Immersive Visualization of Adaptive Grids based on CAVE-Technology*

Jürgen Zauner
Spezialforschungsbereich SFB F013
Numerical and Symbolic Scientific Computing
Freistädter Strasse 313
Universität Linz, A-4040 Linz, Austria
(http://www.sfb013.uni-linz.ac.at)

Abstract

This report is an introduction of the newest features of VIPP and the possibility of presenting grid-based finite element results on graphic workstations and in an immersive visualization environment like the CAVE.

1 Introduction

Fast visualization of large data sets is an important task in context with computational mathematics like [1]. Usual visualization techniques do often not provide enough insight into the results. In order to increase the understanding of the data and make better conclusions out of them it is important to use an immersive technology of visualization. The CAVE is one of the best immersive technologies.

2 Previous work

In a previous work we integrated a fast visualization system into FEPP [1, 2]. This system is able to handle two and three dimensional finite element grids. One important feature is its tight coupling to FEPP. That is the reason why our system is able to visualize larger grids than visualization tools which use their own stored data set. Another consequence is that the latency between producing and visualization of data is low.

To get an interactive frame rate, which means at least 7 frames per second, two acceleration methods are used. Multiple levels of detail (LOD) [4] created by FEPP are used to reduce the data set for the visualization methods and the rendering time is accelerated by OpenGL display lists [3]. This means the visualization methods are applied to the data set of the current detail level and the rendering instructions of all visualization methods are optimized stored into

*This research has been supported by the Austrian Science Fund FWF within the SFB "Numerical and Symbolic Scientific Computing" under the grant SFB F013

a display list.

For a better understanding of the raw data set we have implemented the following visualization methods:

- color mapping
- isolines
- configurable single isosurface
- multisurfaces
- single cutplane
- vector plot
- element shrinking
- deformation

3 New features

3.1 Permanent storage

3.1.1 Interfaces to the Finite Element Code FEPP

In the previous version of the tool offline visualization was not possible. Therefore we added an abstract interface class, an inherited interface class for direct FEPP access and an inherited storable interface class based on C++ streams [5] to our package. These two classes allow us to realize two versions of our visualization system: a standalone tool and an FEPP integrated version.

The stream interface also helped us to create a small export filter for NETGEN [6]. By this measure the applicability of the tool VIPP has been increased. The main part of this interface are the VFI classes (visualization fepp interface). This datastructure stores the geometrical information and grid functions. Thereby different timesteps of these functions can be recorded. This feature provides visualization of dynamical grid functions. But the main feature of the stream interface is that no calculation time is needed for visualizing the data.

3.1.2 Configuration

The storability of both the mesh data and the gridfunctions is very important for an easy to use visualization tool. But the reconfiguration of previously used visualization methods was a main problem of the previous version of VIPP. This disadvantage has been fixed by storeable visualization methods based on C++ streams. Now they can be stored into vmc file by the VIPP GUI.

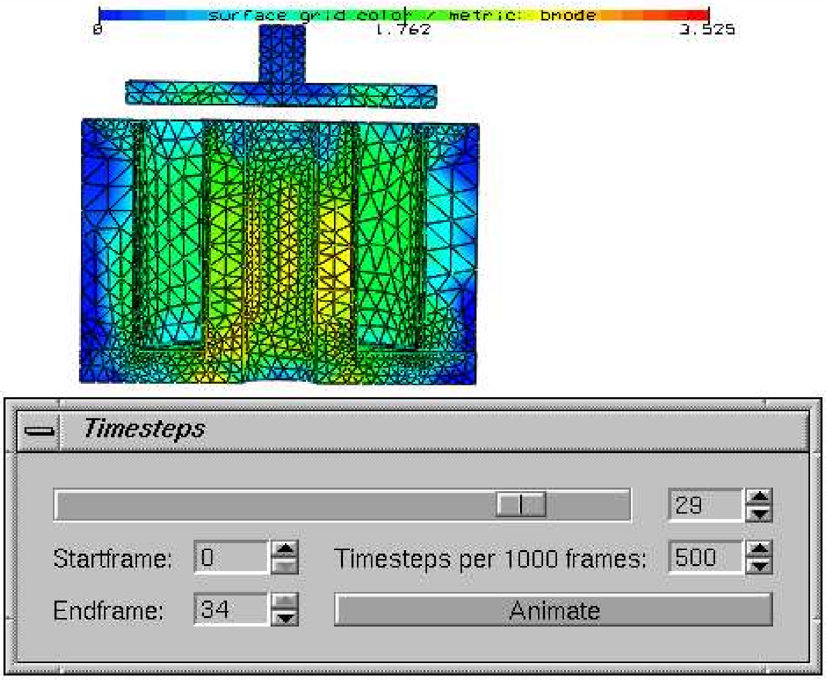

Figure 1: Visualization dialog for timesteps

3.2 Timesteps

FEPP is able to calculate dynamic gridfunctions with interesting intermediate result. To visualize this dynamic process we have to store the results of the current timestep before FEPP calculates the next results which are based on the values of the actual solution of the gridfunctions.

This storage procedure is very important because FEPP discards the old results. Due to this fact a tightly coupled visualization is impossible. So we use the datastructure of the stream interface to store the timestep results.

In figure 1 the dialog for the visualization of timesteps is shown. The timestep can be visualized using an animation or by scrolling through them with a slider.

3.3 Element-constant gridfunctions

Gridfunctions can be calculated for each vertex (nodal gridfunctions seen in figure 2) or for each element (element-constant gridfunction seen in figure 3).

Nodal gridfunctions are very easy to handle in the visualization because we can interpolate the value for each three or two dimensional point. This possibility is also important for the visualization of a coarser grid (LOD) because the coarser levels use only vertices of the finest grid for which the gridfunction is calculated.

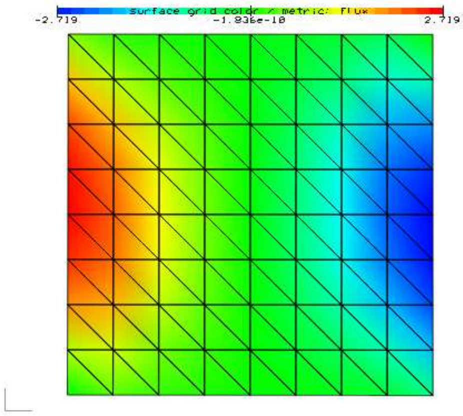

Figure 2: First component of flux as nodal gridfunction (demo example poisson.in)

Element-constant gridfunction are harder to handle. They have no defined solution for vertices, edges or surfaces which are used by more that one element. In this case we have decided to use the solution value with the maximal absolute value of the involved elements. The reason for this decision is that such interface surfaces are often surfaces between air and a material where the gridfunction exists. So the absolute solution value of the air is very low compared with the other material.

An other mechanism to eliminate this problem is the selection of preferred or displayed materials. To realize this in the visualization methods we use the material information delivered by FEPP.

But this is not the only problem occurred by element-constant gridfunction. Due to the fact that there is no solution for vertices we have to calculate a solution value for all elements of a coarser grid involved in the visualization process. This value is the solution mean value of the finest refined elements of the visualized coarse element. This calculation step can be done as a preprocessing step with the consequence that a lot of additionally memory is needed for the LOD solution values in the datastructure. In order to save memory we decided to make this calculation in the visualization step. This is the reason why the visualization of element-constant gridfunctions is slower than the visualization of nodal gridfunctions.

3.4 Materials selection

In many grids we have the problem that material is located around the interesting parts of the mesh. E.g. in the visualization of a magnet or transformer the core is covered by the air. To visualize the core we can use a cutting plane. But many cases the effect is not good enough. That is why we have implemented a method to deselect the surface material in the visualization process. In lower left field of figure 4 we can see the material selection field of the surface visu-

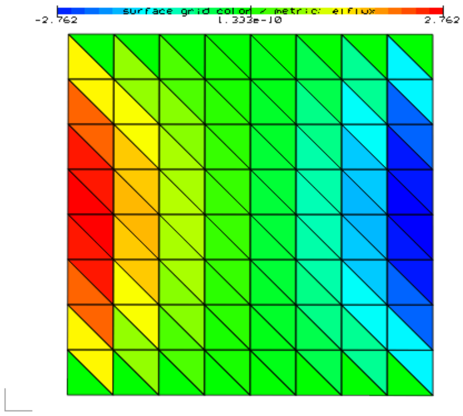

Figure 3: First component of flux as element-constant gridfunction (demo example poisson.in)

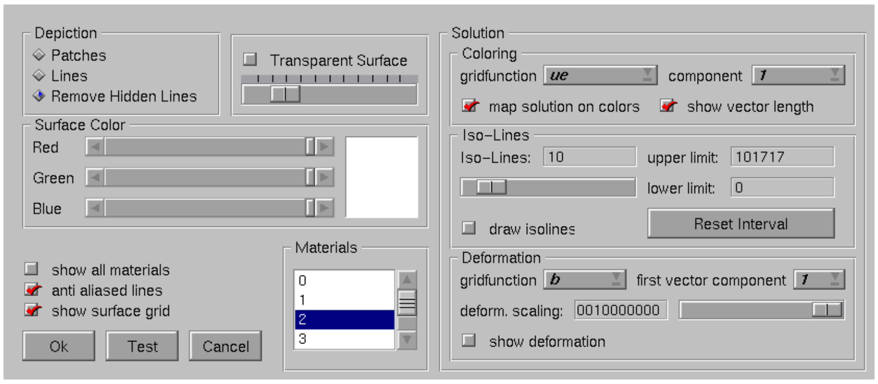

Figure 4: Configuration dialog of the surface visualization method

alization configuration dialog. In our example the shown selection results in a transformer visualized without air (figure 5).

3.5 Gridfunction length

Unfortunately until now VIPP has not supported the visualization of the gridfunction amount. This means each grid function component could be shown but not the value of its length. In the new version this feature is integrated as an online calculation to save storage. For a fast visualization you have to use a new gridfunction. Like the material selection this new feature can be configured in the surface visualization dialog seen in figure 4.

3.6 Animation

We implemented a set of classes to make the creation of animation scripts easier. Each animation consists of so called animation instructions which have associated points of time where they start and where they end. These instructions

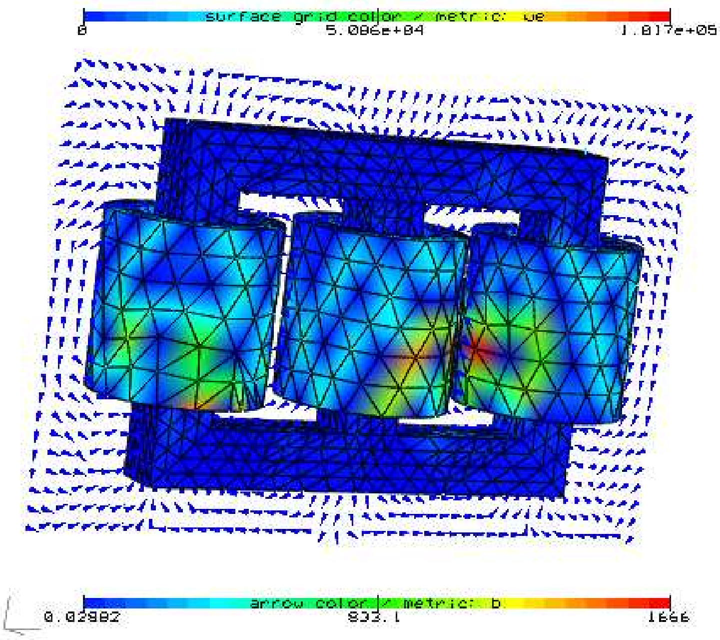

Figure 5: The length the gridfunction *ue* with selected materials (demo example trafo.in)

can be different kinds of moves and rotations. For example, the movement of the camera from one position to another where both, position and orientation are computed by linear interpolation can be realized. Another important application is the automatic reload of new visualization method configuration files. With these classes it is very easy to produce good looking presentations. A movie demonstrating this new feature can be found in [8].

3.7 Movie

The visualization of timesteps is often relatively slow because it is not possible to use displaylists. So for large problems we integrated a semiautomatically generation of movies. First you have to load the grid. After this step you configure the visualization methods. Now you can tell VIPP to store a ppm picture of each timestep (as seen in figure 6). You have to tell VIPP the first part of the name which should describe the content of the movie. All generated images starts with this name and ends with the frame number and the *.ppm* extension. This images can be processed to a movie by a additional tool. For example on a SGI workstation there exists a tool called *dmconvert*. When you use this tool you have to convert the images into a bmp format. This can be done with *convert* and the *tcsh* buildin command *foreach*. After this you can apply *dmconvert* on the bmp images. E.g: *dmconvert -f mpeg1v,brate=4500000 -p video,size=400x400 -n filename##.bmp,start=00,end=50,step=1 filename##.bmp filename.mpeg*

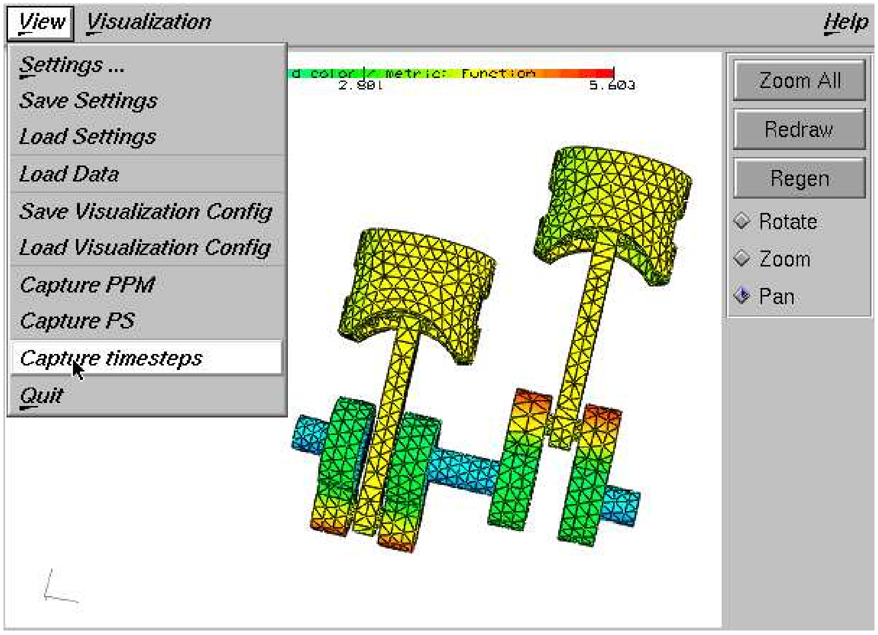

Figure 6: Menu entry to create a ppm file for each timestep

3.8 CAVE

A desktop visualization of the mesh is very useful but to get a better comprehension of the solution you have to use a very large display. This effect can be increased by a surrounding spatial display. To get a immersive impression of the grid you also have to use a three dimensional display. So we have decided to use the CAVE [7] to visualize the grid in a very immersive way.

3.8.1 Interaction

One of the most important problem of the visualization in the CAVE was: *How can I animate, scale, move and rotate the mesh in the CAVE?*

To get a good solution for this problem we have analyzed the hardware of our CAVE. We have eyeglasses with two position sensors and a wand (a navigation tool in the cave) with one position sensor, three buttons and a joystick. In figure 7 you see a person using the eyeglasses and wand to examine the tetrahedron elements of a crankshaft in the CAVE.

Normally a CAVE provides the information about position and orientation of head and hand (holding the wand). In our case the information about the hand is limited to its position because the CAVE library needs at least two sensors to get the direction of the wand.

To implement a navigation and interaction system with a high usability we need more than one position. First we made a navigation system which approximated the wand direction with the head and wand position. We moved the head position a bit down along the y-axis. This movement results in a pointer

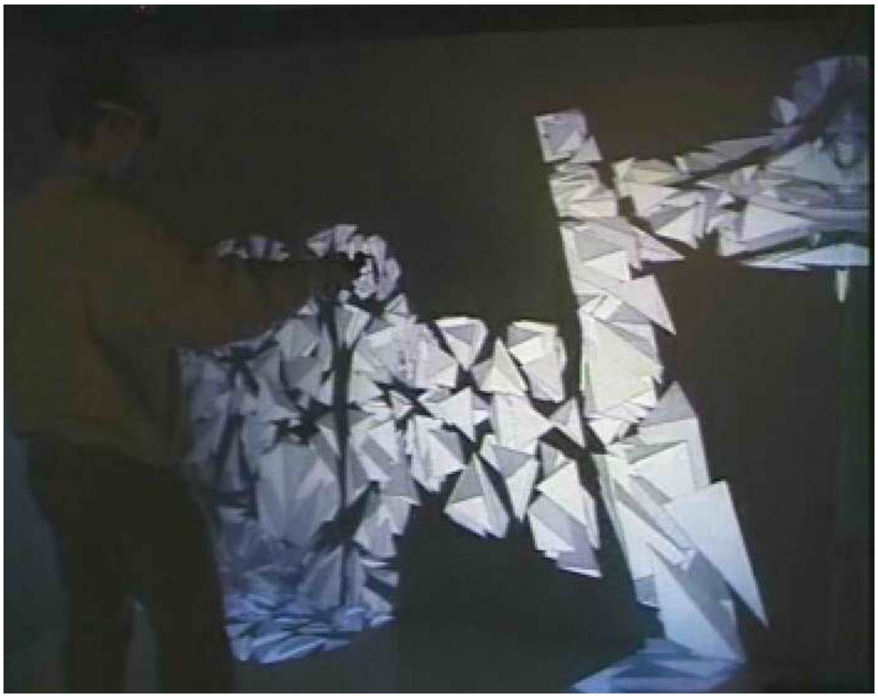

Figure 7: Crankshaft example in the CAVE

line given by this point and the wand position. By pressing a move button the user can define a default length of this pointer line. The movement in the CAVE on this line is proportional to the movement of the wand in the reality. This moving mechanism is easy to use for that person who has the sensor eyeglasses and the wand. But it is useless when you want present the mesh to another person who is wearing the sensor eyeglasses since the perspective in the CAVE only fits the person with the sensor eyeglasses. Therefore we decided to develop an other navigation method.

This new method is independent from the head position. The movement in the CAVE is the same movement of the wand in the reality. This means you grab the CAVE's virtual reality at the position of the wand and drag it through the CAVE room. This moving technique is not very useful for architectural walkthroughs but it's very effectively for visualization of single objects.

To show the object from all sides you can walk around it. But the problem is, that the CAVE only has four walls. So we also need an rotation mechanism. We solved this problem by using a fix point in the center of the CAVE. By pressing the rotation button you define a start position. A base vector for the rotation is defined between the center position and the start position. When you move the wand in CAVE room a current position is defined. The current vector is defined between the current position and the center position. By building the crossproduct of the current vector and the base vector we get the rotation axis. The length of the difference vector between the current vector and the base vector is proportional to the rotation angle.

With this moving and rotation methods we are able to present the mesh without wearing the sensor eyeglasses. To scale the object we used a scale key and the joystick on the wand. For a movement on one of the axis (x, y and z) or a rotation around them by using the joystick we also added keys. So rotation, movement and scaling can be done without entering the CAVE. To get a higher usability we added also a reset key which resets all navigation value.

For defined animations using the animation classes we only use three buttons. The first is to accelerate the animation forward, the second effects a backward acceleration and the third halts the animation. This is very useful for presentation to make explanations or go backwards to give answer on question concerning a part of the animation.

3.8.2 Timesteps

The visualization of timesteps in the CAVE is accelerated by displaylists. Displaylist can't be shared between the displaypipes of the CAVE machine. So this means for each timestep there exists one display list per displaypipe. Our CAVE has two pipes so in sum there are two displaylists for each timestep. When you want to present your mesh with timesteps you have to reduce the grid and the solution value to a very low limit because each timestep uses a lot of memory.

4 Users guide

4.1 Syntax

vipp [-gfm file.gfm] [-vmc file.vmc] [-cave [-anim2 — -timesteps]]

-gfm file.gfm	loads the mesh and the gridfunctions stored in *file.gfm*.
-vmc file.gfm	loads the configuration of the visualization methods stored in *file.gfm*.
-cave	activates the visualization in the CAVE and doesn't show the configuration GUI (only works with given gfm and vmc files).
-anim2	a default animation path which rotates the object an loads different vmc files for each rotation (only works in combination with *-cave*). The following vmc file have to exist: 1.vmc, 2.vmc, 3.vmc, 4.vmc, 5.vmc and 6.vmc
-timesteps	the second animation type is the timestep animation. This option causes VIPP to create a display list for each timestep. This timesteps will be animated in the cave (only works in combination with *-cave*).

4.2 Key bindings

4.2.1 Keys for the option *-timesteps*

x	activate the rotation mode around the x axis
y	activate the rotation mode around the y axis
z	activate the rotation mode around the z axis
q	activate the move mode along the x axis
w	activate the move mode along the y axis
e	activate the move mode along the z axis
1	increase the animation speed of the timesteps
2	halt the timestep animation
3	decrease the animation speed of the timesteps (the speed also can be negative)
s	activate then scaling mode
r	reset all navigation settings like position, object orientation, ...
space key	deactivate all modes

4.2.2 Wand bindings for the option *-timesteps*

middle button	activate/deactivate the wand move mode
right button	activate/deactivate the wand rotate mode
joystick	pushing the joystick (or trackball) up or down modifies the setting value of the current navigation mode (works with the following key modes: *x*, *y*, *z*, *q*, *w*, *e* and *s*

4.2.3 Wand bindings for the option *anim2*

left button1	increase the animation speed
middle button	halt the animation
right button	decrease the animation speed (the speed also can be negative)

There are no key bindings for the option *anim2* because there is no navigation possible.

5 Programmers guide

5.1 VIPP versions

There exists three different VIPP versions. The first is the FEPP integrated version. To make this version you have to compile the source code with the define FEPP_INTEGRATED (-DFEPP_INTEGRATED). This is already done when you compile it as a part of the standard FEPP distribution. This version is used to export static FEPP grids. Some new Features like material selection, element-constant gridfunctions and timesteps will not work in this version with direct access to FEPP. They will work when you save the data into a file and load this file. But this data will not update after a refinement step of the FEPP grid.

The second version is the standalone tool VIPP. You have to compile the source code without the define FEPP_INTEGRATED. This can be done in the FEPP directory fepp/libsrc/visualization. You have to call *make vipp*, *make* *vipp32* or *make vipp64* to create the standalone version you want. This version is used to visualize the data files and all new features (except the animation classes) will work in this version.

The third version is the CAVE version. For this version you have to compile the sources with the defines CAVE_FEPP_VIS and CAVE_NAV_ROT_OBJ (-DCAVE_FEPP_VIS -DCAVE_NAV_ROT_OBJ) and without FEPP_INTEGRATED. This can also be done in the directory fepp/libsrc/visualization by calling *make cave_vipp32* or *make -f Makefile.CAVE cave_vipp32* when you only have installed the visualization part of the FEPP distribution.

5.2 Stream interface

For all versions you have to define HAVE_CONFIG_H (-DHAVE_CONFIG_H) and set the environment variable MACHINE. This will include the file config.h in the directory fepp/libsrc/visualization which is used to configure the portable stream interface correctly. This works for SGI and Linux PC's. If you have an other machine you have to define WORDS_BIGENDIAN (-DWORDS_BIGENDIAN) when this machine is a bid endian machine. If it is a little endian machine you must not define WORDS_BIGENDIAN to store and load the FEPP data file correctly.

The stream interface can be used to store static and dynamic solutions. You have to use the class VFISolution which can be found in VFI.h and VFI.cc.

Static solutions can be stored into a stream by:

- creating a VFISolution object with the constructor *VFISolution(void)*,
- reading the data with the method *FromDirectData(void)* and
- writing the date to a out stream with the method *ToStream(ostream *os)*.

Dynamic solutions can be stored into a stream by:

- creating a VFISolution object with the constructor *VFISolution(long timeSteps)* (timeSteps is the number of timesteps which should be store),
- reading the data with the method *NextStepOfDirectData(void)* (This will read the mesh geometry and solution data at the first call. At all other calls it will only read the newest timestep.) and
- writing the date to a out stream with the method *ToStream(ostream *os)*.

When you want to integrate the VFI classes into your own none FEPP application you have to create a VFISolution object by calling the constructor *VFISolution(void)* and write the data by yourself into the structure. For further details on this topic you have to examine the files VFI.h and VFI.cc the methods *FromDirectData(void)* and *NextStepOfDirectData(void)* of the class VFISolution will help you understand how to fill the datastructure.

5.3 Animation

As mentioned in the new feature section we have implemented a set of animation classes. This classes can be found in Animation.h and Animation.cc. An example animation can be found in Main.cc (method Animate2(void)).

First you have to create a Animation class which contains a Anination-Clock object (RealtimeClock or FrameClock) and a linear linked list of AnimationInstructions (e.g. LinearCameraMotion, DistConstCenteredCameraMotion, ZoomInstr, LoadVMethSettingsInstr, ...). This instructions have a activation and deactivation time. They will be called by the Animation during their activation time.

The time is controlled by the AnimationClock object. A RealtimeClock object will deliver the difference between the the animation begin and the current time to the Animation object. A FrameClock calculates the time for the actual frame. This means it knows how much frames have to be animated during one animation seconds (not realtime seconds!).

LoadVMethSettingsInstr are a little bit tricky instructions because to the activation begin and end they have additional trigger times at the begin and end. Between the trigger times and the activation times there have to be a buffer of a few frames to get a correct animation.

References

[1] Gerhard Kurka: *A fast Visualization System for Adaptive Grids*, SFB-Report 98-11, Oktober 1998.

[2] Joachim Schöberl: *FEPP – A Finite Element Package written in C++*, http://www.numa.uni-linz.ac.at/Staff/joachim/cpp/doc/index.html, 2001.

[3] Silicon Graphics: *OpenGL Reference Manual*, Addison-Wesley Publishing Company, ISBN-0-201-63276-4, 1992.

[4] Amitabh Varshney, Pankaj K. Agarwal, Frederick P. Brooks Jr., William V. Wright, Hans Weber: *Generating Levels of Detail for Large-Scale Polygonal Models*, Department of Computer Science, Duke University, North Carolina, CS-1995-20, 1995.

[5] Bjarne Stroustrup: *The C++ Programming Language*, third Edition, Addison-Wesley Publishing Company, ISBN-0-201-8895-4.

[6] Joachim Schöberl: *NETGEN*, http://www.sfb013.uni-linz.ac.at/ joachim/netgen, 2001.

[7] Electronic Visualization Laboratory: *CAVE User's Guide*, http://www.evl.uic.edu/pape/CAVE/prog/CAVEGuide.html, University of Illinois at Chicago, 1997.

[8] SFB: *Example movies*, http://www.sfb013.uni-linz.ac.at, University of Linz, 2001.