THREE DIMENSIONAL COMPUTER ANIMATION FOR PRESENTING WEATHER FORECAST

Alphan Es

Computer Graphics and Virtual Reality Group
TÜBITAK-BILTEN
METU -ANKARA, 06531, TURKEY
email: alphan.es@bilten.metu.edu.tr

Veysi Isler

Department of Computer Engineering
Middle East Technical University,
ANKARA, 06531, TURKEY
email: isler@ceng.metu.edu.tr

ABSTRACT

Three-dimensional computer animation can be very useful in presenting weather forecast on TV channels. However, generation of the animation for weather forecast using general-purpose tools might be difficult and time-consuming since this kind of animation requires special visual effects and various animatable parameters. Furthermore, the animation is to be generated by ordinary users with no training in animation. In this paper, we present a three-dimensional computer animation system specifically designed for presenting weather forecast. The system is designed and implemented using object-oriented techniques that are very helpful for computer animation and motion abstraction. The animation system also incorporates multiresolution modeling and paging of huge terrains for fast rendering.

Keywords: Three dimensional computer animation, object oriented programming, key frame animation, multiresolution modeling.

INTRODUCTION

Three dimensional computer animation has a wide range of application areas such as education, simulation, advertisement and entertainment. Another area of application can be weather forecast presentation if the animation system has a friendly and easy user interface which is to be used frequently a few times a day by the users with no animation training. In this paper an animation system, called Weather Forecast Animator (WFA) which has an easy and friendly user interface specifically developed to create animations for weather forecast presentation is described.

WFA is a key frame based animation tool that uses three-dimensional graphics and actual geographic data to visualize weather conditions. It is intended for television channels, and simplifies the generation of weather forecast presentations with superior visual quality. WFA has object-oriented design and written in C++ [Strou91]. Although it is developed for weather condition animations, it can easily be extended and adapted for other purposes. Almost everything in a scene is animatable which greatly stems from its object-oriented design. Its animation system uses key frame technique. This technique is widely used, and provides a simpler interface compared to other animation techniques such as language based, or simulation based animation systems. Program uses a graphical user interface and uses OpenGL [Neidr96, Opengl] graphics library. Thus, it supports hardware acceleration.

Another topic that is considered is the usage of multiresolution modeling [Heckb94] for both rendered images and the animation editor of WFA. Multiresolution modeling is to use different detail levels for the scene objects. The appropriate detail level for an object is determined by measuring its benefit and cost with respect to the position of the viewpoint [Funkh93, Yuan97, Horvz97]. Benefit of an object is, in the simplest sense, the contribution of the object to the overall scene. Cost on the other hand is the measure of the resource consumption for rendering that object. High benefit implies being more important and those objects must be drawn using a higher detail level. For optimizing rendering quality and speed, benefit over cost ratio must be kept as high as possible. Being different from other animation systems, WFA enables the user to limit the total animation time. In order to render whole animation in this limited time, cost and benefit scheme is being implemented.

The second and the third sections discuss object-oriented architecture of the animation system. The second section focuses on animatable classes, while the third section pays attention to key frame implementation. The fourth section presents the user interface of the system, its integration to the key frame system and the use of multi-resolution modeling for the editor. The fifth section describes rendering and the way to use multiresolution modeling for rendering.

ANIMATABLE CLASSES

WFA is an object-oriented application. Object oriented approach gives flexibility in terms of programming and specification of complex animation sequences [Erkan95]. Furthermore, it is easy to insert new object types to the application.

The simplified class structure of the program is shown in Fig. 1. This graph represents animation objects' hierarchy. Object Modeling Technique (OMT) [Rumba91] is used to depict the graph. Briefly in this technique, triangle denotes inheritance, diamond denotes aggregation and dot denotes multiplicity in association. Class **Animatable** is the base class for all animatable objects. It is a pure virtual class and has some general functions for all animatable objects. These functions include type definition, loading and saving operation and graphical user interface related basic processes. Each object that is designed to be animatable class has to overload these general functions to correctly manage their animatable parameters.

There are four animatable object types: **Material, Environment, 3DObject** and **DirectionalLight**. All these objects have at least one animatable parameters.

Material class is used for defining the objects material attributes. It has ambient, diffuse, emission, specular, shininess and transparency parameters, all of which can be animated. Each object to be rasterized must have one material object assigned. OpenGL uses this material object for coloring during rendering.

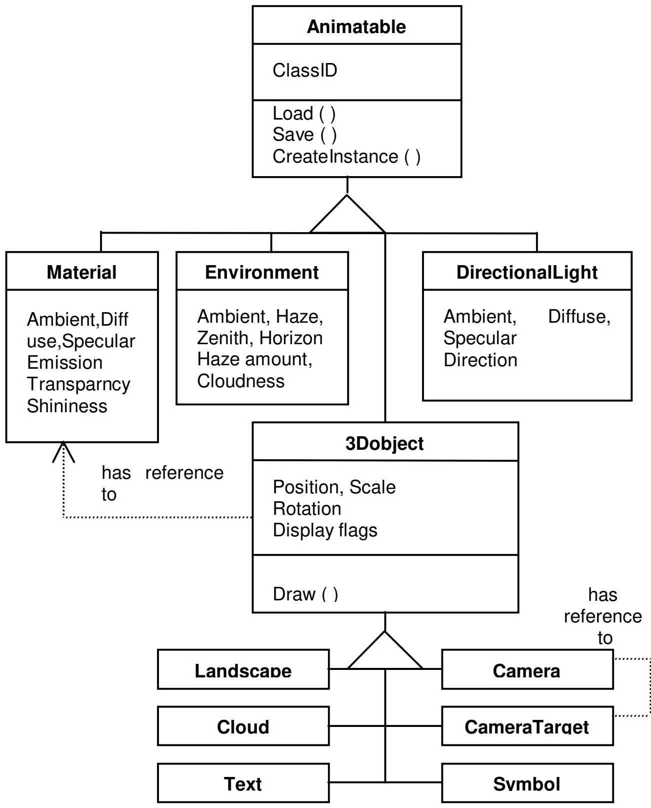
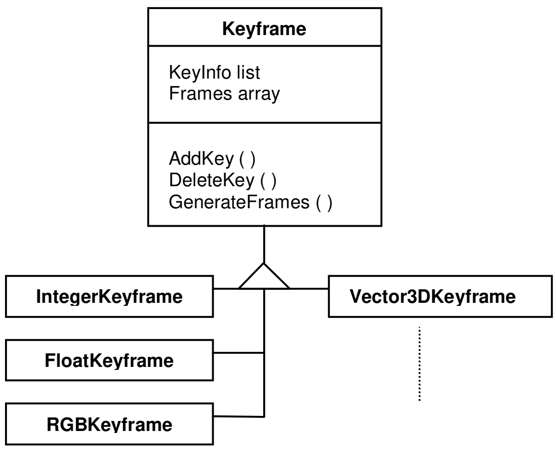


Figure 1: Simplified class structure of animatable objects

Environment object defines the general properties of the scene. Some of its parameters are ambient, haze, horizon and zenith colors, haze amount and horizon height. These parameters can be animated also. Some other parameters such as ground texture, cloud density are also define scene properties but can not be animated.

DirectionalLight class is used for lighting. There are up to three directional lights that can be used. Although OpenGL can handle more lights (at least eight), this constraint is intended for the sake of rendering efficiency. All of the light parameters can be animated. These are light direction, ambient, diffuse, specular colors.

3DObject class is the most complex class among animatables. It is the base class of the camera and the objects to be rendered. Each class that is inherited from object has position, scale and rotation values. These objects are displayed in the viewport, so each of them also has to implement a method to display itself in viewport. **Landscape** class is used for visualization of landscape data, **Cloud** class is used to simulate weather conditions (i.e. rainy, snowy or sunny) and **Text** class is used to display some information on the landscape (such as city names and temperatures). **Symbol** class is used to add general three-dimensional models to the scene. Using this class it is possible to insert weather forecast related symbols and landmarks. **Camera** and **CameraTarget** classes are used for positioning and animating the virtual camera.



struct **KeyInfo** {
float Tension, Continuity, Bias, EaseTo, EaseFrom
void* FrameInfoPointer
}

Figure 2: Keyframe classes

Keyframe class is responsible for generating and managing key frames and generating in-between frames. **Keyframe** class can be used with any type of variable. Base class does not yield a restriction over the type of key frame values. Key frame values may be real number, RGB triplet or any other complex data structures may be used as seen in Fig. 2. Different data types are linked to *FrameInfoPointer* member of *KeyInfo* structure. For instance, **IntegerKeyframe** class is used to manage integer type key values and *FrameInfoPointer* is linked to an integer variable, while **Vector3DKeyframe** class cares for three dimensional vectors and is linked to a three dimensional vector. Main methods of this class are to create, add, delete, move keys, allocating memory for in-between frames and an abstract function for generation of in-between frames. In between frames are generated using by *Kochanek-Bartels splines* [Koch84] or by linear interpolation methods. These methods are widely used for 3D animation packages.

Handling Animatable Parameters

All animatable classes have at least one animatable parameter. The parameters can be of any type. Program uses real numbers, integers, RGB colors or three dimensional vectors as parameters. It is possible to add other parameter types. Parameters are handled by **Parameter** class. This class cooperates with a keyframe if the parameter is animatable. In order to achieve this, each parameter contians a pointer to its associated **KeyFrame** class. For example, **IntegerParameter** class cooperates with **IntegerKeyframe** class to save and manage integer type key values.

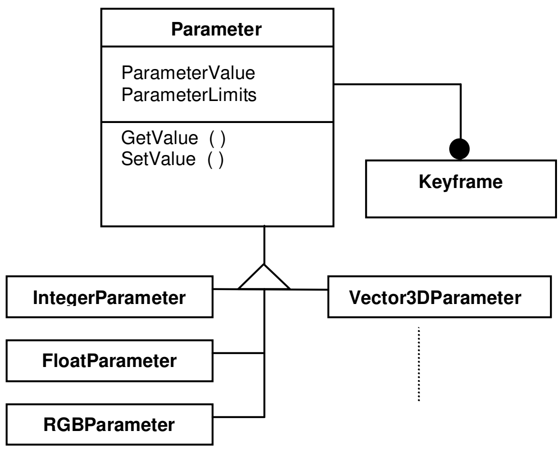


Figure 3: Parameter classes

Parameter class also defines minimum and maximum values of a parameter and can automatically justify the value of the parameter if it falls out of the range. There are also methods for setting and getting parameters.

Parameters can also be grouped for easier management. The class **ParameterBlock** provides this grouping. It can hold any number of parameters and handle their keyframe management automatically. This class is also useful for loading and saving included parameters' contents from and to a file and also for displaying key frames in the trackview editor automatically. Each animatable object has a **ParameterBlock** that contains its animatable or non-animatable parameters.

KEYFRAME SYSTEM

Program uses key frame animation technique for describing visual changes. These visual changes can be in object position, orientation, size, color, environment parameters or camera, light, and other object parameters.

Animator defines basic motions in certain frames. These basic frames are called keys. Key frame system interpolates these keys by producing in between frames. Animator also can control kinetic characteristics of the movement.

Defined key frames are viewed in a track-view window for enhanced control of motion. Track-view window displays a time chart and divides time to animation frames. Each animatable parameter is represented by a row, and each frame is represented by a column. Animator can create, delete, move or modify keys using track-viewer (Fig. 4).

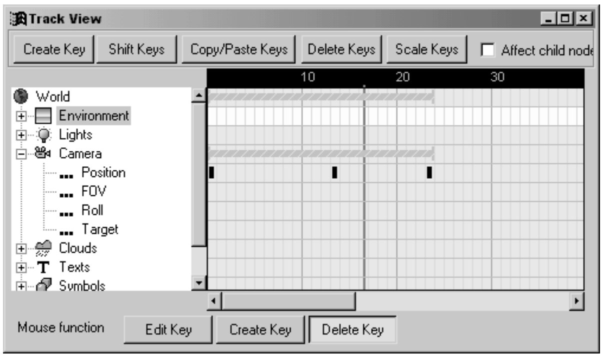


Figure 4: Trackview editor. On the right, animatable objects and their parameters are listed. Each column on the left panel represents a frame. Black dots denote key frames.

Interpolation Methods

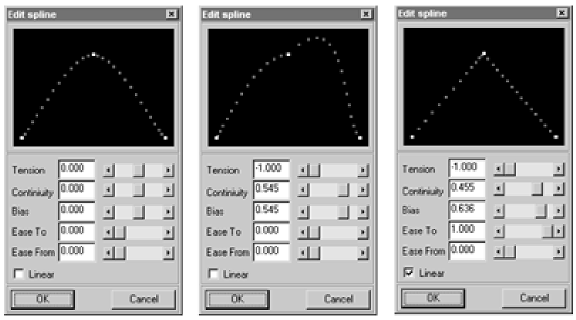
Key frame system uses Kochanek-Bartels spline and linear interpolation methods to produce the in-between frames. User defines the method to be used for interpolation, and for each key either method can be applied. Spline is used by default, and it is intended for smooth interpolation.

Kochanek-Bartels spline [Kocha84] has useful properties for defining motion paths in an animation. There are three parameters for each key when spline is used. These are tension, continuity and bias parameters. Tension defines the curvature of the spline segment, continuity defines the velocity continuity (first order continuity) and bias defines the biasing amount of spline segment.

The other method that can be used for interpolation is linear interpolation. This is simpler and more efficient than spline, but it is not smooth and intended for the spline segments where the motion is totally linear.

Simulating Accelerations

Interpolation techniques are used for in-betweening frames. It just defines the trajectory of the motion. In order to simulate acceleration and deceleration, each key has been given two parameters: *ease in* and *ease out*. *Ease in* parameter defines the deceleration amount to the key, and *ease out* defines the acceleration amount from the key. Using these two parameters animator can control the accelerations in key frames, and this gives more realistic look to the animation (Fig. 5).



(a) (b) (c)

Figure 5: Spline parameters for key frames. Middle white point represents current key frame. (a) shows initial parameter of a created key frame. In (b), tension, continuity and bias parameters are changed. In (c), ease to parameter is altered and applied linear interpolation.

USER INTERFACE

WFA provides the user with an easy and friendly graphical interface. There are several elements of the interface including menu, toolbars, dialog boxes for editing parameters and a viewport as in Fig. 6. The references [Foley96, Burgr89, Hearn96] include important factors and programming considerations about graphical user interfaces, such as graphical design, input device interactions and associated transition diagrams, windowing system interface etc.

User can look at the scene in any position and direction. There are several control buttons such as panning, rotating, zooming and field of view on the interface. User can use orthographic or perspective projection methods.

Viewport window displays scene objects and their trajectory. User can give motion to the objects using this viewport. Basically there are three common manipulation methods for the objects which are selecting, moving, scaling and rotating. These operations are processed by **3DObject** class, and thus all scene objects that are inherited from **3DObject** class can be manipulated using this operations. There is no need to define different move operations for each object types i.e. **Camera** and **Text** classes use the same function.

Besides these general operations, each scene object can also be attached a specialized editor. Using these editors, it is possible to edit object specific parameters. In order to achieve this, objects need to overwrite base **3DObject** class' related functions. These functions include creating modification dialog and accepting changes after manipulation.

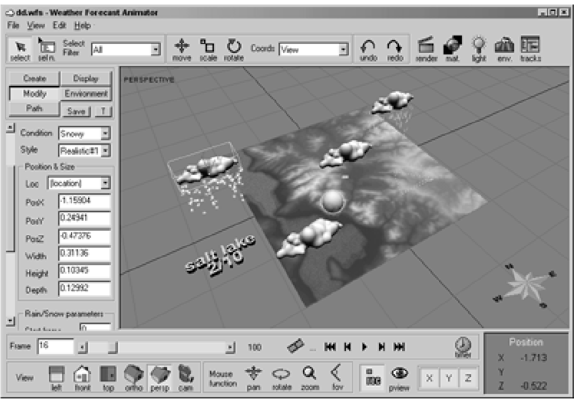


Figure 6: User interface of the program

Parameter-User Interface Integration

In order to easily edit contents of parameters, a class named **ParameterControl** is provided. This control contains a **Parameter** type variable and binds this variable to the operating system's standard user interface controls. These standard controls include edit boxes, scroll boxes, spin buttons or any other type that operating system supports. Binding is achieved using multi inheritance. Application is developed using Microsoft Foundation Class (MFC) [Krugl97, Sheph96] library. An edit box control that has the ability to edit parameters automatically is inherited from both **CButton** class (in MFC this class stands for the standard edit box control of the operating system) and from **ParameterControl** class. When the user edits the contents of the edit box and accepts the changes (i.e. presses enter key), **ParameterControl** automatically validates parameters value, saves the value to the parameter's associated key frame (if it is an animatable parameter) and stores changes to a history buffer for further undo/redo operations. This scheme enormously reduces code size and helps the maintenance of the application.

Using parameters and parameter bound controls, animatable object parameters are simple to modify. All animatable classes have associated editors. These editors contain **ParameterControl** derived user interface controls. When the user changes an animatable parameter, it is automatically saved to the parameter's key frame. In order to edit parameters of a **3DObject** user has to select the object. When a **3DObject** is selected, if it has implemented base class' modify dialog related functions, a modify dialog appears in a dialog bar for editing.

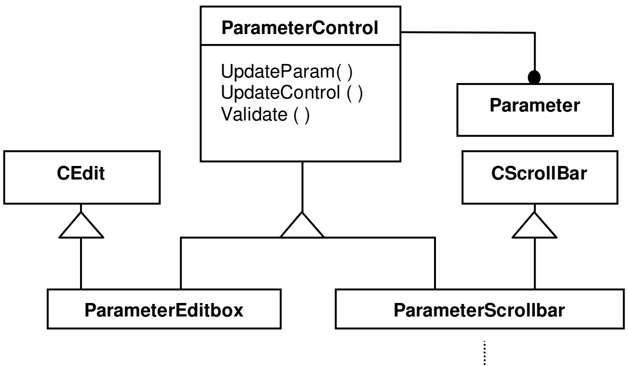


Figure 7: ParameterControl class

Fast Interaction Using Multiresolution Modeling

For a couple of objects, user interface works well. But if there are too many objects in the scene, refresh times of the viewport and response to the user during interactivity gets slower to unacceptable levels. To alleviate this problem, the program lets user adjust detail levels for the scene objects or draws objects as bounding boxes. In the current version this is done manually and detail level selection is applicable to landscape and text objects only. Clever methods for automatically choosing appropriate detail level for the objects are being added to the system. For achieving real time detail level switching, several models of an object in different detail levels may be prepared and used. But this scheme may consume too much memory. So instead of that, we use a compact object format that contains different detail levels inside such as progressive meshes [Hoppe96]. The application is not an object modeler, so it is very reasonable to use progressive meshes for the scene objects, because detail levels need not be calculated during execution of the program. Progressive mesh model can be used for texts, clouds and symbols easily.

RENDERING

Rendering process may be partitioned into two parts. One of them is rendering landscape and the other is rendering of the other generic three-dimensional objects. Rendering methods of these two differs in order to optimize rendering speed and memory usage. The basic drawing primitives are triangles.

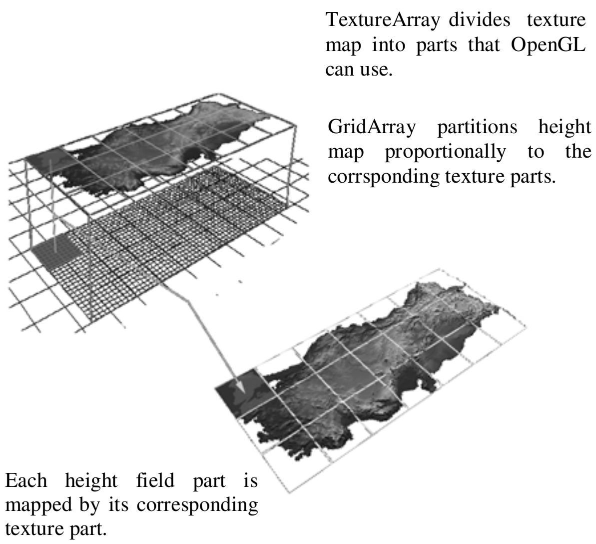


Figure 8: Rendering phases of landscape object

The height fields are generated from digital elevation maps. The height field is a uniform grid of height values. This form is chosen because of its easier management over irregular triangular meshes and detail levels manipulation is left as a future work. At the each element of the height field map, a height value and an associated normal is stored. Being uniform facilitates rendering data as quad strips without need of any complex algorithms and data structures. Height values are stored by a class named **GridArray**. **GridArray** is responsible for loading and manipulating height fields.

OpenGL has a limit in terms of texture sizes. This limit may changes on different hardware and software implementations. In the current version of Microsoft OpenGL this limit is 1024x1024. But most of the current accelerators set this value to lower dimensions such as 512x512 or 256x256.

Program allows using larger textures than OpenGL allows. In order to overcome texture size limits, big texture map is divided into smaller parts so that OpenGL can use them. In the program TextureArray class is responsible for dividing textures into smaller parts. It simply loads a big texture map and partitions it into parts that can be used by OpenGL. Texture map may also contain opacity information. Thus unwanted parts such as out of map borders can be masked out easily.

TextureArray and **GridArray** classes cooperate for rendering landscape image. The **Landscape** object contains these two classes. Firstly, height field is loaded using **GridArray**, then **TextureArray** loads texture to be mapped and partitions it. Then height map is divided into subsections proportionally sized to its corresponding texture part. While rendering, each height field part is rendered using its corresponding texture parts (Fig. 8).

Multiresolution Modeling for Rendering

As in the editor, multiresolution modeling is used in final rendering, as well. Landscape and other objects are rendered using different levels of detail. Because there are specialized algorithms for landscape rendering, different multiresolution modeling techniques for landscape and other scene objects is chosen [Falby93, Rabin97, Linds95].

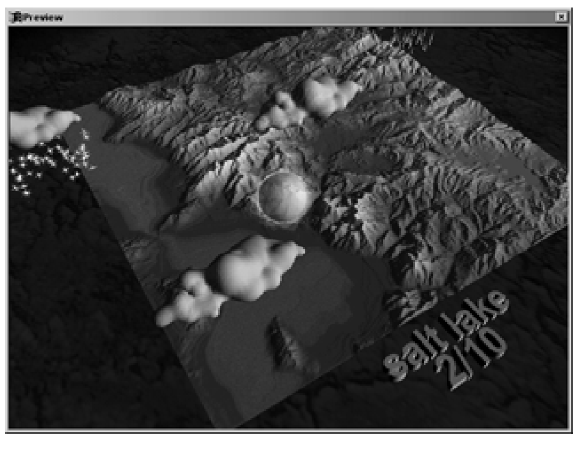


Figure 9: An image rendered by the program.

Landscape object consists of height field and may be larger than the available primary memory. For very large terrain databases, paging operation is required to keep only the needed part of the database in the main memory. In paging, database must be divided into regions and only visible regions are loaded to the memory during rendering, and appropriate detail levels can be selected for different regions. This also decreases the total memory usage. For the other objects, progressive meshes [Hoppe96] or texture caching methods [Shade96, Lengy97] may be used. In both of the methods, detail level selection is done using benefit over cost ratio [Funkh93, Yuan97, Horvz97].

Because the application is a keyframe based animation system and so the motion paths are defined before rendering, rendering can benefit from path coherence. For landscape object, visible sections and detail levels may be pre-calculated, for the other objects detail switching frames or texture cache lives can be accurately calculated. These calculation can be done prior to rendering. This may also facilitate, interactive frame rates and thus instead of recording animation to a video file, user can fly through the map in real time.

CONCLUSION

An animation system that can ease the generation of weather forecast presentations is described. The system has a very user-friendly interface, which enables even ordinary users to specify complex animations with special effects for presenting weather forecast.

The system is designed using object oriented techniques that provides motion abstraction to be able to specify and control complex animation sequences easily.

The system also includes the idea of multiresolution modeling for fast rendering and user interaction.

REFERENCES:

[Burgr89] Burger P., Gillies D., *Interactive Computer Graphics, Addison-Wesley*, ISBN: 0-201-17439-1, p. 33-73, 419-440, 1989.

[Erkan95] Erkan B., Özgüç B., *Object-Oriented Motion Abstraction, The Journal of Visualization and Computer Animation*, Vol. 6 : 49-65, 1995.

[Falby93] Falby J. S., Zyda M. J., Pratt D. R., Mackey R. L., *NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation, In Computer & Graphics*, Vol. 17, No. 1, pp. 65-69., 1993.

[Foley96] Foley J., Van Dam A., Feiner P., Hughes J., *Computer Graphics: Principles and Practice, Addison-Wesley*, ISBN: 0-201-84840-6, pp. 1057-1080, 1996.

[Funkh93] Funkhauser, T., Sequin, C., *Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments*. *Computer Graphics SIGGRAPH'93 Conference Proceedings*, 247-254, 1993.

[Hearn96] Hearn D., Baker M. P., *Computer Graphics, Prentice-Hall*, ISBN: 0-13-530924-7, pp.271-295, 584-596, 1996.

[Heckb94] Heckbert P. S., Garland M., *Multiresolution modeling for fast rendering. In Proc. Graphics Interface'94*, Banff, Canada May 1994. Canadian Inf. Proc. Soc., pp. 43–50, 1994.

[Hoppe96] Hoppe H., *Progressive Meshes, ACM Computer Graphics SIGGRAPH'96 Proc.*, pp. 99-108, 1996.

[Horvz97] Horvitz E., Lengyel J., *Perception, Attention, and Resources: A Decision-Theoretic Approach to Graphics Rendering, Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI '97), Providence*, RI, pp. 238-249, August 1-3, 1997.

[Kocha84] Kochanek, D., Bartels, R., *Interpolating Splines with Local Tension, Continuity, and Bias Control, SIGGRAPH'84*, pp. 33-41, 1984.

[Krugl97] Kruglinski D. J., *Inside Visual C++, Fourth Edition, MS Press*, ISBN :1-57231-565-2, 1997.

[Lengy97] Lengyel J., Snyder J., *Rendering With Coherent Layers. SIGGRAPH'97 Proc.,* pp. 233-242, 1997.

[Linds95] Lindstom P., Koller D., Hodges L. F., Ribarsky W., Faust N., Turner G., *Level-of-Detail Management For Real-Time Rendering of Phototextured Terrain, Technical Report TR 95-06*, Graphics, Visualization, and Usability Center, Georgia Tech, 1995.

[Neidr96] Neider J., Davis T., Woo M. *OpenGL Programming Guide, Addison Wesley*, 1996.

[Opengl] *OpenGL Official Web Site, http://www.opengl.org*

[Rabin97] Rabinovich B., Gotsman C., *Visualization of Large Terrains in Resource-Limited Computing Environments, Proceedings of Visualization*, Phoenix, October 1997.

[Rumba91] Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorense W., *Object-Oriented Modeling and Design, Prentice Hall*, 1991.

[Shade96] Shade J., Linchinski D., Salesin D., DeRose T., Snyder J., *Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. In SIGGRAPH'96 Conference Proceedings.*, Annual Conference Series, H. Rushmeier, Ed.: ACM SIGGRAPH, pp. 75-82, August 1996.

[Sheph96] Shepherd G., Wingo S., *MFC Internals, Addison Wesley*, ISBN 0-201-40721-3, 1996.

[Strou91] Stroustrup B., *The C++ Programming Language, 2nd Edition, Addison-Wesley*, 1991.

[Watt89] Watt A., *Fundamentals of Three-Dimensional Computer Graphics, Addison-Wesley*, ISBN 0-201-15442-0, pp. 285-306, 1989.

[Yuan97] Yuan P., Lau R. W. H., *A Framework for Performance Evalution of Real-time Rendering Algorithms in Virtual Reality, ACM VRST'97*, pp. 51 - 58, 1997.