional representation demonstrate compact and flexible representation of surfaces and objects that are the results of logical operations on volumes.

The creation of complex models for such applications as movie special effects, graphic art, and computer-aided design can be a time-consuming, tedious and error-prone process. Most geometric modeling systems expect the user to manipulate control points of NURBS, individual mesh vertices and polygons, or use conventional, higher-level operations such as volume deformations and Boolean operations. In an image processing system, vertex and control point manipulations would be equivalent to painting an image pixel-by-pixel. The process of 3-D object production is also tedious and requires a variety of different techniques. While it may be useful to have access to such low-level operations in certain cases, most image manipulations are done using higher-level tools. Recently, computer graphics associated with interactive modeling and editing of 3-D objects has been developing. Well-known

commercial systems such as Softimage, Alias Wavefront (Maya), 3DstudioMax, LightWave 3D, Modo 3D offer an interactive editing of polygonal models. There are interactive systems based on implicits[1], triangle meshes[2], images[3], volumes[4], function-based models[5].

For interactive mode, many function-based systems use polygonization algorithms. This leads to a loss of accuracy calculations. As well as modeling system is complicated. This paper describes an interactive system modeling based on perturbation functions[6,7]. The system modeling supports a great variety of different interaction techniques to alter shape, including sculpting, carving, texturing, etc. The system reads, processes, and writes models without intermediate tesselation. This paper extends the work[7], where an application of perturbation functions in computer graphics was presented.

AN INTERACTIVE SYSTEM MODELING

Shape modeling seen as a process of creating the final object by gradual local shape deformations. Under local deformation is meant that the area of each individual shape deformation is significantly smaller than the overall size of the shape. Perturbation functions and set-theoretic operations to represent shapes and their properties have been used.

Common description

An algorithms and a set of C++ classes for system modeling have been developed: recursive multilevel ray casting [7] (hereinafter RMLRC) for scenes containing functionally defined objects (including OpenGL color/depth buffers compatibility); C++ classes for functionally defined objects representation; C++ classes for rendering of functionally defined objects; C++ classes interface, these classes provided to make the whole system to be easily extended to incorporate new algorithms and features. Thus, the resulting is designed as collection of classes (hereinafter VxFramework) to facilitate the development of system modeling applications.

Modules overview

These modules are subdivided other tasks of the projects into independent parts that use the classes of *VXFramework* and are integrated into the system through inheritance of interface classes. The VxDll module is responsible for Base classes for rendering and objects representation. The VxSceneBuilder.dll module is responsible for scene parsing/saving. An interactive system modeling stuff is grouped in *VxManipulator* class.

Geometric objects and recursive multilevel ray casting algorithm overview

It is proposed to describe complex geometric objects by defining the function of deviation (of the second order) from the basic quadric in the form:

$$F(x,y,z) = A11x^2 + A22y^2 + A33z^2 + A12xy + A13\,xz + A23yz + A14x + A24y + A34z + A44 \geq 0 \quad \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

The freeform is a composition of the base surface and the perturbation functions

$$F'(x,y,z) = F(x,y,z) + \sum_{i=1}^{N} R_i(x,y,z) \quad \ldots. (2)$$

where the perturbation function $R(x, y, z)$ is found as follows

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

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

Since max[Q + R] ≤max[Q] + max[R], for estimating the maximum Q on some interval we have to calculate the maximum perturbation function on the same interval. The obtained surfaces are smooth (see Fig. 1), and creation of complex surface forms requires a few perturbation functions.

The multilevel ray casting algorithm[7], which performs efficient search for volume elements - voxels which participate in image generation was used.

Before rendering could start the renderer should be initialized or "announced". The main steps of the process are following ():

```
Renderer::Init (_flags)
{
  //initializing m_vxXi - space subdivision description structure
        m_vxXi.Init(given_quatro_level, given_binary_level, _flags);
        m_vxXi.S= given_camera_matrix;
        m_vxXi.P= given_perspective_matrix;
        pGeometryToRender->Announce(m_vxXi);
}
```

For rendering to be correct the scene should be announced too. This recursive process is done by virtual 'Announce' method to guarantees the tree traversal including perturbation. While the implementation of this function by VxOp – operations class simply send calls to its content, the VxQuadr initializes special internal structure encapsulated quadric representation by 10 coefficients and their changes.

Within the proposed paradigm it is possible to derive your own subclass with new rendering properties (e.g. with special shading or texturing) simply by overriding the virtual Get Material() and Get Normal() methods of any functionally defined object class or even Calculate Color method of Renderer itself.

This main space traversing cycle is performed after checking that context and buffer(s) are presented and valid by Renderer like following:

```
Init(_flags);// Initialization of rendering setting, scene preparation
Render (Scene);
```

Base classes, reference counting, smart pointers

All VxFramework classes (such as buffers, geometry, light sources, etc.) are reference counted. To provide certain functionality they are inherited from one or more Object-derived classes along with using DECLARE/IMPLEMENT macros in MFC fashion to provide runtime type functionality. However some hacks allow using multiple inheritances for your classes, so you need not to stick to single inheritance like if you're using MFC.

The reference-counter is incremented through Add Ref() and decremented through Release() and if it's zero, the object is deleted to recover memory. To guarantee that object is deleting from the same heap it was created in all the destructors are made private.

There were some kind of smart pointers – vx_ptr template, to store pointers to dynamically allocated objects. They behave much like built-in C++ pointers except that they automatically delete the object pointed to at the appropriate time. They can also be used to keep track of dynamically allocated objects shared by multiple owners. One of the advantages of use vx_ptr is that they are objects which could be created at the stack while encapsulating all dynamic memory management stuff. Another advantage is that the *operator* = of vx_ptr decrements counter of previously stored object before as well as

its constructors increments reference counter.

Rendering classes

The set of classes are used to create and setup different attributes of the renderer, its context, buffers manipulations and etc. *VxRender* Context is the most important class for the rendering control. Like in all 3D rendering, we need the concept of a virtual camera. This is set up with the following calls:

void SetCameraMatrix(Matr4);
void SetProjectionMatrix(Matr4);

VxFramework supports the Phong light model as class *'LightModel'* with the ability to control light sources as exemplars of *'LightSource'* class.

The context is send to appropriate renderer to perform render. It is shareable amongst different renderers. For UI applications it is common practice that context user currently working with is become globally available for all kind of plug-ins and etc.

This is done through *gGetCurrentRenderContext()* and *gSet-CurrentRenderContext()* methods correspondingly.

Depth/Normal and Color Renderers

You could use one the two renderers (depth/normal and color) or both for scene rendering by filling appropriate buffers. Also it is possible to derive your own renderer subclass simply by overriding *VxBaseRenderer* pure-virtual prototypes. Most of rendering settings are controlled through assigning of the rendering context. Each of renders has its own implementation of the following pure virtual function (inherited from *VxBaseRenderer*). Depth/Normal and Color Buffers are presented as implementations of Render Data ptototype. These renderers always ask their buffers (set by Set Buffer) that they are valid for requested rendering by calling IsValidFor method. All memory allocation, clearing, data access and etc. are done by flags specifying what the data is used for. As mentioned above, buffers are reference-counted and therefore are shareable. The sample of using buffers for rendering as well as OpenGL-compatibility issue.

Objects

Objects are the building blocks of the scene. The following paragraphs describe the existing types of objects.

Functionally defined objects types corresponding to base quadric types:

‘ell’, ‘seat’, ‘ellpar’, ‘unihyp’, ‘duahyp’, ‘cone’, ‘cylinder’, ‘layer’, ’plane’, ’clin’, ’parabola’, ’hyperbola’.

Sample of the scene:

```
ell
{
}
cone
```

VxBuilder will automatically made union of these objects by default. The following complex objects were added to standard library:

Torus (defined by ‘torus’ keyword);

Cube (defined by ‘cube’ keyword).

Also as mentioned above, you could use objects simply by referencing to its file name using ‘object’ keyword. Please note that feature doesn’t supported for file saving. It always saved to one file through traversal saving of all object regardless to their own paths.

Sample of the scene:

```
//x.scene
cube
torus
//y.scn
object x
```

In addition the primitive types described above, you can also combine multiple objects of any type into complex one by Constructive Solid Geometry (CSG). Two basic types of CSG operations supported through keywords: 'union', 'intersection' ('inter').

Sample of the scene containing intersection of the two primitive objects:

```
inter
{
  ell
  cone
}
```

The following items may be applied to object of any type. It is often useful to invert an object using 'inverse', for example use of 'inverse' for union changes it to intersection. Transformations: 'move', 'rotate', 'scale', 'skewx', 'skewx', 'skewy', 'skewz', use 'matrix' keyword. Specification of the color uses 'color' keyword. Colors consist of three values for each of color components.

Interface concept

All Vx manipulations are object(s)-oriented thus you apply commands by first selecting the object and then selecting the command (a noun-verb interface). You select object (s) directly by left mouse button clicking in the render window (you could see the list of their name(s) objects at the top of the window) or through scene tree control using context menu. The selected object (if it is not perturbation) is marked with red and remains red until you change the selected object. The three-color tripod (with corresponding axes highlighted) is visible for each selected object in order to help you understand the direction of the movement or other manipulations (see Fig. 2). The area of perturbation if it is selected is outlined by blue line and its perturbation factor is shown after its name.

Proposed interface for design system provides 4 basic manipulations: Move, Rotate, Scale and Deform. Objects are selected by left mouse button clicking. If 'Ctrl' is pressed then only perturbations could be selected. By choosing the action you could transform selected objects. When the rendering of the scene is completed you could see the name(s) of the object(s) under cursor in Info mode (when Shift is pressed). You also could apply modifications to object(s) selected during Info mode after you turned it off (by releasing the Shift - key). The Scene control is the tree view window that displays a hierarchical list of objects, such as operations (union/ intersection), quadrics and perturbations. The drag-n-drop operations are supported. Each item consists of a label that names object. User is able to add/ delete objects, change the properties of the selected item (e.g. type of quadric or operation, perturbation factor value etc.) through property sheet at the bottom of the control. The context menu allows user to cut/copy/paste/delete/create objects.

CONCLUSION

In this article, interactive shape modeling of geometric objects defined by perturbation functions. Gradual modification of an initial shape with interactive modifications is the central concept of system modeling. To ensure interactivity, the efficient acceleration methods for function evaluation have been proposed. A system modeling allowing for implementation of different functionally defined interactive shape modeling applications was considered.

REFERENCES

1. R.N. Perry and S.F. Frisken. Kizami: A System for Sculpting Digital Characters, in *SIGGRAPH'01*, 2001, pp. 47-56.

2. M. Agrawala, A.C. Beers and M. Levoy, M. *In Proceedings, Symposium on Interactive* 3D *Graphics,* 1995, pp. 145-150.
3. B.M. Oh, M. Chen, J. Dorsey and F. Durand. Image-based modeling and photo editing, in *SIGGRAPH'01*, 2001, pp. 433-442.
4. S. Wang and A. Kaufman, Volume Sculpting, *Symposium on Interactive 3D Graphics, ACM Press, 1995,* pp.151-156.
5. K. Levinski and A. Sourin. Interactive function-based artistic shape modeling// *International Symposium Cyber Worlds: Theory and Practice,* Tokyo, Japan 6-8 November, 2002 pp.521-528.
6. S.I. Vyatkin, Complex Surface Modeling Using Perturbation Functions, *Opto-electronics, Instrumentation and Data Processing*, Volume 43, Number 3, 2007, pp. 226-231.
7. Vyatkin Sergey I., Dolgovesov Boris S., Gorodilov Mikhail A. Perturbation Functions In Computer Graphics//Modern Instrumentation, 2013, vol. 2, № 2. pp. 26-32.

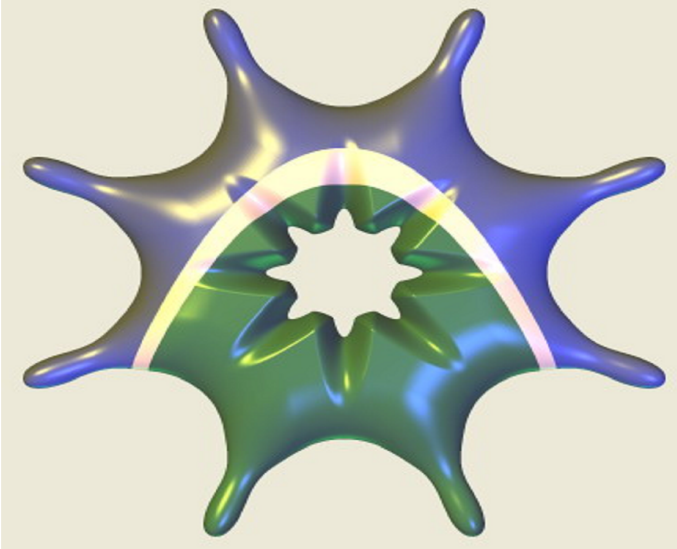

Figure 1. Functionally defined object with perturbation functions.

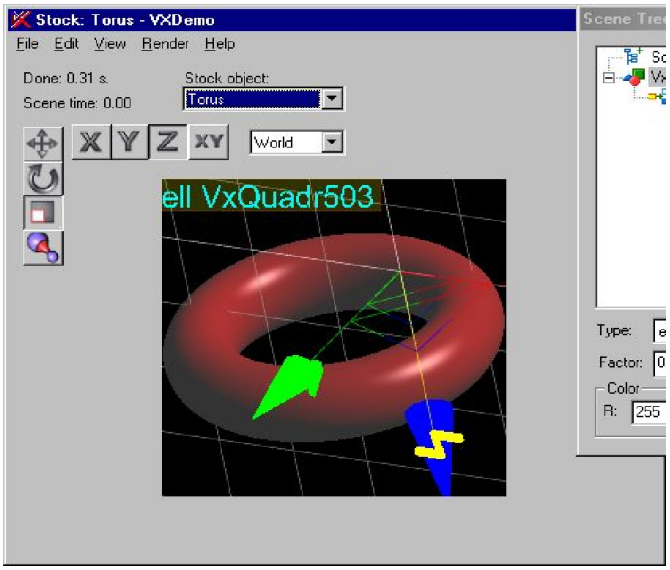

Figure 2. Main window of the program.