Web Based VRML Modeling

Szilárd Kiss [S.Kiss@cs.utwente.nl]
Department of Computer Science - Language, Knowledge and Interaction group,
University of Twente, Postbus 217, 7500 AE Enschede, The Netherlands

Abstract

We present a method to connect VRML and Java components in a web page using EAI which makes possible to interactively generate and edit VRML meshes. The meshes used are based on regular grids, to provide an interaction and modeling approach that uses the internal semantics of such a mesh by linking the available modeling operations either to single vertices, the vertices of a ring or column, or all the vertices from the VRML mesh. Our method permits strict mesh complexity control and scales the operations according to the mesh properties. We describe the structure of the system and provide a few examples of the meshes that were created.

Keywords: Virtual Reality, VRML, modeling, WWW.

1 Introduction and background

Our aim is to create modeling systems that can be used to produce meshes for virtual environments based on the VRML (Virtual Reality Modeling Language [10]) standard. We created a prototype system based on VRML and EAI (External Authoring Interface [2]) which allows a Java binding to the VRML world, makig possible a dynamic mesh handling approach. Instead of using conventional editing operations as described in [7] and using automatically generated complex surfaces (see [3], [4] and [9]) we use an approach based on vertices and sets of vertices grouped in rings and columns. We also would like to provide an easy to use interface, for which a good example would be the "Teddy" system [6].

2 Editor appearance

Figure 1. is a screenshot of the editor in action. The system provides a simple, easy to use interface for all the operations possible, without any elements that must be looked up from menus or lists. The visual elements are grouped and marked according to their functionality in the editor system, and their functions are presented below.

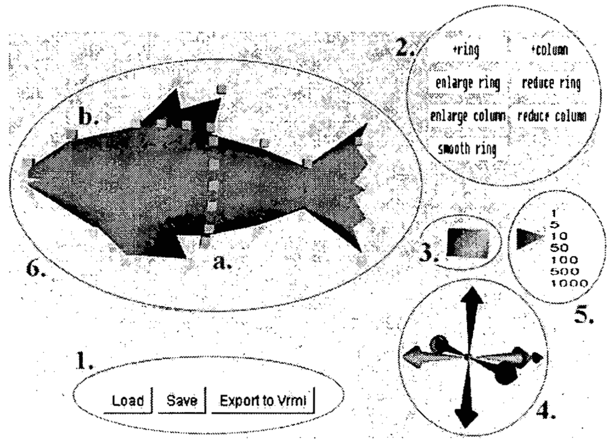

Figure 1. Annotated screenshot of the editor

1. Buttons of the Java applet for loading, saving the mesh data and exporting to VRML format. They use the familiar load/save window of the underlaying platform, provided by the Java AWT classes, and they perform file readings and writings of the internal format and file writings to VRML file format.

2. Editing buttons in the VRML environment. From left to right and from top to bottom in order: add ring and add column (which increase the mesh complexity), enlarge ring and reduce ring (for the ring-wise scaling of the mesh), enlarge column and reduce column (for the column-wise scaling, which has a global scope, affecting the whole mesh) and last the smooth ring button which is used for smoothing the mesh surface ring-wise.

3. Rotation widget (box). On the figure it has been already rotated, and the elements noted with 4. and 6. follow its rotation exactly. Used for positioning the mesh in a manner that allows to perform the desired selection of vertices and to preview the mesh from different positions.

4. Editing widget. Composited of 6 arrows, with different colour for each pair of arrows that represent the positive and negative direction of the main axes. A click on an arrow

will displace the selected vertices in the direction the arrow is pointing by the length selected using the unit selection widget (next widget).

5. Editing unit selection widget. The default value is set to 100 millimeters, this can be changed by dragging the triangle on the left. A wide range of values (from one millimeter to one meter) gives the possibility to handle meshes at different scales, and to perform scale transitions easily.

6. Edited mesh. a) is the current selection, a ring, while b) is a column which is not yet selected, but the pointing device is over the column sensor, and the column is highlighted as a possible next selection.

3 Data

The editor uses an internal format for the storing of mesh data. This is done for two reasons. First, there is a data parsing reason, meaning that a simple data format would allow a more comprehensible parsing method. Second, an internal format ensures no complications regarding the use of a data file with possibly non-regular structure. The latter would not be anyway a good idea (to create the mesh elsewhere and import it in the application) as the goal is to use as simple meshes as possible. Users are not required to edit this data manually as it is done generally in VRML. This it would be a low-level editing intervention, and the application provides a visual and also low-level approach to editing. Although manual edit is certainly possible, it is more cumbersome than the possibilities offered by the editor, for instance to select a single vertex, and move it in any of 6 degrees of freedom even with a displacement unit of one millimeter. Of course, the same operation can be effectuated more globally, on a set of selected vertices.

4 Structure of the editor

We have already mentioned that the editor uses the combination of VRML, Java and EAI technologies to provide it's functionality. Since these are distinct technologies, and the EAI component handles all the interaction between the VRML and Java components, it is possible to discuss these components separately. Taking in account that the communication between the components is a two-way communication, it is best to show it in the flow diagram from figure 2. and explain it's mechanisms step by step.

The VRML component is a VRML file which at the beginning contains no mesh, just an anchor point for binding the generated mesh geometry into the scene graph (this will be loaded and generated at user request). The file does contain beside the anchor point for geometry a set of "widgets" that are tools inside the virtual environment that aid in the editing process. Depending on their functionality, these widgets act directly upon the virtual environment, or indirectly by sending their events to be handled by the Java applet, which in turn passes modified data back to the VRML component. This second component (the Java applet) is used for the data handling necessary for opening, generating, modifying and saving meshes. This applet is used also for translating into the environment the effects of user actions performed through some of the widget tools.

These two components that work together using EAI are packed in an HTML file for presentation in a web browser. As they are side by side in a common environment (that of a web browser), it is possible to use code that enables the communication between the Java and VRML components, and this code is the EAI.

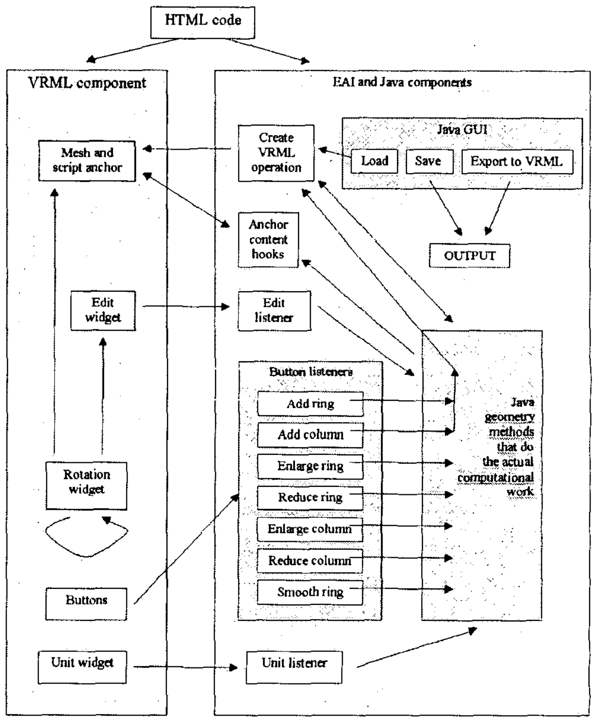

Figure 2. Flow graph

4.1 EAI component

This is the simplest component of the structure, it connects the virtual environment and the applet that are inside the same web page. It gives the possibility for the Java methods to access the VRML content, and does this by providing Java wrapper objects around the internal VRML browser, nodes and fields. As it can be observed from the figure 2. there are a multitude of connections between the

VRML and Java components, and yet these are not all. These are in fact representing only conceptual connections, and the number of real connections are much higher, since the VRML "objects" (single nodes, grouped nodes or new prototypes) usually have multiple event sending/receiving possibilities, which have to be used to provide interaction possibilities to the system.

4.2 VRML component

The VRML component provides the "graphics engine" for the editor. It has the role of visualizing the mesh and all the user controls in real-time. It allows the run-time linking of code into the system, which permits truly dynamical interaction to be built.

In the following we will present the elements of the virtual environment one by one, this will allow a more accurate view over the component. We start with the VRML prototypes, which are custom groups of nodes generally used to store reusable code (groups of nodes in this case), then we present the scene graph, how these elements build up the environment.

4.2.1 VRML "PROTO" prototypes

The first reusable code segment inside the VRML file is a HUD (Heads Up Display) prototype. This prototype is used generally for creating custom display panels inside the virtual environment, for instance a custom navigation bar, or a tool-bar. Its purpose is to mirror the effect of user movement on these panels, in other words keeping the panel unchanged, in its initial position on the screen, indifferently how the user navigates in the environment. This particular HUD prototype accepts any subtree of nodes as parameter. Therefore it can be used for different purposes, as we will see when discussing the other elements of the VRML component.

The second prototype is a geometrical assemble, representing a three-dimensional arrow. Its functionality is related for specifying the DOF (or axis) operations, which will be discussed as the prototype is used in the scenegraph (section 4.2.2).

The last prototype of the application is a button prototype. It consists of a small rectangle, on top of which a picture is mapped, picture that explains in text (this is the method currently used) the functionality of that particular button. Its main purpose is however the capability to track user actions with a sensor and the capability of switching the button on/off.

4.2.2 Scene graph

After the prototype declaration and the usual general settings code of a VRML file, the editing environment is created using the prototypes defined earlier and some additional nodes. First, an editing "widget" (or tool) is created. This is embedded in a HUD instance, thus user navigation does not affect it. The widget itself consists of 6 instantiations of the arrow widget, sensors attached to all of them, and a script that handles the events of the sensors. The 6 arrows represent the 6 DOFs of a three dimensional space, the +X, -X, +Y, -Y, +Z, and -Z directions. As a visual aid, X directions are colored red, Y directions green and Z directions blue (as a sort of direct mapping between the enumeration of the three dimensional coordinates and the enumeration of the three basic RGB color components). Their purpose is to collect the user's displacement commands regarding the selected mesh vertices. As the user clicks on such an arrow, the sensor attached to the arrow sends a "clicked" message to the script. Each arrow has this functionality, and the script, depending on which arrow was activated and depending on the current displacement unit, will issue a displacement event containing the value of displacement. This value is catched by an event listener defined in the Java component.

The next elements of the virtual environment are seven instantiated buttons. Their functionality is to transmit their activation status to functions from the Java component that will effectuate the actions related to the buttons. The button functionalities are:

- adding a ring or a column. These button trigger Java methods that add a new ring or new column right after the current selection. The order of the rings is defined by the data enumeration order. The column order is calculated from the data using the right hand rule, with the thumb representing the ring data direction and the remaining bent fingers representing the vertex enumeration direction.
- positive or negative ring scale. The selected ring is enlarged or reduced by adding respectively substracting the value of the displacement unit vector to/from the selected vertex values. Ring scaling introduces changes in the object thickness. The buttons are not active if a column is selected.
- positive or negative column scale. Unlike the ring scale operations that are acting on a subset of mesh vertices (rings), a column scale operation acts on every vertex of the mesh. A positive scale in this case means that the object will become a longer (taller) object and a negative scale that the object will become shorter, while the thickness does not change. Scale column operations are distributed between the mesh rings based on percentages calculated from the ring positions relative to the mesh center, this way the mesh will retain its original shape.

- smooth ring. A fraction of the relative distances of vertices from the center point of the mesh are used to smooth the contour line of the ring vertices. Eventually, with enough smoothing operations, the vertices will approximate the same distance from the center point, as if they were on the same circle.

Another element embedded in a HUD is a container element, which at start is empty, and is filled by the code generated by the Java component. The container is used only as a linking point, and contains no other functionality. The functionalities will be added at the same time with the generated geometry as the Java component generates the code simultaneously and links it to the VRML container element as one piece of code.

The next item is the rotating widget or tool. The tool is represented as a three dimensional box, which has a rotation sensor linked to it. There is a backdraw to this construction, which lies in the mapping of the two-dimensional input device (the mouse) to three-dimensional manipulation. However, as the user becomes acquainted with this particular manipulation possibility, it can provide enough control for editing purposes. The function of this widget is to enable the viewing of the mesh from all angles and also the favorable positioning of the mesh in the course of the editing process. As the user drags the rotating widget, the same rotation is conveyed to the edited mesh container (and consequently to the edited mesh itself) and to the editing arrows. This way, not only the mesh is positioned, but the axes of the editing arrows and the actual axes of the vertices remain aligned.

The last element of the environment is an editing unit selection slider. The user has the possibility to select the edit unit between the 1 mm, 5 mm, 10 mm, 50 mm, 100 mm, 500 mm and 1000 mm values, allowing a wide scale of the meshes to be edited. A triangle is pointing to the current value, represented as a panel with the enumerated values written on it. By dragging the triangle, these edit values can be selected.

4.3 Java component

The Java component is the part of the editor that essentially does all the work related to the interaction that occurs inside the editor. It receives the events of the VRML component, effectuates the functions/commands associated with the particular events and it returns the result in some form to the environment.

4.3.1 Connections to the elements of the VRML component

To have access to the VRML component, the Java component uses code from the EAI packages. This way, it can get hold of the objects inside the virtual environment. It is not necessary to explain how exactly this has to be accomplished, a list of connections and their purpose should give enough insight to the event flow represented in figure 2.

Not all the connections will be set up when the environment is created. To be able to communicate fully with the VRML component, the dynamic code created by the system must also be linked with the Java component. This is done using the EAI in the same manner as used for the static code of the VRML component, after each VRML code generating operation.

Below is the list of connections made with the EAI:

- connection to the browser. Without this, the environment cannot be accessed at all by the Java component.
- connection to the editing buttons, to their generated events, and to their active/inactive properties.
- a mesh node and a coordinate setting event. The mesh node will always hold the VRML representation of the edited mesh, and the coordinate setting event is used to refresh the visualized geometry.
- selector setting event. As the coordinates change, the position of sensor boxes must also change to provide an accurate representation for the location of mesh vertices.
- events such as those for removing the generated geometry (before the new geometry is created) and centering, scaling or translating the mesh (to provide a maximal view of the mesh).
- events to transfer vertex selection statuses (a single vertex, a ring of vertices or a column of vertices can be selected at a time).
- events to handle the editing unit selection process, which sets the magnitude of the editing operations.

Connections are made for all the nodes and events enumerated above. To each event there is a serial number assigned, which makes it possible for a callback method to identify the source event and dependig on what this event's function is, it handles the event itself, or in case the event is more complicated, it triggers the method responsible for handling that particular event.

4.3.2 File operations

The first file handling method is the file reading (loading) method. If the data file is opened succesfully, its content is read into a string variable. A parsing method extracts the relevant data from the string and sets up the data parameters used by the Java component, like number of rings, number of columns. It also creates a dynamical parameter depending on the previous values to store the vertex coordinates (the number of vertex coordinates is seldomly the same, therefore the variable holding this data is always created dynamically). At the end, the method calls the VRML code generation method described next, sets up initial values for a number of parameters for data consistency, and calls VRML-Java connection initializing methods for the newly generated VRML content.

The remaining two file operation methods deal with the storage of the edited mesh. The save method opens a window to specify the desired name and directory of the file to be saved, then it constructs the internal data file format and fills it with the required data extracted from the Java component variables. The last method, exporting onto VRML format resembles the previous method, only there is extra data to be generated beside the geometry (header, settings nodes), and the vertex data must be accompanied by mesh facets indexing data, extracted also from data variables stored inside the Java component.

4.3.3 Generating VRML code

The VRML code creation method is responsible for creating the mesh geometry, the selection geometry and sensors, the script code that handles the selection process internally in the VRML component, and the events used for communicating with the Java component. It does all this by creating a single string of code, and sending it to the VRML component to be integrated in it. We call this VRML code dynamical because it is generated on the fly (multiple times during the editing process, when the mesh changes complexity) and it is modified constantly, with every operation commited. This dynamic code generation process has the following steps:

- Creating the indexes for the mesh geometry. These indexes specify the polygons of the mesh. Since a regular grid of vertices is used which provides a logical connection between the vertices, every item (cell) of a grid made from the vertices can be covered algorithmically by two triangle facets.

- Creating the mesh geometry string itself, an IndexedFaceSet VRML node, instantiating the data from the parsed vertex coordinates and using the previously created triangle indexes.

- The next step is the creation of sensors, handlers and a script with functions (events) to manage the properties and events of vertex operations. They will be created for *each* of the vertices from the mesh.

- Creating the code that handles the vertex grouping into rings or columns. It contains a line-set geometry, which connects the vertices of the ring respectively column together, reusing the same coordinates the mesh uses. A touch sensor is connected to the line-set, which triggers the behaviors of the individual box sensors within the line-set.

4.3.4 Geometry operations

The most frequently used method is a method witch catches the displacement event transmitted from the VRML component. In function of the current selection, applies the displacement to the data stored in the Java component and sends that data back to the VRML component. It also sends the same displacements to the sensor box(es) assigned to the selected vertex or vertices.

The size of a sensor box assigned to a certain vertex depends on the distance of the vertex from its neighbors. This way, sensors do not overlap, and remain scaled according to the complexity and the size of the edited mesh geometry.

A mesh positioning method provides an interface where the visibility of the edited mesh is maximal, being entirely visible and without occluding by rotation the other elements of the interface. It scales and translates the geometry to a convenient position.

The ring adding method inserts a new ring in order after the selected ring (or after the ring containing the selected vertex), except when the selection is the last ring, when no action is performed. The data for the new vertices are calculated as the mean values of the vertices from the neighboring rings, and the dimension of the vertex data variable is changed to be able to hold the new values along the old values.

The column addition method is essentially the same, the only difference between the two methods lies in the structure of the vertices, and there is no restriction in adding a column after the last column like in the case of adding rings.

If the ring smmothing method is activated, it calculates the center of the current ring and depending on the distance of vertices from the center and from the mean center distance, the vertex values are increased or decreased to achieve smoother surfaces. The new data is transmitted then to the VRML component.

The ring scaling method affects the vertices of the current ring. For every vertex, distances from the center position are calculated in the horizontal (XZ) plane, and the scale value (the current editing unit is used as the scaling value) is distributed to the vertex X and Y values proportional to the previous values.

Column scaling is different in functionality from ring scaling. While ring scaling acts locally, only affecting one ring of vertices, column scaling affects all vertices of the mesh. It can be used for enlarging the distance between the ring vertices, thus longer/taller meshes can be obtained. A column centerpoint is calculated (from the current column) and depending on the distances of the current column vertices from the center point, each ring is displaced proportionally by a certain calculated fraction of the scale value.

5 Symmetry

The editor was continuously tested and modified during the developing process to eliminate undesirable effects or enhance its usability. At first the symmetry which is present in many real-world objects had to be introduced by hand to the mesh, making the user's task unnecessarely complicated. To eliminate this overhead with symmetric object modeling, the system was modified to provide this symmetric editing functionality. The mesh, according to this approach, is edited only on one side of the symmetry plane, the other side follows the changes automatically. This method yields in a few advantages. With fewer control points, the size of the generated VRML code drops also, thus gain in speed can be achieved. By using one side of the object for editing, the other side can be used for preview purposes, as selections are not visible on that side. And as a separate preview side is available, the edited side can use a semi-transparent appearance for non-selected controls, giving more comprehensibility to the mesh structure.

6 Conclusions

The editor in its current state gives a high level of control over a loaded mesh, making possible precise editing, new mesh construction, instant visibility and it is web deployable. However, the system can model only single meshes. Our intent is to extend it to be able to model hierarchies of meshes, thus being able to model complex meshes such as the H-anim [5] hierarchy or other hierarchies based on bone structures [1] to produce the avatar and agent meshes needed for the virtual environment experiments our group is working on [8]. Figure 3. shows a few models created with our system.

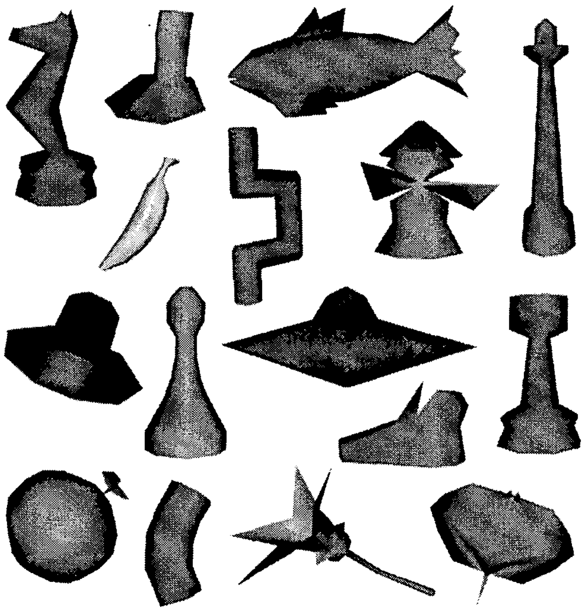
Figure 3. Example models

References

[1] N. I. Badler, C. B. Phillips, and B. L. Webber. *Simulating Humans: Computer Graphics, Animation, and Control.* Oxford University Press, 1993.

[2] Chris Marrin - Silicon Graphics, Inc. External Authoring Interface Reference. http://www.graphcomp.com/info/specs/eai.html, 1997. Proposal for a VRML 2.0 Informative Annex.

[3] Y. Goto and A. Pasko. Interactive Modeling of Convolution Surfaces with an Extendable User Interface. In A. de Sousa and J. C. Torres, editors, *EuroGraphics 2000 - Short presentations*, pages 37–42. EuroGraphics, 2000.

[4] H. Grahn, T. Volk, and H. J. Wolters. NURBS in VRML. In *Proceedings of the Web3D-VRML 2000 Fifth Symposium on Virtual Reality Modeling Language*, pages 35–43. ACM Press, 2000.

[5] Humanoid Animation Working Group - Web3D Consortium. H-Anim: Specification for a Standard VRML Humanoid. http://ece.uwaterloo.ca/ h-anim/spec1.1/index.html.

[6] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A Sketching Interface for 3D Freeform Design. In *Proceedings of the SIGGRAPH 1999 Annual Conference on Computer Graphics*, pages 409–416. ACM Press, 1999.

[7] G. Maestri. *Digital Character Animation 2*, volume 1 - Essential Techniques. New Riders Publishing, 1999.

[8] A. Nijholt and H. Hondorp. Towards Communicating Agents and Avatars in Virtual Worlds. In A. de Sousa and J. C. Torres, editors, *EuroGraphics 2000 - Short presentations*, pages 91–95. EuroGraphics, 2000.

[9] A. Wakita, M. Yajima, T. Harada, H. Toriya, and H. Chiyokura. XVL: A Compact And Qualified 3D Representation With Lattice Mesh and Surface for the Internet. In *Proceedings of the Web3D-VRML 2000 Fifth Symposium on Virtual Reality Modeling Language*, pages 45–51. ACM Press, 2000.

[10] Web3D Consortium. VRML97: The Virtual Reality Modeling Language. http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm, 1997.